MME User Guide

(CRAY T90™ Series)

HDM-102-B

Cray Research/Silicon Graphics Proprietary

CRAY
RESEARCH
A Silicon Graphics Company

Record of Revision

REVISION	DESCRIPTION
	August 1995. Original printing.
A	March 1996. This revision corresponds to the MT-T2.2.0 offline diagnostic release.
B	August 1997. This revision corresponds to the MT-T2.3.0 offline diagnostic release.

Any shipment to a country outside of the United States requires a letter of assurance from Cray Research, Inc.

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

CRAY RESEARCH, INC.
Customer Service Logistics
1100 Lowater Road
P.O. Box 4000
Chippewa Falls, WI 54729-0078
USA

Comments about this publication should be directed to:

CRAY RESEARCH, INC.
Service Publications and Training
890 Industrial Blvd.
P.O. Box 4000
Chippewa Falls, WI 54729-0078
USA

MME USER GUIDE

ENVIRONMENT 0 (continued)

ENVIRONMENT 0 (continued)

ENVIRONMENT 1 77

Figures

Figures (continued)

Tables

Tables (continued)

Description of this Document

This document provides procedures that describe how to use Mainframe Maintenance Environment (MME) environments 0, 1, and 2 to troubleshoot CRAY T90™ series mainframes.

This document is one component of the MME documentation set, which also includes the following documents:

MME Interface Reference, publication number HDM-008-A.

> This document describes the interfaces used with MME environments 0, 1, and 2. It also describes all available menu button commands.

MME Diagnostic Tests and Utilities, publication number HDM-103-B.

> This document provides quick-reference information for all diagnostic tests and utilities you can use with MME.

ENVIRONMENT 0

Environment 0 is one component of the Mainframe Maintenance Environment (MME) software package that field engineers use to troubleshoot CRAY T90 series mainframes. Environment 0 provides basic mainframe testing; use environment 0 to ensure that the mainframe is operating at a level that permits environment 1 and environment 2 based testing.

Environment 0 runs in the maintenance workstation (MWS) or system workstation (SWS) and creates maintenance channel functions that are sent to the mainframe through the maintenance channel to test the following areas of the mainframe:

- Boundary scan communication
- Configuration
- Memory
- I/O error correction
- Logic monitor
- Exchange
- Instruction buffers
- Miscellaneous

Environment 0 comprises three testing modes (automatic, manual, and compose) for varying levels of user control:

- Automatic mode runs predefined sequences of maintenance channel functions.

- Manual mode runs user-selected sequences from the predefined set with user-selected parameters.

- Compose mode runs user-defined sequences of maintenance channel functions. This enables testing beyond the areas tested in automatic and manual modes but requires you to create or modify the sequence that is sent to the mainframe.

Compose mode also enables you to view and modify the predefined sequences that are used in automatic and manual modes. You can run these modified sequences or save them for later use. Normally, you should use compose mode to examine or modify existing sequences rather than create new ones because creating new sequences requires a detailed understanding of the maintenance channel functions.

Environment 0 uses a 256-Kword data buffer (64-bit words) in the MWS or SWS that is called the MME buffer. This buffer collects data coming from the mainframe through the maintenance channel and creates data blocks that are sent to the mainframe through the maintenance channel. The MME buffer also stores data that is used for comparisons of actual and expected data.

This section describes how to start environment 0 and use automatic, manual, and compose modes to test the mainframe.

Starting MME in Environment 0

You can start MME in environment 0 from a UNIX® command prompt or from the OpenWindows™ Workspace menu.

NOTE: For information about starting MME environment 0 from a Service Center through a hub, refer to the *Remote Support* document, publication number HMM-106-A.

CAUTION

MME performs maintenance channel functions that will hang UNICOS if UNICOS is running in the mainframe when you start MME.

To prevent this from accidentally occurring, ensure that the Owner setting in the SCE base window is set to OS for the logical partition in which UNICOS is running when UNICOS is running in the mainframe. MME cannot access a logical partition if the OS owns it.

From a UNIX Prompt

To start MME environment 0 from a UNIX prompt, enter one of the following commands:

• **`mme –0`**	to use a front-end interface (FEI) channel
• **`mme –0 –sim`**	to use the simulator
• **`mme –0 –debug`**	to use the simulator and bugger/debugger

NOTE: You may also enter any of the command line options that Table 1 lists.

Table 1. Environment 0 Command Line Options

Option	Description
`–client`	Start the MME client only
`–config` *file*	Configure MME with the configuration data that is stored in the file specified by *file*
`–copy` *num*	Connect to maintenance software that is assigned the copy number specified by *num* **NOTE:** Copy numbers are necessary only when you run multiple copies of MME on the same MWS or SWS (for example, when you run several MME copies with the simulator or when you use MME to support multiple CRAY T90 series mainframes that are connected to the same MWS or SWS).
`–io` *num*	Use the CPU specified by *num* to perform input and output operations
`–kill`	Kill any running MME, SCE, or LME applications before starting a new copy of MME
`–remote` *host*	Start the MME client only and connect the client to the MME server that is running on the remote host specified by *host*
`–server`	Start the MME server only

From the OpenWindows Workspace Menu

You can start environment 0 from the OpenWindows Workspace menu on either an MWS or an SWS.

MWS Workspace Menu Options

Figure 1 shows the OpenWindows Workspace menu options that you should choose on an MWS to start environment 0 with an FEI channel. Choose any copy number.

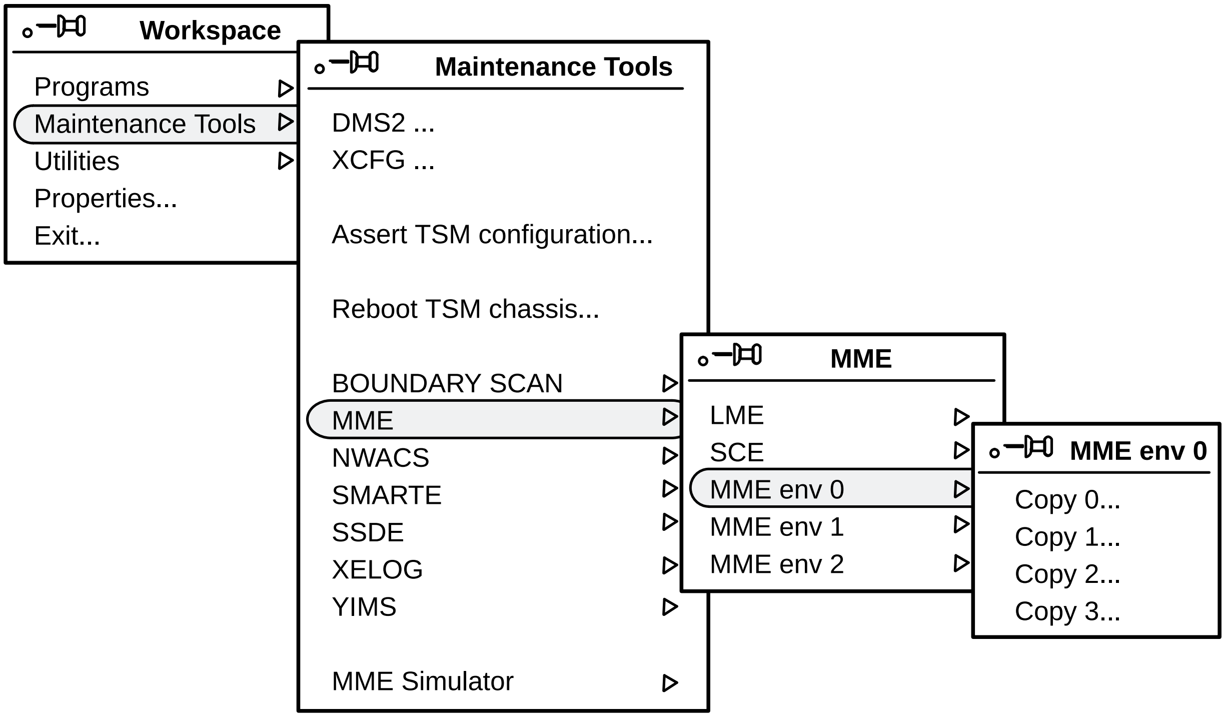


Figure 1. MWS Workspace Menu Options to Start Environment 0 with an FEI Channel

Figure 2 shows the OpenWindows Workspace menu options that you should choose on an MWS to start environment 0 with the simulator or with the simulator and bugger/debugger.

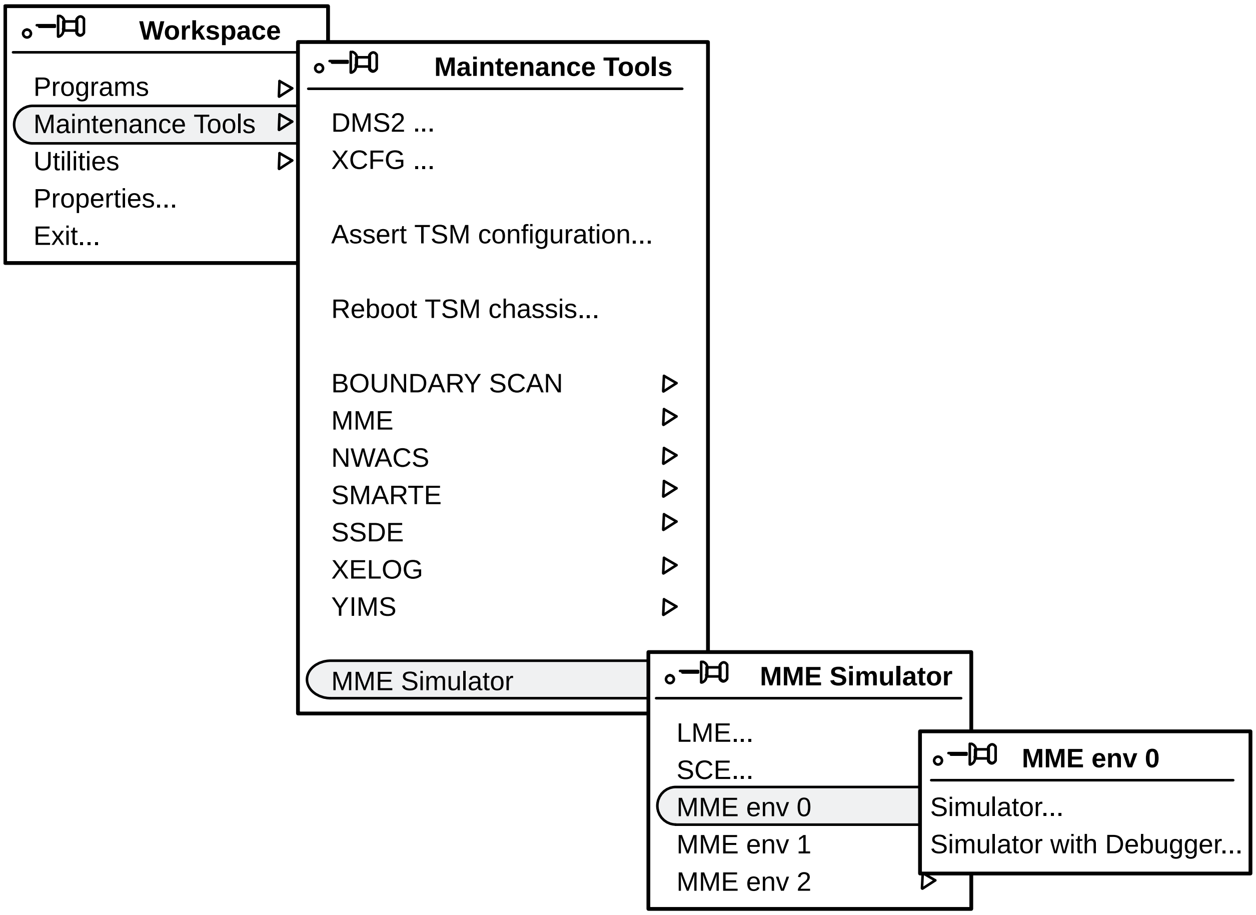


Figure 2. MWS Workspace Menu Options to Start Environment 0 with the Simulator or with the Simulator and Bugger/Debugger

SWS Workspace Menu Options

Figure 3 shows the OpenWindows Workspace menu options that you should choose on an SWS to start environment 0 with an FEI channel. Choose any copy number.

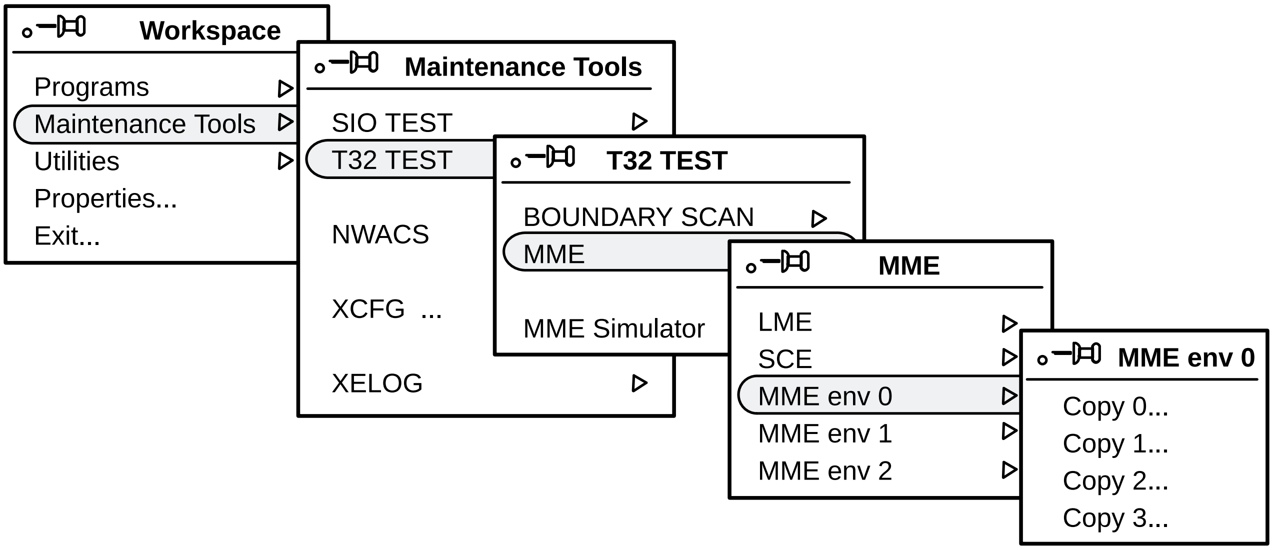


Figure 3. SWS Workspace Menu Options to Start Environment 0 with an FEI Channel

Figure 4 shows the OpenWindows Workspace menu options that you should choose on an SWS to start environment 0 with the simulator or with the simulator and bugger/debugger.

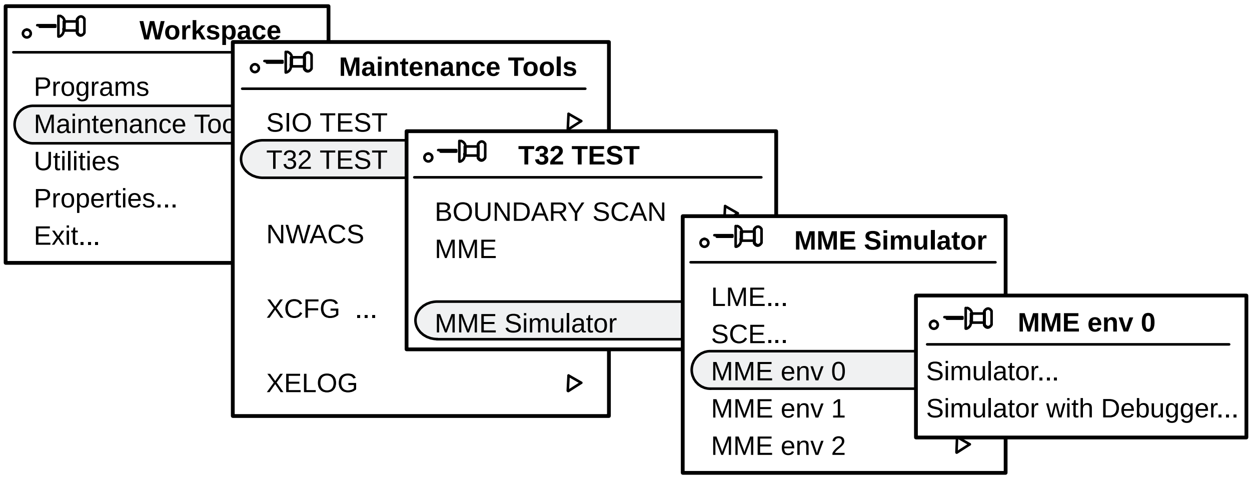


Figure 4. SWS Workspace Menu Options to Start Environment 0 with the Simulator or with the Simulator and Bugger/Debugger

What Happens When You Start Environment 0?

The following actions occur when you start MME:

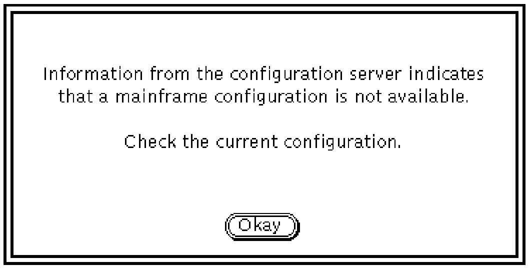
1. The MME server attempts to connect with the System Configuration Environment (SCE) server.

   If MME cannot connect with a running SCE server, MME starts a new SCE server and tries to connect to the new SCE server. (If you specified a configuration file with the `–config` command line option, MME sends this file to SCE through the SCE `–default` command line option. SCE loads the configuration that is stored in the file.)

2. Once MME establishes a connection with SCE, MME attempts to receive a configuration from SCE:

   - If a configuration is available, SCE provides MME with the components that are available for use by the maintenance system. MME automatically configures itself to use these components.

   - If a configuration is not available, MME displays the message shown in the following snap:



Information from the configuration server indicates
that a mainframe configuration is not available.

Check the current configuration.

   If MME displays this message, then you need to create a configuration using SCE before you continue using MME. Refer to the *SCE User Guide*, publication number HDM-069-C, for more information about creating a configuration.

Using Automatic Mode

When you click on Test Mode: Automatic, environment 0 runs in automatic mode. Automatic mode enables you to run all or any combination of the environment 0 tests. Perform the following procedure to run tests in automatic mode:

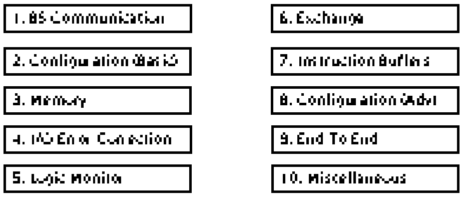
1. Click on Test Mode: Automatic to indicate that you want to run the test(s) in automatic mode.

2. Click on the modules that you want to assign to the tests. The selected tests are run on these modules.

   For information about which modules can be tested by the tests, refer to the “Environment 0 Tests” section of the *MME Diagnostic Tests and Utilities* document, publication number HDM-102-B.

   Click on Select All Modules to select all valid modules in the current configuration. Click on Deselect All Modules to deselect all modules that are currently selected.

3. Click on one or more of the test settings:

   Click on Select All Tests to select all of the tests. Click on Deselect All Tests to deselect all tests.

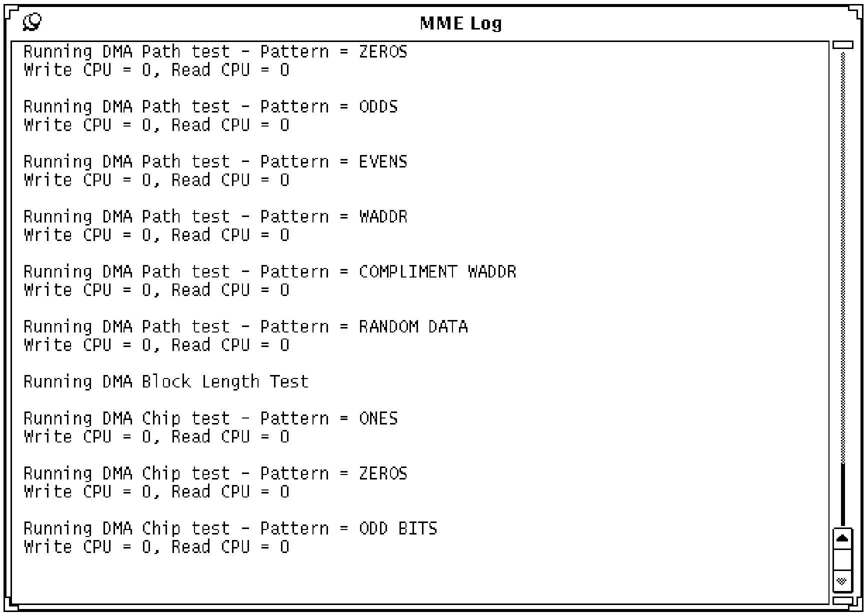
4. Specify an Error Mode:

   Click on Error Mode: Stop On Channel Error to stop testing when a channel error occurs. Click on Error Mode: Stop On Sequence Error to stop testing when a sequence error occurs.

   Use this option to isolate an error when it occurs. After the error occurs, click on Test Mode: Compose; the failing function is highlighted in the Sequence scroll box in compose mode.

5. Click on Go ▷; MME executes the specified tests.

6. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.



```
Running DMA Path test - Pattern = ZEROS
Write CPU = 0, Read CPU = 0

Running DMA Path test - Pattern = ODDS
Write CPU = 0, Read CPU = 0

Running DMA Path test - Pattern = EVENS
Write CPU = 0, Read CPU = 0

Running DMA Path test - Pattern = WADDR
Write CPU = 0, Read CPU = 0

Running DMA Path test - Pattern = COMPLIMENT WADDR
Write CPU = 0, Read CPU = 0

Running DMA Path test - Pattern = RANDOM DATA
Write CPU = 0, Read CPU = 0

Running DMA Block Length Test

Running DMA Chip test - Pattern = ONES
Write CPU = 0, Read CPU = 0

Running DMA Chip test - Pattern = ZEROS
Write CPU = 0, Read CPU = 0

Running DMA Chip test - Pattern = ODD BITS
Write CPU = 0, Read CPU = 0
```

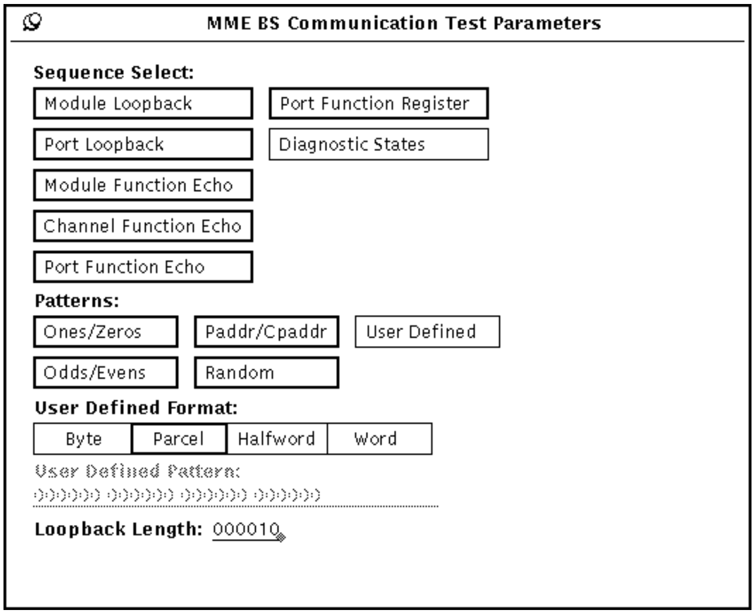
Using Manual Mode

When you click on Test Mode: Manual, environment 0 runs in manual mode. Manual mode enables you to select which sequences of the predefined tests will run. This enables you to isolate certain areas for testing. In manual mode, you can run only one test at a time. The following procedures describe how to run each environment 0 test in manual mode.

Running the Boundary Scan (BS) Communication Test

Perform the following procedure to run the BS communication test:

1. Click on Test Mode: Manual to enter manual mode.
2. Click on the BS module(s) that you want to test.
3. Click on Tests: 1. BS Communication to select the boundary scan test. The MME Boundary Scan Test Parameters window appears:



Loopback Length: 000010

4. Click on the sequences that you want to run. The sequences perform the following functions:

Sequence:	**Description:**
Module Loopback	This sequence loops back module data.

Sequence:	**Description:**
Port Loopback	This sequence loops back port data.
Module Function Echo	This sequence echoes the module function word.
Channel Function Echo	This sequence echoes the channel function word.
Port Function Echo	This sequence echoes the port function word.
Port Function Register	This sequence loads the port register and reads the value back.
Diagnostic Status	This sequence checks front-end interface (FEI) errors, such as parity errors on the channel.

5. If you are running a loop-back sequence, an echo sequence, or a port function register sequence, click on the patterns that you want to use for testing:

Pattern:	**Description:**
Ones/Zeros	The sequence uses 000000_8 and 177777_8 parcel patterns.
Odds/Evens	The sequence uses 125252_8 and 052525_8 parcel patterns.
Paddr/Cpaddr	The sequence uses parcel address and complement parcel address patterns.
Random	The sequence uses random data parcel patterns.
User Defined	The sequence uses user-specified parcel patterns. Specify the format (click on User Defined Format: Byte, Parcel, Halfword, or Word). In the User Defined Pattern field, enter the pattern that you want to use.

6. In the Loopback Length field, enter the length of the data block that you want to loop back if you are running a loop-back sequence.

7. Click on Go ▷; MME tests the selected sequences.

8. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

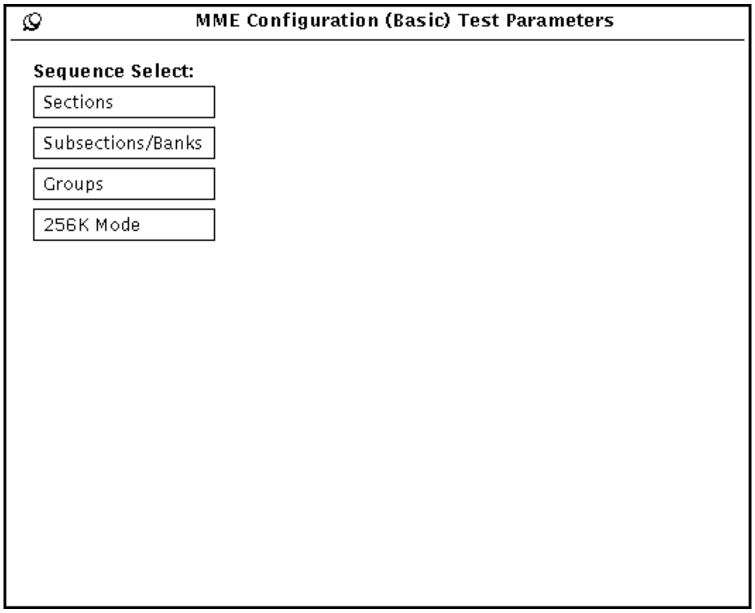
Running the Configuration (Basic) Test

Perform the following procedure to run the configuration (basic) test in manual mode:

1. Click on Test Mode: Manual to enter manual mode.

2. Click on the CPU module(s) that you want to test.

3. Click on Tests: 2. Configuration (Basic) to select the basic configuration test. The MME Configuration (Basic) Test Parameters window appears:



4. Click on the sequence that you want to test:

Sequence:	**Description:**
Sections	This sequence checks all memory section configuration settings.

Sequence:	**Description:**
Subsections/Banks	This sequence checks all memory subsection and bank configuration settings.
Groups	This sequence checks all memory group configuration settings.
256K Mode	This sequence checks CPU memory addressing when a CPU is configured in upper 256-Kword addressing mode.

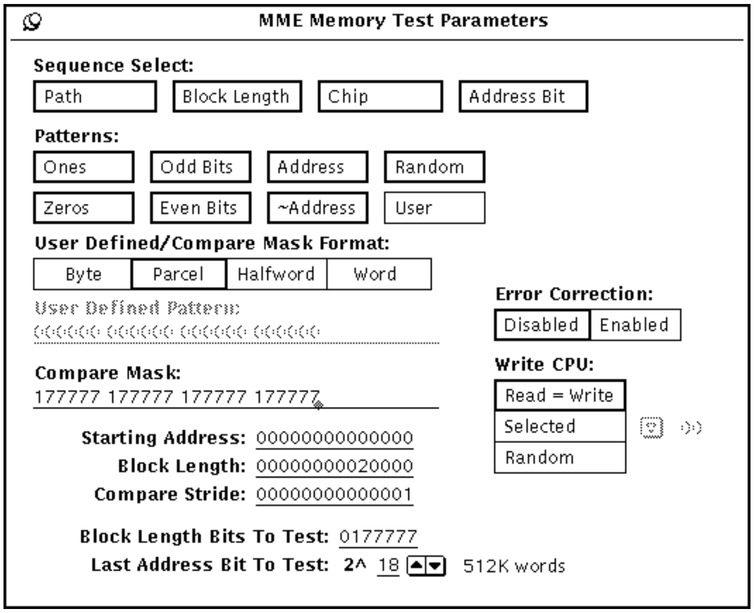
5. Click on (Go ▷); MME runs the selected test sequence(s).

6. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

Running the Memory Test

Perform the following procedure to run the memory test in manual mode:

1. Click on Test Mode: Manual to enter manual mode.
2. Click on the CPU module(s) that you want to test.
3. Click on Tests: 3. Memory to select the memory test. The MME Memory Test Parameters window appears:



MME Memory Test Parameters

User Defined Pattern: 000000 000000 000000 000000

Compare Mask: 177777 177777 177777 177777

Starting Address: 00000000000000

Block Length: 00000000020000

Compare Stride: 00000000000001

Block Length Bits To Test: 0177777

Last Address Bit To Test: 2^ 18 512K words

4. Click on the sequence(s) that you want to use:

Sequence:	**Description:**
Path	This sequence writes the MME buffer with the selected pattern, writes the MME buffer data to memory with error correction enabled, reads the memory data back to the MME buffer, and compares sent and received data.

Sequence: **Description:**

Block Length — This sequence tests the block length of DMA write and read function words.

Chip — This sequence writes the MME buffer with the selected pattern, writes the MME buffer data to memory with error correction disabled, reads the memory data back to the MME buffer, and compares the sent and received data.

In the Starting Address field, enter the starting address of the data block. In the Block Length field, enter the length of the data block.

Address Bit — This sequence tests the address bits in a DMA function word.

In the Last Address Bit To Test field, specify the number of the last address bit that you want to test. The memory size that you have selected to test appears to the right of the field.

5. If you are using the Path or Chip sequences, you need to specify the data pattern(s). Click on the pattern(s) that you want to use:

Pattern:	Description:
Ones	The sequence uses a 177777_8 parcel pattern.
Zeros	The sequence uses a 000000_8 parcel pattern.
Odd Bits	The sequence uses a 125252_8 parcel pattern.
Even Bits	The sequence uses a 052525_8 parcel pattern.
Address	The sequence uses an address parcel pattern. (Memory locations are written with their addresses.)
~Address	The sequence uses a complement address parcel pattern. (Memory locations are written with the complements of their addresses.)

Pattern:	**Description:**
Random	The sequence uses a random data parcel pattern.
User	The sequence uses a user-defined parcel pattern. Click on User Defined/Compare Mask Format: Byte, Parcel, Halfword, or Word setting. In the User Defined Pattern field, enter the pattern that you want to use.

6. In the Compare Mask field, enter a mask to indicate the bit positions that you want to compare (0_2 = do not compare bit position; 1_2 = compare bit position).

7. If you are using the Chip sequence, specify the following items:

   - Starting address: in the Starting Address field, enter the starting address that you want to use.

   - Block Length: in the Block Length field, enter the block length that you want to use.

   - Error correction mode: error correction is disabled by default. If you want to enable error correction, click on Error Correction: Enabled.

   - Write CPU: specify the CPU that writes the data by clicking on one of the following settings:

     - Write CPU: Read = Write to use the same CPU to read and write the data,

     - Write CPU: Selected to specify which CPU writes the data (choose the CPU from ▽), or

     - Write CPU: Random to randomly choose the CPU that writes the data.

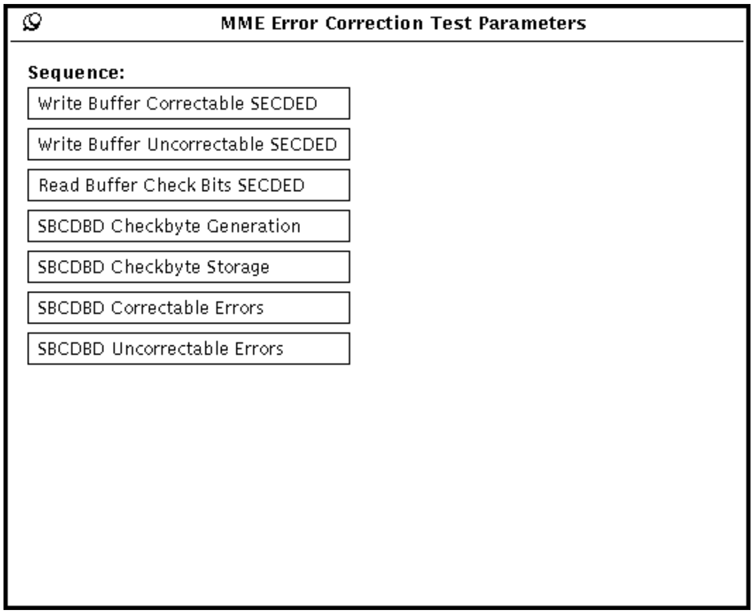
8. Click on Go ▷; MME tests the selected patterns.

9. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

Running the Input and Output (I/O) Error Correction Test

Perform the following procedure to run the I/O error correction test in manual mode:

1. Click on Test Mode: Manual to enter manual mode.
2. Click on the CPU module(s) that you want to test.
3. Click on Tests: 4. I/O Error Correction to select the I/O error correction test. The MME Error Correction Test Parameters window appears:



MME Error Correction Test Parameters

Sequence:

- Write Buffer Correctable SECDED
- Write Buffer Uncorrectable SECDED
- Read Buffer Check Bits SECDED
- SBCDBD Checkbyte Generation
- SBCDBD Checkbyte Storage
- SBCDBD Correctable Errors
- SBCDBD Uncorrectable Errors

4. Click on the sequence(s) that you want to use:

Sequence:	**Description:**
Write Buffer Correctable SECDED	This sequence writes data that forces a correctable single-error correction/ double-error detection (SECDED) error and verifies that the data error is corrected.
Write Buffer Uncorrectable SECDED	This sequence writes data that forces an uncorrectable SECDED error and verifies that the data error is detected.

Sequence:	**Description:**
Read Buffer Check Bits SECDED	This sequence writes different data patterns and reads and verifies the check bits.
SBCDBD Checkbyte Generation	This sequence writes different data patterns and reads and verifies the checkbytes.
SBCDBD Checkbyte Storage	This sequence writes a series of checkbytes, reads the checkbytes back, and verifies the checkbyte storage mechanism.
SBCDBD Correctable Errors	This sequence writes data that forces a correctable single-byte correction/ double-byte detection (SBCDBD) error and verifies that the data error is corrected.
SBCDBD Uncorrectable Errors	This sequence writes data that forces an uncorrectable SBCDBD error and verifies that the data error is detected.

5. Click on (Go ▷); MME runs the selected sequences.

6. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

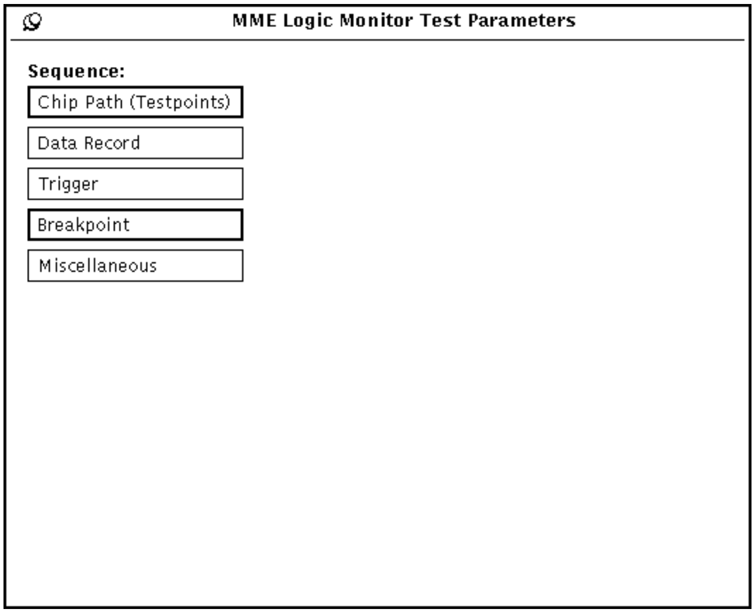
Running the Logic Monitor Test

Perform the following procedure to run the logic monitor test in manual mode:

1. Click on Test Mode: Manual to enter manual mode.

2. Click on the CPU, I/O, and/or SHR module(s) that you want to test.

3. Click on Tests: 5. Logic Monitor to select the logic monitor test. The MME Logic Monitor Parameters window appears.



4. Click on the test sequence(s) that you want to run:

   Sequence: **Description:**

   Chip Path (Testpoints) This sequence tests the capability of the logic monitor(s) to read a known 0 and 1 value test point on each chip, which verifies the chip paths to the HM options.

   Data Record This sequence tests the capability of the logic monitor(s) to record preselected values and compares the results with expected values.

   Trigger This sequence tests the capability of the logic monitor(s) to trigger on preselected values and compares the results with expected values.

   Breakpoint This sequence tests the capability of the logic monitor(s) to perform breakpoint functions for preselected values and compares the results with expected values.

   Miscellaneous *This sequence is not implemented.*

5. Click on Go ▷; MME runs the selected sequences.

6. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

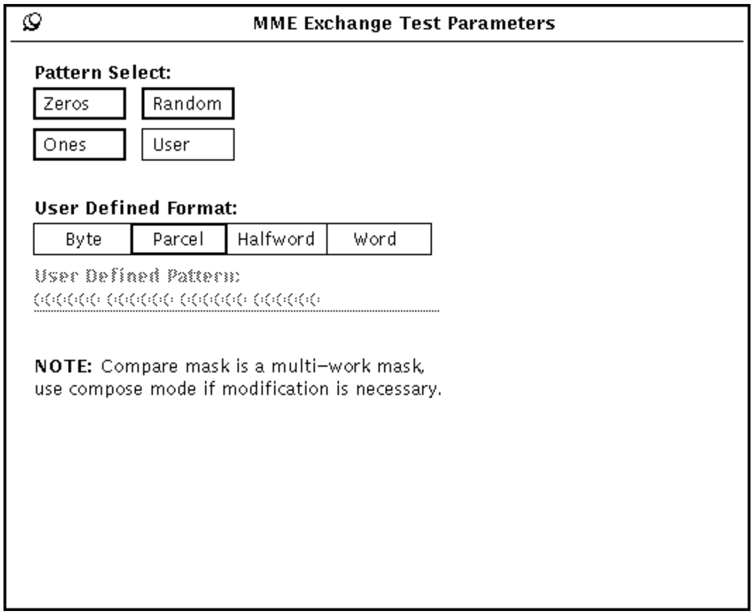
Running the Exchange Test

Perform the following procedure to run the exchange test in manual mode:

1. Click on Test Mode: Manual to enter manual mode.

2. Click on the CPU module(s) that you want to test.

3. Click on Tests: 6. Exchange to select the exchange test. The MME Exchange Test Parameters window appears:


MME Exchange Test Parameters

Pattern Select:

Zeros Random
Ones User

User Defined Format:

Byte Parcel Halfword Word

User Defined Pattern:
000000 000000 000000 000000

NOTE: Compare mask is a multi–work mask,
use compose mode if modification is necessary.

4. Click on the pattern(s) that you want to test:

Pattern:	Description:
Zeros	This sequence sends 40_8 words of a 0’s pattern to the MME buffer, performs a DMA transfer to memory, exchanges in to the CPU, exchanges out to memory, performs a DMA transfer from memory to the MME buffer, and compares the sent and received data.

Pattern:	**Description:**
Ones	This sequence sends 40_8 words of a 1’s pattern to the MME buffer, performs a DMA transfer to memory, exchanges in to the CPU, exchanges out to memory, performs a DMA transfer from memory to the MME buffer, and compares the sent and received data.
Random	This sequence sends 40_8 words of a random pattern to the MME buffer, performs a DMA transfer to memory, exchanges in to the CPU, exchanges out to memory, performs a DMA transfer from memory to the MME buffer, and compares the sent and received data.
User	This sequence sends 40_8 words of a user-defined pattern to the MME buffer, performs a DMA transfer to memory, exchanges in to the CPU, exchanges out to memory, performs a DMA transfer from memory to the MME buffer, and compares the sent and received data. To specify the user-defined pattern format, click on User Defined/Compare Mask Format: Byte, Parcel, Halfword, or Word. In the User Defined Pattern field, enter the pattern that you want to test.

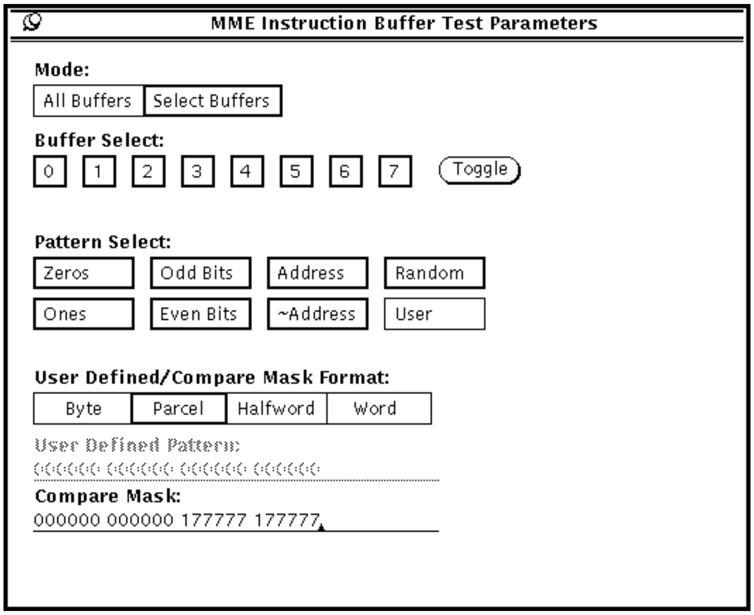
5. Click on Go ▷; MME tests the selected patterns.

6. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

Running the Instruction Buffers Test

Perform the following procedure to run the instruction buffers test in manual mode:

1. Click on Test Mode: [Manual] to enter manual mode.
2. Click on the CPU module(s) that you want to test.
3. Click on Tests: [7. Instruction Buffers] to select the instruction buffers test. The MME Instruction Buffer Test Parameters window appears:



MME Instruction Buffer Test Parameters

Mode:
All Buffers | Select Buffers

Buffer Select:
0 1 2 3 4 5 6 7 Toggle

Pattern Select:
Zeros | Odd Bits | Address | Random
Ones | Even Bits | ~Address | User

User Defined/Compare Mask Format:
Byte | Parcel | Halfword | Word

User Defined Pattern:
000000 000000 000000 000000

Compare Mask:
000000 000000 177777 177777

4. Click on Mode: [All Buffers] to test all instruction buffers, or click on Mode: [Select Buffers] to test specific instruction buffers.

   To select specific instruction buffers to test, click on the Buffer Select numbers that you want (any or all of [0], [1], [2], [3],[4],[5],[6], and [7]). You can toggle your selections with the (Toggle) button.

5. Click on the pattern(s) that you want to use:

Pattern: **Description:**

This sequence writes the MME buffer with a 0’s pattern, writes the MME buffer contents to memory, loads the instruction buffers from memory, stores selected instruction buffers to memory, reads the data back to the MME buffer, and compares the expected and actual data.

This sequence writes the MME buffer with a 1’s pattern, writes the MME buffer contents to memory, loads the instruction buffers from memory, stores selected instruction buffers to memory, reads the data back to the MME buffer, and compares the expected and actual data.

This sequence writes the MME buffer with an odd bits pattern, writes the MME buffer contents to memory, loads the instruction buffers from memory, stores selected instruction buffers to memory, reads the data back to the MME buffer, and compares the expected and actual data.

This sequence writes the MME buffer with an even bits pattern, writes the MME buffer contents to memory, loads the instruction buffers from memory, stores selected instruction buffers to memory, reads the data back to the MME buffer, and compares the expected and actual data.

This sequence writes the MME buffer with an address pattern, writes the MME buffer contents to memory, loads the instruction buffers from memory, stores selected instruction buffers to memory, reads the data back to the MME buffer, and compares the expected and actual data.

Pattern:	**Description:**
~Address	This sequence writes the MME buffer with a complement address pattern, writes the MME buffer contents to memory, loads the instruction buffers from memory, stores selected instruction buffers to memory, reads the data back to the MME buffer, and compares the expected and actual data.
Random	This sequence writes the MME buffer with a random pattern, writes the MME buffer contents to memory, loads the instruction buffers from memory, stores selected instruction buffers to memory, reads the data back to the MME buffer, and compares the expected and actual data.
User	This sequence writes the MME buffer with a user-defined pattern, writes the MME buffer contents to memory, loads the instruction buffers from memory, stores selected instruction buffers to memory, reads the data back to the MME buffer, and compares the expected and actual data. To specify the user-defined pattern format, click on User Defined/Compare Mask Format: Byte, Parcel, Halfword, or Word. In the User Defined Pattern field, enter the pattern that you want to use.

6. In the Compare Mask field, enter a mask to indicate the bit positions that you want to compare (0_2 = do not compare bit position; 1_2 = compare bit position).

7. Click on (Go ▷); MME tests the selected patterns.

8. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

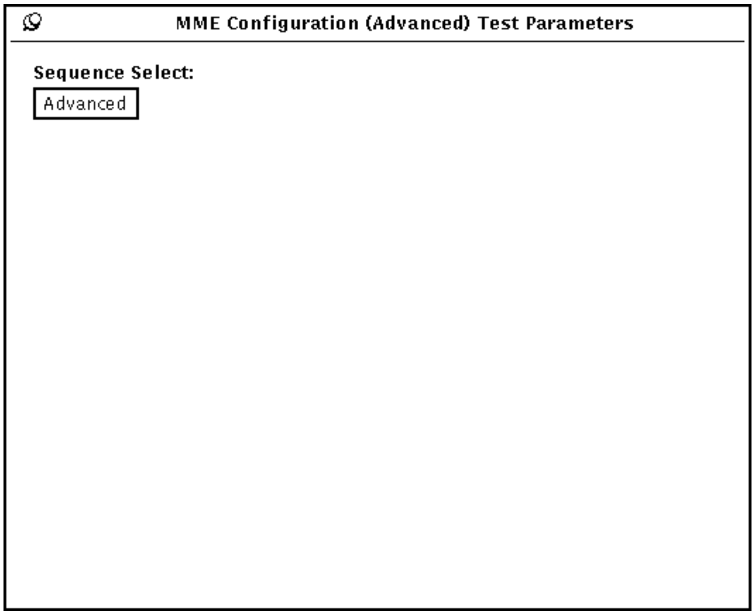
Running the Configuration (Advanced) Test

Perform the following procedure to run the configuration (advanced) test in manual mode:

1. Click on Test Mode: [Manual] to enter manual mode.

2. Click on the BS, CPU, I/O, and/or SHR module(s) that you want to test.

3. Click on Tests: [8. Configuration (Adv] to select the advanced configuration test. The MME Configuration (Advanced) Test Parameters window appears:



MME Configuration (Advanced) Test Parameters

Sequence Select:
[Advanced]

4. Click on the sequence that you want to test:

Sequence:	**Description:**
[Advanced]	This sequence checks I/O group and shared group configuration settings and interprocessor interrupts within shared groups for each CPU.

5. Click on (Go ▷); MME runs the selected test sequence.

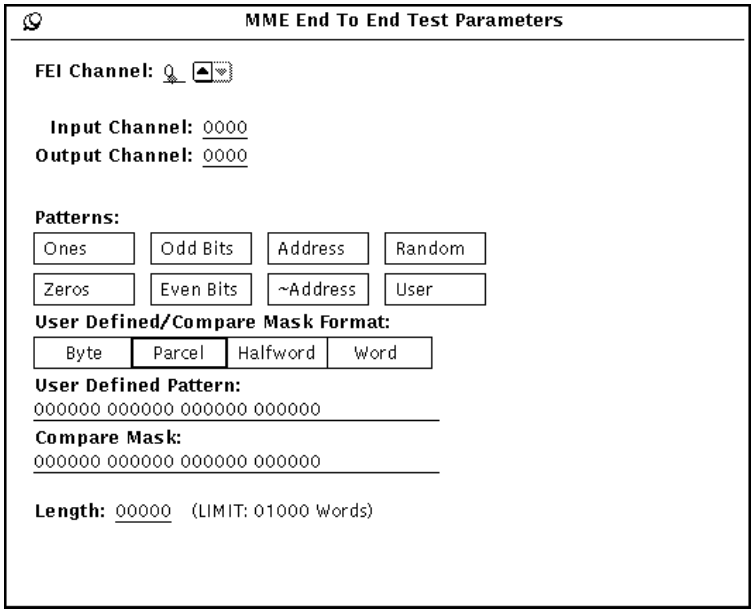
6. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

Running the End-to-end Test

1. Click on Test Mode: Manual to enter manual mode.

2. Click on the CPU module(s) that you want to test.

3. Click on Tests: 9. End To End to select the end-to-end test. The MME End to End Test Parameters window appears:



MME End To End Test Parameters

FEI Channel: 0

Input Channel: 0000
Output Channel: 0000

Patterns:

User Defined/Compare Mask Format:

User Defined Pattern:
000000 000000 000000 000000

Compare Mask:
000000 000000 000000 000000

Length: 00000 (LIMIT: 01000 Words)

The end-to-end test starts a small program in the CPU that you are testing. When the program receives input from the input channel, it returns the same data on the output channel. The end-to-end test then reads the final data and compares it to the original data.

4. In the FEI Channel field, specify the logical FEI channel that you want to use. This channel defaults to the FEI channel that SCE assigned to the support channel.

5. In the Input Channel field, specify the input channel that you want to use.

6. In the Output Channel field, specify the output channel that you want to use.

7. Click on the data pattern that you want to use:

Pattern:	**Description:**
Ones	The test uses a 177777_8 parcel pattern.
Zeros	The test uses a 000000_8 parcel pattern.
Odd Bits	The test uses a 125252_8 parcel pattern.
Even Bits	The test uses a 052525_8 parcel pattern.
Address	The test uses an address parcel pattern.
~Address	The test uses a complement address parcel pattern.
Random	The test uses a random data parcel pattern.
User	The test uses a user-defined parcel pattern. Click on User Defined/Compare Mask Format: Byte, Parcel, Halfword, or Word setting. In the User Defined Pattern field, enter the pattern that you want to use.

8. In the Compare Mask field, enter a mask to indicate the bit positions that you want to compare (0_2 = do not compare bit position; 1_2 = compare bit position).

9. In the Length field, specify the size of the data block that the test should use.

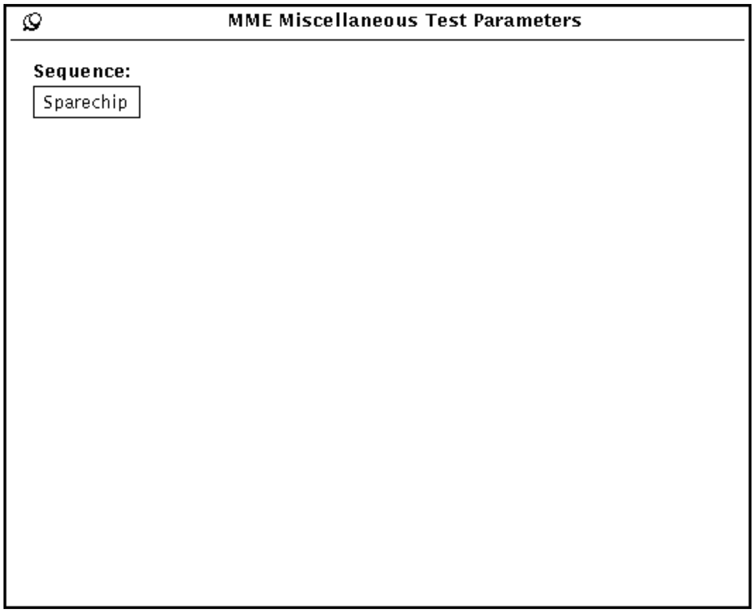
10. Click on Go ▷; MME runs the end-to-end test with the specified parameters.

11. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

Running the Miscellaneous Test

Perform the following procedure to run the miscellaneous test in manual mode:

1. Click on Test Mode: Manual to enter manual mode.
2. Click on the CPU module(s) that you want to test.
3. Click on Tests: 10. Miscellaneous to select the miscellaneous test. The MME Miscellaneous Test Parameters window appears:



MME Miscellaneous Test Parameters

Sequence:
Sparechip

4. Click on the sequence that you want to test:

Sequence:	**Description:**
Sparechip	This sequence tests the spare memory chips.

5. Click on Go ▷; MME tests the selected sequence.
6. View the MME Log window to see any errors that occur. If the MME Log window is not open, choose **View –> Log** to open it.

Using Compose Mode

When you click on Test Mode: Compose, environment 0 runs in compose mode. Compose mode enables you to create sequences of maintenance channel functions to test specific areas of the mainframe. Using the graphic interface in the compose mode base window, you can easily create or modify a sequence of functions that MME converts into the commands that are necessary to perform the functions.

This subsection describes modifying and creating sequences. Normally, you should modify an existing sequence rather than create a new one.

Use the MME Compose Sequence Entry window to create sequences of the following maintenance channel functions that run in the mainframe:

- Boundary scan loop controller functions
- Boundary scan module functions
- Boundary scan channel functions
- Boundary scan port functions
- Shared loop controller functions
- Shared logic monitor functions
- CPU loop controller functions
- CPU logic monitor functions
- CPU DMA functions
- Input/output loop controller functions
- Input/output logic monitor functions
- Input/output sanity generator functions

Use the MME Compose Sequence Entry window to create sequences of the following functions and utilities that run in the MWS or SWS:

- Channel functions: close, disconnect, lock, masterclear, open, read, reset, unlock, write, and write/read

- Comment functions: quiet and verbose

- Compare function

- Control functions: goto, label, and stop

- File operation functions: read, write, append, and delete

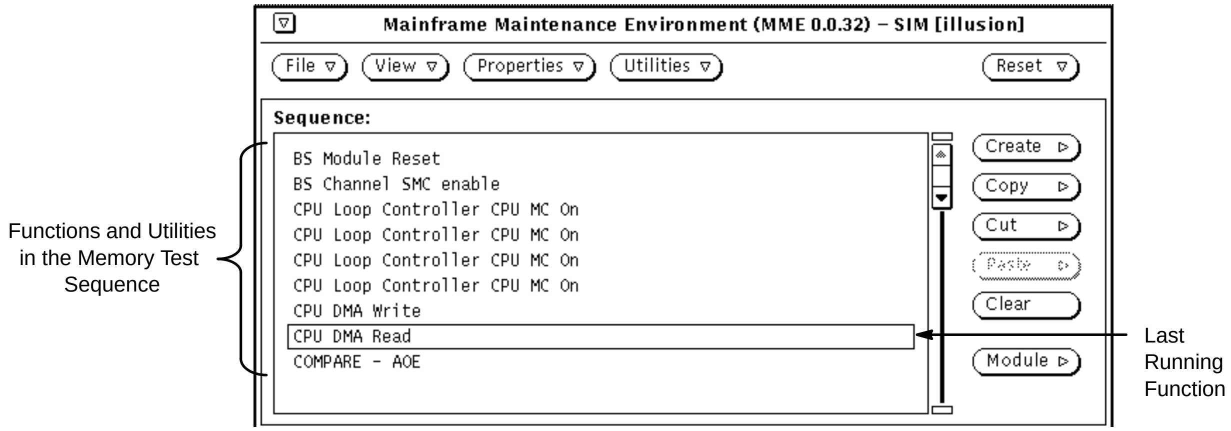
- Utilities: delay, mask, move, pattern, and squish

Modifying an Existing Sequence

The following example shows how to modify an existing sequence. This example modifies the memory test sequence to test data starting at mainframe address 40000_8.

1. Run a sequence in automatic or manual mode and click on (Halt).

2. Click on Test Mode: [Compose] to switch to compose mode. The memory test sequence is shown in the Sequence scroll box; refer to Figure 5. Notice that the last running function is highlighted.



```
BS Module Reset
BS Channel SMC enable
CPU Loop Controller CPU MC On
CPU Loop Controller CPU MC On
CPU Loop Controller CPU MC On
CPU Loop Controller CPU MC On
CPU DMA Write
CPU DMA Read
COMPARE - AOE
```

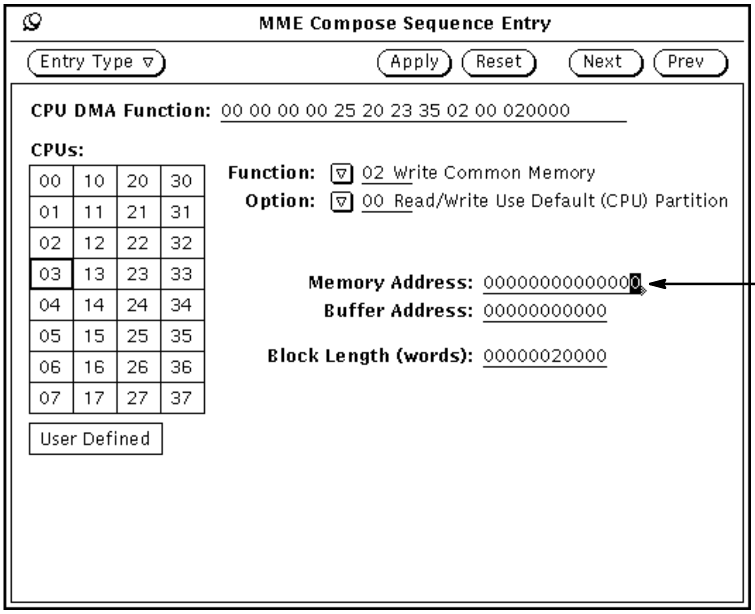
Functions and Utilities in the Memory Test Sequence

Last Running Function

Figure 5. Viewing the Original Sequence

3. Click on the CPU DMA Write function that is displayed in the Sequence scroll box. MME displays the MME Compose Sequence Entry window for the function.

4. Change where the direct memory access (DMA) function starts writing data by changing the value stored in the Memory Address field. For example, Figure 6 shows how to change the write address from 0_8 to 40000_8.



CPU DMA Function: 00 00 00 00 25 20 23 35 02 00 020000

CPUs:

00	10	20	30
01	11	21	31
02	12	22	32
03	13	23	33
04	14	24	34
05	15	25	35
06	16	26	36
07	17	27	37

Function: ▿ 02 Write Common Memory

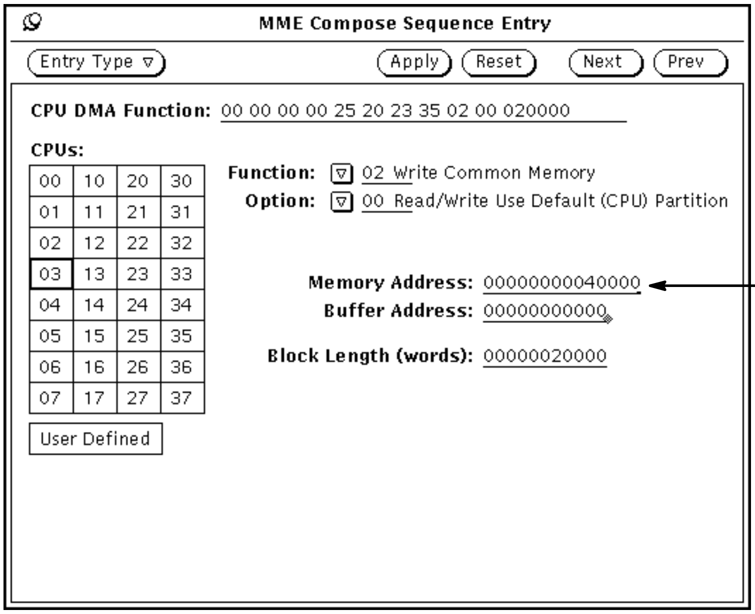
Option: ▿ 00 Read/Write Use Default (CPU) Partition

Memory Address: 00000000000000

Buffer Address: 00000000000

Block Length (words): 00000020000

The original DMA function writes data starting at mainframe address 0_8



CPU DMA Function: 00 00 00 00 25 20 23 35 02 00 020000

CPUs:

00	10	20	30
01	11	21	31
02	12	22	32
03	13	23	33
04	14	24	34
05	15	25	35
06	16	26	36
07	17	27	37

Function: ▿ 02 Write Common Memory

Option: ▿ 00 Read/Write Use Default (CPU) Partition

Memory Address: 00000000040000

Buffer Address: 00000000000

Block Length (words): 00000020000

This modified DMA function writes data starting at mainframe address 40000_8

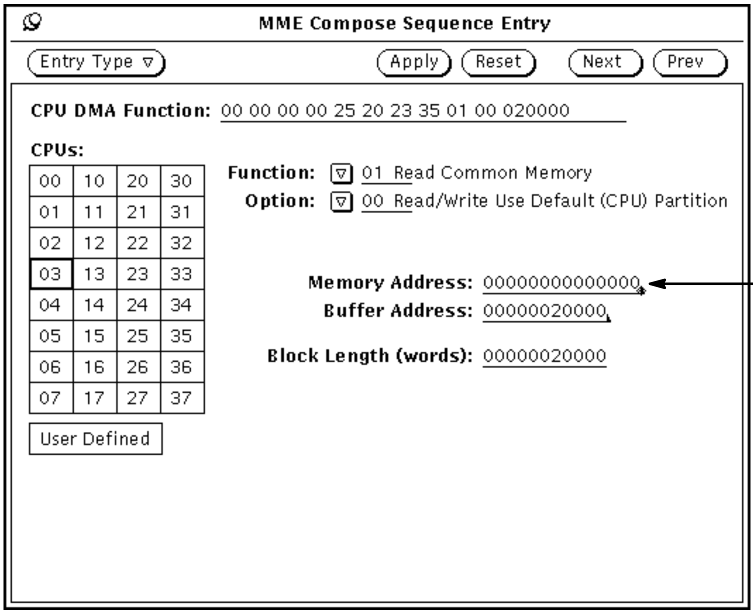
Figure 6. Modifying Where the DMA Function Writes the Data

5. Click on (Apply) to send the function change to the sequence.

NOTE: If (Apply) is not present in the MME Compose Sequence Entry window, MME has been configured to enable the auto apply function with the Properties –> Enable Auto Apply command. Move the cursor to the MME base window, and the function changes are automatically applied.

6. Click on the CPU DMA Read function that is displayed in the Sequence scroll box. MME displays the MME Compose Sequence Entry window for the function.

7. Change where the DMA function starts reading data by changing the value stored in the Memory Address field. For example, Figure 7 shows how to change the read address from 0_8 to 40000_8.



CPU DMA Function: 00 00 00 00 25 20 23 35 01 00 020000

CPUs:

00	10	20	30
01	11	21	31
02	12	22	32
03	13	23	33
04	14	24	34
05	15	25	35
06	16	26	36
07	17	27	37

Function: ▽ 01 Read Common Memory

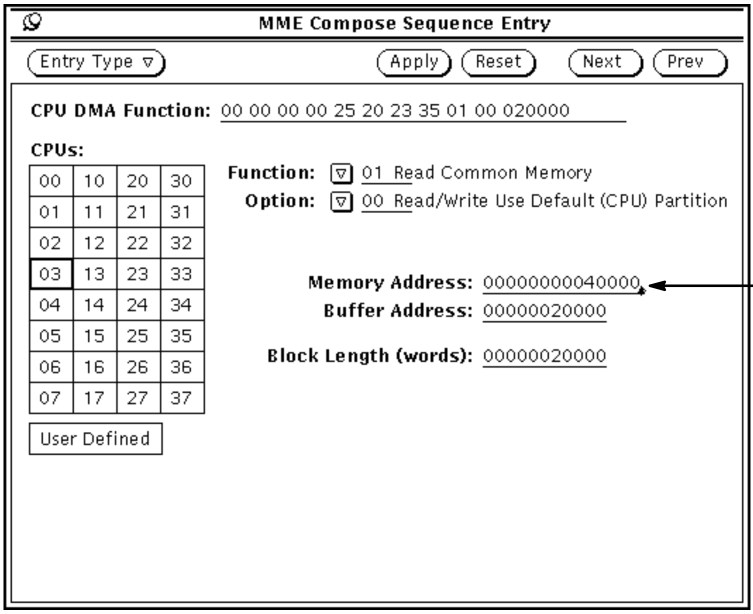
Option: ▽ 00 Read/Write Use Default (CPU) Partition

Memory Address: 00000000000000

Buffer Address: 00000020000

Block Length (words): 00000020000

The original DMA function reads data starting at mainframe address 0_8



CPU DMA Function: 00 00 00 00 25 20 23 35 01 00 020000

CPUs:

00	10	20	30
01	11	21	31
02	12	22	32
03	13	23	33
04	14	24	34
05	15	25	35
06	16	26	36
07	17	27	37

Function: ▽ 01 Read Common Memory

Option: ▽ 00 Read/Write Use Default (CPU) Partition

Memory Address: 00000000040000

Buffer Address: 00000020000

Block Length (words): 00000020000

This modified DMA function reads data starting at mainframe address 40000_8

Figure 7. Modifying Where the DMA Function Reads the Data

8. Click on (Apply) to send the function change to the sequence.

NOTE: If (Apply) is not present in the MME Compose Sequence Entry window, MME has been configured to enable the auto apply function with the Properties –> Enable Auto Apply command. Move the cursor to the MME base window, and the function changes are automatically applied.

9. Save or run the sequence:

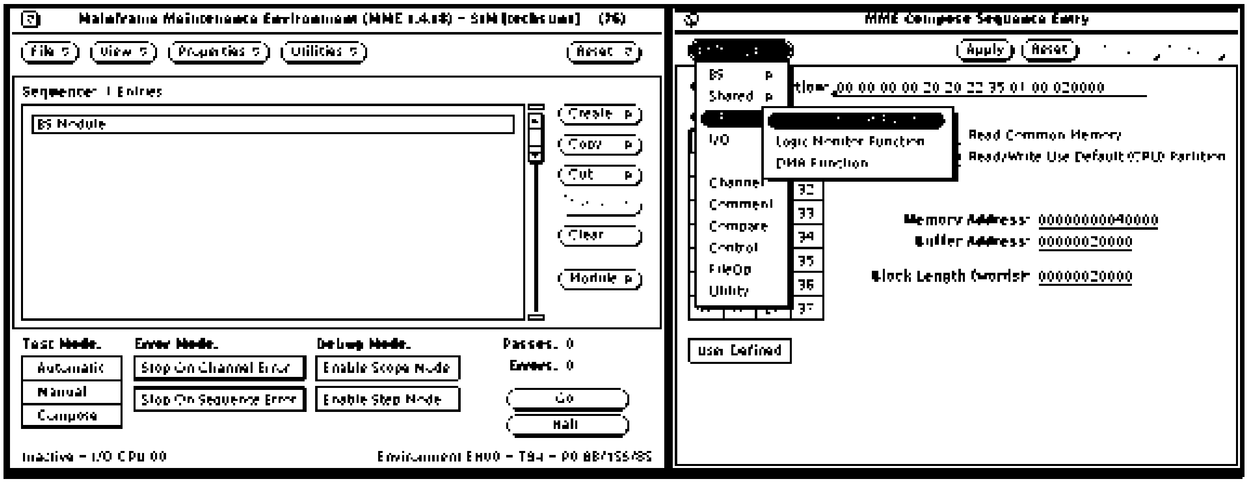
- To run the modified sequence, click on (Go ▷).

- To save the sequence, choose **File –> Save –> Sequence**. You should also save the data to use with the sequence; choose **File –> Save –> Data**.

  When you want to reuse the sequence, load the sequence with the File –> Load –> Sequence command and load the data with the File –> Load –> Data command.

  For more information, refer to the “File –> Save –> Sequence,” “File –> Save –> Data,” “File –> Load –> Sequence,” and “File –> Load –> Data” subsections of the *MME Interface Reference*, publication number HDM-008-A.

You can also change the functions in the current sequence. Figure 8 shows an example of how to change a selected function.



Passes. 0

Errors. 0

Inactive – I/O CPU 00

Environment EN00 – T94 – P0 88/155/85

tion: 00 00 00 00 20 20 22 35 01 00 020000

Memory Address: 0000000040000

Buffer Address: 00000020000

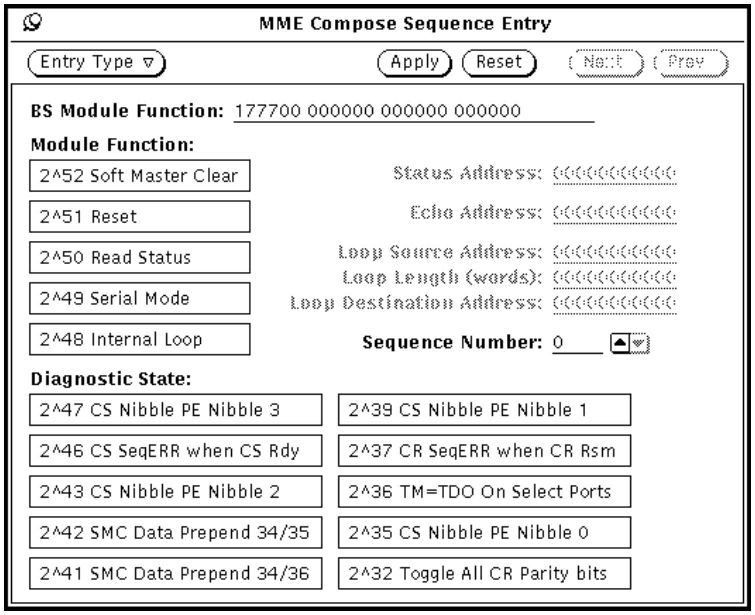
Block Length (words): 00000020000

Figure 8. Changing a Selected Function in the Sequence

Creating a New Sequence

Perform the following procedure to create a new sequence of maintenance channel functions or utilities:

1. In the Mainframe Maintenance Environment base window, choose **Create –> Before**, **Create –> After**, **Create –> Top**, or **Create –> Bottom** to specify where in the Sequence scroll box you want to create the new entry. The MME Compose Sequence Entry window appears:



MME Compose Sequence Entry

BS Module Function: 177700 000000 000000 000000

Module Function:

Status Address: 0000000000

Echo Address: 0000000000

Loop Source Address: 0000000000

Loop Length (words): 0000000000

Loop Destination Address: 0000000000

Sequence Number: 0

Diagnostic State:

Entry Type ▽
Apply
Reset
Next
Prev
2^52 Soft Master Clear
2^51 Reset
2^50 Read Status
2^49 Serial Mode
2^48 Internal Loop
2^47 CS Nibble PE Nibble 3
2^39 CS Nibble PE Nibble 1
2^46 CS SeqERR when CS Rdy
2^37 CR SeqERR when CR Rsm
2^43 CS Nibble PE Nibble 2
2^36 TM=TDO On Select Ports
2^42 SMC Data Prepend 34/35
2^35 CS Nibble PE Nibble 0
2^41 SMC Data Prepend 34/36
2^32 Toggle All CR Parity bits

2. Choose a different entry type from (Entry Type ▽) if you want a function or utility other than the default. Refer to the descriptions of the individual functions and utilities later in this subsection for more information about the functions and utilities available.

3. Modify the information in the MME Compose Sequence Entry window to create the specific function or utility that you need.

4. Click on (Apply) to place the entry in the sequence or (Reset) to reset the MME Compose Sequence Entry window.

NOTE: If (Apply) is not present in the MME Compose Sequence Entry window, MME has been configured to enable the auto apply function with the Properties –> Enable Auto Apply command. Move the cursor to the MME base window, and the function changes are automatically applied.

When there is more than one entry in the Sequence scroll box, use the (Next) button to move forward one entry in the sequence and the (Prev) button to move backward one entry in the sequence.

5. Repeat Steps 1 through 4 to create more entries in the sequence.

6. Choose a module from (Module ▷) if you want to assign the sequence to a specific module.

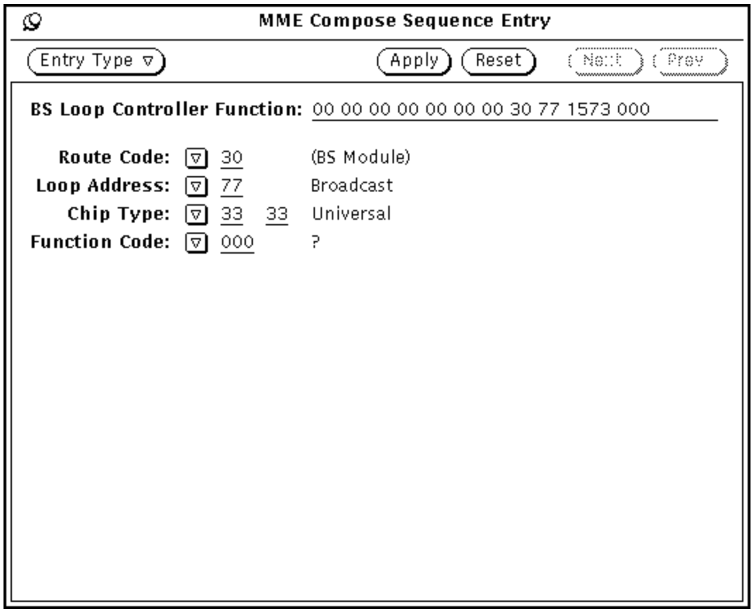
7. Click on (Go ▷). MME sends the commands to the mainframe through the maintenance channel to perform the functions that you have requested.

Boundary Scan Functions

The boundary scan functions are used to manipulate the boundary scan module, channel, port, and loop controller components.

Boundary Scan Loop Controller Functions

The boundary scan loop controller functions enable you to send functions to any of the chips on a boundary scan or IO02 module. Choose **Entry Type –> BS –> Loop Controller Function** to access the boundary scan loop controller functions. The following MME Compose Sequence Entry window appears:



BS Loop Controller Function: 00 00 00 00 00 00 00 30 77 1573 000

Route Code: ▽ 30 (BS Module)
Loop Address: ▽ 77 Broadcast
Chip Type: ▽ 33 33 Universal
Function Code: ▽ 000 ?

Notice that the maintenance channel command data for the current function is generated and displayed in the BS Loop Controller Function field. This data is updated as you select different loop controller functions. To modify the boundary scan loop controller function, perform the following procedure:

1. From Route Code: ▽, choose the route code of the module to which you want to send the loop controller functions (boundary scan module or IO02 module).

2. From Loop Address: ▽, choose the loop address to which you want the function to go. Currently All Loops is the only option, which corresponds to a broadcast function 77_8.

3. Specify the chip type(s) where you want the function to go. From Chip Type: ▽, choose the chip type(s) to which you want the function to go.

4. From Function Code: ▽, choose the function that you want to run.

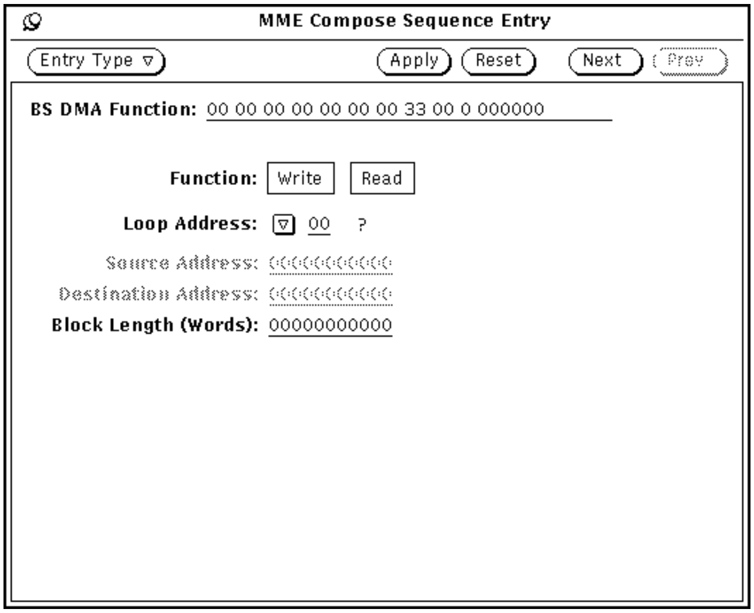
   For more information about the function codes, refer to the *Boundary Scan Module (BS02)* document, publication number HTM-005-A, and the *Triton Maintenance System Engineering Note*, publication number PRN-0957.

5. Click on (Apply) to place the function in the Sequence scroll box.

When execution reaches the boundary scan loop controller function in the Sequence scroll box, the maintenance channel command data that was displayed in the BS Loop Controller Function field is executed.

Boundary Scan DMA Functions

The boundary scan DMA functions enable you to perform direct memory access (DMA) reads and writes. Choose **Entry Type –> BS –> DMA** to access the boundary scan DMA functions. The following MME Compose Sequence Entry window appears:



BS DMA Function: 00 00 00 00 00 00 00 33 00 0 000000

Loop Address: 00

Source Address: 00000000000

Destination Address: 00000000000

Block Length (Words): 00000000000

Notice that the maintenance channel command parcels for the current function are generated and displayed in the BS DMA Function field. These parcels are updated as you select different module functions or diagnostic states. To create an entry with boundary scan DMA functions, perform the following procedure:

1. Click on the type of DMA function that you want to create (Function: Write or Read).

2. From the Loop Address: ▽, choose the loop address to which you want the function to go.

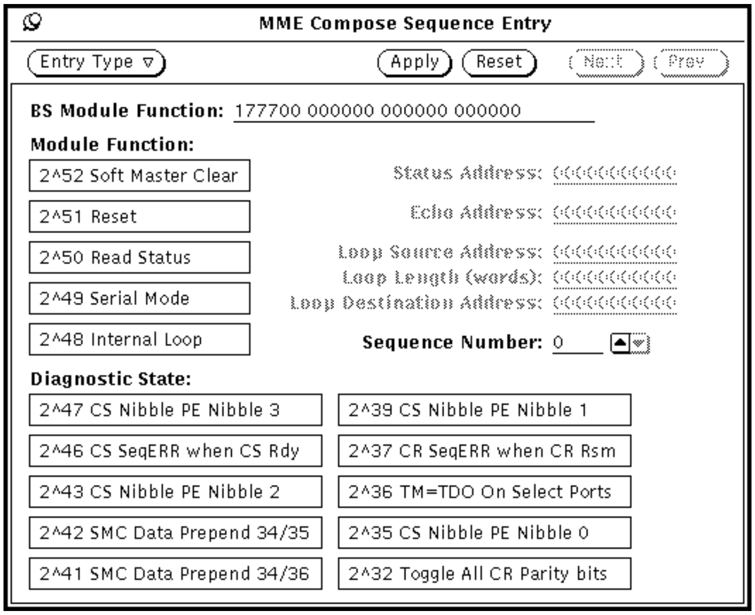
3. If the function is a write function, enter the address of the source data in the Source Address field. If the function is a read function, enter the address that you want to read in the Destination address field.

4. In the Block Length (Words) field, enter the number of words that you want to read or write.

Boundary Scan Module Functions

The boundary scan module functions enable you to access the functionality of the module and to modify the diagnostic state of the module. Choose **Entry Type –> BS –> Module Function** to access the boundary scan module functions. The following MME Compose Sequence Entry window appears:

NOTE: Always run a disconnect function before you run a boundary scan module function.



BS Module Function: 177700 000000 000000 000000

Module Function:

- 2^52 Soft Master Clear
- 2^51 Reset
- 2^50 Read Status
- 2^49 Serial Mode
- 2^48 Internal Loop

Status Address: 0000000000

Echo Address: 0000000000

Loop Source Address: 0000000000

Loop Length (words): 0000000000

Loop Destination Address: 0000000000

Sequence Number: 0

Diagnostic State:

2^47 CS Nibble PE Nibble 3	2^39 CS Nibble PE Nibble 1
2^46 CS SeqERR when CS Rdy	2^37 CR SeqERR when CR Rsm
2^43 CS Nibble PE Nibble 2	2^36 TM=TDO On Select Ports
2^42 SMC Data Prepend 34/35	2^35 CS Nibble PE Nibble 0
2^41 SMC Data Prepend 34/36	2^32 Toggle All CR Parity bits

Notice that the maintenance channel command parcels for the current function are generated and displayed in the BS Module Function field. These parcels are updated as you select different module functions or diagnostic states. To create an entry with boundary scan module functions, perform the following procedure.

1. Click on the module functions that you want to use:

Module Function: **Description:**

2^52 Soft Master Clear

This function performs a soft master clear, which sends a disconnect signal through the control channel to the VME support system to force the control channel to a known state. The soft master clear function also turns off the sanity code generator, clears any entries in the error logger, and disables the SMC by ignoring SMC and error logger inputs.

2^51 Reset

This function performs a reset, which sends a disconnect signal through the control channel to the VME support system to force the control channel to a known state.

2^50 Read Status

This function returns 4 parcels of module status and clears error status (for serial mode). Specify the address to which the status is returned in the Status Address field.

2^49 Serial Mode

This function causes the boundary scan module to enter serial mode. If this setting is not selected, passon mode is used.

2^48 Internal Loop

This function loops source parcels back to the return channel (for serial mode).

In the Echo Address field, enter the address to which you want to echo the loop-back function word.

In the Loop Source Address field, enter the address of the data block in the MME buffer that you want to loop back.

In the Loop Length (words) field, enter the length of the data block that you want to loop back.

In the Loop Destination Address field, enter the address in the MME buffer that will receive the returned data.

2. Click on the diagnostic states that you want to modify:

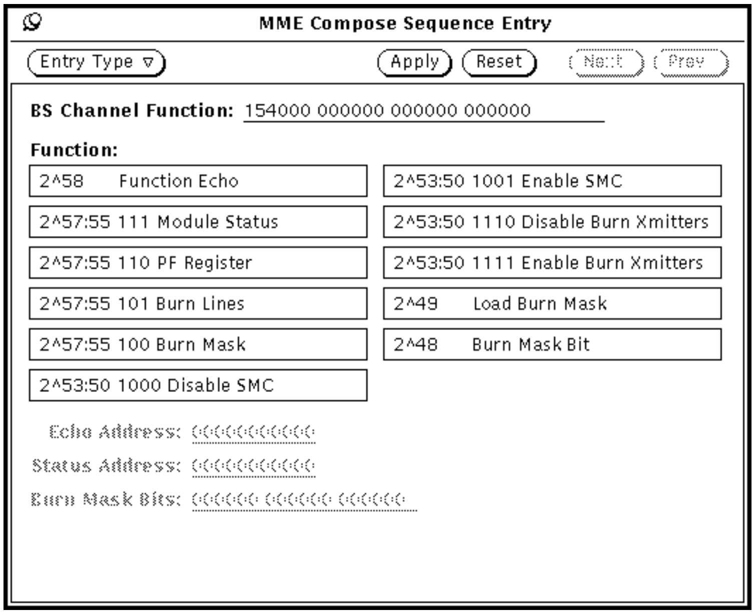
Diagnostic State:	**Description:**
2^47 CS Nibble PE Nibble 3	This diagnostic state forces CS_NibblePE for nibble 3.
2^46 CS SeqERR when CS Rdy	This diagnostic state forces CS_SeqErr when CS_Rdy.
2^43 CS Nibble PE Nibble 2	This diagnostic state forces CS_NibblePE for nibble 2.
2^42 SMC Data Prepend 34/35	This diagnostic state forces the serial maintenance channel (SMC) data prepend to equal 34_8 or 35_8.
2^41 SMC Data Prepend 34/36	This diagnostic state forces the SMC data prepend to equal 34_8 or 36_8.
2^39 CS Nibble PE Nibble 1	This diagnostic state forces CS_NibblePE for nibble 1.
2^37 CR SeqERR when CR Rsm	This diagnostic state forces CS_SeqErr when CS_Rsm.
2^36 TM=TDO On Select Ports	This diagnostic state forces TM = TDO on all selected ports.
2^35 CS Nibble PE Nibble 0	This diagnostic state forces CS_NibblePE for nibble 0.
2^32 Toggle All CR Parity bits	This diagnostic state toggles all CR_parity bits.

3. In the Sequence Number field, specify the sequence number.

4. Click on (Apply) to place the function in the Sequence scroll box.

Boundary Scan Channel Functions

Choose **Entry Type –> BS –> Channel Function** to access the boundary scan channel functions. The following MME Compose Sequence Entry window appears:



BS Channel Function: 154000 000000 000000 000000

Echo Address: 0000000000

Status Address: 0000000000

Burn Mask Bits: 000000 000000 000000

Notice that the maintenance channel command parcels for the current function are generated and displayed in the BS Channel Function field. These parcels are updated as you select different channel functions. To create an entry with boundary scan channel functions, perform the following procedure.

1. Click on the channel functions that you want to run:

Channel Function:	**Description:**
2^58 Function Echo	This function returns the 4 parcels of the channel function command word to the MME buffer. In the Echo Address field, enter the address in the MME buffer that you want to echo.

NOTE: You can use only one of the following four functions at a time: 2^57:55 111 Module Status, 2^57:55 110 PF Register, 2^57:55 101 Burn Lines, or 2^57:55 100 Burn Mask.

Channel Function:	**Description:**
2^57.SS 111 Module Status	This function returns 4 parcels of module status to the MME buffer. In the Status Address field, enter the address in the MME buffer that will receive the data.
2^57.SS 110 PF Register	This function returns 4 parcels of the PF register to the MME buffer. In the Status Address field, enter the address in the MME buffer that will receive the data.
2^57.SS 101 Burn Lines	This function returns 4 parcels of continuity line status to the MME buffer. In the Status Address field, enter the address in the MME buffer that will receive the data.
2^57.SS 100 Burn Mask	This function returns 4 parcels of the burn mask to the MME buffer. In the Status Address field, enter the address in the MME buffer that will receive the data.

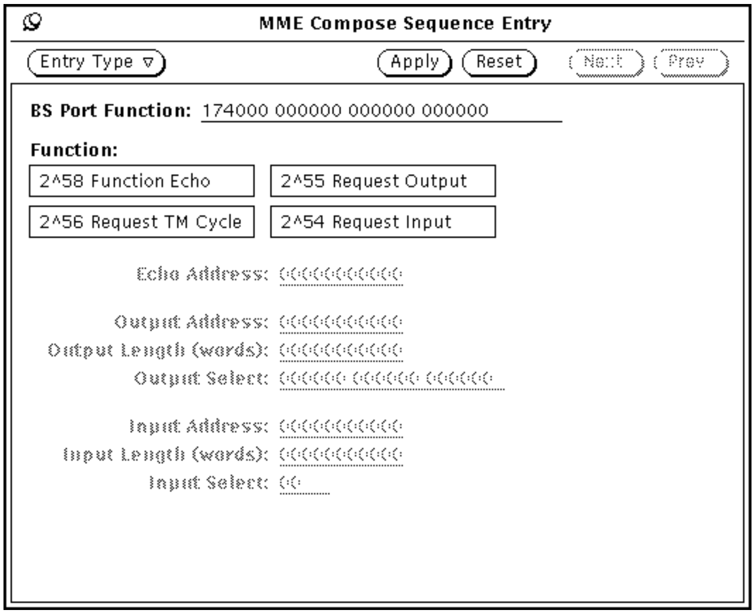
NOTE: You can use only one of the 2^53.S0 1000 Disable SMC, 2^53.S0 1001 Enable SMC, 2^53.S0 1110 Disable Burn Xmitters, and 2^53.S0 1111 Enable Burn Xmitters functions at a time. (When you click on a setting, the previously selected setting deselects.)

2^53.S0 1000 Disable SMC	This function disables the SMC.
2^53.S0 1001 Enable SMC	This function enables the SMC.
2^53.S0 1110 Disable Burn Xmitters	This function disables the continuity line transmitters.
2^53.S0 1111 Enable Burn Xmitters	This function enables the continuity line transmitters.
2^49 Load Burn Mask	This function loads the bits specified in the Burn Mask Bits field into the burn mask.
2^48 Burn Mask Bit	This function disables detection by the WACS for this boundary scan module.

2. Click on (Apply) to place the function in the Sequence scroll box.

Boundary Scan Port Functions

Choose **Entry Type –> BS –> Port Function** to access the boundary scan port functions. The following MME Compose Sequence Entry window appears:



MME Compose Sequence Entry

Entry Type ▽ Apply Reset Next Prev

BS Port Function: 174000 000000 000000 000000

Function:

2^58 Function Echo 2^55 Request Output

2^56 Request TM Cycle 2^54 Request Input

Echo Address: 0000000000

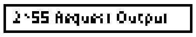
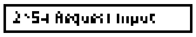
Output Address: 0000000000
Output Length (words): 0000000000
Output Select: 000000 000000 000000

Input Address: 0000000000
Input Length (words): 0000000000
Input Select: 00

Notice that the maintenance channel command parcels for the current function are generated and displayed in the BS Port Function field. These parcels are updated as you select different port functions. To create an entry with boundary scan port functions, perform the following procedure:

1. Click on the channel functions that you want to run:

Channel Function:	**Description:**
2^58 Function Echo	This function returns the 4 parcels of the function command word to the MME buffer. In the Echo Address field, enter the address of the MME buffer data you want to use.
2^56 Request TM Cycle	This function performs the test_mode cycle.

Channel Function:	**Description:**
	This function sends output to the specified output ports. In the Output Address field, enter the MME buffer address of the data that you want to send as output. In the Output Length (words) field, enter the length of the output data block. In the Output Select field, enter a bit mask to select to which of the 48 ports the output is sent.
	This function returns parcels from the specified input ports or asserts a test mode and leaves it active. In the Input Address field, enter the MME buffer address that you want to receive the data. In the Input Length (words) field, enter the length of the input data block. In the Input Select field, enter a bit mask to select the input ports from which the data is received.

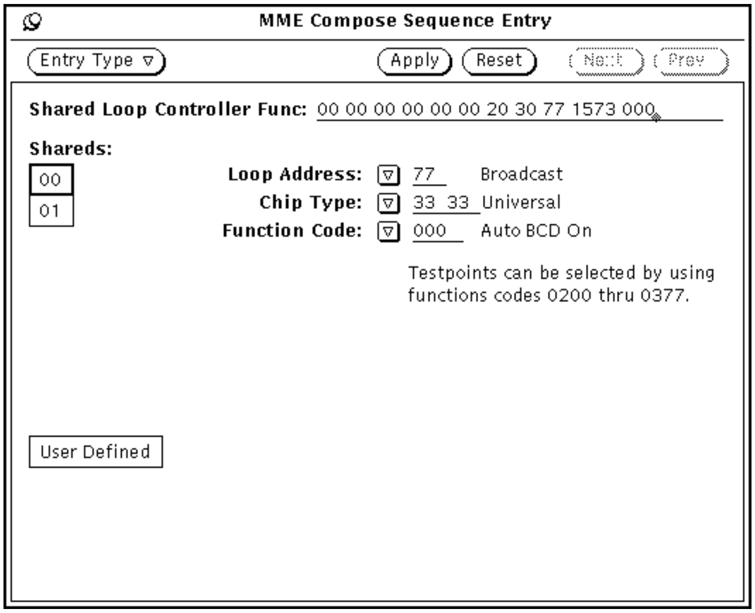
2. Click on (Apply) to place the function in the Sequence scroll box.

Shared Functions

The shared functions enable you to perform shared maintenance and configuration functions and shared logic monitor functions.

Shared Loop Controller Functions

The shared loop controller functions are configuration and maintenance functions that you can send to a shared module through a shared module loop controller. Choose **Entry Type –> Shared –> Loop Controller Function** to access the I/O loop controller functions. The following MME Compose Sequence Entry window appears.



Shared Loop Controller Func: 00 00 00 00 00 00 20 30 77 1573 000

Shareds:
00
01

Loop Address: ▽ 77 Broadcast
Chip Type: ▽ 33 33 Universal
Function Code: ▽ 000 Auto BCD On

Testpoints can be selected by using functions codes 0200 thru 0377.

Notice that the maintenance channel command data for the current function is generated and displayed in the Shared Loop Controller Function field. This data is updated as you select different functions. To create an entry with I/O loop controller functions, perform the following procedure:

1. Click on the shared module that you want to use.

   NOTE: Based on the current configuration data and the shared module that you select, MME automatically generates a route code for the function. If you want to force the route code to a specific value, you must click on User Defined and change the route code information in the Shared Loop Controller Func field.

2. From Loop Address: ▽, choose the loop address to which you want the function to go.

3. From Chip Type: ▽, choose the chip type(s) to which you want the function to go.

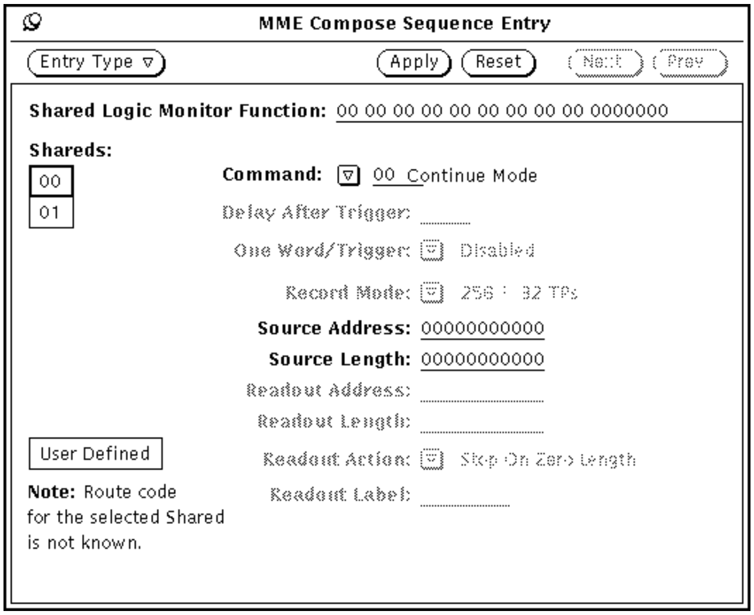
4. From Function Code: ▽, choose a configuration or maintenance function.

   Refer to the *Maintenance Channel* document, publication number HTM-006-B; and the *Triton Maintenance System Engineering Note*, publication number PRN-0957; for more information about the function codes.

5. Click on (Apply) to place the function in the Sequence scroll box.

Shared Logic Monitor Functions

The shared logic monitor functions enable you to control the activity of the logic monitors on the shared modules. Choose **Entry Type –> Shared –> Logic Monitor Function** to access the shared logic monitor functions. The following MME Compose Sequence Entry window appears:



Shared Logic Monitor Function: 00 00 00 00 00 00 00 00 00 0000000

Shareds:
00
01

Command: ▽ 00 Continue Mode
Delay After Trigger:
One Word/Trigger: ▽ Disabled
Record Mode: ▽ 256 : 32 TPs
Source Address: 00000000000
Source Length: 00000000000
Readout Address:
Readout Length:
Readout Action: ▽ Skip On Zero Length
Readout Label:

Note: Route code for the selected Shared is not known.

Notice that the maintenance channel command data for the current function is generated and displayed in the Shared Logic Monitor Function field. This data is updated as you select different functions. To create an entry with shared logic monitor functions, perform the following procedure:

1. Click on the shared module that you want to use.

   NOTE: Based on the current configuration data and the shared module that you select, MME automatically generates a route code for the function. If you want to force the route code to a specific value, you must click on [User Defined] and change the route code information in the Shared Logic Monitor Function field.

2. From Command: ▽, choose a shared logic monitor function.

Refer to the *Maintenance Channel* document, publication number HTM-006-B; and the *Triton Maintenance System Engineering Note*, publication number PRN-0957; for more information about these commands.

3. Update any fields that activate. Table 2 describes the fields.

Table 2. Shared Logic Monitor Command Fields

Field	Description
Delay After Trigger	Number of clock periods the logic monitor continues recording after a trigger condition occurs
One Word/Trigger	Record 1 word per trigger condition option (enabled or disabled)
Record Mode	Type of recording the logic monitor should do (number of clock periods to record and number of test points to record per clock period)
Source Address	MME buffer address of data to write to the logic monitor
Source Length	Length of data block to write to the logic monitor
Readout Address	MME buffer address to receive data from a logic monitor readout buffer command
Readout Length	Length of data block for data received from logic monitor readout buffer command
Readout Action	Action to perform based on results of data from logic monitor readout buffer command
Readout Label	Label to jump to if the readout action is a goto label command

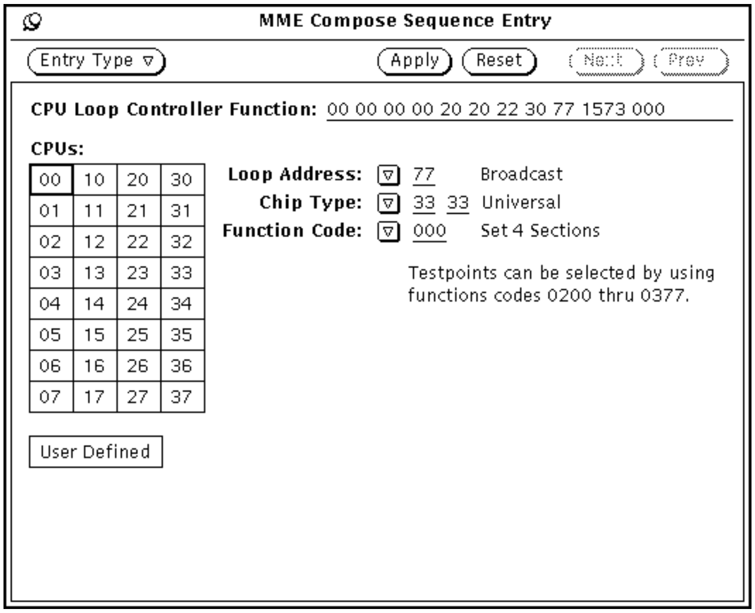
4. Click on Apply to place the function in the Sequence scroll box.

CPU Functions

The CPU functions enable you to perform maintenance and configuration functions, diagnostic monitor functions, and DMA functions.

CPU Loop Controller Functions

The CPU loop controller functions are configuration or maintenance functions that you can send through the CPU module loop controllers to any chip or loop of chips on the CPU modules. Choose **Entry Type –> CPU –> Loop Controller Function** to access the CPU loop controller functions. The following MME Compose Sequence Entry window appears.



CPU Loop Controller Function: 00 00 00 00 20 20 22 30 77 1573 000

CPUs:

00	10	20	30
01	11	21	31
02	12	22	32
03	13	23	33
04	14	24	34
05	15	25	35
06	16	26	36
07	17	27	37

Loop Address: ▽ 77 Broadcast
Chip Type: ▽ 33 33 Universal
Function Code: ▽ 000 Set 4 Sections

Testpoints can be selected by using functions codes 0200 thru 0377.

Notice that the maintenance channel command data for the current function is generated and displayed in the CPU Loop Controller Function field. This data is updated as you select different functions. To create an entry with CPU loop controller functions, perform the following procedure:

1. Click on the CPU that you want to use.

NOTE: Based on the current configuration data and the CPU that you select, MME automatically generates a route code for the function. If you want to force the route code to a specific value, you must click on User Defined and change the route code information in the CPU Loop Controller Function field.

2. From Loop Address: ▿, choose the loop address to which you want the function to go.

3. From Chip Type: ▿, choose the chip type(s) to which you want the function to go.

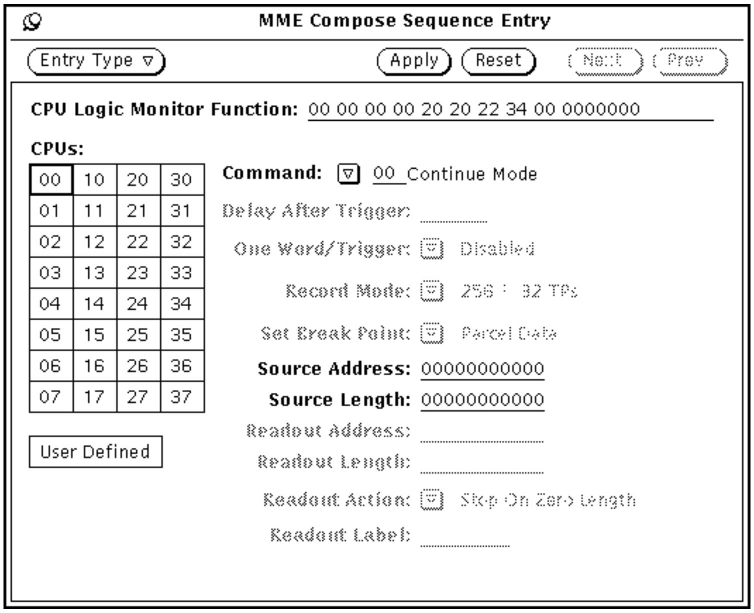
4. From Function Code: ▿, choose a configuration or maintenance function.

   Refer to the *Maintenance Channel* document, publication number HTM-006-B; and the *Triton Maintenance System Engineering Note*, publication number PRN-0957; for more information about the function codes.

5. Click on (Apply) to place the function in the Sequence scroll box.

CPU Logic Monitor Functions

The CPU logic monitor functions enable you to control the activity of the logic monitors on the CPUs. Choose **Entry Type –> CPU –> Logic Monitor Function** to access the CPU logic monitor functions. The following MME Compose Sequence Entry window appears:



CPU Logic Monitor Function: 00 00 00 00 20 20 22 34 00 0000000

CPUs:

00	10	20	30
01	11	21	31
02	12	22	32
03	13	23	33
04	14	24	34
05	15	25	35
06	16	26	36
07	17	27	37

Command: ▿ 00 Continue Mode
Delay After Trigger:
One Word/Trigger: ▿ Disabled
Record Mode: ▿ 256 : 32 TPs
Set Break Point: ▿ Parcel Data
Source Address: 00000000000
Source Length: 00000000000
Readout Address:
Readout Length:
Readout Action: ▿ Skip On Zero Length
Readout Label:

Notice that the maintenance channel command data for the current function is generated and displayed in the CPU Logic Monitor Function field. This data is updated as you select different functions. To create an entry with CPU logic monitor functions, perform the following procedure.

1. Click on the CPU that you want to use.

   NOTE: Based on the current configuration data and the CPU that you select, MME automatically generates a route code for the function. If you want to force the route code to a specific value, you must click on [User Defined] and change the route code information in the CPU Logic Monitor Function field.

2. From Command: ▽, choose a CPU logic monitor function.

   Refer to the *Maintenance Channel* document, publication number HTM-006-B; and the *Triton Maintenance System Engineering Note*, publication number PRN-0957; for more information about these commands.

3. Update any fields that activate. Table 3 describes the fields.

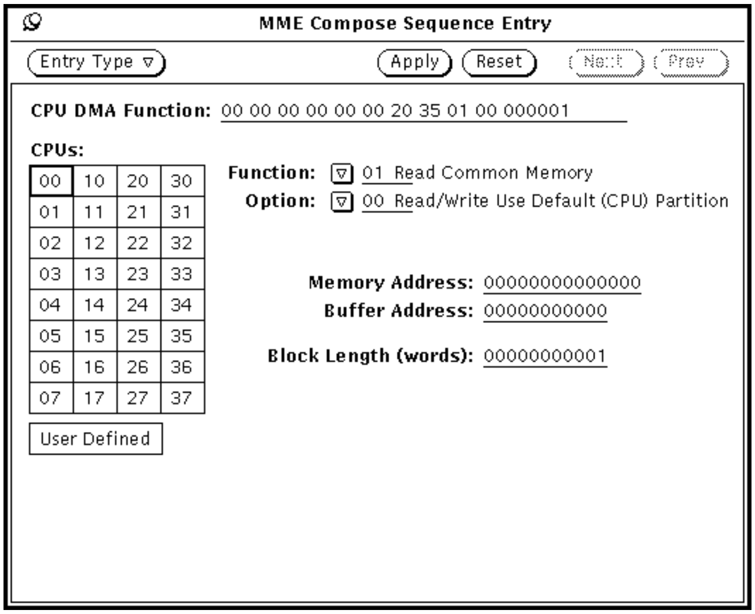
Table 3. CPU Logic Monitor Command Fields

Field	Description
Delay After Trigger	Number of clock periods the logic monitor continues recording after a trigger condition occurs
One Word/Trigger	Record 1 word per trigger condition option (enabled or disabled)
Record Mode	Type of recording the logic monitor should do (number of clock periods to record and number of test points to record per clock period)
Set Break Point	Breakpoint used to stop CPU execution
Source Address	MME buffer address of data to write to the logic monitor
Source Length	Length of data block to write to the logic monitor
Readout Address	MME buffer address to receive data from a logic monitor readout buffer command
Readout Length	Length of data block for data received from logic monitor readout buffer command
Readout Action	Action to perform based on results of data from logic monitor readout buffer command
Readout Label	Label to jump to if the readout action is a goto label command

4. Click on (Apply) to place the function in the Sequence scroll box.

CPU DMA Functions

The CPU DMA functions enable you to control the direct memory access (DMA) activity that can be performed. Choose **Entry Type –> CPU –> DMA Function** to access the CPU DMA functions. The following MME Compose Sequence Entry window appears:



CPU DMA Function: 00 00 00 00 00 00 20 35 01 00 000001

CPUs:

00	10	20	30
01	11	21	31
02	12	22	32
03	13	23	33
04	14	24	34
05	15	25	35
06	16	26	36
07	17	27	37

Function: ▽ 01 Read Common Memory
Option: ▽ 00 Read/Write Use Default (CPU) Partition

Memory Address: 00000000000000
Buffer Address: 00000000000

Block Length (words): 00000000001

Notice that the maintenance channel command data for the current function is generated and displayed in the CPU DMA Function field. This data is updated as you select different functions. Perform the following procedure to create an entry with CPU DMA functions:

1. Click on the CPU that you want to use.

   NOTE: Based on the current configuration data and the CPU that you select, MME automatically generates a route code for the function. If you want to force the route code to a specific value, you must click on User Defined and change the route code information in the CPU DMA Function field.

2. From Function: ▽, choose a DMA function.

   Refer to the *Maintenance Channel* document, publication number HTM-006-B; and the *Triton Maintenance System Engineering Note*, number PRN-0957; for more information about the DMA functions.

3. From Option: ▽, choose an option.

Refer to the *Maintenance Channel* document, publication number HTM-006-B; and the *Triton Maintenance System Engineering Note*, publication number PRN-0957; for more information about the available options.

4. In the Memory Address field, enter the starting address of the data block in mainframe memory that you want to manipulate.

5. In the Buffer Address field, enter the starting address of the data block in the MME buffer that you want to manipulate.

6. In the Block Length (words) field, enter the size of the data block (in words) that you want manipulate.

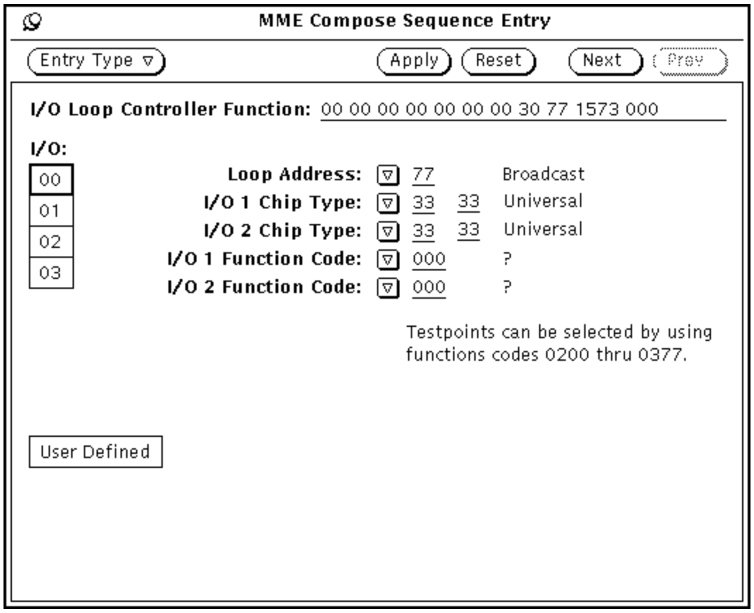
7. Click on Apply to place the function in the Sequence scroll box.

I/O Functions

The I/O functions enable you to perform I/O maintenance and configuration functions and I/O logic monitor functions.

I/O Loop Controller Functions

The I/O loop controller functions are configuration and maintenance functions that you can send to an IO module through an IO module loop controller. Choose **Entry Type –> I/O –> Loop Controller Function** to access the I/O loop controller functions. The following MME Compose Sequence Entry window appears:



I/O Loop Controller Function: 00 00 00 00 00 00 00 30 77 1573 000

I/O: 00 01 02 03

Loop Address:	▽	77		Broadcast
I/O 1 Chip Type:	▽	33	33	Universal
I/O 2 Chip Type:	▽	33	33	Universal
I/O 1 Function Code:	▽	000		?
I/O 2 Function Code:	▽	000		?

Testpoints can be selected by using functions codes 0200 thru 0377.

Notice that the maintenance channel command data for the current function is generated and displayed in the I/O Loop Controller Function field. This data is updated as you select different functions. To create an entry with I/O loop controller functions, perform the following procedure:

1. Click on the IO module that you want to use.

   NOTE: Based on the current configuration data and the IO module that you select, MME automatically generates a route code for the function. If you want to force the route code to a specific value, you must click on User Defined and change the route code information in the I/O Loop Controller Function field.

2. From Loop Address: ▽, choose the loop address to which you want the function to go.

3. If the IO module that you want to use is an IO01 module: from I/O 1 Chip Type: ▽, choose the chip type(s) to which you want the function to go.

4. If the IO module that you want to use is an IO02 module: from I/O 2 Chip Type: ▽, choose the chip type(s) to which you want the function to go.

5. If the IO module that you want to use is an IO01 module: from I/O 1 Function Code: ▽, choose a configuration or maintenance function.

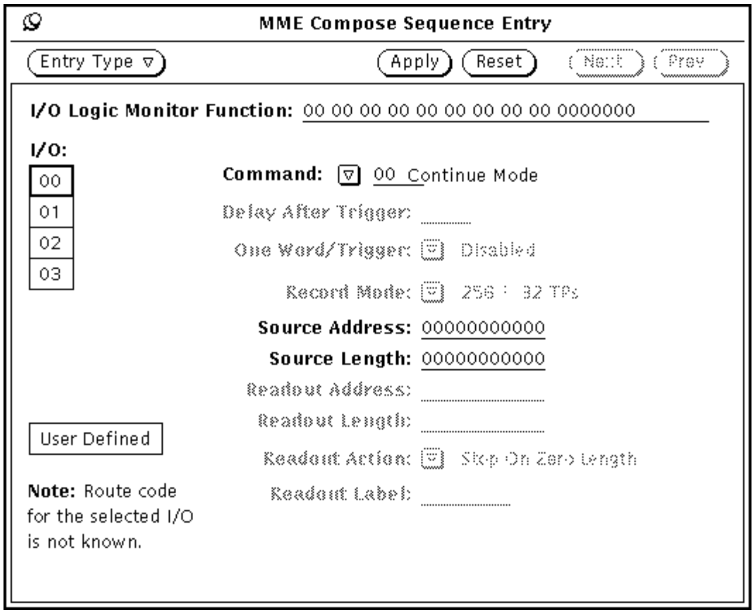
Refer to the *Maintenance Channel* document, publication number HTM-006-B; and the *Triton Maintenance System Engineering Note*, publication number PRN-0957; for more information about the IO01 module functions.

6. If the IO module that you want to use is an IO02 module: from I/O 2 Function Code: ▽, choose a configuration or maintenance function.

7. Click on Apply to place the function in the Sequence scroll box.

I/O Logic Monitor Functions

The I/O logic monitor functions enable you to control the activity of the logic monitors on the IO modules. Choose **Entry Type –> I/O –> Logic Monitor Function** to access the I/O logic monitor functions. The following MME Compose Sequence Entry window appears:



I/O Logic Monitor Function: 00 00 00 00 00 00 00 00 00 0000000

I/O: 00 01 02 03

Command: ▽ 00 Continue Mode

Delay After Trigger:

One Word/Trigger: ▽ Disabled

Record Mode: ▽ 256 : 32 TPs

Source Address: 00000000000

Source Length: 00000000000

Readout Address:

Readout Length:

Readout Action: ▽ Skip On Zero Length

Readout Label:

Note: Route code for the selected I/O is not known.

Notice that the maintenance channel command data for the current function is generated and displayed in the I/O Logic Monitor Function field. This data is updated as you select different functions. Perform the following procedure to create an entry with I/O logic monitor functions:

1. Click on the IO module that you want to use.

   NOTE: Based on the current configuration data and the IO module that you select, MME automatically generates a route code for the function. If you want to force the route code to a specific value, you must click on User Defined and change the route code information in the I/O Logic Monitor Function field.

2. From Command: ▽, choose an I/O logic monitor function.

   Refer to the *Maintenance Channel* document, publication number HTM-006-B; and the *Triton Maintenance System Engineering Note*, publication number PRN-0957; for more information about these commands.

3. Update any fields that activate. Table 4 describes the fields.

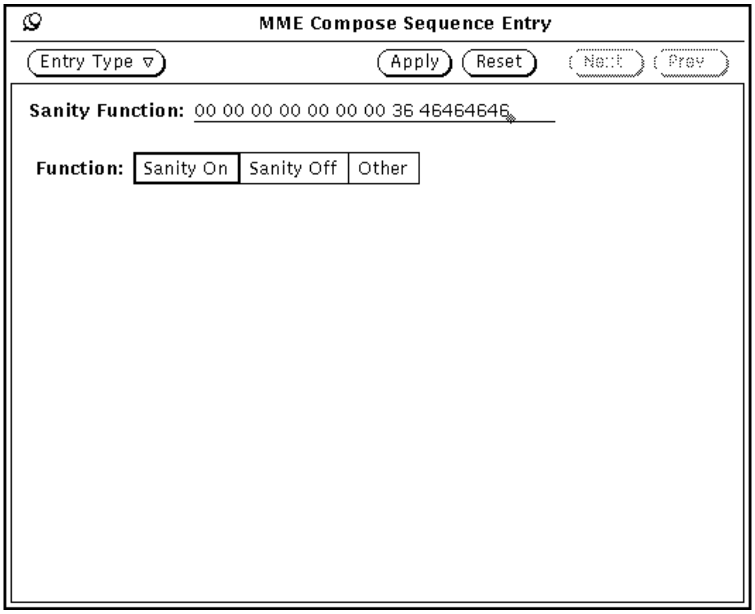
Table 4. I/O Logic Monitor Command Fields

Field	Description
Delay After Trigger	Number of clock periods the logic monitor continues recording after a trigger condition occurs
One Word/Trigger	Record 1 word per trigger condition option (enabled or disabled)
Record Mode	Type of recording the logic monitor should do (number of clock periods to record and number of test points to record per clock period)
Source Address	MME buffer address of data to write to the logic monitor
Source Length	Length of data block to write to the logic monitor
Readout Address	MME buffer address to receive data from a logic monitor readout buffer command
Readout Length	Length of data block for data received from logic monitor readout buffer command
Readout Action	Action to perform based on results of data from logic monitor readout buffer command
Readout Label	Label to jump to if the readout action is a goto label command

4. Click on (Apply) to place the function in the Sequence scroll box.

I/O Sanity Generator Functions

The I/O sanity generator functions enable you to start and stop the sanity generator. Choose **Entry Type –> I/O –> Sanity Generator** to access the I/O sanity generator functions. The following MME Compose Sequence Entry window appears:



Sanity Function: 00 00 00 00 00 00 00 36 46464646

Function: [✓ Sanity On] Sanity Off Other

Sanity On

The Sanity On function starts the sanity generator. Perform the following procedure to create a Sanity On function:

1. Click on Function: Sanity On.
2. Click on Apply to place the function in the Sequence scroll box.

Sanity Off

The Sanity Off function stops the sanity generator. Perform the following procedure to create a Sanity Off function:

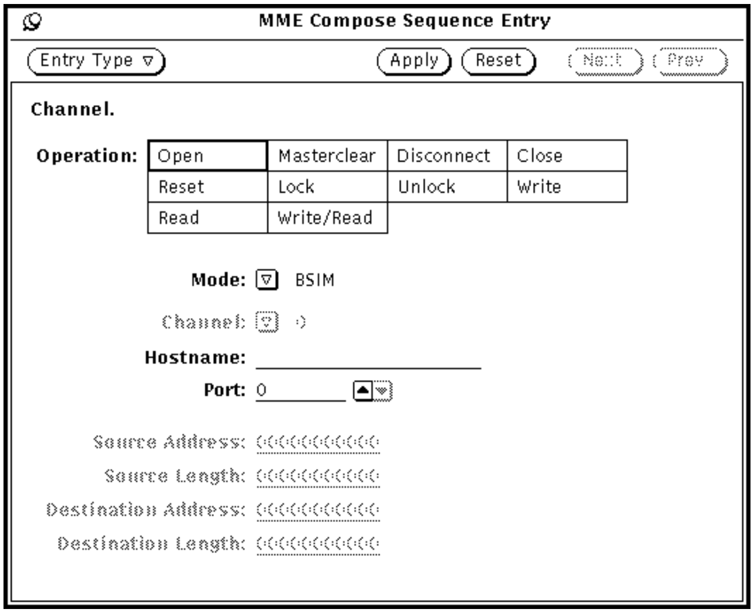
1. Click on Function: Sanity Off.
2. Click on Apply to place the function in the Sequence scroll box.

Other

Currently, this function is not implemented.

Channel Functions

The channel functions enable you to control a front-end interface channel. Choose **Entry Type –> Channel** to access the channel functions. The following MME Compose Sequence Entry window appears:



Channel.

Operation:

Open	Masterclear	Disconnect	Close
Reset	Lock	Unlock	Write
Read	Write/Read		

Mode: ▽ BSIM

Channel: ▽ 0

Hostname:

Port: 0

Source Address: 0000000000

Source Length: 0000000000

Destination Address: 0000000000

Destination Length: 0000000000

Open

The open channel function opens an FEI channel. Perform the following procedure to create an open channel function:

1. Click on Operation: Open to select the open channel function.
2. Click Channel: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, or 15 to specify the channel that you want to open.
3. From Mode: ▽, choose the channel mode that you want to use. Refer to Table 5 for descriptions of the options.

Table 5. Channel Mode Options

Option	Description
BSIM	Used for Boolean simulation mode
ISIM	Used for instruction simulation mode
Sun4 Boundary Scan (fymt_bs)	Used for boundary scan mode functions with a device driver for a boundary scan module in the IO module slot (tester only)
Sun4 Maintenance (fymt_mc)	Used for maintenance channel functions other than boundary scan mode functions with a device driver for a boundary scan module in the IO module slot (tester only)
Sun5 Maintenance (fymtm_bs)	Used for boundary scan mode functions with a device driver for a boundary scan module in the normal operating location
Sun5 Maintenance (fymtm_mc)	Used for maintenance channel functions other than boundary scan mode functions with a device driver for a boundary scan module in the normal operating location
SPV BS/Maintenance (SIM)	This option is no longer valid and will be removed from future versions of MME. Do not use this option.
SPV BS/Maintenance	Used with the boundary scan/maintenance channel subchannel of the supervisory channel
SPV Loopback	Used with the loopback subchannel of the supervisory channel
Data Channel (fymc)	Used for the special driver needed to run the end-to-end test on a LOSP channel

4. Click on (Apply) to place the function in the Sequence scroll box.

Masterclear

The masterclear channel function master clears the FEI channel. Perform the following procedure to create a masterclear function:

1. Click on Operation: [illegible] to select the masterclear function.
2. Click on (Apply) to place the function in the Sequence scroll box.

Disconnect

The disconnect channel function disconnects the FEI channel. Perform the following procedure to create a disconnect channel function:

1. Click on Operation: Disconnect to select the disconnect function.
2. Click on Apply to place the function in the Sequence scroll box.

Close

The close channel function closes the open FEI channel. Perform the following procedure to create a close channel function:

1. Click on Operation: Close to select the close function.
2. Click on Apply to place the function in the Sequence scroll box.

Reset

The reset channel function resets the FEI channel. Perform the following procedure to create a reset channel function:

1. Click on Operation: Reset to select the reset function.
2. Click on Apply to place the function in the Sequence scroll box.

Lock

The lock channel function locks the FEI channel. Perform the following procedure to create a lock channel function:

1. Click on Operation: Lock to select the lock function.
2. Click on Apply to place the function in the Sequence scroll box.

Unlock

The unlock channel function unlocks the FEI channel. Perform the following procedure to create an unlock channel function:

1. Click on Operation: Unlock to select the unlock function.
2. Click on Apply to place the function in the Sequence scroll box.

Write

NOTE: You must open a channel with the open command before you initiate a write command. If you do not open a channel, MME displays Can't do raw write on current channel in the MME Log window. This message indicates that you tried to write data to the maintenance channel.

The write channel function writes data to the FEI channel; MME gets the data from the MME buffer. Perform the following procedure to create a write channel function:

1. Click on Operation: Write to select the write function.
2. In the Source Address field, enter the starting address of the block of data (in the MME buffer) that you want to write to the FEI channel.
3. In the Source Length field, enter the length of the block of data that you want to write to the channel.
4. Click on Apply to place the function in the Sequence scroll box.

Read

NOTE: You must open a channel with the open command before you initiate a read command. If you do not open a channel, MME displays Can't do raw read on current channel in the MME Log window. This message indicates that you tried to read data from the maintenance channel.

The read channel function reads data from the FEI channel; MME puts the data into the MME buffer. Perform the following procedure to create a read channel function:

1. Click on Operation: Read to select the read function.
2. In the Destination Address field, enter the first address in the MME buffer that should receive the data that is read from the FEI channel.
3. In the Destination Length field, enter the length of the block of data that you want to read from the channel.
4. Click on Apply to place the function in the Sequence scroll box.

Write/Read

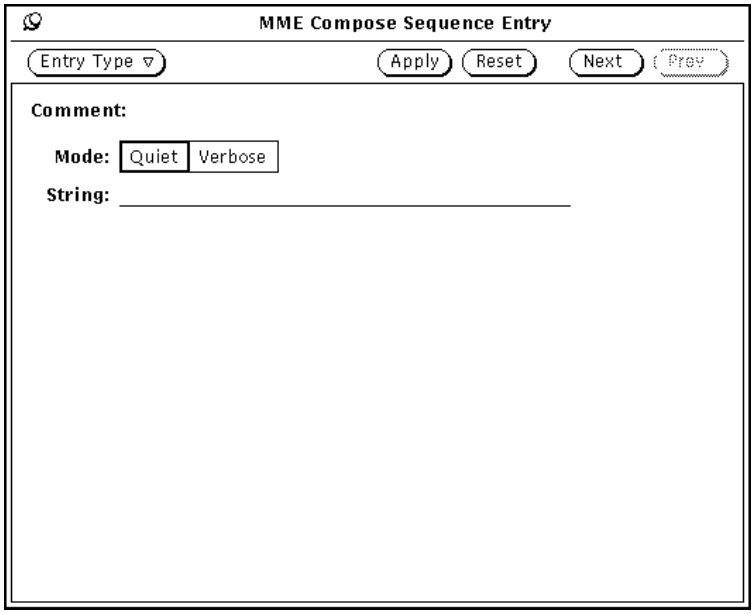
NOTE: You must open a channel with the open command before you initiate a write or read command. If you do not open a channel, MME displays Can't do raw write on current channel in the MME Log window. This message indicates that you tried to write data to the maintenance channel. (The message is for the write function because the write function executes first.)

The write/read channel function writes data to the FEI channel and then reads data from the FEI channel. MME gets the data to be written from the MME buffer, and MME puts the data that is read from the channel into the MME buffer. Perform the following procedure to create a write/read channel function:

1. Click on Operation: Write/Read to select the write/read function.
2. In the Source Address field, enter the starting address of the block of data (in the MME buffer) that you want to write to the FEI channel.
3. In the Source Length field, enter the length of the block of data that you want to write to the channel.
4. In the Destination Address field, enter the first address in the MME buffer that should receive the data that is read from the FEI channel.
5. In the Destination Length field, enter the length of the block of data that you want to read from the channel.
6. Click on Apply to place the function in the Sequence scroll box.

Comments

You can enter comments into a sequence to document what the sequence does. There are two types of comments: quiet and verbose. Choose **Entry Type –> Comment** to access the comment functions. The following MME Compose Sequence Entry window appears:



Comment:

Mode: Quiet Verbose

String:

Quiet

Quiet comments are not displayed in the MME Log window as the sequence executes. Perform the following procedure to create a quiet comment:

1. Click on Mode: Quiet to select a quiet comment.
2. In the String field, type the comment and press the Return key.
3. Click on (Apply) to place the comment in the Sequence scroll box.

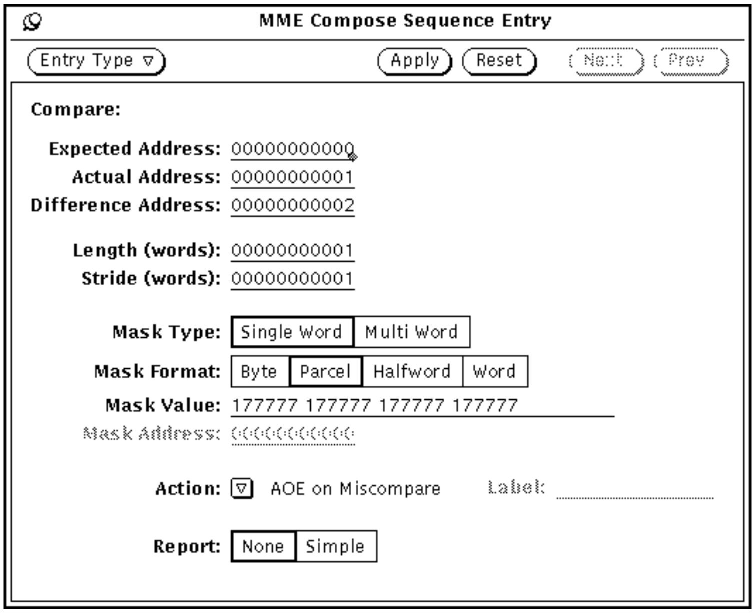
Verbose

Verbose comments are displayed in the MME Log window as the sequence executes. Perform the following procedure to create a verbose comment:

1. Click on Mode: Verbose to select a verbose comment.
2. In the String field, type the comment and press the Return key.
3. Click on (Apply) to place the comment in the Sequence scroll box.

Compare Function

The compare function compares data in the MME buffer. This function is used to compare actual values with expected values. Choose **Entry Type –> Compare** to access the compare function. The following MME Compose Sequence Entry window appears:



Compare:

Expected Address: 00000000000
Actual Address: 00000000001
Difference Address: 00000000002

Length (words): 00000000001
Stride (words): 00000000001

Mask Type: Single Word | Multi Word
Mask Format: Byte | Parcel | Halfword | Word
Mask Value: 177777 177777 177777 177777
Mask Address: 00000000000

Action: ▽ AOE on Miscompare Label:

Report: None | Simple

Perform the following procedure to create a compare function:

1. In the Expected Address field, enter the MME buffer address that contains the expected data and press the Return key.

2. In the Actual Address field, enter the MME buffer address that contains the actual data and press the Return key.

3. In the Difference Address field, enter the MME buffer address where you want to store the difference between the expected and actual values and press the Return key.

4. In the Length (words) field, enter the length of the data block (in words) that you want to compare and press the Return key.

5. In the Stride (words) field, enter the stride that you want to use and press the Return key. The stride specifies which words you want to compare. For example, a stride of 1 compares every word, and a stride of 2 compares every other word.

6. Click on the mask type that you want to use. The mask indicates which bits are compared. Click on Mask Type: [Single word] to use the 1-word mask that is specified in the Mask Value field (the format of this word is specified by the Mask Format settings). Click on Mask Type: [Multi word] to use a multiple-word mask that is located at the memory location specified in the Mask Address field.

7. Specify the action that MME should perform based on the results of the comparison:

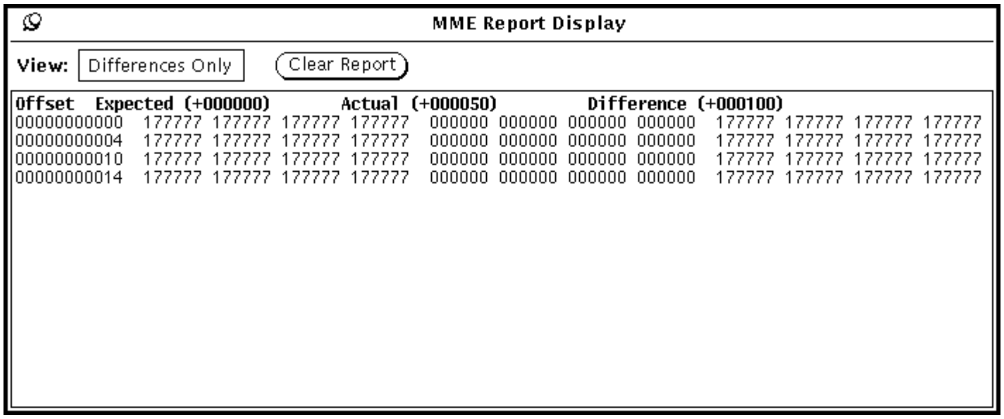
   - Choose Action: ▽ **AOE on Miscompare** to have MME stop sequence execution when the function detects a difference between the actual and expected data.

   - Choose Action: ▽ **Branch on Miscompare** to have MME jump to a label in the sequence (specify the label in the Label field) when the function detects a difference between the actual and expected data.

   - Choose Action: ▽ **AOE on Compare** to have MME stop sequence execution when the actual data matches the expected data.

   - Choose Action: ▽ **Branch on Compare** to have MME jump to a label in the sequence (specify the label in the Label field) when the actual data matches the expected data.

8. Click on Report: [None] or [Simple] to specify that you want a report when a difference is detected. To view the report, choose **View –> Report**. For example, the following report might be generated.



MME Report Display

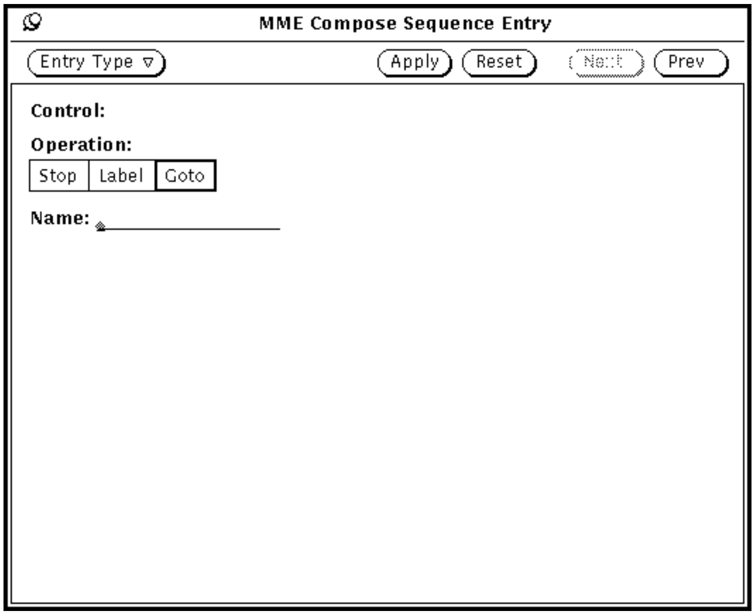
View: Differences Only

```
Offset   Expected (+000000)          Actual (+000050)           Difference (+000100)
00000000000   177777 177777 177777 177777   000000 000000 000000 000000   177777 177777 177777 177777
00000000004   177777 177777 177777 177777   000000 000000 000000 000000   177777 177777 177777 177777
00000000010   177777 177777 177777 177777   000000 000000 000000 000000   177777 177777 177777 177777
00000000014   177777 177777 177777 177777   000000 000000 000000 000000   177777 177777 177777 177777
```

9. Click on (Apply) to place the function in the Sequence scroll box.

Control Functions

The control functions are used to direct the flow of function execution within a sequence. These functions change the program flow from the normal top-to-bottom execution within the scroll box. This enables conditional execution of functions in a sequence. Choose **Entry Type –> Control** to access the control functions. The following MME Compose Sequence Entry window appears:



Goto

The goto function transfers sequence execution to the command that follows the label specified in the goto function; the label is defined with the label function. Perform the following procedure to create a goto function:

1. Click on Operation: Goto.
2. In the Name field, enter the name of the label to which you want to go (for example, Label1).
3. Click on Apply to place the function in the Sequence scroll box.

When execution reaches the goto function in a sequence, execution transfers to the command that follows the label that is specified in the goto function.

Label

The label function creates a label in the sequence that acts as a marker to which execution can be transferred by a goto or compare function. Perform the following procedure to create a label:

1. Click on Operation: Label.
2. In the Name field, enter the name of the label (for example, Label1).
3. Click on Apply to place the function in the Sequence scroll box.

Stop

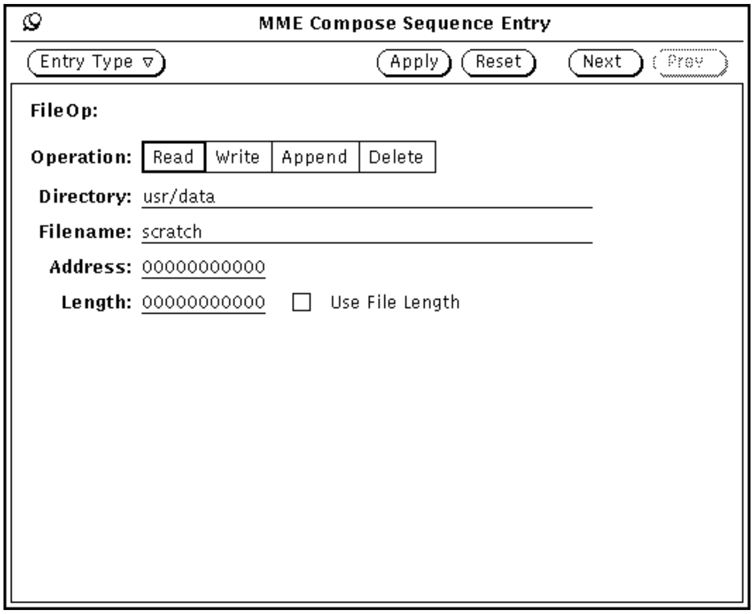
The stop function stops execution of the sequence. Perform the following procedure to create a stop function:

1. Click on Operation: Stop.
2. Click on Apply to place the function in the Sequence scroll box.

When execution reaches the stop function in the sequence, execution of the sequence stops.

File Operation Functions

The file operation functions enable you to read, write, append, and delete data files. Choose **Entry Type –> FileOp** to access the file operation functions. The following MME Compose Sequence Entry window appears:



FileOp:

Operation: Read | Write | Append | Delete

Directory: usr/data

Filename: scratch

Address: 00000000000

Length: 00000000000 □ Use File Length

Read

The read file operation function enables you to read data from a file into the MME buffer. Perform the following procedure to create a read file operation function:

1. Click on Operation: Read.

2. In the Directory field, enter the directory where the data file is located.

3. In the Filename field, enter the name of the file that you want to read.

4. In the Address field, enter the MME buffer address where you want to store the data that is read.

5. In the Length field, enter the size of the data block that you want to read.

NOTE: Click on the Use File Length check box to read the entire file.

6. Click on Apply to place the function in the Sequence scroll box.

Write

The write file operation function enables you to write data from the MME buffer to a file. Perform the following procedure to create a write file operation function:

1. Click on Operation: Write.
2. In the Directory field, enter the directory where the data file is located.
3. In the Filename field, enter the name of the file that you want to write.
4. In the Address field, enter the MME buffer address of the data that you want to write.
5. In the Length field, enter the size of the data block that you want to write.
6. Click on Apply to place the function in the Sequence scroll box.

Append

The append file operation function enables you to append data from the MME buffer to a file that you previously created. Perform the following procedure to create an append file operation function:

1. Click on Operation: Append.
2. In the Directory field, enter the directory where the data file is located.
3. In the Filename field, enter the name of the file that you want to append.
4. In the Address field, enter the MME buffer address of the data that you want to append to the file.
5. In the Length field, enter the size of the data block that you want to append to the file.
6. Click on Apply to place the function in the Sequence scroll box.

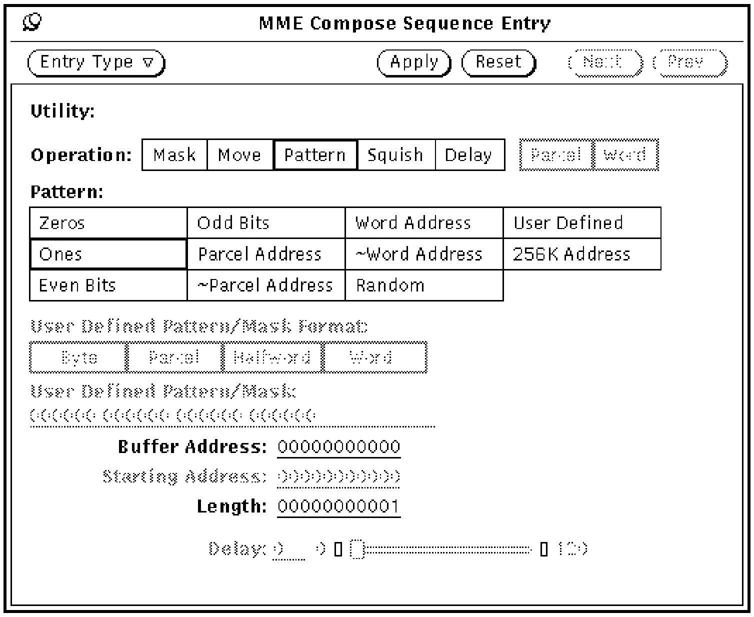
Delete

The delete file operation function enables you to delete a data file that you no longer need. Perform the following procedure to create a delete file operation function:

1. Click on Operation: Delete.
2. In the Directory field, enter the directory where the data file is located.
3. In the Filename field, enter the name of the file that you want to delete.
4. Click on (Apply) to place the function in the Sequence scroll box.

Utilities

Several utilities can be added to sequences that you execute in compose mode. Choose **Entry Type –> Utility** to access the utilities. The following MME Compose Sequence Entry window appears:



Operation: Pattern

Pattern: Ones

Buffer Address: 00000000000

Length: 00000000001

Delay

The delay utility creates a pause (in seconds) in the execution of a sequence of functions. Perform the following procedure to create a delay utility:

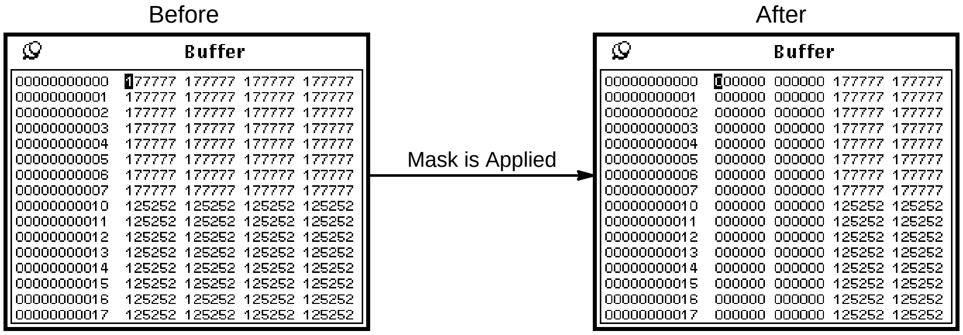
1. Click on Operation: Delay.

2. Specify the delay by entering a value in the Delay field or by moving the slider until the desired value is displayed in the field.

3. Click on Apply to place the utility in the Sequence scroll box.

Mask

The mask utility applies a mask to an area in the MME buffer. If a bit in the mask is set to 0_2, the data value in that bit position in an MME buffer word is set to 0_2. If a bit in the mask is set to 1_2, the data value in that bit position in an MME buffer word retains its value; for example, a mask value of 000000_8 000000_8 177777_8 177777_8 masks off the top half of each word, as shown in Figure 9.



Buffer

```
00000000000   177777 177777 177777 177777
00000000001   177777 177777 177777 177777
00000000002   177777 177777 177777 177777
00000000003   177777 177777 177777 177777
00000000004   177777 177777 177777 177777
00000000005   177777 177777 177777 177777
00000000006   177777 177777 177777 177777
00000000007   177777 177777 177777 177777
00000000010   125252 125252 125252 125252
00000000011   125252 125252 125252 125252
00000000012   125252 125252 125252 125252
00000000013   125252 125252 125252 125252
00000000014   125252 125252 125252 125252
00000000015   125252 125252 125252 125252
00000000016   125252 125252 125252 125252
00000000017   125252 125252 125252 125252
```

Buffer

```
00000000000   000000 000000 177777 177777
00000000001   000000 000000 177777 177777
00000000002   000000 000000 177777 177777
00000000003   000000 000000 177777 177777
00000000004   000000 000000 177777 177777
00000000005   000000 000000 177777 177777
00000000006   000000 000000 177777 177777
00000000007   000000 000000 177777 177777
00000000010   000000 000000 125252 125252
00000000011   000000 000000 125252 125252
00000000012   000000 000000 125252 125252
00000000013   000000 000000 125252 125252
00000000014   000000 000000 125252 125252
00000000015   000000 000000 125252 125252
00000000016   000000 000000 125252 125252
00000000017   000000 000000 125252 125252
```

Figure 9. Mask Utility Example

Perform the following procedure to create a mask utility:

1. Click on Operation: Mask.

2. Click on Pattern: Byte, Parcel, Halfword, or Word to specify the mask format.

3. In the User Defined Pattern/Mask field, enter the mask that you want to use.

4. In the Source Address field, specify the starting address of the data block in the MME buffer that you want to mask.

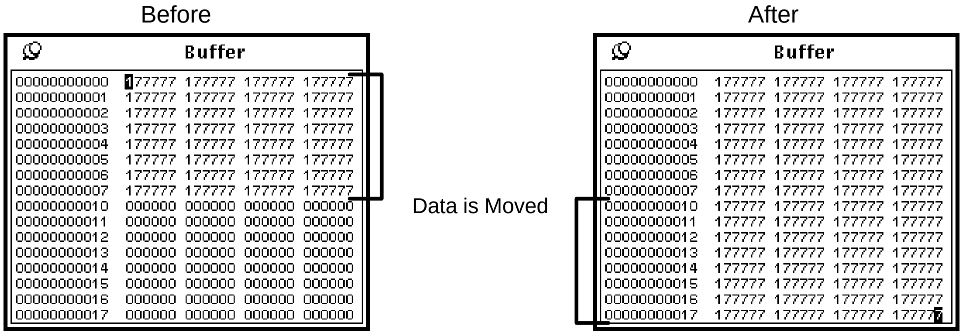
5. In the Length field, specify the length of the data block that you want to mask.

6. Click on Apply to place the utility in the Sequence scroll box.

When execution reaches the mask utility in the scroll box, the specified mask is applied to the specified data in the MME buffer. Use this utility to mask out (set to zero) specific bits in a word.

Move

The move utility copies a block of data from one location in the MME buffer to another location. Figure 10 shows an example of the move utility.



Before

```
Buffer
00000000000  177777 177777 177777 177777
00000000001  177777 177777 177777 177777
00000000002  177777 177777 177777 177777
00000000003  177777 177777 177777 177777
00000000004  177777 177777 177777 177777
00000000005  177777 177777 177777 177777
00000000006  177777 177777 177777 177777
00000000007  177777 177777 177777 177777
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
```

Data is Moved

After

```
Buffer
00000000000  177777 177777 177777 177777
00000000001  177777 177777 177777 177777
00000000002  177777 177777 177777 177777
00000000003  177777 177777 177777 177777
00000000004  177777 177777 177777 177777
00000000005  177777 177777 177777 177777
00000000006  177777 177777 177777 177777
00000000007  177777 177777 177777 177777
00000000010  177777 177777 177777 177777
00000000011  177777 177777 177777 177777
00000000012  177777 177777 177777 177777
00000000013  177777 177777 177777 177777
00000000014  177777 177777 177777 177777
00000000015  177777 177777 177777 177777
00000000016  177777 177777 177777 177777
00000000017  177777 177777 177777 177777
```

Figure 10. Move Utility Example

Perform the following procedure to create a move utility:

1. Click on Operation: Move.

2. Click on Parcel or Word to specify the type of data that you want to move.

3. In the Source Address field, enter the starting MME buffer address of the data block that you want to move.

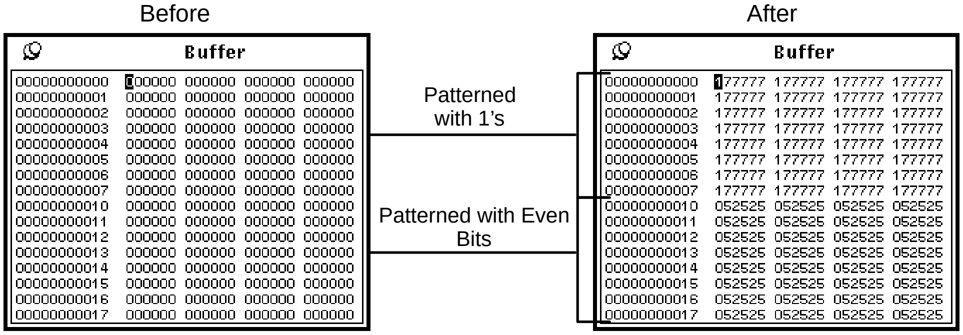
4. In the Destination Address field, enter the MME buffer address to which you want to move the data.

5. In the Length field, enter the length of the data block.

6. Click on (Apply) to place the utility in the Sequence scroll box.

Pattern

The pattern utility patterns a block of MME buffer memory with 0’s, 1’s, even bits, odd bits, address, complement address, random, or user data. Figure 11 shows an example of the pattern utility.



```
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
```

```
00000000000   177777 177777 177777 177777
00000000001   177777 177777 177777 177777
00000000002   177777 177777 177777 177777
00000000003   177777 177777 177777 177777
00000000004   177777 177777 177777 177777
00000000005   177777 177777 177777 177777
00000000006   177777 177777 177777 177777
00000000007   177777 177777 177777 177777
00000000010   052525 052525 052525 052525
00000000011   052525 052525 052525 052525
00000000012   052525 052525 052525 052525
00000000013   052525 052525 052525 052525
00000000014   052525 052525 052525 052525
00000000015   052525 052525 052525 052525
00000000016   052525 052525 052525 052525
00000000017   052525 052525 052525 052525
```

Figure 11. Pattern Utility Example

Perform the following procedure to create a pattern utility:

1. Click on Operation: [Pattern].

2. Click on Pattern: [Zeros], [Ones], [Even Bits], [Odd Bits], [Address (Parcel)], [~Address (Parcel)], [Address (Word)], [~Address (Word)], [Random], or [User Defined] to specify the pattern that you want to use.

   If you clicked on [User], click on User Defined Pattern/Mask Format: [Byte], [Parcel], [Halfword], or [Word]. Then, enter the pattern that you want to use in the User Defined Pattern/Mask field.

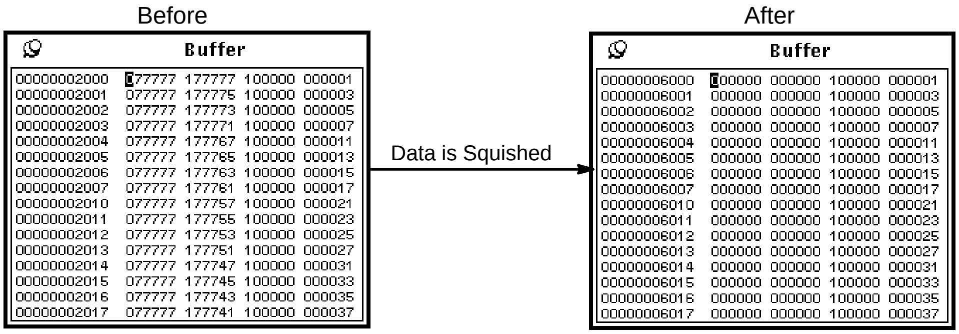
3. In the Buffer Address field, enter the starting address of the data block you want to pattern.

4. In the Length field, enter the length of the data block that you want to pattern.

5. Click on (Apply) to place the utility in the Sequence scroll box.

Squish

The squish utility manipulates data that is returned from a logic monitor read-out buffer function when a logic monitor is in 8 x 1024 mode. In 8 x 1024 mode, valid data bits are contained in half a word. The squish utility reads the validity flags (bits 63 and 31) to determine which halfword contains the valid data; the valid data is placed in the lower halfword of the destination location, and the upper halfword is filled with zeroes. Figure 12 shows an example of the squish utility.



```
00000002000  077777 177777 100000 000001
00000002001  077777 177775 100000 000003
00000002002  077777 177773 100000 000005
00000002003  077777 177771 100000 000007
00000002004  077777 177767 100000 000011
00000002005  077777 177765 100000 000013
00000002006  077777 177763 100000 000015
00000002007  077777 177761 100000 000017
00000002010  077777 177757 100000 000021
00000002011  077777 177755 100000 000023
00000002012  077777 177753 100000 000025
00000002013  077777 177751 100000 000027
00000002014  077777 177747 100000 000031
00000002015  077777 177745 100000 000033
00000002016  077777 177743 100000 000035
00000002017  077777 177741 100000 000037
```

```
00000006000  000000 000000 100000 000001
00000006001  000000 000000 100000 000003
00000006002  000000 000000 100000 000005
00000006003  000000 000000 100000 000007
00000006004  000000 000000 100000 000011
00000006005  000000 000000 100000 000013
00000006006  000000 000000 100000 000015
00000006007  000000 000000 100000 000017
00000006010  000000 000000 100000 000021
00000006011  000000 000000 100000 000023
00000006012  000000 000000 100000 000025
00000006013  000000 000000 100000 000027
00000006014  000000 000000 100000 000031
00000006015  000000 000000 100000 000033
00000006016  000000 000000 100000 000035
00000006017  000000 000000 100000 000037
```

Figure 12. Squish Utility Example

Perform the following procedure to create a squish utility:

1. Click on Operation: Squish.

2. In the Source Address field, specify the starting address of the data block of 8 x 1024 mode data.

3. In the Destination Address field, enter the starting address of the destination data block.

4. In the Length field, enter the length of the data block that you want to squish.

5. Click on Apply to place the utility in the Sequence scroll box.

ENVIRONMENT 1

Environment 1 is a component of the Mainframe Maintenance Environment (MME) software package that field engineers use to troubleshoot CRAY T90 series mainframes. Typically, in environment 1, only one diagnostic program, utility, or loop is loaded into mainframe memory at a time. Once this program is loaded into mainframe memory, it is called a control point. Because only one control point is usually loaded in mainframe memory at a time, control points used in environment 1 have access to the entire mainframe or portion of the mainframe that MME is using. Control points can be single- or multiple-CPU control points. To test multiple CPUs, you can assign multiple CPUs to one control point, and each CPU runs the same code that is stored once in memory.

MME still runs in the maintenance workstation (MWS) or system workstation (SWS), but all testing occurs in the mainframe. Information passes through the maintenance channel to MME. MME monitors the performance of the control point that is active in mainframe memory and updates information that is available through the MME runtime information displays. When only one control point is loaded (which is the normal use of environment 1), all mainframe memory addresses are absolute, which means that they are based on a starting address of zero.

Control point sections are stored as individual files. All sections for a control point are stored in a common directory. Only one section is actually loaded in mainframe memory at a time. That section is called the current section. MME loads and removes the test sections from mainframe memory according to the minimum and maximum pass counts that the code for that section specifies. The minimum value specifies the minimum number of passes that must occur before MME can load a different section into memory. The maximum value specifies the number of passes necessary before a section is no longer loaded into memory and run. Diagnostic programmers define these values in the program code; however, you can customize the values by saving a new version of a control point. Refer to “File –> Save –> Control Point” in the *MME Interface Reference*, publication number HDM-008-A, for more information.

The following procedure provides a general overview of the process for using MME environment 1. This section includes related information for each of the following steps of the process.

1. Start MME in environment 1.
2. Load a layout (optional).
3. Allocate resources (optional).
4. Load a control point.
5. Assign a CPU to the current control point.
6. Click on (Go).
7. Monitor the progress of control point execution.
8. Click on (Halt ▷).

Start MME in Environment 1

You can start MME in environment 1 from a UNIX command prompt or from the OpenWindows Workspace menu.

NOTE: For information about starting MME environment 1 from a Service Center through a hub, refer to the *Remote Support* document, publication number HMM-106-A.

CAUTION

MME performs maintenance channel functions that will hang UNICOS if UNICOS is running in the mainframe when you start MME.

To prevent this from accidentally occurring, ensure that the Owner setting in the SCE base window is set to OS for the logical partition in which UNICOS is running when UNICOS is running in the mainframe. MME cannot access a logical partition if the OS owns it.

From a UNIX Prompt

To start MME environment 1 from a UNIX prompt, enter one of the following commands:

•	**`mme –1`**	to use a front-end interface (FEI) channel
•	**`mme –1 –sim`**	to use the simulator
•	**`mme –1 –debug`**	to use the simulator and bugger/debugger

NOTE: You may also enter any of the command line options that Table 6 lists.

Table 6. Environment 1 Command Line Options

Option	Description
`–client`	Start the MME client only
`–config` *file*	Configure MME with the configuration data stored in the file specified by *file*.
`–copy` *num*	Connect to maintenance software assigned the copy number specified by *num*. **NOTE:** Copy numbers are necessary only when you run multiple copies of MME on the same MWS or SWS (for example, when you run several MME copies with the simulator or when you use MME to support multiple CRAY T90 series mainframes connected to the same MWS or SWS).
`–io` *num*	Use the CPU specified by *num* to perform input and output operations
`–kill`	Kill any running MME, SCE, or LME applications before starting a new copy of MME
`–remote` *host*	Start the MME client only and connect the client to the MME server that is running on the remote host specified by *host*
`–server`	Start the MME server only

From the OpenWindows Workspace Menu

You can start environment 1 from the OpenWindows Workspace menu on either an MWS or an SWS.

MWS Workspace Menu Options

Figure 13 shows the OpenWindows Workspace menu options that you should choose on an MWS to start environment 1 with an FEI channel. Choose any copy number.

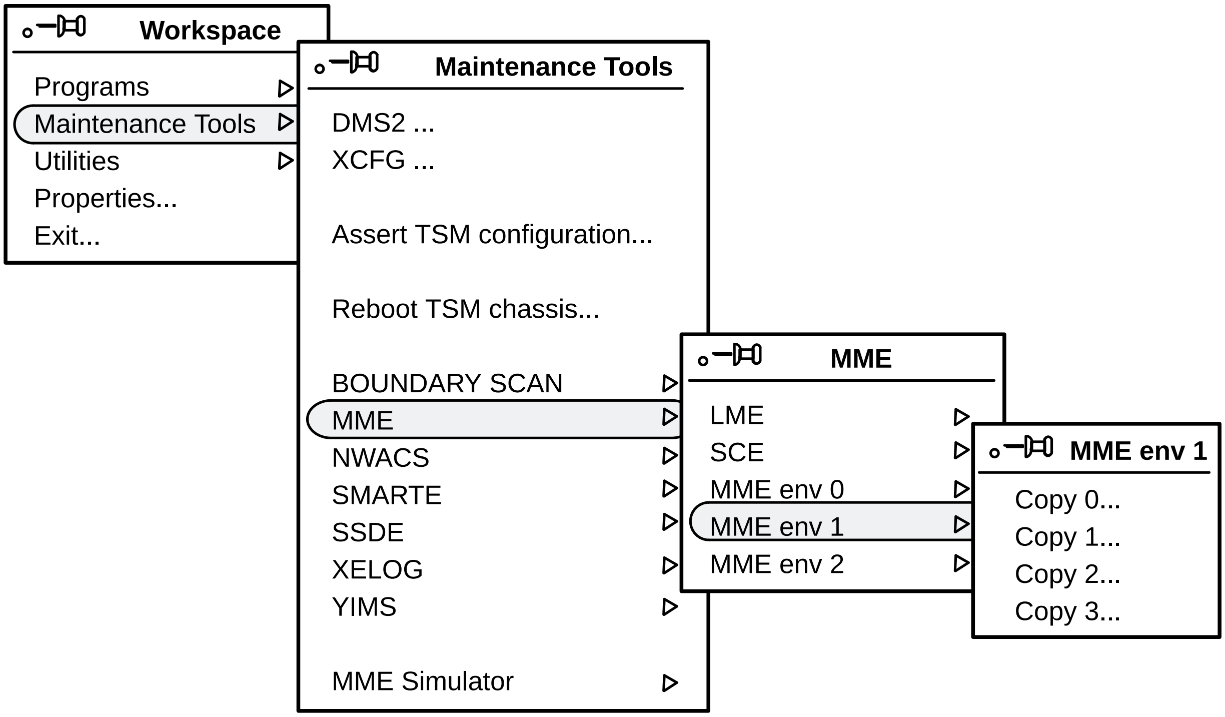


Figure 13. MWS Workspace Menu Options to Start Environment 1 with an FEI Channel

Figure 14 shows the OpenWindows Workspace menu options that you should choose on an MWS to start environment 1 with the simulator or with the simulator and bugger/debugger.

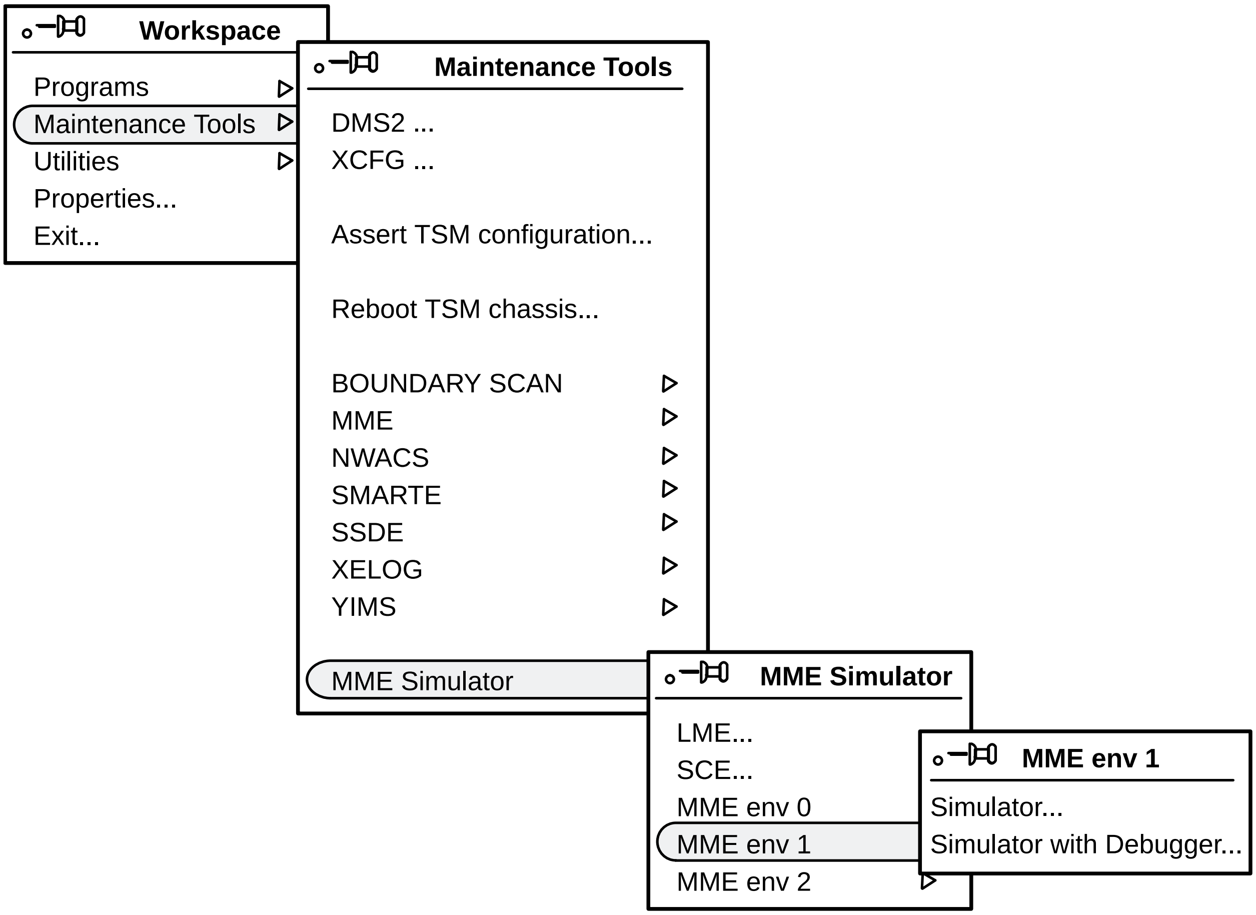


Figure 14. MWS Workspace Menu Options to Start Environment 1 with the Simulator or with the Simulator and Bugger/Debugger

SWS Workspace Menu Options

Figure 15 shows the OpenWindows Workspace menu options that you should choose on an SWS to start environment 1 with an FEI channel. Choose any copy number.

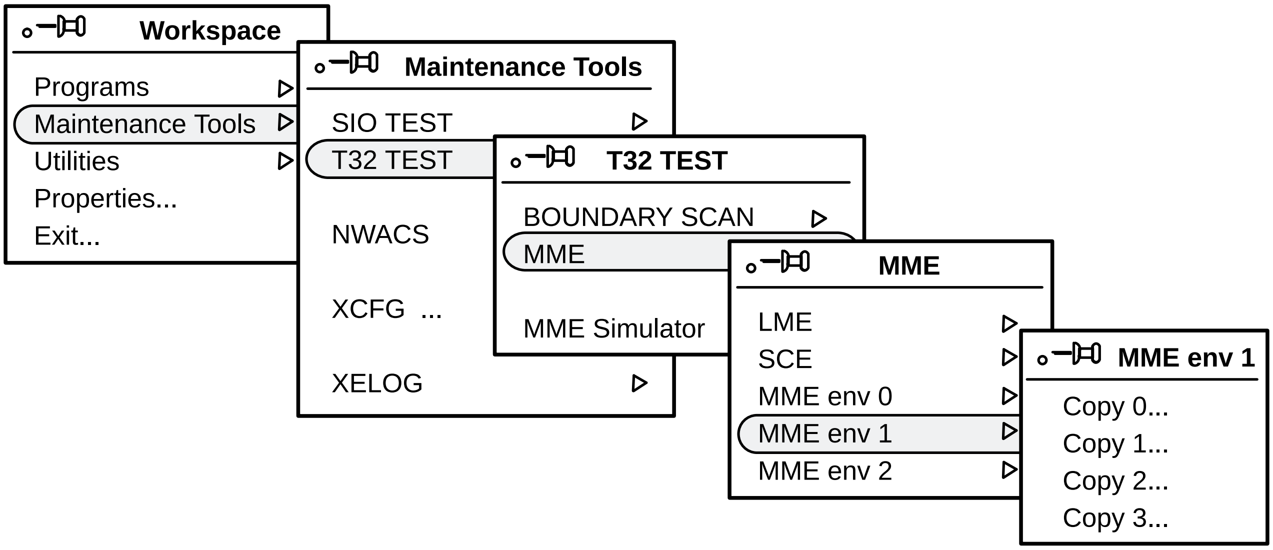


Figure 15. SWS Workspace Menu Options to Start Environment 1 with an FEI Channel

Figure 16 shows the OpenWindows Workspace menu options that you should choose on an SWS to start environment 1 with the simulator or with the simulator and bugger/debugger.

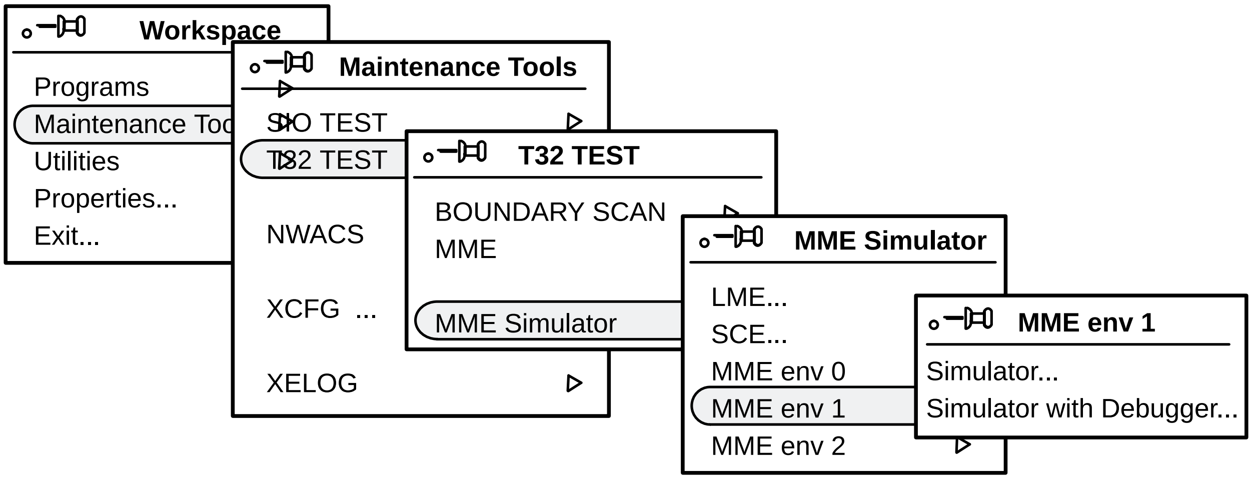


Figure 16. SWS Workspace Menu Options to Start Environment 1 with the Simulator or with the Simulator and Bugger/Debugger

What Happens When You Start Environment 1?

The following actions occur when you start MME:

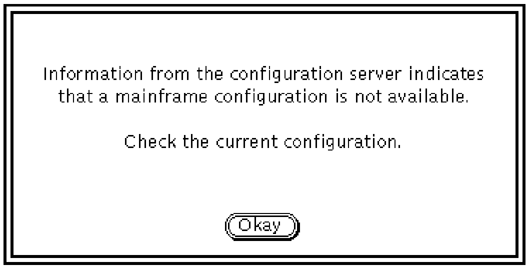
1. The MME server attempts to connect with the System Configuration Environment (SCE) server.

   If MME cannot connect with a running SCE server, MME starts a new SCE server and tries to connect to the new SCE server. (If you specified a configuration file with the `–config` command line option, MME sends this file to SCE through the SCE `–default` command line option. SCE loads the configuration stored in the file.)

2. Once MME establishes a connection with SCE, MME attempts to receive a configuration from SCE:

   - If a configuration is available, SCE provides MME with the components that are available for use by the maintenance system. MME automatically configures itself to use these components.

   - If a configuration is not available, MME displays the message shown in the following snap:



Information from the configuration server indicates that a mainframe configuration is not available.

Check the current configuration.

   If MME displays this message, then you need to create a configuration using SCE before you continue using MME. Refer to the *SCE User Guide*, publication number HDM-069-C, for more information about creating a configuration.

Load a Layout (Optional)

Layouts are not implemented yet.

Allocate Resources (Optional)

MME enables you to change the CPU automatic assignment options and CPU modes, the CPU-to-memory delays, and the section swap interval. Refer to the “Properties –> Resource Allocation” description in the *MME Interface Reference*, publication number HDM-008-A, for more information.

Load a Control Point

To perform testing with MME, you need to load a diagnostic program, utility, or loop into mainframe memory. When you load one of these Cray Assembly Language (CAL) programs into memory, it is called a control point. Because you load only one control point into mainframe memory at a time in environment 1, the control point has access to the entire mainframe or portion of the mainframe that MME is using.

MME performs the following functions to load a control point:

1. MME loads the code located in addresses 0 through the end of the standard location block into an MME data buffer.

2. MME configures the diagnostic or utility code in the MME buffer based on the data stored in the standard locations. For example, MME configures the memory configuration and CPU select standard locations.

3. MME writes the code in the MME data buffer into mainframe memory.

4. MME writes the code from the end of the standard locations to the end of the initialized data into mainframe memory.

5. Optionally, MME clears the dump area.

6. MME overlays any global user changes to the control point sections.

7. MME overlays any section user changes.

Control Point Components

Figure 17 shows the control point components that are loaded into mainframe memory.

NOTE: All memory addresses in Figure 17 are octal numbers.

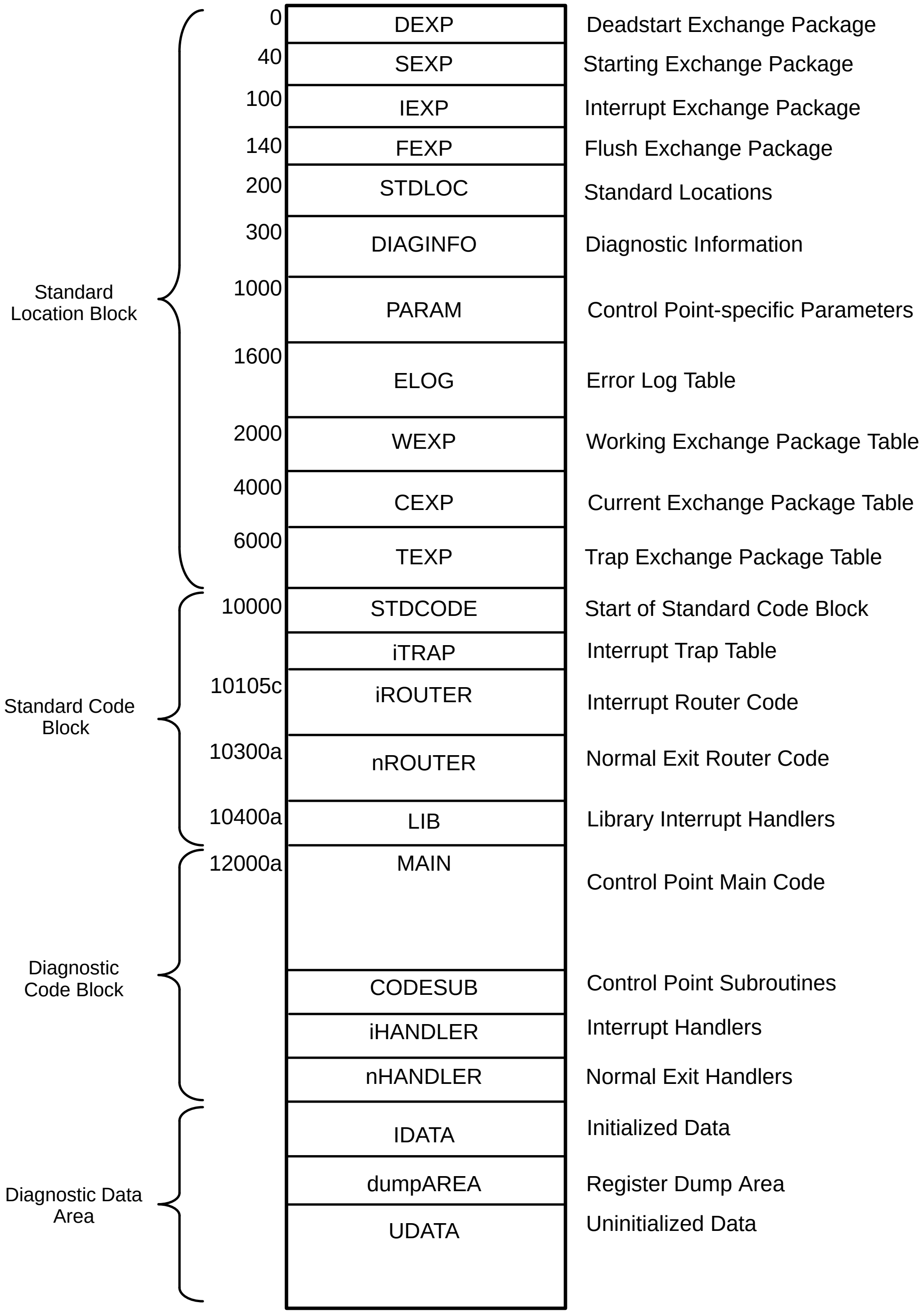


Figure 17. Control Point Components

Control points have four main parts: a standard location block, a standard code block, a diagnostic code block, and a diagnostic data area.

Standard Location Block

The standard location block contains parameters at fixed locations for all control points. This block includes the deadstart exchange package, starting exchange package, interrupt exchange package, flush exchange package, standard locations, diagnostic information, parameters, error log, working exchange package table, current exchange package table, and trap exchange package table.

Deadstart Exchange Package

The deadstart exchange package (DEXP) is located at address 0_8. MME uses the DEXP to deadstart a CPU when the Go button is clicked. MME reads and modifies the starting exchange package and writes this data into the DEXP. MME exchanges the DEXP into the CPU to deadstart the CPU.

Starting Exchange Package

The starting exchange package (SEXP) is located at address 40_8. MME reads the SEXP through the maintenance channel and modifies the data to build the DEXP that is used to deadstart a CPU.

Interrupt Exchange Package

The interrupt exchange package (IEXP) is located at address 100_8. IEXP is not used in environment 1.

Flush Exchange Package

The flush exchange package (FEXP) is stored at address 140_8. A CPU uses the FEXP to perform a dummy exchange to clear any pending interrupts.

Standard Locations

The standard locations are diagnostic parameters that are the same for all diagnostic test and utility programs. Table 7 describes the standard locations, which start at address 200_8 (labeled STDLOC).

Table 7. Standard Locations

Address	Label	Description
0200	LPASS	Last pass to be executed (0 = forever)
0201	SECS	Section select bit mask
0202	CONDS	Conditions select bit mask
0203	MRMASK	Error log mask (mask of significant bits to compare when repeated errors are logged)
0204	STOP	Stop flag bit mask: 00 = Continue (update CPU information and continue processing) 01 = Stop (update CPU information and stop processing) 02 = Not available 10 = Isolate (restart and isolate the error) 20 = Wait on 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 test
0214	CLNN	Number of clusters
0215	CLNU	Bit mask of clusters being tested
0216	CLNS	Bit mask of the clusters to be tested
0217	CLNB	Bit mask of background clusters
0220	DPB	Diagnostic physical bias
0221	DLL	Diagnostic logical base
0224	MFRST	First memory word to test (BSS)
0225	MLIMT	Memory limit address (similar to data limit address)
0226	BANKS	Number of bank bits and number of memory banks
0227	MCFG	Memory configuration (cache enable, number of memory banks, number of memory subsections, and number of memory sections)
0230	SSDBA	SSD base (starting) address
0231	SSDL	SSD limit address
0233	TIFM	Trigger interrupt flag mask

Table 7. Standard Locations (continued)

Address	Label	Description
0234	DIFM	Diagnostic program interrupt-handled flag mask
0235	SIFM	System interrupt flag mask
0236	SIFR	System interrupt flag return
0237	ENV	Current diagnostic environment
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	Starting address of the dump area
0242	DMPJUMP	Dump and idle routine address
0244	LASTREQ	Copy of the last diagnostic-to-controller request
0245	LASTRET	Copy of controller return status
0246	HARDware	Hardware configuration information
0250	MMEREQ0	CPU-to-MME request 0
0251	MMEREQ1	CPU-to-MME request 1
0252	MMEREQ2	CPU-to-MME request 2
0253	MMEREQ3	CPU-to-MME request 3
0254	MMERESP0	MME-to-CPU response 0
0255	MMERESP1	MME-to-CPU response 1
0256	MMERESP2	MME-to-CPU response 2
0257	MMERESP3	MME-to-CPU response 3

Diagnostic Information

Diagnostic information is standardized status information for an executing diagnostic test or utility program. The diagnostic information is located at addresses 300_8 through 377_8, as shown in Table 8. These memory locations are the same for all diagnostic test and utility programs. The current executing control point updates these locations.

Table 8. Diagnostic Information

Address	Label	Description
0300	DIF	Difference between expected and actual diagnostic information
0301	ACT	Actual information
0302	EXP	Expected information
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 being tested
0311	CUT	Condition being tested
0312	SCUT	Subcondition being tested
0313	TSUT	Test sequence being tested
0314	CLOOP	Remaining condition loop count
0315	SLOOP	Remaining subcondition loop count
0316	TLOOP	Remaining test sequence loop count
0332	LOSPT	LOSP table length and table address
0333	VHISPT	VHISP table length and table address
0334	CRMASK	Channel 077 to 000 reserve mask
0335	CRMASKu	Channel 177 to 100 reserve mask
0336	CIMASK	Channel 077 to 000 interrupt mask
0337	CIMASKu	Channel 177 to 100 interrupt mask
0340	CPUREQ0	MME-to-CPU request 0
0341	CPUREQ1	MME-to-CPU request 1
0342	CPUREQ2	MME-to-CPU request 2
0342	CPUREQ3	MME-to-CPU request 3
0344	CPURESP0	CPU-to-MME response 0
0345	CPURESP1	CPU-to-MME response 1
0346	CPURESP2	CPU-to-MME response 2
0347	CPURESP3	CPU-to-MME response 3

Parameters

The control point-specific parameters start at address 1000_8 (labeled PARAM).

Error Log Table

The error log table organizes all memory and register parity errors from the running control point into one area of memory. The error log table begins at address 1600_8.

Refer to the “View –> Error Log” description in the *MME Interface Reference*, publication number HDM-008-A, for information on how to view the error log table.

Working Exchange Package Table

The working exchange package (WEXP) table starts at address 2000_8. This table contains one WEXP for each CPU (32 total). The WEXPs are located on 40_8 word boundaries (for example, the WEXP for CPU 0 is at address 2000_8, the WEXP for CPU 1 is at address 2040_8, and the WEXP for CPU 2 is at address 2100_8).

When a control point receives an interrupt, it exchanges to the WEXP for the CPU to which the control point is assigned.

Current Exchange Package Table

The current exchange package (CEXP) table starts at address 4000_8. This table contains one CEXP for each CPU (32 total). The CEXPs are located on 40_8 word boundaries (for example, the CEXP for CPU 0 is at address 4000_8, the CEXP for CPU 1 is at address 4040_8, and the CEXP for CPU 2 is at address 4100_8).

The CEXP is empty when a control point is loaded. When a control point receives an interrupt and exchanges to WEXP, WEXP is copied to CEXP for the CPU to which the control point is assigned. This process stores the address in the control point code where the exchange occurred in the P register and saves the interrupt flags that caused the exchange.

Trap Exchange Package Table

The trap exchange package (TEXP) table starts at address 6000_8. This table contains one TEXP for each CPU (32 total). The TEXPs are located on 40_8 word boundaries (for example, the TEXP for CPU 0 is at address 6000_8, the TEXP for CPU 1 is at address 6040_8, and the TEXP for CPU 2 is at address 6100_8).

The TEXP for the CPU exchanges into the CPU when the interrupt routers or handlers receive an intolerable interrupt. This causes the iTRAP code at address 11000_8 to execute.

Standard Code Block

The standard code block contains library code that is common to all diagnostics. The standard code block includes the interrupt trap table, the interrupt router (iROUTER) code, the normal exit router (nROUTER) code, and the library interrupt handlers (LIB). The standard code exchanges into the CPU when an interrupt occurs.

Interrupt Trap Table

The interrupt trap table contains a table of hang addresses. Currently, there is only one hang address in the table.

Interrupt Router Code

The interrupt router (iROUTER) code is the first level of interrupt processing. This code determines what interrupts exist and passes the interrupts to the appropriate handler routines.

Normal Exit Router Code

The normal exit router (nROUTER) code receives normal exit interrupts from the interrupt router code and passes the interrupts to the appropriate normal exit handler code.

Library Interrupt Handlers

The library interrupt handlers are standardized handlers that are used for interrupt processing. These handlers are the same for all diagnostic tests and utilities.

Diagnostic Code Block

The diagnostic code block contains all code for the current diagnostic test or utility program. This block includes the control point main code, the control point subroutines, the interrupt handlers, and the normal exit handlers. The size of this block varies for the different diagnostic test and utility programs.

Control Point Main Code

The control point main code contains the actual diagnostic test or utility program code that performs the testing or utility functions.

Control Point Subroutines

Any subroutines that the control point main code uses are stored starting at the memory location labeled CODESUB.

Interrupt Handlers

The interrupt handlers contain the code used that processes the interrupts that occur while a control point is executing.

Normal Exit Handlers

The normal exit handlers contain the code that processes the normal exit calls that occur while a control point is executing.

Diagnostic Data Area

The diagnostic data area is memory that is reserved for data that is used or created by the current diagnostic test or utility program. This area includes the error information block, initialized data, register dump area, and uninitialized data.

Initialized Data

The initialized data is preset data that is used by the diagnostic test or utility program. This data includes constants and predetermined (sometimes called *canned*) answers.

Register Dump Area

The register dump area is a block of memory that is reserved for any register data that is dumped by the diagnostic test or utility program or by the Halt –> Register Dump option.

Uninitialized Data

The uninitialized data is a data area in which the diagnostic test or utility program stores the data that it uses. This data is not initialized or stored on the MWS or SWS hard disk with the diagnostic test or utility program. The diagnostic test or utility program must initialize this data.

Assign a CPU to the Current Control Point

You must assign a CPU to the current control point to perform any troubleshooting for the CPU. To assign a CPU to the current control point, click on the CPU in the CPU selection, control point, and status area in the MME base window. Refer to “CPU Selection, Control Point, and Status Area” in the *MME Interface Reference*, publication number HDM-008-A, for more information.

Click on Go

Click on Go to start control point execution; the control point executes through the sequence of events that Figure 18 illustrates and the text following the figure describes. (The circled numbers in the figure correspond to the numbered steps in the text that follows.)

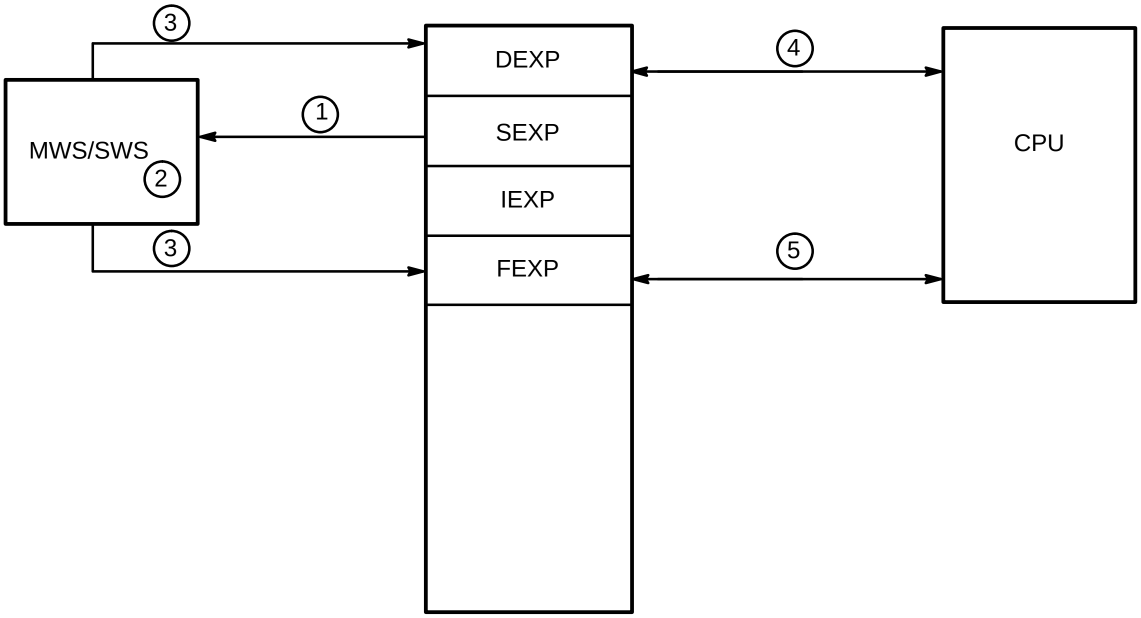


Figure 18. Control Point Execution Sequence (Go Clicked)

1. MME reads SEXP from mainframe memory through the maintenance channel into MWS or SWS memory.

2. MME modifies the copied SEXP to create exchange packages that will be written into DEXP and FEXP. MME modifies the exchange packages as follows:

   For the DEXP copy, MME performs the following steps.

   a. MME sets A7 to the physical CPU number.

   b. Depending on the values set in the MME Resource Allocation window, MME modifies the interrupt on correctable memory error (ICM) mode bit, interrupt on uncorrectable memory error (IUM) mode bit, interrupt on register parity error (IRP) mode bit, and the cache LAT bits.

      For more information about the MME Resource Allocation window, refer to “Properties –> Resource Allocation” in the *MME Interface Reference,* publication number HDM-008-A.

   NOTE: The XA and EA registers are left at the defaults, which point to FEXP.

   For the FEXP copy, MME performs the following steps:

   a. MME sets A7 to the physical CPU number.

   b. Depending on the values set in the MME Resource Allocation window, MME modifies the ICM, IUM, and IRP mode bits.

   c. MME sets the exchange address (XA) parameter to its original value plus A7 multiplied by 40_8 [XA = XA + (A7 * 40_8)]; this makes XA point to the WEXP for the CPU.

      MME also sets exit address 0 (EA0) through EA4 to the original value plus A7 multiplied by 40_8:

      EA0 = EA0 + (A7 * 40_8)
      EA1 = EA1 + (A7 * 40_8)
      EA2 = EA2 + (A7 * 40_8)
      EA3 = EA3 + (A7 * 40_8)
      EA4 = EA4 + (A7 * 40_8)

   d. Depending on the values set in the MME Resource Allocation window, MME modifies the cache LAT bits.

3. MME writes the exchange packages to DEXP and FEXP.
4. The DEXP exchanges into the CPU. The DEXP P register points to the cpuFLUSH routine in the standard code.
5. The CPU executes the cpuFLUSH code, which clears out any interrupts and then exchanges out to EA0, which is FEXP. The FEXP P register points to MAIN, so the CPU starts executing the control point code.

NOTE: If the control point is a multi-CPU control point, this sequence of events repeats until all CPUs that are assigned to the control point are deadstarted.

Monitor the Progress of Control Point Execution

As a control point executes, you should monitor the information MME displays to determine the progress of the control point. It is important to understand what happens during control point execution so that you can determine whether everything is operating properly. Table 9 shows the status information that you need to monitor while a control point executes.

Table 9. Status Information from an Executing Control Point

Status	Description
“ERROR COUNT” flashing next to a CPU	The control point detected an error. Refer to “Diagnostic-detected Errors” for more information.
“Holding” appears next to a CPU	Control point execution is paused. Check the runtime information display for a prompt for the control point.
Indicator (MEM, RPE, SHR, LAT, or UKN) appears in the menu bar	MME detected a memory, register parity, shared, LAT, or unknown error. MME logs these errors in the error log. Choose **View –> Error Log** to view the error log.
Interrupt flag	An interrupt occurred. Refer to “Interrupts” for more information.
P register is incrementing	Everything is operating correctly.
P register is not incrementing	The P register is hung. Check the WEXP and TEXP for the CPU to see if a flag is set. Check the listing to see if hang code is causing the hung P register.
“Waiting” appears next to a CPU	The CPU is waiting to execute a multiple-CPU control point.

Diagnostic-detected Errors

Control point diagnostic test code that is running in a CPU detects and reports an error through the following sequence of events:

1. The control point test code detects a data comparison error for the hardware values being tested.

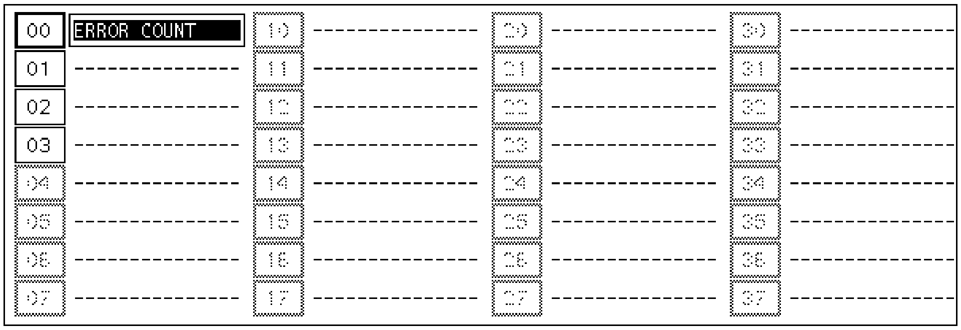
2. The control point test code logs the error in the standard locations.

3. The control point test code performs a dump and wait normal exit request. The handler for this normal exit increments the error count and activates the hold flag in the STOP standard location.

4. MME checks the minimum and maximum error counts that are assigned to the control point.

NOTE: The CPU does not stop for an error.

When a control point increments the error count, ERROR COUNT flashes next to the CPU in the CPU status area. Figure 19 shows the ERROR COUNT indicator highlighted.



00	ERROR COUNT	10	---------------	20	---------------	30	---------------
01	---------------	11	---------------	21	---------------	31	---------------
02	---------------	12	---------------	22	---------------	32	---------------
03	---------------	13	---------------	23	---------------	33	---------------
04	---------------	14	---------------	24	---------------	34	---------------
05	---------------	15	---------------	25	---------------	35	---------------
06	---------------	16	---------------	26	---------------	36	---------------
07	---------------	17	---------------	27	---------------	37	---------------

Figure 19. Error Indicator

When you see an error indicator, refer to the error return address (ERA) in the diagnostic information block. The ERA, which is located at address 0305_8, indicates an area in the listing near the location of the failing code. View the ERA in address mode to determine the address. Look at the code in the listing that is adjacent to the ERA to determine the code that actually failed.

NOTE: The ERA is shown on the DIAGINFO runtime information display, which is currently available for most diagnostic tests and utilities.

For information on how to view a listing, refer to “View –> Listing –> Current” and “View –> Listing –> Other” in the *MME Interface Reference*, publication number HDM-008-A.

Interrupts

Interrupts are either tolerable or intolerable. Tolerable interrupts are interrupts that occur while the CPU is processing the main diagnostic code. Tolerable interrupts can be ignored or routed, depending on the code of the diagnostic program. Intolerable interrupts are interrupts that occur while the CPU is processing code from the standard code block. Intolerable interrupts are trapped by hanging the CPU.

Figure 20 shows the two interrupt classes and the actions performed when interrupts occur.

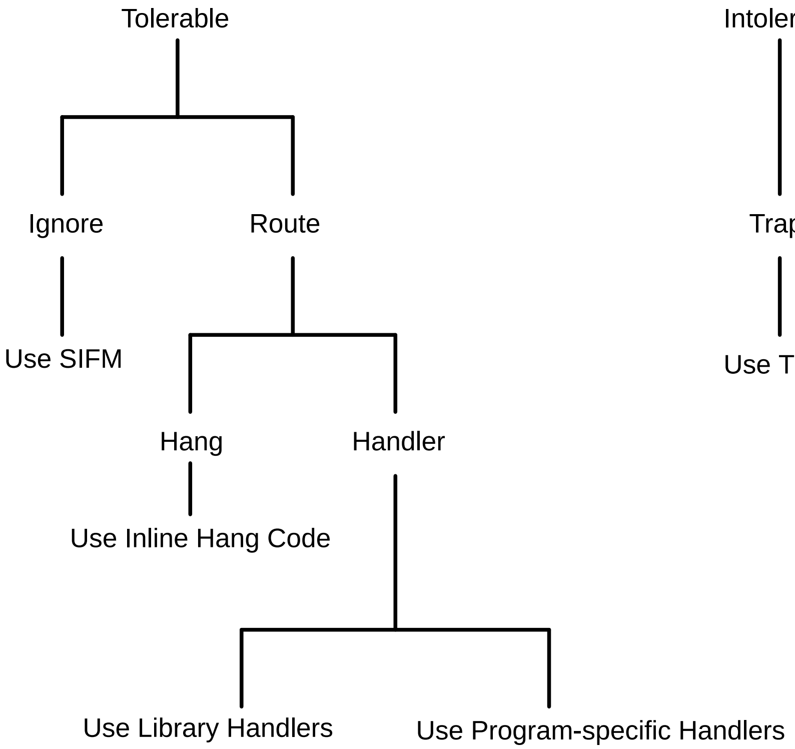


Figure 20. Interrupt Classes (Environment 1)

When interrupts occur, the control point code exchanges to the interrupt router (iROUTER) code in the standard code block of the control point. The iROUTER code routes all interrupts in the current interrupt list through the sequence that Figure 21 shows and the text following the figure describes. (The circled numbers in the figure correspond to the numbered steps in the text that follows.)

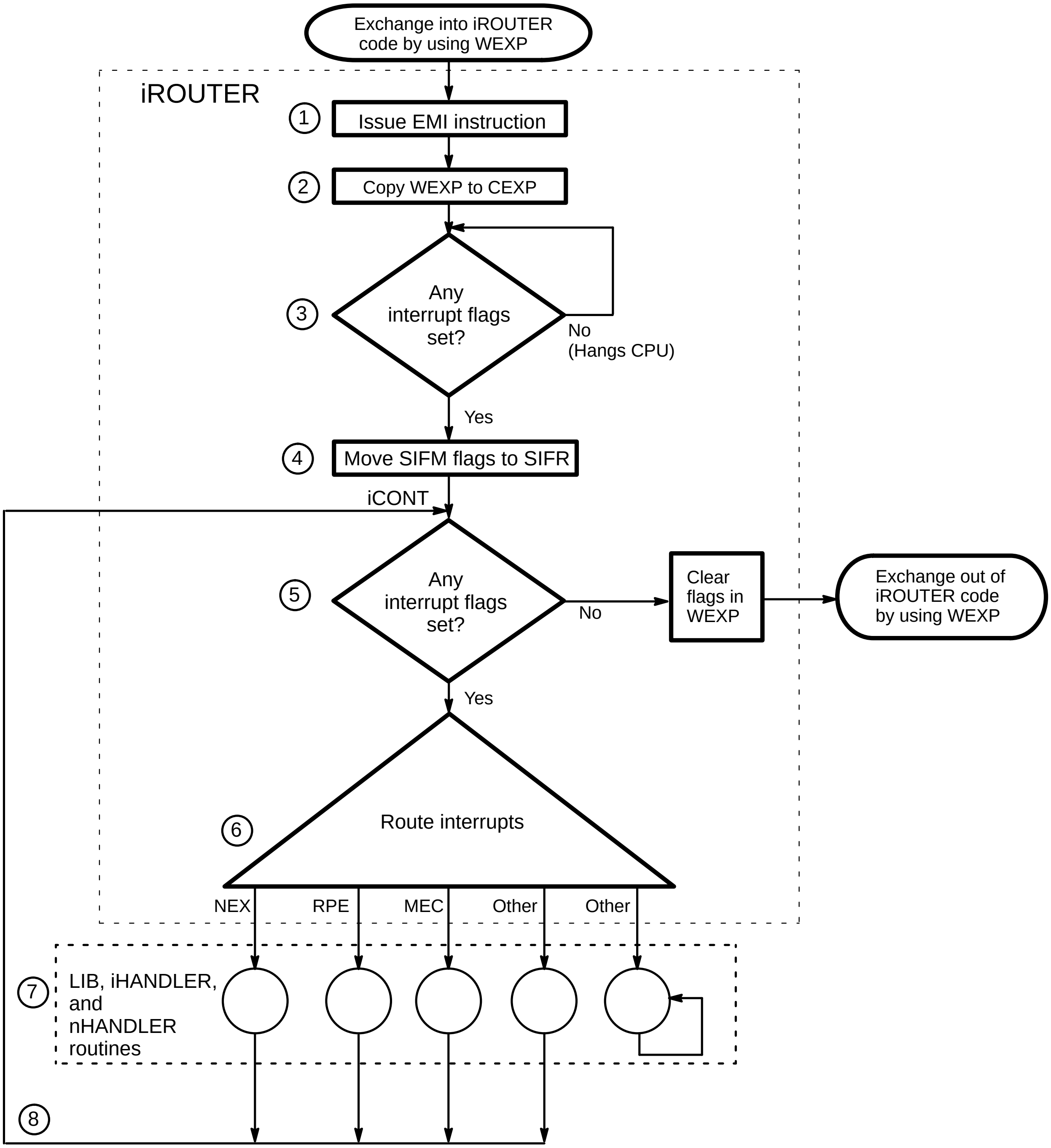


Figure 21. Interrupt Processing in Environment 1

1. The iROUTER code issues an EMI instruction to enable monitor mode interrupts.

2. The iROUTER code copies the WEXP to the CEXP.

3. The iROUTER code verifies that there are interrupt flags in the current interrupt list to process.

   - If there are interrupt flags to process, the iROUTER code continues with Step 4.

   - If there are no interrupt flags to process, the iROUTER code hangs in the CPU.

4. The iROUTER code checks the system interrupt flag mask (SIFM) parameter to determine whether the interrupt should be ignored.

   If the flag for an interrupt is set in the SIFM parameter, the iROUTER code removes the flag from the current list of interrupt flags and places it in the system interrupt return mask (SIFR) parameter. (Refer to the “Ignore” discussion on page 101 for more information.)

5. The iROUTER code checks to determine whether any interrupt flags are still set.

   - If interrupt flags are still set, the iROUTER code continues with Step 6.

   - If no interrupts flags are set, the iROUTER code exchanges out of the CPU, and the CPU resumes main diagnostic code execution.

6. The iROUTER code routes one interrupt in the current interrupt list and clears the interrupt flag in the list.

   - If a handler exists for the interrupt, the iROUTER code routes the interrupt to the handler code. (Refer to the “Route to a Handler” discussion on page 101 for more information.)

   - If no handler exists for the interrupt, the iROUTER code routes the interrupt to hang code, which hangs in the CPU. (Refer to the “Route to a Hang” discussion on page 108 for more information.)

7. The handler routine processes the interrupt.

8. After the current interrupt is processed by a handler, control returns to Step 5 at the label iCONT.

Tolerable Interrupts

Tolerable interrupts occur while the diagnostic test or utility code is executing in a CPU. The iROUTER ignores tolerable interrupts or routes them to a hang or handler.

Ignore

The iROUTER code can remove interrupts from the current list of pending interrupts to be processed to prevent the interrupts from being processed. If a diagnostic test or utility program sets the corresponding bit for an interrupt in the system interrupt flag mask (SIFM) parameter, the iROUTER code removes the interrupt from the current list of interrupts to be processed. The iROUTER code places the flag in the system interrupt return mask (SIFR) parameter. The interrupt is ignored.

Ignored interrupts are typically used so the diagnostic test code can force an interrupt condition. When control returns to the test code, the test code checks the SIFR parameter to verify that the interrupt occurred.

Route to a Handler

The diagnostic code handles interrupts through special code sections called handlers. Handlers contain the code that is necessary to process interrupts. Three types of handlers may be available in a control point: library interrupt handlers, interrupt handlers, and normal exit handlers. If a handler is available for an interrupt, the iROUTER code routes the interrupt to the handler.

NOTE: Some handlers are just inline hangs.

Library Interrupt Handlers

Some interrupts are routed to library interrupt handlers, which are general handlers included in all control points. These handlers contain code that processes common interrupts.

Interrupt Handlers

Some interrupts are routed to interrupt handlers that are located in the diagnostic code area of a control point. These handlers are specific to a control point and contain code that processes special-case interrupts.

Normal Exit Handler

Several control points use normal exit (NEX) interrupts to perform tasks in monitor mode instead of the usual user mode that the control point code runs in. A special router is used to route NEX interrupts. The NEX router (nROUTER) routes a normal interrupt based on the value stored in the S1 register. Figure 22 shows the sequence of events that occur to route and handle a normal exit interrupt. (The circled numbers in the figure correspond to the numbered steps in the text that follows.)

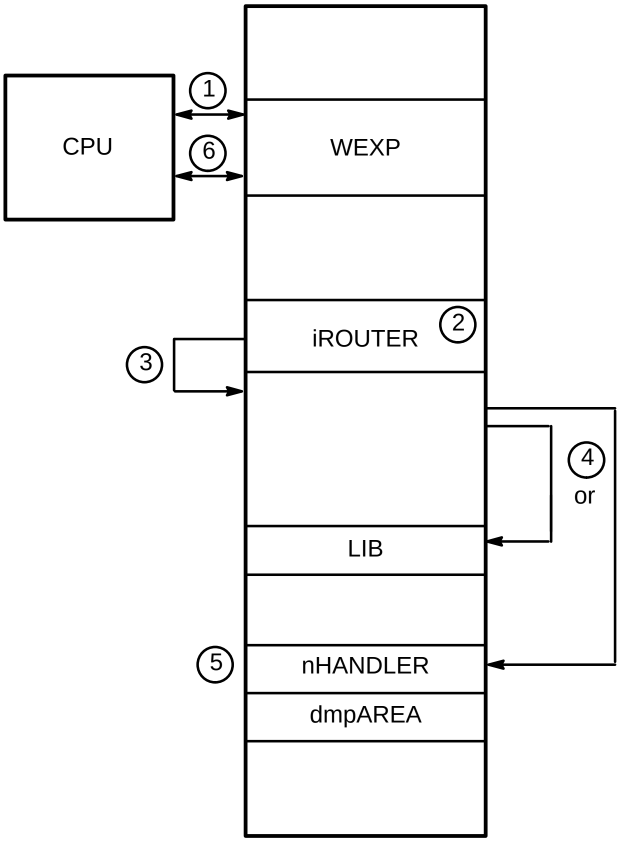


Figure 22. Normal Exit Interrupt

1. An interrupt occurs that causes an exchange between the CPU and WEXP (the WEXP P register points to the iROUTER code). This causes the CPU to execute the iROUTER code.

2. The iROUTER code checks the system interrupt flag mask (SIFM).

3. The iROUTER code routes to the appropriate handler for processing. For a normal exit (NEX) interrupt, control is passed to the normal exit router (nROUTER).

4. The nROUTER code examines the function code in the S1 register and sends control to the corresponding handler, which is a library handler or normal exit handler (nHANDLER). The nROUTER code performs the following actions, as shown in Figure 23.

The circled letters in Figure 23 correspond to the lettered steps that follow this figure.

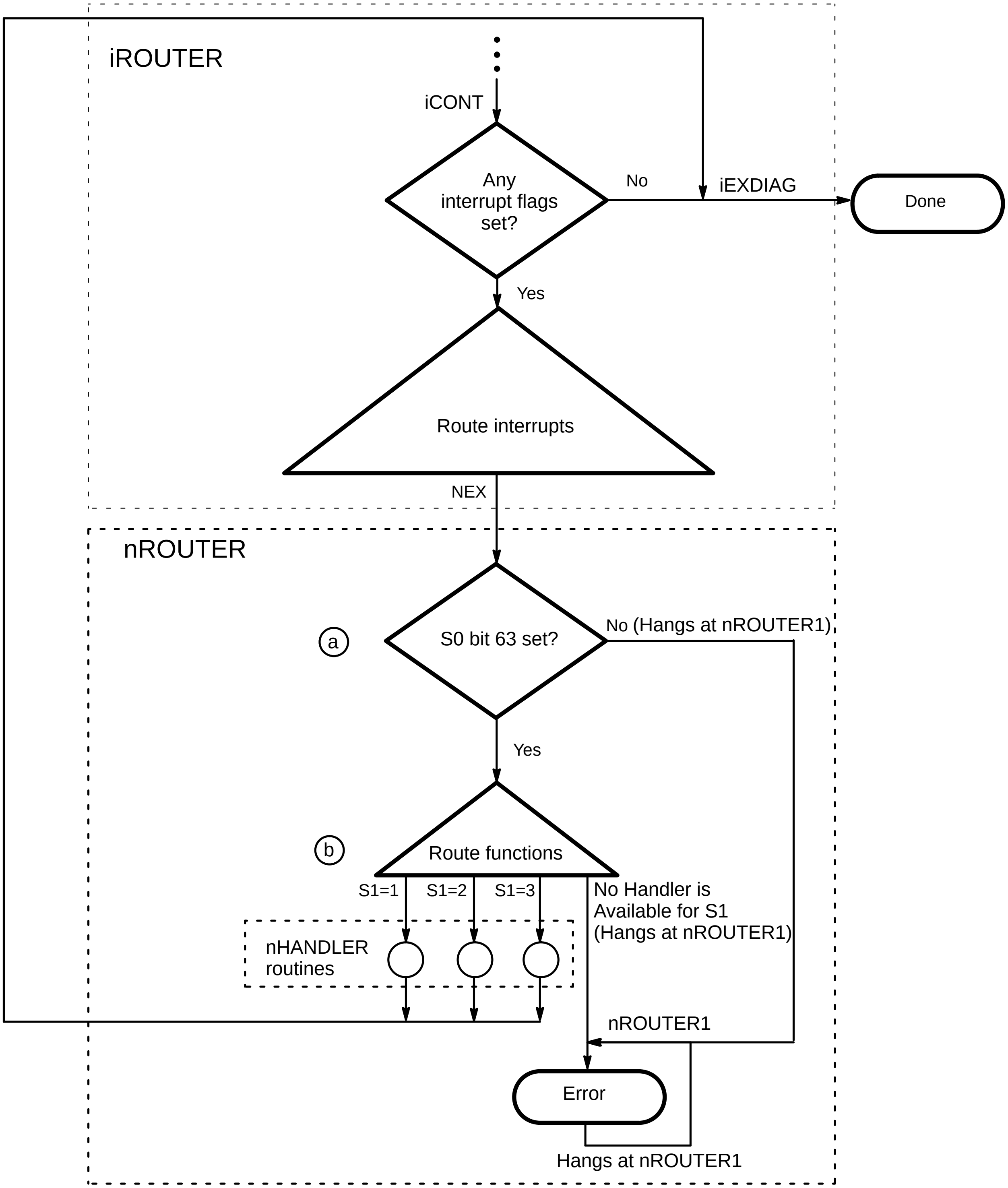


Figure 23. Normal Exit Interrupt Processing

a. The nROUTER code verifies that bit 63 (the sign bit) of S0 is set to 1. This diagnostic code must set this bit to indicate that a normal exit interrupt is actually occurring. If this bit is not set, the CPU hangs at the address labeled nROUTER1.

b. The nROUTER code routes the interrupt to the corresponding handler in the LIB or nHANDLER code. If no handler corresponds to the value in S1, the CPU hangs at the address labeled nROUTER1.

Table 10 shows the normal exit request bit fields. Table 11 shows the normal exit routines that correspond to the bits that are set in S1.

Table 10. Environment 1 Normal Exit Request Bit Fields

Register	Contents			
S0	Valid Request Flag, Bit 63 = 1			
S1	Function Mask for the Controller Only Handlers, Bits 63 – 48	Function Mask for the Controller and Handlers, Bits 47 – 32	Function Mask for the Router Only Handlers, Bits 31 – 16	Function Mask for the Program-defined Handlers, Bits 15 – 0
S2	Parameter 1			
S3	Parameter 2			
S4	Parameter 3			
A0	Parameter 4			
A1	Parameter 5			
A2	Parameter 6			
A3	Parameter 7			

Table 11. Environment 1 Normal Exit Routines

Octal Bit	Name	Description
66	hNOP	Perform no operation
57	hIDLE	Dump registers and idle the CPU
56	hIDLE	Dump registers and wait for the hold bit to clear
46	hHOLD	Hold until the user clicks the Resume button in the MME base window

Table 11. Environment 1 Normal Exit Routines (continued)

Octal Bit	Name	Description
44	hsrLOCK	Set shared register cluster Parameters: S2: Cluster or cluster mask S3: 0 = Release the specified cluster 1 = Reserve the specified cluster 1000 = Release the specified clusters in cluster mask 1001 = Reserve the specified clusters in cluster mask When this routine sets a single cluster number (S3 = 1), the CLN register is set in the exchange package. When this routine sets a group of clusters (S3 = 1001), S2 returns a mask of reserved clusters. S0 returns –22 if another control point controls the requested cluster.
43	hSETPCI	Set up PCI if it is selected in PCITIME
41	hSETM	Set or clear mode flags in WEXP Parameters: S2: Bit mask of flags relative to the register that you are using S3: 0 = Clear flags 1 = Set flags
40	hSETIM	Set or clear interrupt mode bits in WEXP Parameters: S2: Bit mask of mode bits relative to the register that you are using S3: 0 = Clear bits 1 = Set bits
26	hMAINTS	Set maintenance mode S2: 1 = CPU maintenance mode 2 = I/O maintenance mode 3 = SHR maintenance mode S3: Loop controller function code S4: Destination (CPU, channel, or module number)

Table 11. Environment 1 Normal Exit Routines (continued)

Octal Bit	Name	Description
25	hLOG	Enable or disable the error logger Parameter: S2: 0 = Disable 1 = Enable
24	hLOGQ	Start or stop the error logger queue Parameters: S2: Bit 0: 0 = Stop logging 1 = Start logging Bit 1: 0 = Allow I/O activity 1 = Temporarily disable I/O activity Bit 2: 0 = Do not delay before performing reads 1 = Delay before performing reads S3: Time-out value in microseconds (1 – 15000) S4: Number of errors to record (1 – 10000) S5: Queue destination (mainframe address)
23	hQUIET	Temporarily disable MME maintenance channel I/O activity
22	hQWAIT	Resume MME maintenance channel I/O activity

5. Control passes to the code at the memory address labeled iEXDIAG.

6. The iROUTER code exits through an exchange between WEXP and the CPU. The CPU continues control point execution unless the normal exit (NEX) performed an hIDLE routine. If an hIDLE routine was performed, the CPU hangs in STDCODE after the registers are dumped to dmpAREA.

Route to a Hang

If no handler routine exists for an interrupt, the iROUTER code routes the interrupt to inline hang code in the IROUTER code. The hang code causes the CPU to hang in the iROUTER code. Figure 24 shows the sequence of events for a hang and the text following describes it. (The circled numbers in Figure 24 correspond to the numbered steps that follow.)

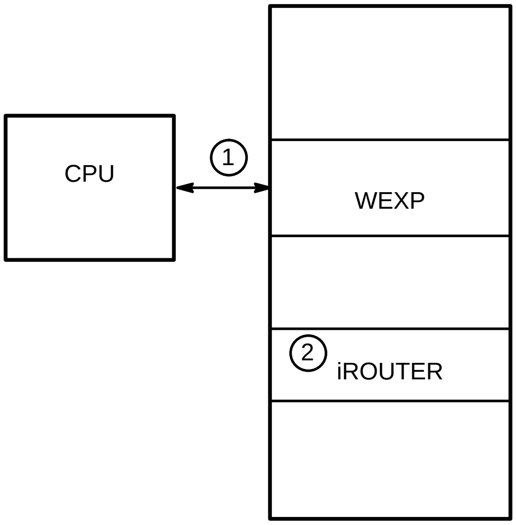


Figure 24. Interrupt Processing (Hang)

1. An interrupt occurs that causes an exchange between the CPU and WEXP. The CPU begins to execute the iROUTER code.

2. When the iROUTER code attempts to route the interrupt to the handler, no handler code is available. The iROUTER code hangs the CPU at the code that tests for the interrupt flag (P register = trap).

NOTE: If memory and register parity errors are detected, these errors are usually logged. Then, the standard code exchanges out of the CPU, and the control point code exchanges back into the CPU.

As tolerable interrupts are processed, various interrupts appear in the CPU status area, which is normal; you may want to monitor the pass count and SIFR.

If the CPU stops executing instructions in the standard code during NEX interrupt processing, look at the WEXP for the CPU. Check the P register, S0 register, and S1 register to determine what was running in the CPU. Click on halt and look at the WEXP to determine where the CPU stopped executing code.

If the P register does not increment in the CPU status area, this indicates that the CPU is hung. Look at the code in the listing where the P register is hung.

Intolerable Interrupts

Intolerable interrupts occur while the iROUTER code is processing an interrupt. Because the standard code is already running, there is no way to process these new interrupts. MME traps intolerable interrupts by exchanging a trap exchange package (TEXP) into the CPU, which Figure 25 shows and the text following the figure describes.

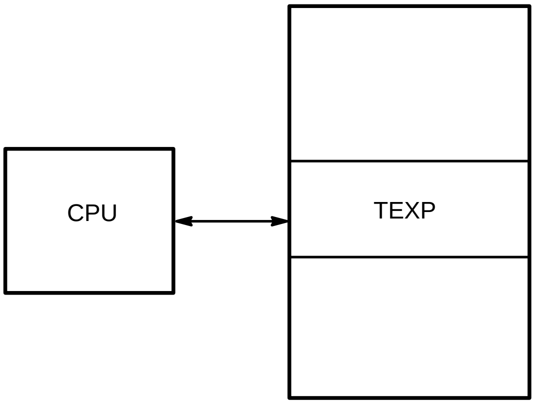


Figure 25. Intolerable Interrupt Processing

The standard code exchanges out of the CPU to the TEXP, and the TEXP exchanges into the CPU. This causes the CPU to hang in a hang instruction at iTRAP. The flag that caused the interrupt is trapped in the TEXP.

For intolerable interrupts, you should notice that the TEXP P register is not pointing to iTRAP (address 10000_8) or that interrupt flags are set in TEXP. This indicates that an intolerable interrupt occurred in the standard code while the interrupts from the diagnostic code were being processed.

Look at the TEXP for the CPU to see the intolerable interrupt(s) that occurred. The WEXP and CEXP P registers show where the interrupt occurred in the diagnostic code.

Click on Halt

Choose **Halt –> No Dump**, **Halt –> Exchange Dump**, or **Halt –> Register Dump** from Halt ▷ to stop control point execution.

Halt –> No Dump

The Halt –> No Dump option halts control point execution by setting Master Clear on the CPU(s). This option does not dump register or exchange information.

Halt –> Exchange Dump

The Halt –> Exchange Dump option halts the executing control point and dumps exchange information. After this dump, the exchange package for whatever was executing when you clicked Halt ▷ is stored in the DEXP and in the WEXP for the CPU. MME performs an exchange using a maintenance channel function.

For multiple-CPU control points, MME performs an exchange for each CPU. DEXP contains the exchange package for the last CPU halted. The WEXP table contains the exchange packages for the CPUs.

NOTE: The CPUs never issue instructions for this command.

Halt –> Register Dump

The Halt –> Register Dump option halts the executing control point and dumps registers into mainframe memory at the address that is assigned to label dmpAREA. You can specify which registers you want to dump by changing the DMPMASK parameter at address 240_8 in the standard locations. MME performs the following sequence of events for a register dump.

1. MME builds a DEXP, where:

   Modes = 016
   Logical base = 0
   Physical base = base address of the control point (usually 0)
   Logical limit = size of the control point

2. MME writes the DEXP to mainframe memory.

3. MME performs a maintenance channel function that causes an exchange, which starts the CPU.

4. MME waits for a flag to set in the dump area. This signals that the register dump is complete.

5. MME performs a maintenance channel function that causes an exchange, which stops the CPU.

For multiple-CPU control points, MME performs this sequence for each CPU. DEXP contains the exchange package for the last CPU halted. The WEXP table contains the exchange packages for the CPUs.

NOTE: If you select a register dump and no dump area (dmpAREA) is available, MME performs an exchange dump.

ENVIRONMENT 2

Environment 2 is one component of the Mainframe Maintenance Environment (MME) software package that field engineers use to troubleshoot CRAY T90 series mainframes. Environment 2 includes many of the features that are available in environment 1 but also enables you to load several diagnostic programs, utilities, or loops into mainframe memory at a time. A special program called the diagnostic controller (DC) resides in lower mainframe memory and controls the mainframe resources that the control points use.

The run system property in environment 2 enables you to perform confidence testing of the mainframe by creating an environment for hardware system evaluation similar to an operating system. The operating system environment is simulated by swapping jobs (control points) between active CPUs.

The following procedure gives a general overview of the process for using MME environment 2. This section provides related information for each step of the process.

1. Start MME in environment 2.
2. Load a layout (optional).
3. Allocate resources (optional).
4. Enable the run system (optional).
5. Load one or more control points.
6. Assign CPU(s) to the control point(s).
7. Click on (Go).
8. Monitor the progress of control point execution.
9. Click on (Halt ▷).

Start MME in Environment 2

You can start MME in environment 2 from a UNIX prompt or from the OpenWindows Workspace menu.

NOTE: For information about starting MME environment 2 from a Service Center through a hub, refer to the *Remote Support* document, publication number HMM-106-A.

CAUTION

MME performs maintenance channel functions that will hang UNICOS if UNICOS is running in the mainframe when you start MME.

To prevent this from accidentally occurring, ensure that the Owner setting in the SCE base window is set to OS for the logical partition in which UNICOS is running when UNICOS is running in the mainframe. MME cannot access a logical partition if the OS owns it.

From a UNIX Prompt

To start MME environment 2 from a UNIX prompt, enter one of the following commands:

- **`mme –2`** to use a front-end interface (FEI) channel
- **`mme –2 –sim`** to use the simulator
- **`mme –2 –debug`** to use the simulator and bugger/debugger

NOTE: You can also enter any of the command line options that Table 12 lists.

Table 12. Environment 2 Command Line Options

Option	Description
`–client`	Start the MME client only
`–config` *file*	Configure MME with the configuration data stored in the file specified by *file*
`–copy` *num*	Connect to maintenance software assigned the copy number specified by *num* **NOTE:** Copy numbers are necessary only when you run multiple copies of MME on the same MWS or SWS (for example, when you run several MME copies with the simulator or when you use MME to support multiple CRAY T90 series mainframes that are connected to the same MWS or SWS).
`–io` *num*	Use the CPU specified by *num* to perform input and output operations

Table 12. Environment 2 Command Line Options (continued)

Option	Description
`–kill`	Kill any running MME, SCE, or LME applications before you start a new copy of MME
`–remote` *host*	Start the MME client only and connect the client to the MME server that is running on the remote host specified by *host*
`–server`	Start the MME server only

From the OpenWindows Workspace Menu

You can start environment 2 from the OpenWindows Workspace menu on either an MWS or an SWS.

MWS Workspace Menu Options

Figure 26 shows the OpenWindows Workspace menu options that you should choose on an MWS to start environment 2 with an FEI channel. Choose any copy number.

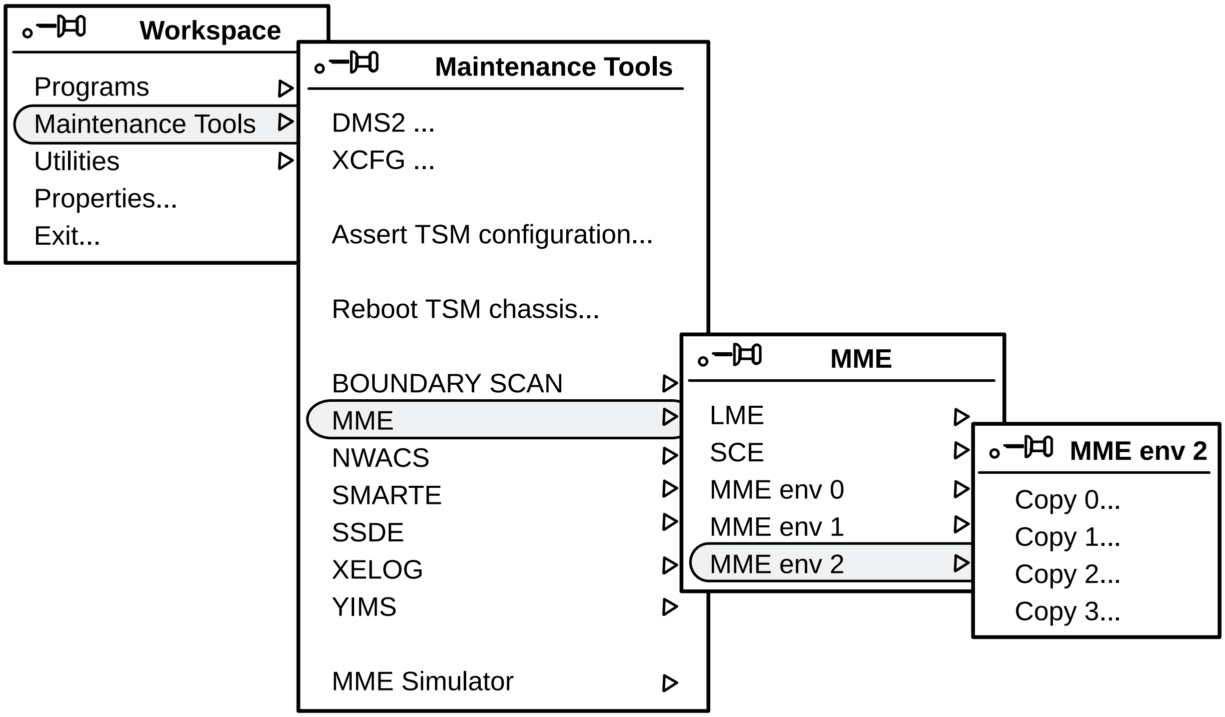


Figure 26. MWS Workspace Menu Options to Start Environment 2 with an FEI Channel

Figure 27 shows the OpenWindows Workspace menu options that you should choose on an MWS to start environment 2 with the simulator or with the simulator and bugger/debugger.

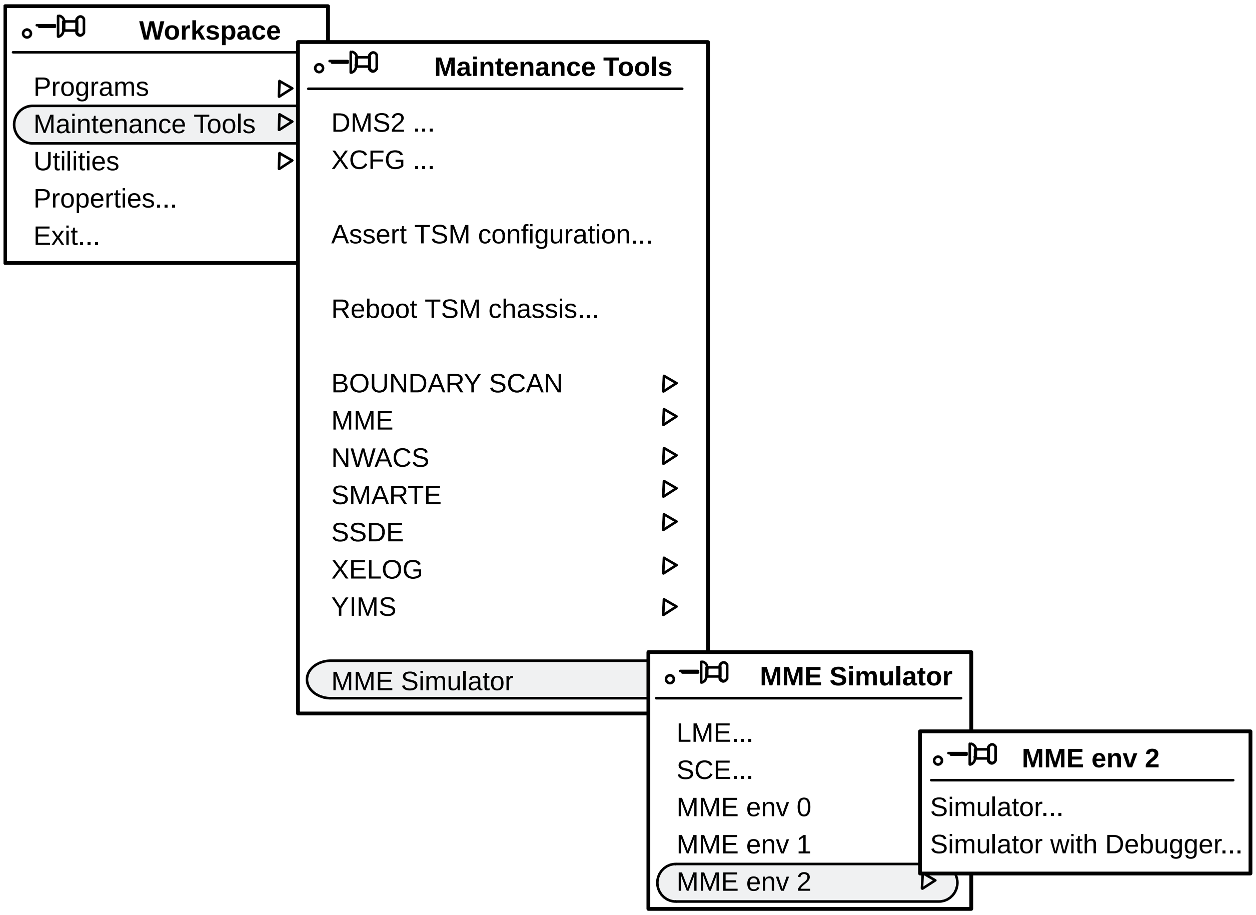


Figure 27. MWS Workspace Menu Options to Start Environment 2 with the Simulator or with the Simulator and Bugger/Debugger

SWS Workspace Menu Options

Figure 28 shows the OpenWindows Workspace menu options that you should choose on an SWS to start environment 2 with an FEI channel. Choose any copy number.

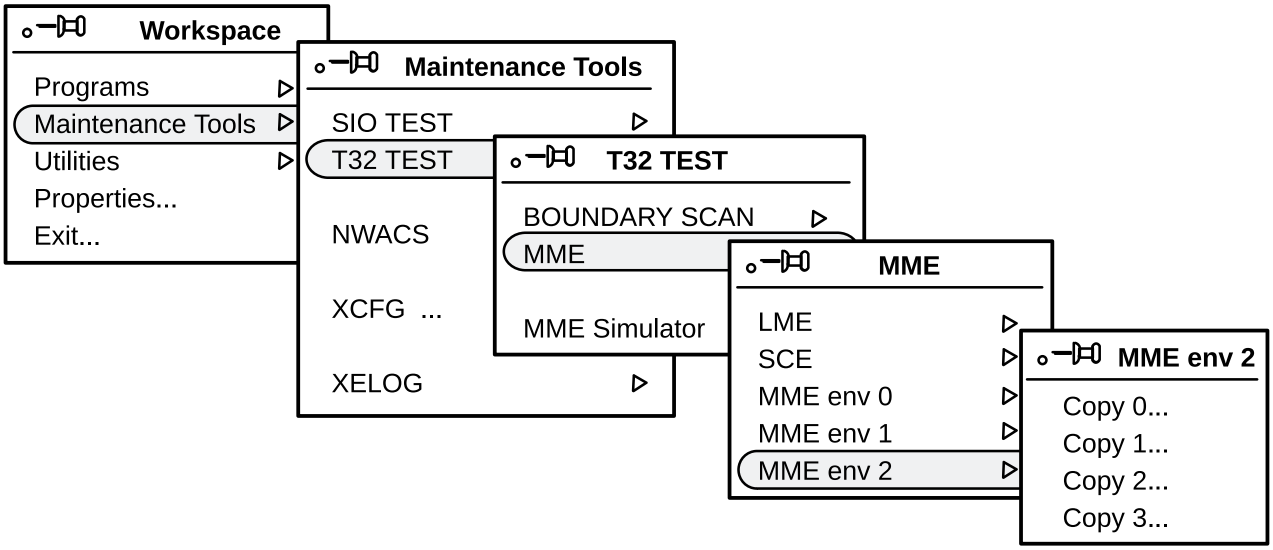


Figure 28. SWS Workspace Menu Options to Start Environment 2 with an FEI Channel

Figure 29 shows the OpenWindows Workspace menu options that you should choose on an SWS to start environment 2 with the simulator or with the simulator and bugger/debugger.

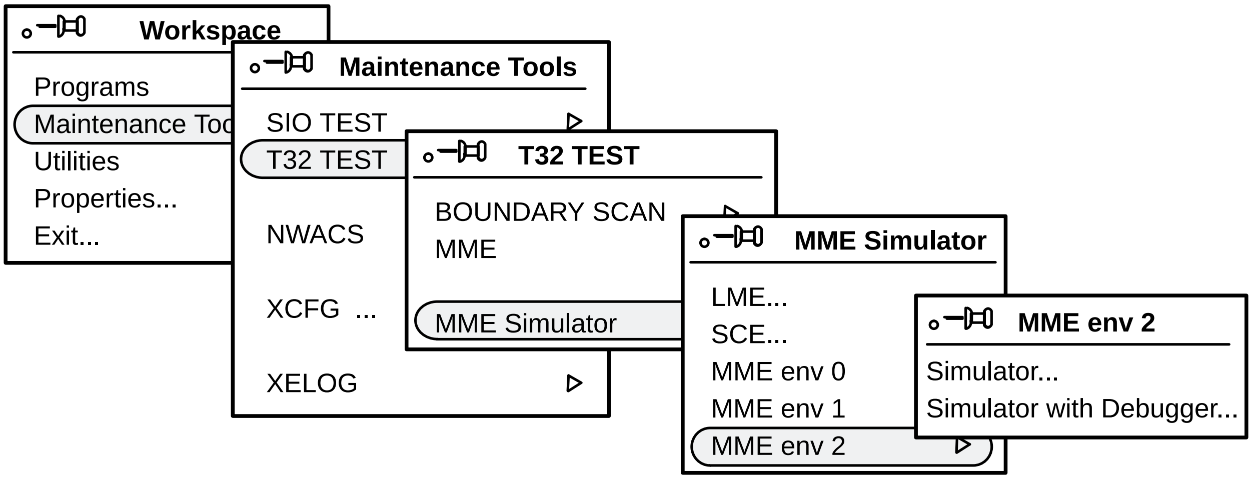


Figure 29. SWS Workspace Menu Options to Start Environment 2 with the Simulator or with the Simulator and Bugger/Debugger

What Happens When You Start Environment 2?

The following actions occur when you start MME:

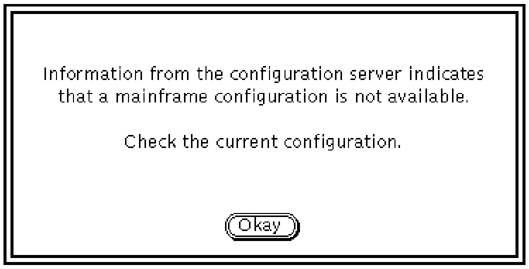
1. The MME server attempts to connect with the System Configuration Environment (SCE) server.

   If MME cannot connect with a running SCE server, MME starts a new SCE server and tries to connect to the new SCE server. (If you specified a configuration file with the `–config` command line option, MME sends this file to SCE through the SCE `–default` command line option. SCE loads the configuration that is stored in the file.)

2. Once MME establishes a connection with SCE, MME attempts to receive a configuration from SCE:

   - If a configuration is available, SCE provides MME with the components that are available for use by the maintenance system. MME automatically configures itself to use these components.

   - If a configuration is not available, MME displays the message shown in the following snap:



Information from the configuration server indicates
that a mainframe configuration is not available.

Check the current configuration.

IF MME displays this message, than you need to create a configuration using SCE before you continue using MME. Refer to the *SCE User Guide*, publication number HDM-069-C, for more information about creating a configuration.

Diagnostic Controller Components

Once MME establishes a configuration, a special Cray Assembly Language (CAL) program, called the diagnostic controller (DC) or controller, automatically loads into the lower 140000_8 words of mainframe memory. The DC acts as an interface between MME in the MWS or SWS and the control points in the mainframe. The controller performs the following actions:

- Manages multiple control points
- Handles memory display updates
- Logs memory and register parity errors
- Routes and handles interrupts
- Handles diagnostic program requests

CAUTION

If UNICOS is running in the mainframe when MME loads the diagnostic controller, the diagnostic controller will overwrite UNICOS in mainframe memory, which will hang the operating system.

To prevent this from accidentally occurring, ensure that the Owner setting in the SCE base window is set to OS for the logical partition in which UNICOS is running when UNICOS is running in the mainframe. MME cannot access a logical partition if the OS owns it.

Figure 30 shows the three main areas of the controller: the standard locations, the code block, and the block storage segment.

Notice that diagnostic controller components begin with a lowercase letter. Remember that control point components begin with an uppercase letter (for example, DEXP, SEXP, and WEXP).

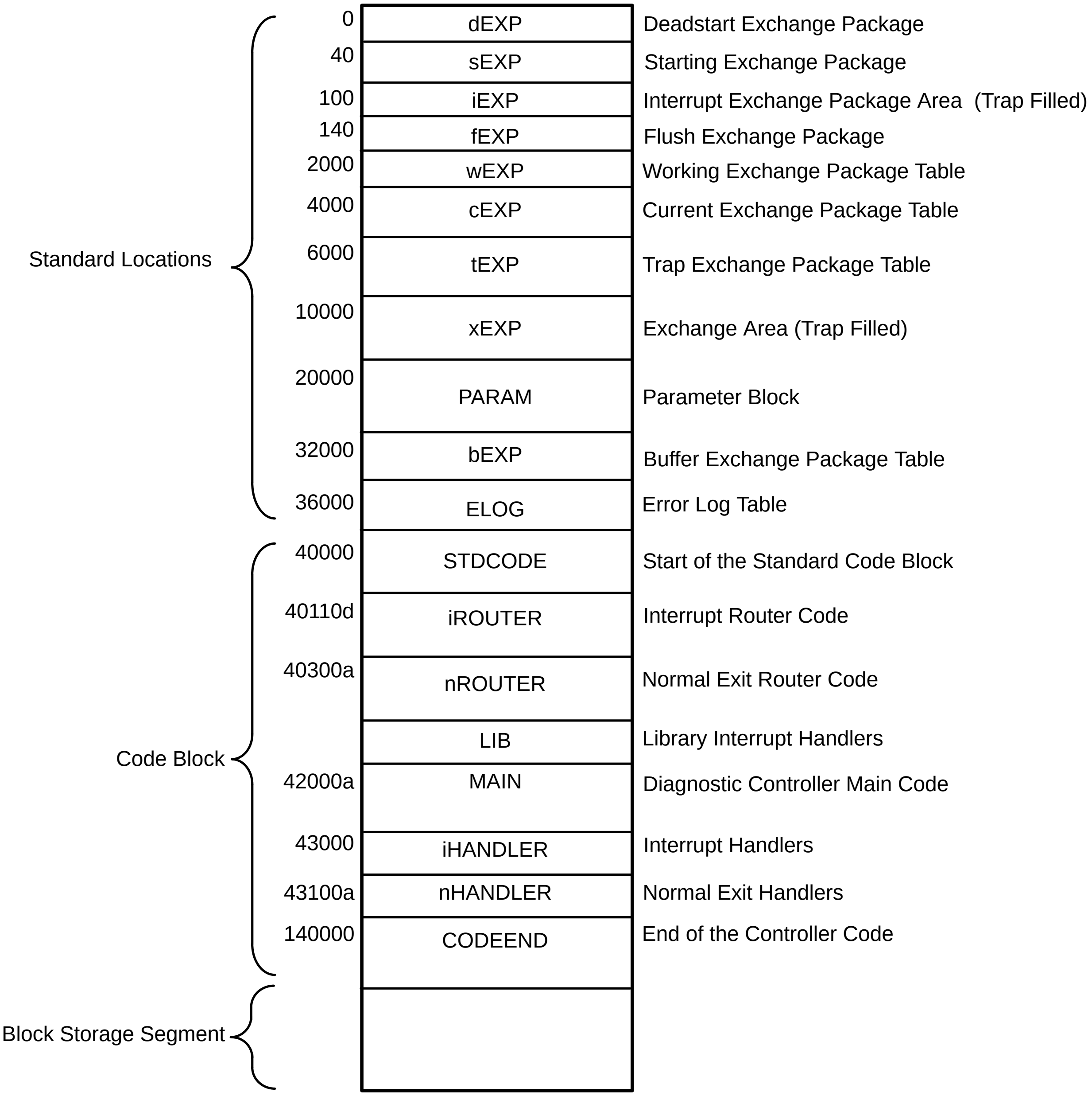


Figure 30. Diagnostic Controller Components

Standard Locations

The standard locations block contains the parameters that the controller uses to operate. The standard locations block includes the deadstart exchange package, the starting exchange package, the interrupt exchange package, the flush exchange package, the working exchange package table, the current exchange package table, the trap exchange package table, the exchange area, the parameter block, the buffer exchange package table, and the error log table.

Deadstart Exchange Package

The deadstart exchange package (dEXP) begins at address 0_8. MME uses this exchange package to exchange the modified copy of sEXP into the CPU when a deadstart occurs.

Starting Exchange Package

The starting exchange package (sEXP) begins at address 40_8. MME uses this exchange package to build the dEXP used to deadstart the CPU. When MME loads the controller, the sEXP includes the following values:

- P register = controller MAIN (42000a)
- XA = wEXP

Interrupt Exchange Package Area

The interrupt exchange package (iEXP) area begins at address 100_8. This area contains exchange packages that have P registers set to iTRAPXA. These exchange packages trap invalid exchanges that occur. There are 76_8 iEXPs in this area.

Flush Exchange Package

The flush exchange package (fEXP) begins at address 140_8. A CPU uses the fEXP to perform a dummy exchange to clear any pending interrupts.

Working Exchange Package Table

The working exchange package (wEXP) table is a group of 40_8 exchange packages that start at address 2000_8. Initially, these exchange packages are identical to iEXP and xEXP, except for the XA and A1 registers. When you click on Go, MME copies the SEXP for a control point into the wEXP for the CPU that is assigned to the control point. The wEXP includes the following values:

- P register = control point MAIN
- XA = wEXP

Current Exchange Package Table

The current exchange package (cEXP) table is a group of 40_8 exchange packages that start at address 3000_8. These exchange packages have P registers set to iTRAPXA. These exchange packages trap invalid exchanges that occur.

Exchange Area

The exchange area (xEXP) begins at 10000_8. This area contains exchange packages that have P registers set to iTRAPXA. These exchange packages trap invalid exchanges that occur.

Parameters

The parameter block, which starts at address 20000_8 (PARAM), contains several tables of parameters that the controller uses to manage control point execution. Refer to Table 13.

Table 13. Controller Parameters

Address	Label	Description
MME updates several tables of base and limit values that the controller uses to access the control points in memory. MME loads these tables, which begin at address 20000_8, before MME makes a request. There are four tables for each CPU: Table 0: Diagnostic LAT logical bases and limits (exchange package format) Table 1: Diagnostic LAT physical (exchange package format) Table 2: Diagnostic absolute base (full address) Table 3: Diagnostic absolute limit (full address)		
20000	mmeLIM	CPU 0 LAT modes, base, limits table
20010	mmePB	CPU 0 LAT physical bias
20020	mmeAB	CPU 0 absolute base
20030	mmeAL	CPU 0 absolute limit
22000	mmeBASE	Control point base address table
22040	mmeCIFM	Clear interrupt flag (1 word per CPU)
22100	diagBASE	Diagnostic base address table
22140	dcCIFM	Copy of actual interrupt mode (IM) (1 word per CPU)

Table 13. Controller Parameters (continued)

Address	Label	Description
The memory allocation tables begin at address 24000_8. These tables contain the currently executing base and limit values. There are four tables for each CPU: Table 0: Diagnostic LAT logical bases and limits (exchange package format) Table 1: Diagnostic LAT physical (exchange package format) Table 2: Diagnostic absolute base (full address) Table 3: Diagnostic absolute limit (full address)		
24000	dcLIM	CPU 0 LAT logical base and limit
24010	dcPB	CPU 0 LAT physical bias
24020	dcAB	CPU 0 absolute base
24030	dcAL	CPU 0 absolute limit
The MME request port contains the requests and responses for communication between the CPU and MME.		
26000	mwsTOcpu	MWS (or SWS)-to-CPU request
26040	mwsACK	CPU-to-MWS (or SWS) response (generated by a CPU)
26100	cpuTOmws	CPU-to-MWS (or SWS) request
26140	cpuACK	MWS (or SWS)-to-CPU response (generated by the MWS or SWS)
The CPU data tables contain data from the CPUs (1 word per CPU for each table).		
26400	hartBEAT	Hartbeat table
26440	idleSTAT	Idle status table
26500	pASS	Diagnostic pass count
26540	eRROR	Diagnostic error count
26600	wEXPP	wEXP P register
26640	wEXPIF	wEXP IF register
26700	iNTFLAGS	Temporary wEXP IF register
26740	sUT	Diagnostic section being tested
27000	cUT	Diagnostic condition being tested
27040	ioLOCKUP	Count of retries in I/O reservation table
27100	srLOCKUP	Count of retries in cluster reservation table
27140	dIFLAGS	Pending diagnostic-handled interrupts
27400	WEXPADDR	WEXP address table
27440	CEXPADDR	CEXP address table
27500	TEXPADDR	TEXP address table
27540	bEXPADDR	bEXP address table
A data block that contains several program variables begins at 27600_8.		
27600	idleHALT	Halt on idle parameter

Table 13. Controller Parameters (continued)

Address	Label	Description
27601	dcHALT	Halt on error active
27602	noRANGE	No base/limit range check parameter: 0 = Check base and limit range 1 = Do not check base and limit range
27603	clrSYS	System (real-time clock [RTC] and I/O channels) clear status parameter: 0 = System was not cleared 1 = System was cleared
27604	cRESBUSY	I/O channel reservation table busy flag
27605	sRESBUSY	Cluster reservation table busy flag
27606	trapSTAT	Save trap status
27607	trapADDR	Save trap address

Exchange Package Swap Buffer

The buffer exchange package table (bEXP) begins at address 32000_8. The controller code uses the bEXP as an exchange package swap buffer.

Error Log Table

The error log table organizes all memory and register parity errors from running control points into one area of memory. The error log table begins at or after address 36000_8. Consult the listing at label ELOG for the actual address.

Refer to the “View –> Error Log” description in the *MME Interface Reference*, publication number HDM-008-A, for information on how to view the error log table.

Code Block

The code block contains all the code necessary for the controller to function. The code block includes the interrupt router code, the normal exit router code, the library handlers, the interrupt handlers, and the normal exit handlers.

Interrupt Router Code

The interrupt router (iROUTER) code begins at address $40110d_8$. The iROUTER code is the first level of interrupt processing. This code determines which interrupts exist and passes the interrupts to the appropriate handler routines.

Normal Exit Router Code

The normal exit router (nROUTER) code begins at address $40300a_8$. The normal router code passes a normal exit to the appropriate normal exit handler code.

Library Interrupt Handlers

The library handlers are standardized handlers that are used for interrupt processing.

Interrupt Handlers

The interrupt handler (iHANDLER) code begins at address 43000_8. This code contains additional handlers that are used to process interrupts.

Normal Exit Handlers

The normal exit handler (nHANDLER) code begins at address $43100a_8$. This code handles normal exit calls.

Block Storage Segment

The block storage segment contains memory that the controller uses to store data as the controller executes. The block storage segment includes the uninitialized data.

Diagnostic Controller Operation

The controller works with MME to control the CPUs. This control ranges from single-CPU diagnostics to multiple CPUs running multiple diagnostics with concurrent I/O activity.

Controller Communication Port

The controller code includes a software bidirectional communication port that is located in mainframe memory. This port consists of an MWS (or SWS)-to-CPU request buffer, an MWS (or SWS)-to-CPU request acknowledge buffer, a CPU-to-MWS (or SWS) request buffer, and a CPU-to-MWS (or SWS) request acknowledge buffer. MME uses this port to make a CPU request by writing a function code into the communication port area of memory. Refer to Table 14 for descriptions of the request function codes. Through this communication port, MME makes requests to the controller, and the controller acknowledges the requests. The controller also uses this port to make requests to MME, and MME uses the port to acknowledge the controller requests.

Table 14. Request Functions

Code	Name	Description	Action
1	START (single CPU)	Start the diagnostic (A7 is unchanged)	MME issues a START@SEXP CPU request for any or all usable CPUs. Upon receipt of the request, a CPU copies the SEXP from its diagnostic data area to the wEXP; the CPU puts the correct instruction base, data base, instruction base limit, and data base limit in the wEXP area and exchanges to the diagnostic.
2	START (multiple CPUs)	Start the diagnostic (A7 = CPU number)	
3	HALT	Copy exchange package at 0 to WEXP, dump the CPU registers, and idle	The CPU writes the contents of its registers to a specified dump buffer located in the diagnostic data area. MME reads the dump buffer and provides a formatted dump display. These requests are also used to rotate control points in the run system.
4	SUSPEND	Dump the CPU registers and idle	
5	UPDATE	Dump the CPU registers and continue diagnostic execution	
6	RESTART (single CPU)	Load registers from the diagnostic dump area and restart (A7 is unchanged)	The CPU loads its register and the WEXP from the dump buffer of the control point and then exchanges to the diagnostic code. The CPU continues execution of the diagnostic where the previous CPU left off. These requests are also used to rotate control points in the run system.
7	RESTART (multiple CPUs)	Load registers from the diagnostic dump area and restart (A7 = CPU number)	

Using this communication port requires significantly less code execution in the mainframe CPU when the MWS (or SWS) is working through the maintenance channel, as opposed to the MWS (or SWS) working through a LOSP channel. Also, the controller will support diagnostic requests that use the maintenance features of the maintenance channel (individual CPU master clear, individual CPU idle, maintenance modes, and the diagnostic monitor).

There are 40_8 function request locations in memory, one for each CPU. They are mwsTOcpu for CPU 0, mwsTOcpu+1 for CPU 1, and so on. When the controller is in the idle loop of the main code block, it monitors the location for the CPU in which it is running. Once that location becomes nonzero, the function code is decoded and acted upon; then the location is zeroed out, which MME reads as an acknowledgement. After a set period of time, MME reads that same location. If the location is zero, the function has been acted upon. If the location remains nonzero, there is a CPU error, and MME prints the appropriate error message.

If, for example, the MWS (or SWS) function is a GO, the controller copies the SEXP of the control point to the wEXP of the controller and then the controller does a normal exchange, which starts the diagnostic program. The control point runs until it receives a halt or an interrupt. If an interrupt occurs, the control point exchanges out to the wEXP, which points to the controller interrupt router.

CPU Deadstart and Control

The controller does not deadstart any CPUs. MME uses the direct memory access (DMA) and the individual CPU control capabilities of the maintenance channel to deadstart the CPUs. Because MME is able to master clear and deadstart CPUs individually through the maintenance channel, you can easily move any CPU into or out of controller code execution. The active CPU handles MME requests through the individual CPU request ports, which are memory locations that are organized by CPU number. MME uses the DMA capacity of the maintenance channel to load diagnostics into memory and to read memory to update all displays, which does not interrupt any CPU that is executing either the controller or diagnostic programs.

MME-to-controller Communications

MME uses the software bidirectional communication port in the controller to control the CPUs. MME writes a start request, dump request, or restart request directly to the MWS (or SWS)-to-CPU request buffer. There are no MCU I/O requests from MME because MCU I/O is done through the DMA of the maintenance channel.

Load a Layout (Optional)

Layouts are not implemented yet.

Allocate Resources (Optional)

MME enables you to change the memory allocation options, the CPU automatic assignment options and CPU modes, the CPU-to-memory delays, and the section swap interval. Refer to the “Properties –> Resource Allocation” description in the *MME Interface Reference*, publication number HDM-008-A, for more information.

Enable the Run System (Optional)

The run system enables you to perform confidence testing of the mainframe by creating an environment for hardware system evaluation that is similar to an operating system. The operating system environment is simulated by swapping jobs (control points) between active CPUs.

Choose **Properties –> Run System** to access the MME Run System window. Use this window to enable the run system and set the properties of the run system parameters. Refer to the “Properties –> Run System (Environment 2 Only)” description in the *MME Interface Reference*, publication number HDM-008-A, for more information about the MME Run System window.

Load One or More Control Points

To perform testing with environment 2, you need to load one or more diagnostic programs, utilities, or loops into mainframe memory. When you load one of these CAL programs into memory, it is called a control point. Because you typically load more than one control point into mainframe memory at a time in environment 2, the control points share the resources of the mainframe; the controller coordinates the resource sharing.

MME performs the following functions to load each control point:

1. MME loads the code that is located in addresses 0 through the end of the standard location block into an MME data buffer.
2. MME configures the diagnostic or utility code in the MME buffer based on the data that is stored in the standard locations. For example, MME configures the memory configuration and CPU select standard locations.
3. MME writes the code that is in the MME data buffer into mainframe memory.
4. MME writes the code from the end of the standard locations to the end of the initialized data into mainframe memory.
5. Optionally, MME clears the dump area.
6. MME overlays any global user changes to the control point sections.
7. MME overlays any section user changes.

Control Point Components

The control point components in environment 2 are similar to the control point components in environment 1, with the following differences:

- Environment 2 control point addressing is relative to the instruction base address of the control point. This occurs because MME does not load the control points at address 0_8, which is where the controller resides. For more information, refer to the following “Control Point Addressing” discussion.
- Environment 2 control points use the interrupt exchange package (IEXP). The IEXP starts at address 100_8. Typically, the controller code uses IEXP to exchange control to the iROUTER code in the control point.

Control Point Addressing

Because the controller starts at address 0_8 and you usually load more than one control point in mainframe memory at a time, control point addressing is not based on address 0_8 (sometimes called absolute addressing) as it is in environment 1. Instead, environment 2 uses relative control point addressing, which bases the addressing of the control point code on the instruction base address (IBA) of the control point. For example, if the control point has an IBA of 240000_8, the actual addresses of the control point components in memory are the addresses in the control point plus the IBA (240000_8). Refer to Figure 31.

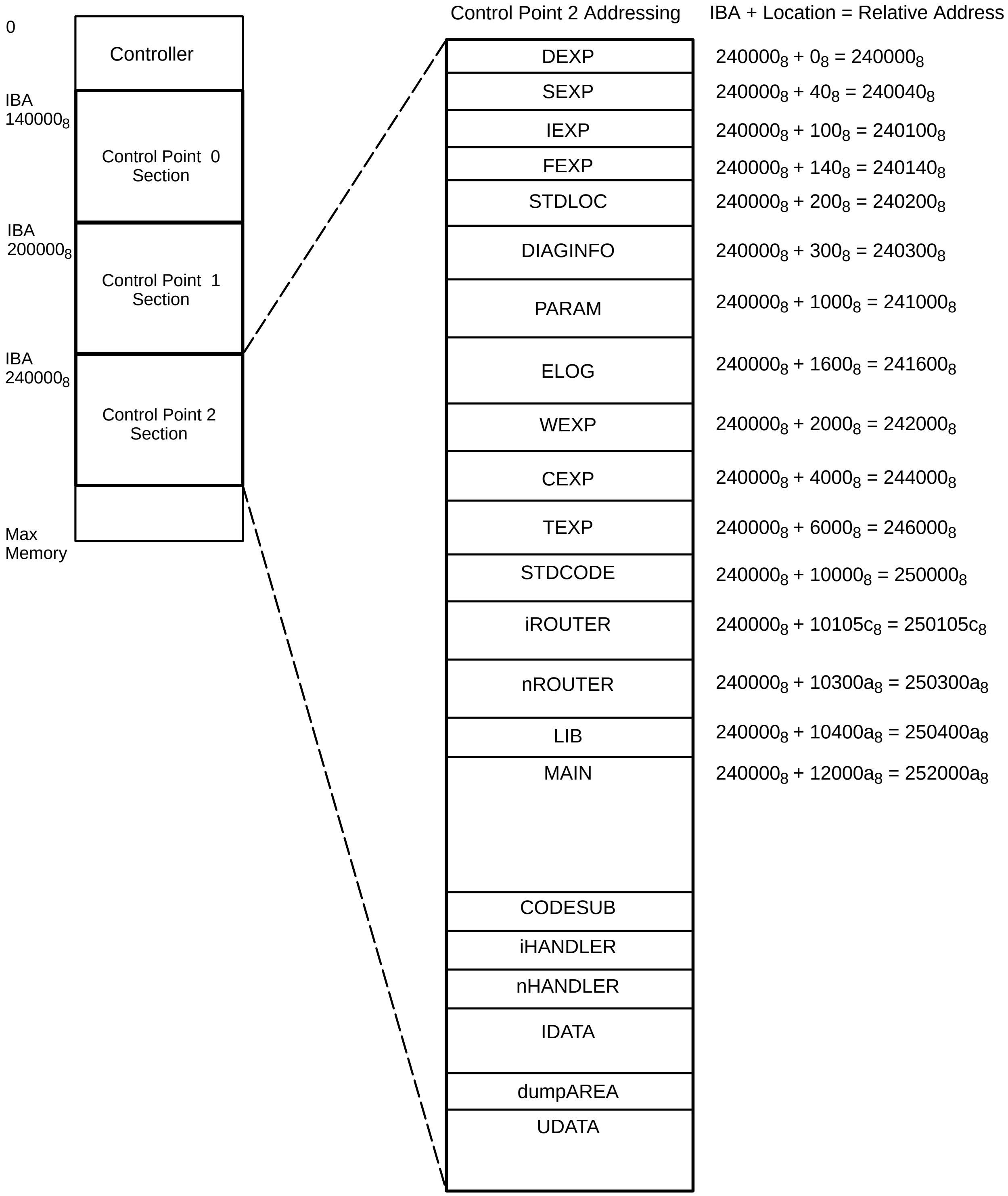


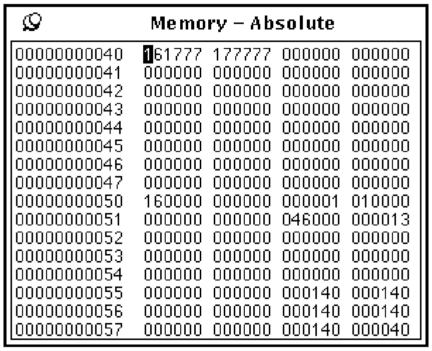
Figure 31. Control Point Addressing

Viewing Memory Addresses

Because addressing in environment 2 is absolute for the controller and relative for the control points, the MME View Memory Setup window enables you to display memory both ways. The following examples illustrate how you can use the settings in the MME View Memory Setup window. Refer to the “View –> Memory” discussion in the *MME Interface Reference*, publication number HDM-008-A, for more information about the MME View Memory Setup window.

Controller Addresses

Absolute addressing uses a base address of 0_8. Controller component addresses in memory location 0_8 through 140000_8 are absolute because the controller code starts at address 0_8. To view these absolute addresses, click on the Base: Absolute setting and enter the address in the Address: field. When you click on View..., a Memory — Absolute window appears, which shows the exact address that you entered. Figure 32 shows a Memory — Absolute window that displays the first 20_8 words of the sEXP for the controller, which is located at absolute address 40_8.



Memory – Absolute

```
00000000040   161777 177777 000000 000000
00000000041   000000 000000 000000 000000
00000000042   000000 000000 000000 000000
00000000043   000000 000000 000000 000000
00000000044   000000 000000 000000 000000
00000000045   000000 000000 000000 000000
00000000046   000000 000000 000000 000000
00000000047   000000 000000 000000 000000
00000000050   160000 000000 000001 010000
00000000051   000000 000000 046000 000013
00000000052   000000 000000 000000 000000
00000000053   000000 000000 000000 000000
00000000054   000000 000000 000000 000000
00000000055   000000 000000 000140 000140
00000000056   000000 000000 000140 000140
00000000057   000000 000000 000140 000040
```

Figure 32. Absolute Memory Display

Control Point Addresses

Relative addressing uses a base address other than 0_8. Control point component addresses are relative because the control point code starts at an instruction base address (IBA), which typically is not 0_8. To view relative addresses, click on the Base: Relative setting and enter the value that you want in the Address: field (for example, enter 40_8 for the SEXP). MME reads the appropriate address based on the IBA for the current control point and displays the data in a Memory (*######*) window.

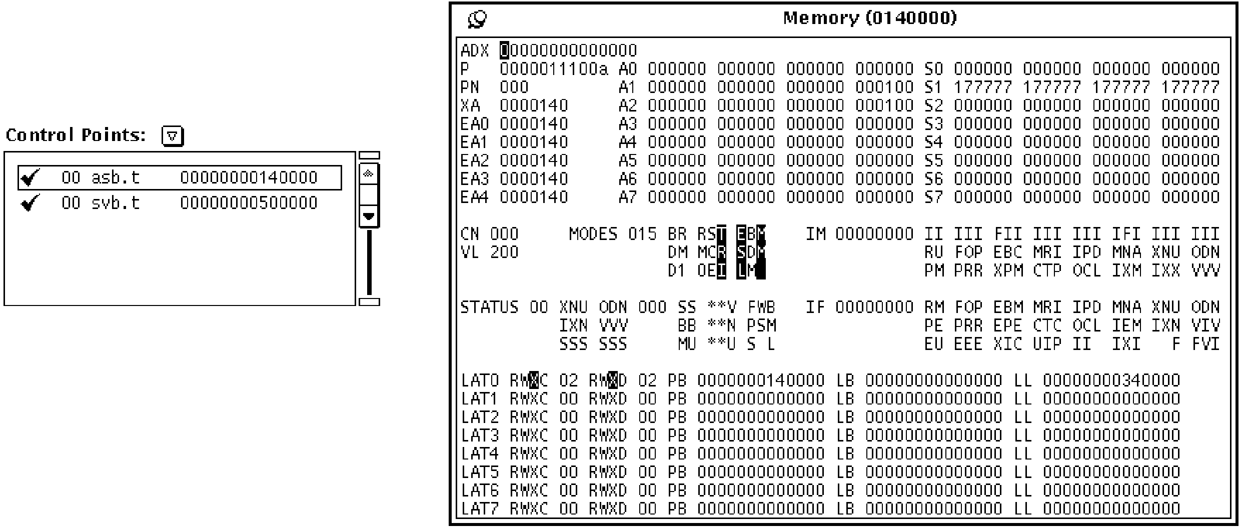
MME includes two options for viewing relative addresses: drifting and anchored modes.

NOTE: In the following examples, all of the exchange packages are for CPUs that use the Cray Research, Inc. (CRI) floating-point number format. If you view an exchange package for a CPU that uses the IEEE floating-point number format, the format of the exchange package will be different from the format of the exchange packages shown in these examples.

Using Drifting Mode

Drifting mode displays memory for the current control point as you change control points. The memory window “drifts” to the base address for the current control point. Figure 33 and Figure 34 provide an example of using drifting mode.

In the following example, MME has two loaded control points. Control point 0 is `asb.t`, which has an IBA of 140000_8. Control point 1 is `svb.t`, which has an IBA of 500000_8. When you first view the memory window, the window displays memory relative to the IBA of `asb.t`, as shown in Figure 33.



The Initial Base Address is the IBA of Control Point 0 (140000_8)

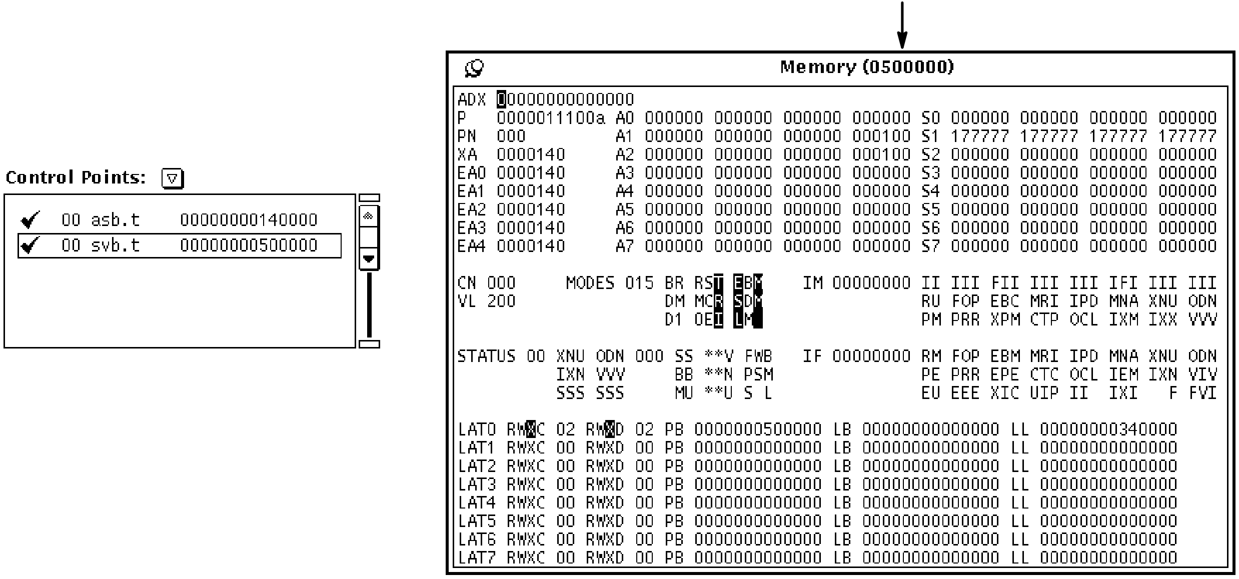
```
Control Points:
✓  00 asb.t     00000000140000
✓  00 svb.t     00000000500000
```

```
Memory (0140000)
ADX 00000000000000
P   0000011100a A0 000000 000000 000000 000000 S0 000000 000000 000000 000000
PN  000         A1 000000 000000 000000 000100 S1 177777 177777 177777 177777
XA  0000140     A2 000000 000000 000000 000100 S2 000000 000000 000000 000000
EA0 0000140     A3 000000 000000 000000 000000 S3 000000 000000 000000 000000
EA1 0000140     A4 000000 000000 000000 000000 S4 000000 000000 000000 000000
EA2 0000140     A5 000000 000000 000000 000000 S5 000000 000000 000000 000000
EA3 0000140     A6 000000 000000 000000 000000 S6 000000 000000 000000 000000
EA4 0000140     A7 000000 000000 000000 000000 S7 000000 000000 000000 000000

CN 000     MODES 015 BR RST EBM    IM 00000000 II III FII III III IFI III III
VL 200               DM MCR SDM                RU FOP EBC MRI IPD MNA XNU ODN
                     D1 OEI LM                 PM PRR XPM CTP OCL IXM IXX VVV

STATUS 00 XNU ODN 000 SS **V FWB   IF 00000000 RM FOP EBM MRI IPD MNA XNU ODN
          IXN VVV     BB **N PSM               PE PRR EPE CTC OCL IEM IXN VIV
          SSS SSS     MU **U S L               EU EEE XIC UIP II  IXI   F FVI

LAT0 RWXC 02 RWXD 02 PB 0000000140000 LB 00000000000000 LL 00000000340000
LAT1 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT2 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT3 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT4 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT5 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT6 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT7 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
```

Figure 33. Drifting Display for the Current Control Point

When you click on control point 1 in the Control Points scroll box, the memory window drifts to the new current control point, which displays memory based on the IBA of `svb.t`, as shown in Figure 34.

The Base Address Drifts to the IBA of Control Point 1 (500000_8)



```
Control Points:
✓ 00 asb.t      00000000140000
✓ 00 svb.t      00000000500000
```

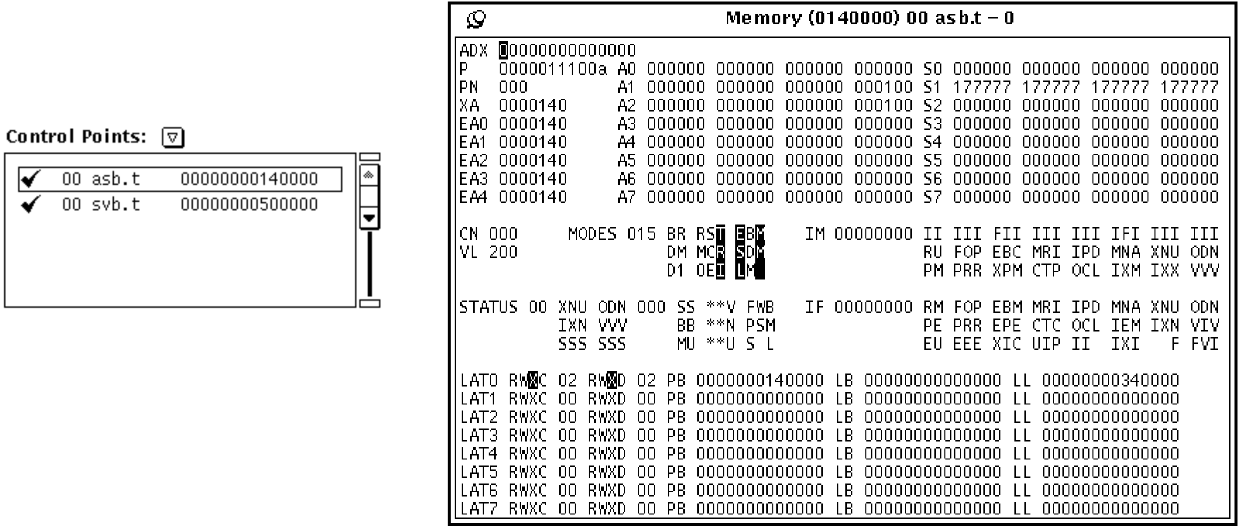
```
Memory (0500000)
ADX 00000000000000
P   0000011100a A0 000000 000000 000000 000000 S0 000000 000000 000000 000000
PN  000         A1 000000 000000 000000 000100 S1 177777 177777 177777 177777
XA  0000140     A2 000000 000000 000000 000100 S2 000000 000000 000000 000000
EA0 0000140     A3 000000 000000 000000 000000 S3 000000 000000 000000 000000
EA1 0000140     A4 000000 000000 000000 000000 S4 000000 000000 000000 000000
EA2 0000140     A5 000000 000000 000000 000000 S5 000000 000000 000000 000000
EA3 0000140     A6 000000 000000 000000 000000 S6 000000 000000 000000 000000
EA4 0000140     A7 000000 000000 000000 000000 S7 000000 000000 000000 000000

CN 000      MODES 015 BR RST EBM    IM 00000000 II III FII III III IFI III III
VL 200                DM MCR SDM                RU FOP EBC MRI IPD MNA XNU ODN
                      D1 OEI LM                 PM PRR XPM CTP OCL IXM IXX VVV

STATUS 00 XNU ODN 000 SS **V FWB   IF 00000000 RM FOP EBM MRI IPD MNA XNU ODN
          IXN VVV     BB **N PSM               PE PRR EPE CTC OCL IEM IXN VIV
          SSS SSS     MU **U S L               EU EEE XIC UIP II  IXI   F FVI

LAT0 RWXC 02 RWXD 02 PB 0000000500000 LB 00000000000000 LL 00000000340000
LAT1 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT2 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT3 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT4 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT5 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT6 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT7 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
```

Figure 34. Drifting Display for the New Current Control Point

Using Anchored Mode

Anchored mode always displays memory for the control point that was current when the memory window was first displayed. The memory window becomes “anchored” to the base address window and always displays memory for that control point, as shown in Figure 35 and Figure 36.

Although the window stays anchored to one control point, the window data changes for the section of the control point that you select, as shown in Figure 37 and Figure 38. The window data changes because individual sections are loaded into mainframe memory and removed from mainframe memory as the current section changes.



The Initial Base Address is the IBA of Control Point 0 (140000_8)

Control Points:

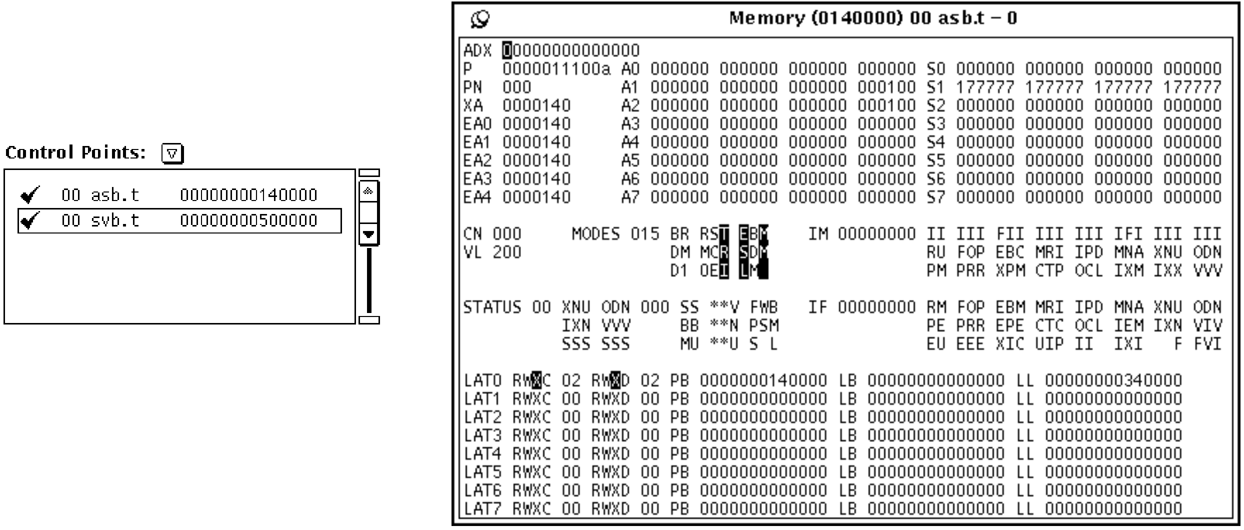
```
✓ 00 asb.t     00000000140000
✓ 00 svb.t     00000000500000
```

Memory (0140000) 00 asb.t – 0

```
ADX  00000000000000
P    0000011100a A0 000000 000000 000000 000000 S0 000000 000000 000000 000000
PN   000         A1 000000 000000 000000 000100 S1 177777 177777 177777 177777
XA   0000140     A2 000000 000000 000000 000100 S2 000000 000000 000000 000000
EA0  0000140     A3 000000 000000 000000 000000 S3 000000 000000 000000 000000
EA1  0000140     A4 000000 000000 000000 000000 S4 000000 000000 000000 000000
EA2  0000140     A5 000000 000000 000000 000000 S5 000000 000000 000000 000000
EA3  0000140     A6 000000 000000 000000 000000 S6 000000 000000 000000 000000
EA4  0000140     A7 000000 000000 000000 000000 S7 000000 000000 000000 000000

CN 000      MODES 015 BR RST EBM    IM 00000000 II  III FII III III IFI III III
VL 200                DM MCR SDM                RU  FOP EBC MRI IPD MNA XNU ODN
                      D1 OEI LM                 PM  PRR XPM CTP OCL IXM IXX VVV

STATUS 00 XNU ODN 000 SS **V FWB    IF 00000000 RM  FOP EBM MRI IPD MNA XNU ODN
          IXN VVV     BB **N PSM                PE  PRR EPE CTC OCL IEM IXN VIV
          SSS SSS     MU **U S L                EU  EEE XIC UIP II  IXI   F FVI

LAT0 RWXC 02 RWXD 02 PB 0000000140000 LB 00000000000000 LL 00000000340000
LAT1 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT2 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT3 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT4 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT5 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT6 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT7 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
```

Figure 35. Anchored Memory Display for the Current Control Point

When you switch control points, the window remains anchored to the IBA of the first control point, as shown in Figure 36.



The Window Remains Anchored to the IBA of Control Point 0

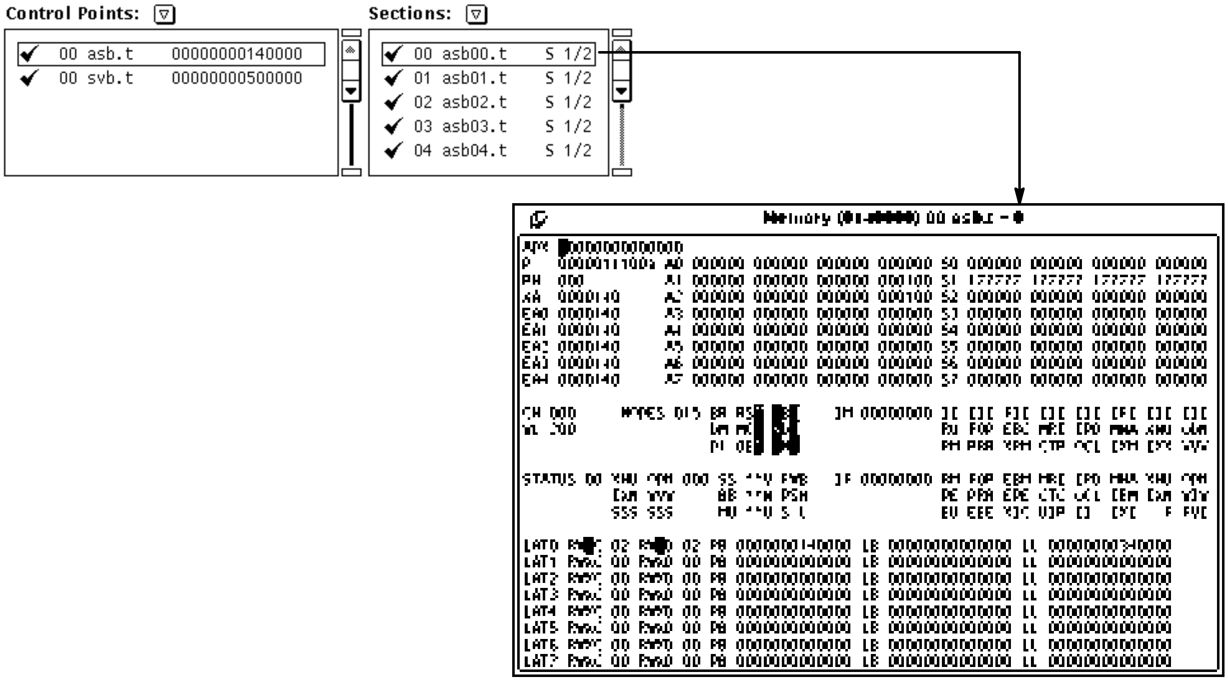
Control Points:

```
✓ 00 asb.t     00000000140000
✓ 00 svb.t     00000000500000
```

Memory (0140000) 00 asb.t – 0

```
ADX  00000000000000
P    0000011100a A0 000000 000000 000000 000000 S0 000000 000000 000000 000000
PN   000         A1 000000 000000 000000 000100 S1 177777 177777 177777 177777
XA   0000140     A2 000000 000000 000000 000100 S2 000000 000000 000000 000000
EA0  0000140     A3 000000 000000 000000 000000 S3 000000 000000 000000 000000
EA1  0000140     A4 000000 000000 000000 000000 S4 000000 000000 000000 000000
EA2  0000140     A5 000000 000000 000000 000000 S5 000000 000000 000000 000000
EA3  0000140     A6 000000 000000 000000 000000 S6 000000 000000 000000 000000
EA4  0000140     A7 000000 000000 000000 000000 S7 000000 000000 000000 000000

CN 000      MODES 015 BR RST EBM    IM 00000000 II  III FII III III IFI III III
VL 200                DM MCR SDM                RU  FOP EBC MRI IPD MNA XNU ODN
                      D1 OEI LM                 PM  PRR XPM CTP OCL IXM IXX VVV

STATUS 00 XNU ODN 000 SS **V FWB    IF 00000000 RM  FOP EBM MRI IPD MNA XNU ODN
          IXN VVV     BB **N PSM                PE  PRR EPE CTC OCL IEM IXN VIV
          SSS SSS     MU **U S L                EU  EEE XIC UIP II  IXI   F FVI

LAT0 RWXC 02 RWXD 02 PB 0000000140000 LB 00000000000000 LL 00000000340000
LAT1 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT2 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT3 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT4 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT5 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT6 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT7 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
```

Figure 36. Anchored Memory Display for the New Current Control Point



Control Points: ▽

✓	00 asb.t	00000000140000
✓	00 svb.t	00000000500000

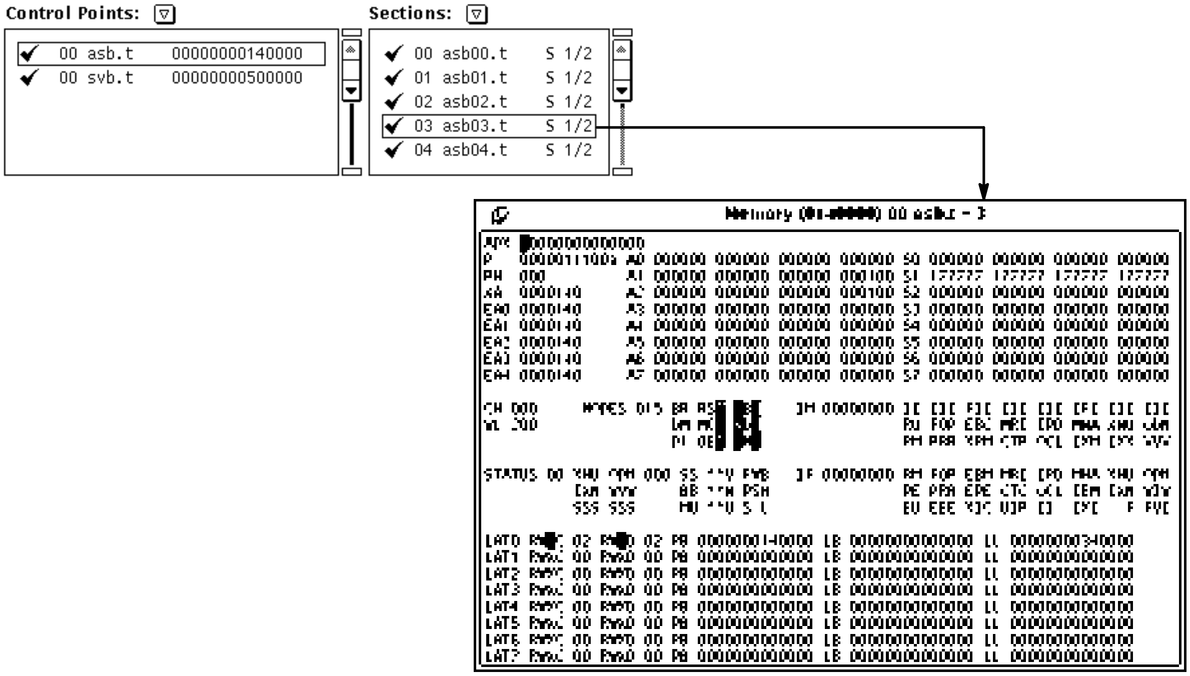
Sections: ▽

✓	00 asb00.t	S 1/2
✓	01 asb01.t	S 1/2
✓	02 asb02.t	S 1/2
✓	03 asb03.t	S 1/2
✓	04 asb04.t	S 1/2

```
Memory ([illegible]) 00 asb.t - 0

APA 0000000000000
P   00000117003 A0 000000 000000 000000 000000 S0 000000 000000 000000 000000
PH  000         A1 000000 000000 000000 000100 S1 177777 177777 177777 177777
XA  0000140     A2 000000 000000 000000 000100 S2 000000 000000 000000 000000
EA0 0000140     A3 000000 000000 000000 000000 S3 000000 000000 000000 000000
EA1 0000140     A4 000000 000000 000000 000000 S4 000000 000000 000000 000000
EA2 0000140     A5 000000 000000 000000 000000 S5 000000 000000 000000 000000
EA3 0000140     A6 000000 000000 000000 000000 S6 000000 000000 000000 000000
EA4 0000140     A7 000000 000000 000000 000000 S7 000000 000000 000000 000000

CH 000     MODES 015 BA AS[illegible]    IM 0000000 [illegible]
VL 200               [illegible]
                     [illegible]

STATUS 00 [illegible]                     IF 0000000 [illegible]

LAT0 [illegible] 02 [illegible] 02 PA 0000000140000 LB 0000000000000 LL 0000000340000
LAT1 RWAC 00 RWAD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT2 RWPC 00 RWPD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT3 RWAC 00 RWAD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT4 RWPC 00 RWPD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT5 RWAC 00 RWAD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT6 RWPC 00 RWPD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT7 RWAC 00 RWAD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
```

Figure 37. Memory Display for the Current Control Point Section

When you switch control point sections, the memory window displays data for the new current section, as shown in Figure 38.



Control Points: ▽

✓	00 asb.t	00000000140000
✓	00 svb.t	00000000500000

Sections: ▽

✓	00 asb00.t	S 1/2
✓	01 asb01.t	S 1/2
✓	02 asb02.t	S 1/2
✓	03 asb03.t	S 1/2
✓	04 asb04.t	S 1/2

```
Memory ([illegible]) 00 asb.t - 3

APA 0000000000000
P   00000117003 A0 000000 000000 000000 000000 S0 000000 000000 000000 000000
PH  000         A1 000000 000000 000000 000100 S1 177777 177777 177777 177777
XA  0000140     A2 000000 000000 000000 000100 S2 000000 000000 000000 000000
EA0 0000140     A3 000000 000000 000000 000000 S3 000000 000000 000000 000000
EA1 0000140     A4 000000 000000 000000 000000 S4 000000 000000 000000 000000
EA2 0000140     A5 000000 000000 000000 000000 S5 000000 000000 000000 000000
EA3 0000140     A6 000000 000000 000000 000000 S6 000000 000000 000000 000000
EA4 0000140     A7 000000 000000 000000 000000 S7 000000 000000 000000 000000

CH 000     MODES 015 BA AS[illegible]    IM 0000000 [illegible]
VL 200               [illegible]
                     [illegible]

STATUS 00 [illegible]                     IF 0000000 [illegible]

LAT0 [illegible] 02 [illegible] 02 PA 0000000140000 LB 0000000000000 LL 0000000340000
LAT1 RWAC 00 RWAD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT2 RWPC 00 RWPD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT3 RWAC 00 RWAD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT4 RWPC 00 RWPD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT5 RWAC 00 RWAD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT6 RWPC 00 RWPD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
LAT7 RWAC 00 RWAD 00 PA 0000000000000 LB 0000000000000 LL 0000000000000
```

Figure 38. Memory Display for the New Current Control Point Section

Assign CPUs to the Control Points

You must assign a CPU to a control point to perform any troubleshooting for the CPU. To assign a CPU to the current control point, click on the CPU in the CPU selection, control point, and status area in the MME base window. Refer to “CPU Selection, Control Point, and Status Area” in the *MME Interface Reference*, publication number HDM-008-A, for more information.

Click on Go

Click on Go to start control point execution; all control points that are assigned CPUs execute the following sequence of events:

1. MME sets the A7 registers for the deadstart exchange package (DEXP) of the control point and the starting exchange package (SEXP) of the control point to the CPU number of the control point:

   DEXP A7 = CPU number
   SEXP A7 = CPU number

2. MME sets the SEXP exchange address to its original value plus the CPU number (in A7) multiplied by 40_8 [SEXP XA = XA + (A7 * 40_8)]. This makes the XA point to the WEXP for the CPU.

   MME also sets exit address 0 (EA0) through EA4 to the original value plus A7 multiplied by 40_8:

   EA0 = EA0 + (A7 * 40_8)
   EA1 = EA1 + (A7 * 40_8)
   EA2 = EA2 + (A7 * 40_8)
   EA3 = EA3 + (A7 * 40_8)
   EA4 = EA4 + (A7 * 40_8)

3. MME copies SEXP to FEXP.

4. MME writes the data to mainframe addresses 0_8 through 200_8, which contain the first four exchange packages of the controller.

5. MME writes the trap exchange package (TEXP) for the current CPU.

6. MME writes the controller tables with base and limit information about the control point.

7. MME writes a START command in the controller communications port. The START command is either START (single CPU) or START (multiple CPUs). Refer again to Table 14 on page 127 for more information about the START commands.

8. The CPU starts executing the control point code.

Monitor the Progress of the Control Points

As control points execute, you should monitor the information that MME displays to determine the progress of the control points. As in environment 1, it is important to understand what happens during control point execution so you can determine whether everything is operating properly. Table 15 lists the status information that you should monitor while the control points execute.

Table 15. Status Information from Executing Control Points

Symptom	Description
"ERROR COUNT" flashing next to a CPU	The control point detected an error. Refer to "Diagnostic-detected Errors" for more information.
"Holding" appears next to a CPU	Control point execution is paused. Check the runtime information display for a prompt for the control point.
Indicator (MEM, RPE, SHR, LAT, or UKN) appears in the menu bar	MME detected a memory, register parity, shared, LAT, or unknown error. MME logs these errors in the error log. Choose **View –> Error Log** to view the error log.
Interrupt flag	An interrupt occurred. Refer to "Interrupts" for more information.
P register is incrementing	Everything is operating correctly.
P register is not incrementing	The P register is hung. Refer to "Intolerable Interrupts" for more information.
"Waiting" appears next to a CPU	The CPU is waiting to execute a multiple-CPU control point.

Diagnostic-detected Errors

Control point test code that is running in a CPU detects and reports an error through the following sequence of events.

1. The control point test code detects a data comparison error for the hardware values that the control point is testing.

2. The control point test code logs the error in the standard locations for the control point.

3. The control point test code performs a dump and idle normal exit (NEX) request, which causes an interrupt.

4. The CPU exchanges from the control point test code into the controller iROUTER, using wEXP.

5. The controller iROUTER code copies wEXP to WEXP. WEXP now contains information about where in the control point test code the exchange occurred.

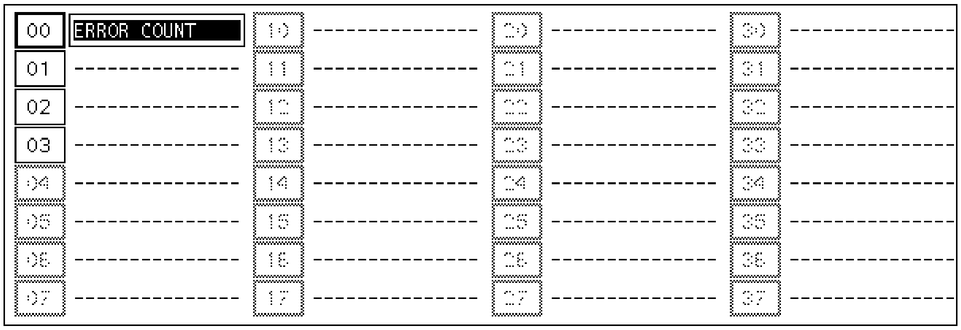
6. The controller iROUTER code updates the diagnostic pass count (pASS), diagnostic error count (eRROR), wEXP P register (wEXPP), wEXP IF register (wEXPIF), diagnostic section under test (sUT), and diagnostic condition under test (cUT) parameters for the controller runtime information display.

7. The controller iROUTER code routes the NEX interrupt to the appropriate handler.

8. MME checks the minimum and maximum error counts that are assigned to the control point.

NOTE: The CPUs do not stop for any errors.

When a diagnostic test program increments the error count, ERROR COUNT flashes next to the CPU in the CPU status area. Figure 39 shows the ERROR COUNT indicator highlighted.



00	ERROR COUNT	10	---------------	20	---------------	30	---------------
01	---------------	11	---------------	21	---------------	31	---------------
02	---------------	12	---------------	22	---------------	32	---------------
03	---------------	13	---------------	23	---------------	33	---------------
04	---------------	14	---------------	24	---------------	34	---------------
05	---------------	15	---------------	25	---------------	35	---------------
06	---------------	16	---------------	26	---------------	36	---------------
07	---------------	17	---------------	27	---------------	37	---------------

Figure 39. Error Indicator

When you see an error indicator, refer to the error return address (ERA) in the diagnostic information block. The ERA, which is located at address 0305, indicates an area in the listing near the location of the failing code. Look at the code in the listing adjacent to the ERA to determine the code that actually failed.

NOTE: The ERA is shown on the DIAGINFO runtime information display, which is currently available for most diagnostic tests and utilities.

For information on how to view a listing, refer to “View –> Listing –> Current” and “View –> Listing –> Other” in the *MME Interface Reference*, publication number HDM-008-A.

Interrupts

As in environment 1, there are two classes of interrupts in environment 2: tolerable and intolerable interrupts. Remember that tolerable interrupts are interrupts that occur while the CPU is processing the main diagnostic code. Tolerable interrupts can be ignored or routed, depending on the code of the diagnostic program. Intolerable interrupts are interrupts that occur while the CPU is processing code from the standard code block. Intolerable interrupts are trapped by hanging the CPU.

Figure 40 shows the two interrupt classes and the actions performed when interrupts occur.

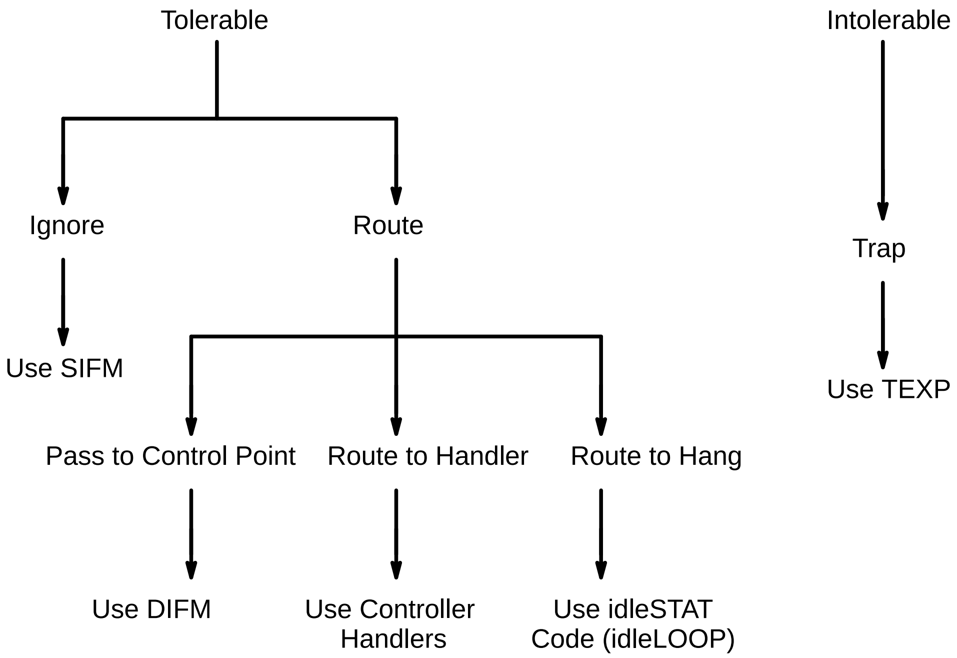


Figure 40. Environment 2 Interrupt Classes and Actions

Environment 2 uses a two-tiered interrupt-processing system. When interrupts occur, the CPU exchanges to the controller iROUTER code to begin processing the interrupts. The first tier of processing uses the controller iROUTER code. The second tier uses the iROUTER code from the control points. The DIFM parameter specifies which tier processes an interrupt.

Typically, environment 2 processes interrupts in the first tier (controller iROUTER code). Occasionally, the control point code contains the routines that are necessary to route and handle an interrupt, so interrupt processing moves to the second tier (control point iROUTER code). To move into the second tier, the controller uses the DIFM parameter, which contains a bit for each interrupt flag. If the bit for a flag is set in the DIFM parameter, the controller passes the interrupt to the control point iROUTER code for processing by the control point.

Environment 2 processes interrupts using the procedure that Figure 41 shows and the text that follows the figure describes. The circled numbers in Figure 41 correspond to the numbered steps in the text that follows the figure.

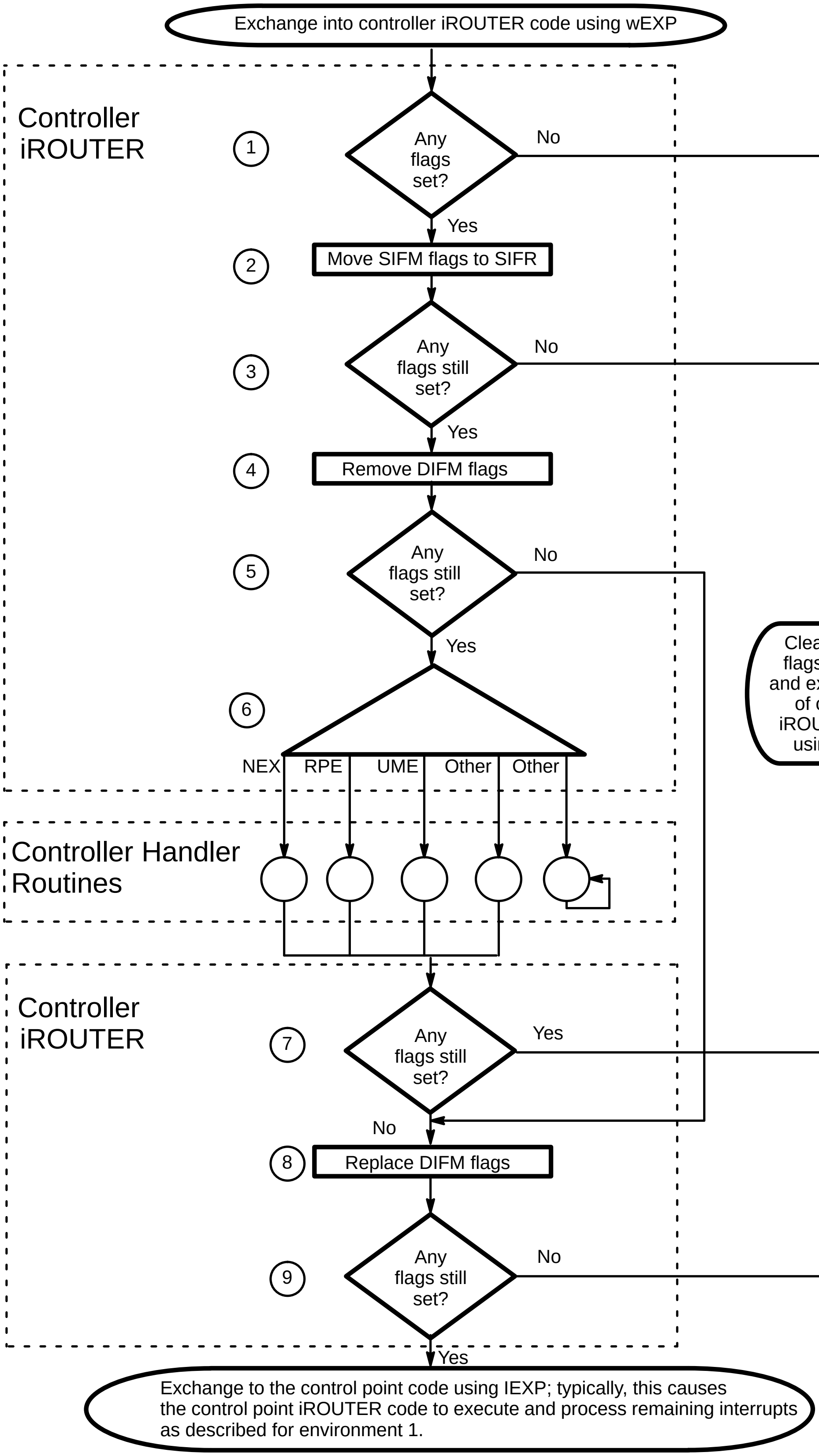

Exchange into controller iROUTER code using wEXP
Controller iROUTER
1
Any flags set?
No
Yes
2
Move SIFM flags to SIFR
3
Any flags still set?
No
Yes
4
Remove DIFM flags
5
Any flags still set?
No
Yes
6
NEX
RPE
UME
Other
Other
Idle Loop (Hangs CPU)
Clear interrupt flags in wEXP and exchange out of controller iROUTER code using wEXP
Controller Handler Routines
Controller iROUTER
7
Any flags still set?
Yes
No
8
Replace DIFM flags
9
Any flags still set?
No
Yes
Exchange to the control point code using IEXP; typically, this causes the control point iROUTER code to execute and process remaining interrupts as described for environment 1.

Figure 41. Interrupt Processing (Controller)

1. The controller iROUTER code verifies that interrupt flags are set in the current interrupt list.

   - If interrupt flags are set, interrupt processing continues with Step 2.

   - If no interrupt flags are set, the iROUTER code causes the CPU to execute an idle loop, which hangs the CPU.

2. The controller iROUTER code checks the SIFM parameter. The controller iROUTER code moves the flags that the SIFM parameter specifies from the current interrupt list to the SIFR parameter. The controller iROUTER code does not process these moved interrupts; the interrupts are ignored.

3. The controller iROUTER code verifies that interrupt flags are set in the current interrupt list.

   - If interrupt flags are still set, more interrupts exist. Interrupt processing continues with Step 4.

   - If no interrupt flags are set, interrupt processing is complete. The controller iROUTER code clears the interrupt flags in wEXP, and MME exchanges the controller iROUTER code out of the CPU and the control point code into the CPU. The CPU continues to execute the control point code.

4. The controller iROUTER code checks the DIFM parameter. The controller iROUTER code moves the flags that the DIFM parameter specifies from the current interrupt list to the dcDIFM parameter. The control point iROUTER code processes these moved parameters after the controller iROUTER code has finished processing interrupts.

5. The controller iROUTER code verifies that interrupt flags are set in the current interrupt list.

   - If interrupt flags are still set, more interrupts exist. Interrupt processing continues with Step 6.

   - If no interrupt flags are set, the controller iROUTER code has completed processing the current list of interrupts. The controller iROUTER code continues with Step 8.

6. The controller iROUTER code routes each interrupt in the current interrupt list. If a handler exists for an interrupt, the controller iROUTER code routes the interrupt to the handler code.

NOTE: The iROUTER code picks off one interrupt and sends it to the handler code; this process (Step 6) repeats until all interrupts are processed.

7. The controller iROUTER code verifies that interrupt flags are set in the current interrupt list.

   - If interrupt flags are still set, the iROUTER code causes the CPU to execute an idle loop, which hangs the CPU.

   - If no interrupt flags are set, the controller iROUTER code has completed processing the current list of interrupts. The controller iROUTER code continues with Step 8.

8. The controller iROUTER code replaces any flags that were removed in Step 4 so the control point iROUTER code can process any interrupts that are indicated in the DIFM parameter.

9. The controller iROUTER code verifies that interrupt flags are set in the current interrupt list.

   - If interrupt flags are still set, these flags came from the DIFM parameter. The controller iROUTER code causes an exchange that typically exchanges the control point iROUTER code into the CPU so the control point can process the interrupts.

   - If no interrupt flags are set, interrupt processing is complete. The controller iROUTER code clears the interrupt flags in wEXP, and MME exchanges the controller iROUTER code out of the CPU and the control point code into the CPU. The CPU continues to execute the control point code.

Tolerable Interrupts

Tolerable interrupts are interrupts that the controller or control point iROUTER code expect. The controller iROUTER code performs three functions for tolerable interrupts: ignore, pass to the control point, and route to handler.

Ignore

In environment 2, the controller code determines which interrupts should be ignored. The controller iROUTER code can remove interrupts from the current list of interrupt flags to be processed. This prevents the interrupts from being processed. If a diagnostic test or utility program sets the corresponding bit for an interrupt in the system interrupt flag mask

(SIFM) parameter, the controller iROUTER code at cKSIFM moves the interrupt from the current list to the system interrupt return mask (SIFR) parameter. The interrupt is ignored.

Ignored interrupts are typically used so the diagnostic test code can force an interrupt condition. When control returns to the test code, the test code checks the SIFR parameter to verify that the interrupt occurred.

Pass to a Control Point

A control point can contain the code necessary to handle certain interrupts. If it does, the control point code sets the bits in the DIFM parameter that correspond to the interrupts that the control point will handle. The controller iROUTER code checks the bits in the DIFM parameter and moves the interrupt flags that are set in DIFM from the current interrupt list to the dcDIFM parameter.

When the controller code finishes processing all remaining interrupts, the controller iROUTER code returns the DIFM flags to the current interrupt list and exchanges the CPU to the control point iROUTER code. The control point iROUTER code processes the interrupts as described for environment 1.

Route to a Handler

The controller handles interrupts through special code sections called handlers. Handlers contain the code that is necessary to process interrupts. Three types of handlers may be available in the controller: library interrupt handlers, interrupt handlers, and normal exit handlers. If a handler is available in the controller code for an interrupt, the controller iROUTER code routes the interrupt to the handler.

NOTE: Some handlers are just inline hangs.

Library Interrupt Handlers and Interrupt Handlers

Library interrupt handlers and interrupt handlers are general handlers that are included in the controller.

Normal Exit Handlers

The controller includes normal exit handlers that enable control points to perform tasks in monitor mode instead of user mode, in which the control points usually execute. The control point code causes a normal exit (NEX) interrupt to access the code that the normal exit handlers contain.

Figure 42 is a flowchart of normal exit interrupt processing. The circled numbers in Figure 42 correspond to the numbered steps in the text that follows the figure.

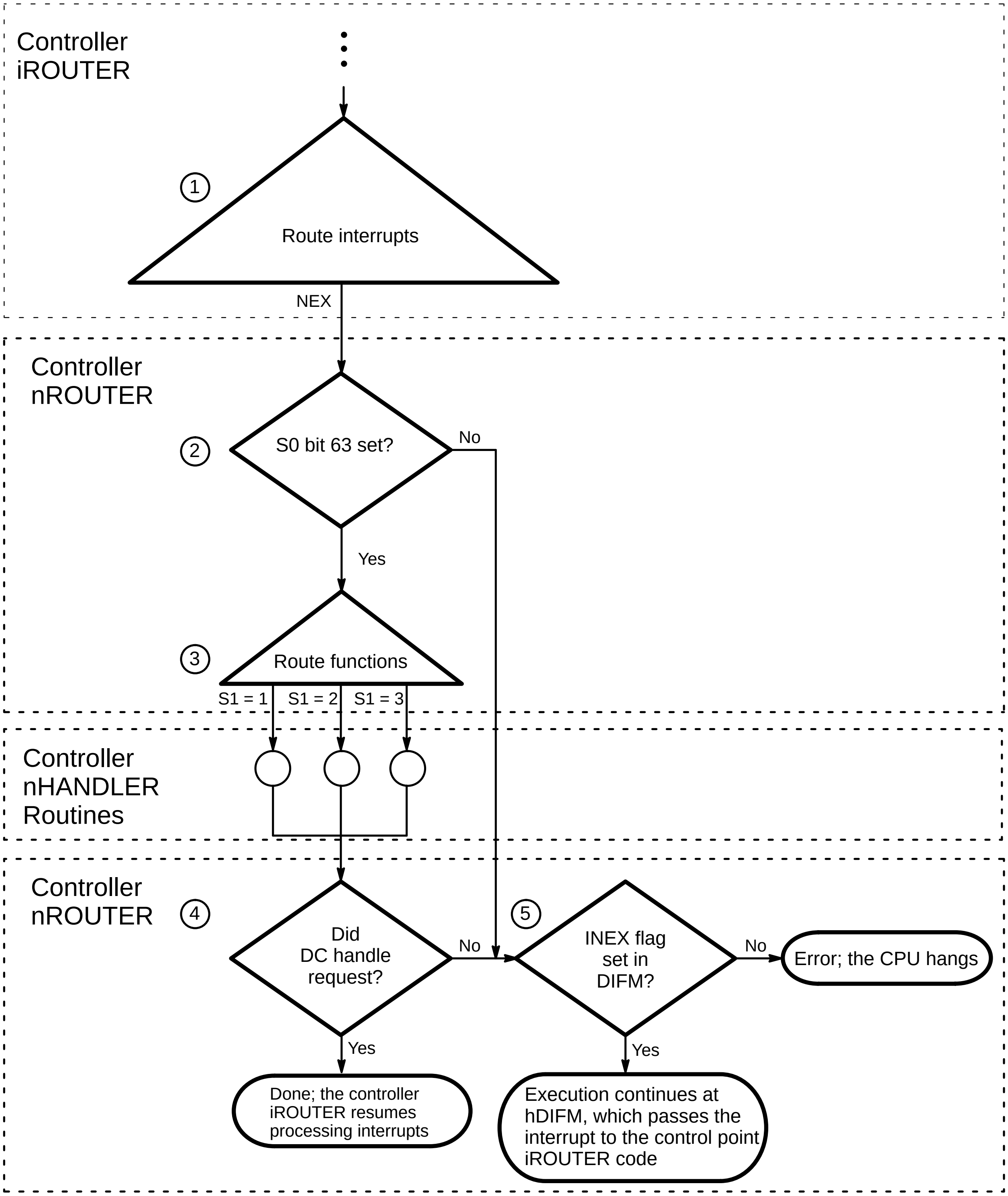


Figure 42. Normal Exit Interrupt Processing

1. The controller iROUTER code routes the NEX interrupt to the controller nROUTER code.

2. The controller nROUTER code verifies that bit 63 (the sign bit) of S0 is set to 1. The control point diagnostic code must set this bit to indicate that a NEX interrupt is occurring. If this bit is not set, processing continues with Step 6.

3. The controller nROUTER code routes the NEX interrupt to the appropriate handler based on the value stored in S1. The handler performs the necessary functions to process the request.

4. The controller nROUTER code verifies that the controller code handled the NEX request.

   - If the request was handled, normal exit processing is complete. Interrupt processing resumes in the controller iROUTER code, which Figure 42 shows.

   - If the request was not handled, NEX processing continues with Step 5.

5. The controller nROUTER code examines the DIFM parameter.

   - If the bit for the NEX flag is set in DIFM, the controller nROUTER code moves the NEX flag to the DIFM copy (dcDIFM) and routes the request back to the control point through the DIFM handler (hDIFM).

   - If the bit for the NEX flag is not set in DIFM, the CPU hangs because an error occurred.

If the CPU cannot complete the request (requested I/O channel busy, mode not allowed and so on), an exchange occurs with the return exchange package that has a fail code in the S0 register. The control point should detect and handle all incomplete requests. If the request is completed by the controller, the content of the S0 register is intact.

Table 16 shows the normal exchange request bit fields. Table 17 shows the normal exit requests that correspond to the bits that are set in the S1 register.

Table 16. Environment 2 Normal Exit Request Bit Fields

Register	Contents			
S0	Valid Request Flag, Bit 63 = 1			
S1	Function Mask for the Controller Only Handlers, Bits 63 – 48	Function Mask for the Controller and Handlers, Bits 47 – 32	Function Mask for the Router Only Handlers, Bits 31 – 16	Function Mask for the Program-defined Handlers, Bits 15 – 0
S2	Parameter 1			
S3	Parameter 2			
S4	Parameter 3			
A0	Parameter 4			
A1	Parameter 5			
A2	Parameter 6			
A3	Parameter 7			

Table 17. Environment 2 Normal Exit Routines

Octal Bit	Name	Description
66	hNOP	Perform no operation
65	hHALT	Halt all CPUs in the controller
62	hioLOCK	Reserve LOSP or VHISP I/O channel Parameters: S2: Bit mask of channels to reserve (channels 077 – 000) S3: Bit mask of channels to reserve (channels 177 – 100) S4: 0 = Release 1 = Reserve 2 = One shot (The channel is released after the first interrupt.)
60	hXEXP	Exchange using the exchange package table Parameters: S2: Pointer to the exchange package table S3: 0 = Copy from the table 1 = Swap with the table
57	hIDLE	Dump registers and idle the CPU
56	hIDLE	Dump registers and wait for the hold bit to clear
46	hHOLD	Hold on WAIT/RESUME

Table 17. Environment 2 Normal Exit Routines (continued)

Octal Bit	Name	Description
44	hsrLOCK	Set shared register cluster Parameters: S2: Cluster or cluster mask S3: 0 = Release cluster 1 = Reserve cluster 1000 = Release clusters in cluster mask 1001 = Reserve clusters in cluster mask When this routine sets a single cluster number (S3 = 1), the CLN register is set upon return. When this routine sets a group of clusters (S3 = 1001), S2 returns a mask of reserved clusters. S0 returns –22 if another control point controls the requested cluster.
43	hSETPCI	Set up PCI if it is selected in PCITIME
41	hSETM	Set or clear mode flags in WEXP Parameters: S2: Bit mask of flags relative to the register you are using S3: 0 = Clear flags 1 = Set flags
40	hSETIM	Set or clear interrupt mode bits in WEXP Parameters: S2: Bit mask of mode bits relative to the register you are using S3: 0 = Clear bits 1 = Set bits

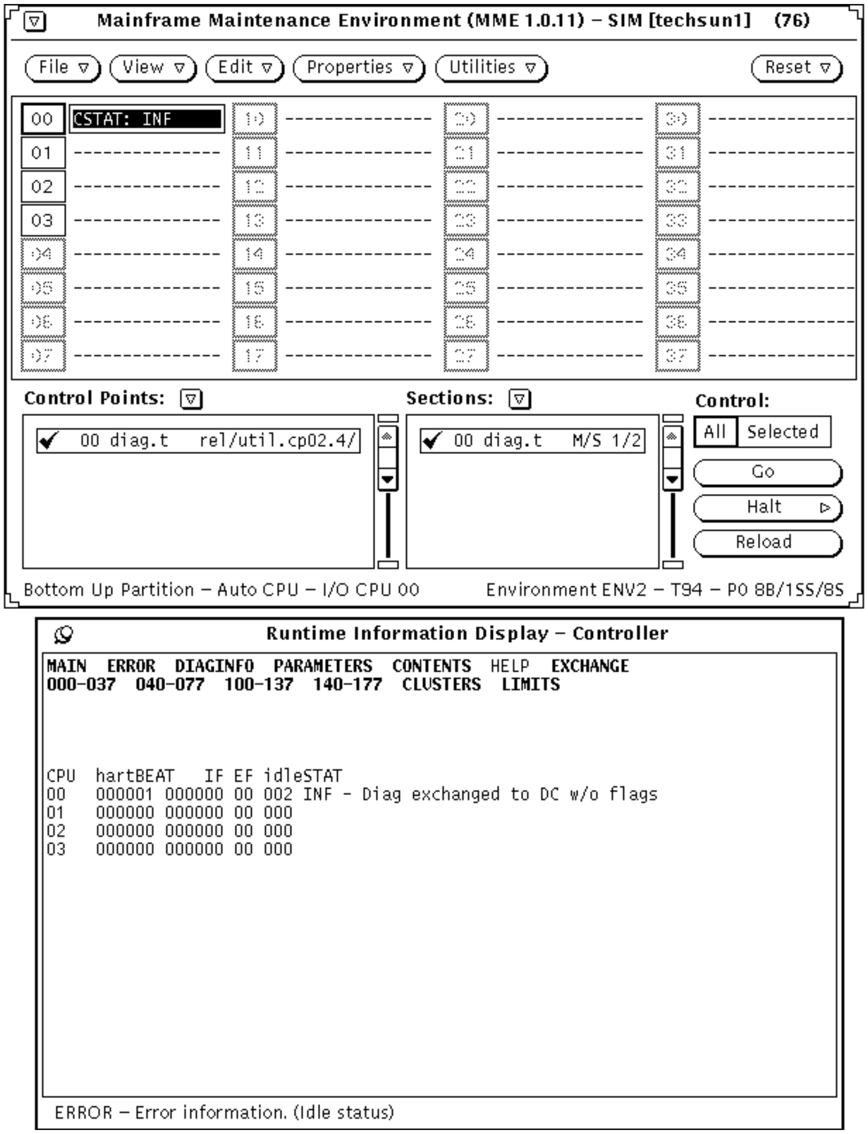
Intolerable Interrupts

An intolerable interrupt is an interrupt that neither the controller nor the control point are expecting. Intolerable interrupts can occur in the main diagnostic code or standard code of the controller or control point.

Exchange into Controller with No Interrupt Flags

If a CPU exchanges into the controller with no interrupt flags set, the controller iROUTER code places the value %%INF in memory location idleSTAT, traps the CPU by jumping to an idle loop, and increments the location hartBEAT, which MME periodically checks.

If hartBEAT is nonzero, MME reads idleSTAT, translates the value into a controller code, displays the code next to the CPU in the MME base window, and prints a message on the ERROR runtime information display for the controller. Refer to Figure 43.


Mainframe Maintenance Environment (MME 1.0.11) – SIM [techsun1] (76)
File ▽
View ▽
Edit ▽
Properties ▽
Utilities ▽
Reset ▽
Control Points: ▽
Sections: ▽
Control:
All
Selected
Go
Halt ▷
Reload
Runtime Information Display – Controller
MAIN ERROR DIAGINFO PARAMETERS CONTENTS HELP EXCHANGE
000-037 040-077 100-137 140-177 CLUSTERS LIMITS

```
00  CSTAT: INF
01  ---------------
02  ---------------
03  ---------------
```

```
✔ 00 diag.t   rel/util.cp02.4/
✔ 00 diag.t   M/S 1/2
```

```
Bottom Up Partition – Auto CPU – I/O CPU 00          Environment ENV2 – T94 – P0 8B/1SS/8S
```

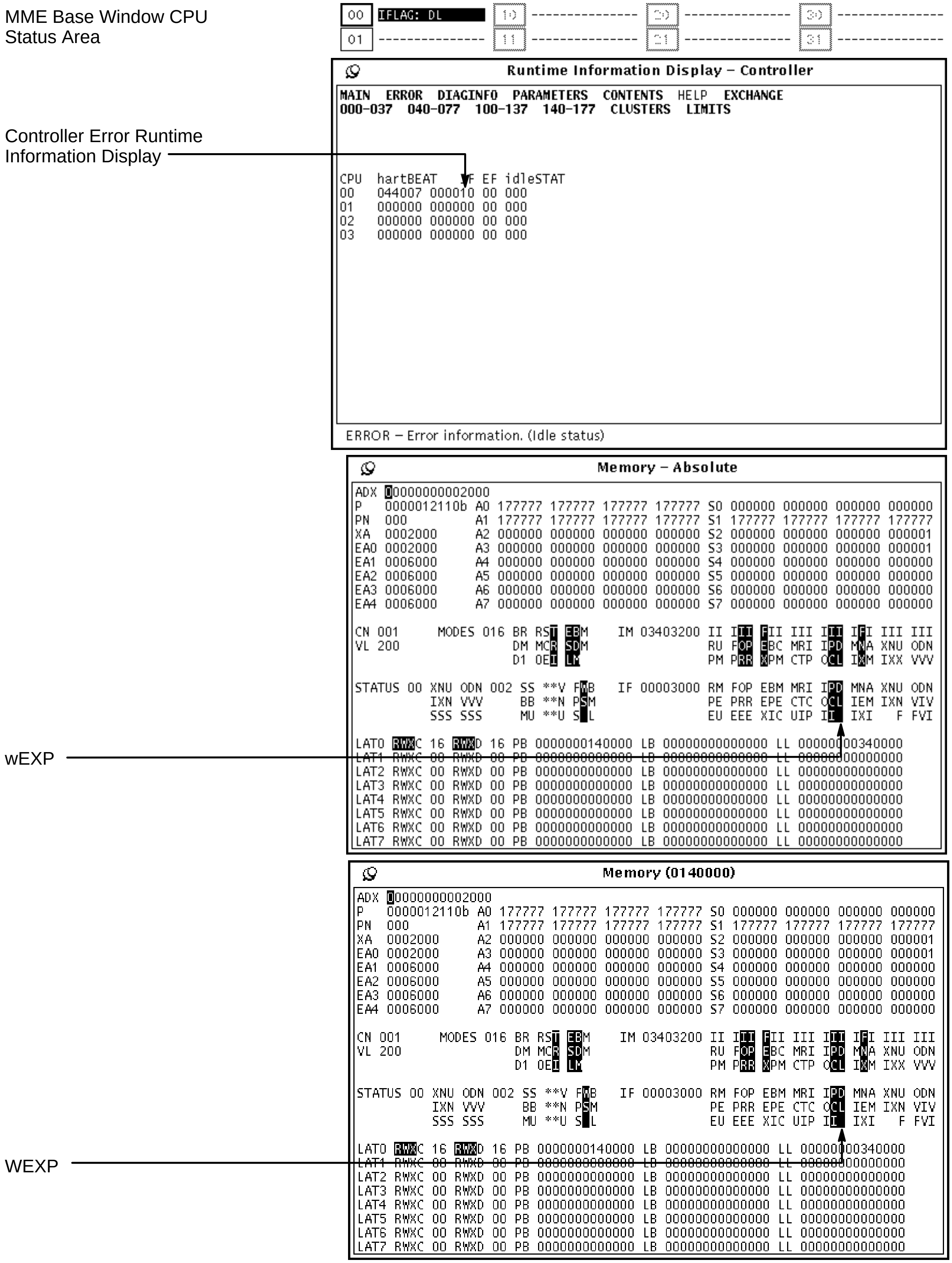
```
CPU   hartBEAT   IF EF idleSTAT
00    000001 000000 00 002 INF - Diag exchanged to DC w/o flags
01    000000 000000 00 000
02    000000 000000 00 000
03    000000 000000 00 000
```

```
ERROR – Error information. (Idle status)
```

Figure 43. Idle Status after an Exchange with No Flags

Exchange into the Controller with an Interrupt the Controller and Control Point Cannot Handle

When an interrupt occurs that the controller and control point cannot handle, the controller iROUTER code moves the flag to the iNTFLAGS parameter, clears all channel and cluster reservations for the control point, and hangs the CPU at location iDLELOOP. Figure 44 shows an example of an interrupt that the controller and control point cannot handle. Notice that the flag for the interrupt is shown in the MME base window CPU status area, the controller ERROR runtime information display, the wEXP, and the WEXP for the CPU.


MME Base Window CPU Status Area
Controller Error Runtime Information Display
wEXP
WEXP

```
00  IFLAG: DL         10  ---------------   20  ---------------   30  ---------------
01  ---------------   11  ---------------   21  ---------------   31  ---------------
```

```
                Runtime Information Display – Controller
MAIN   ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
000-037  040-077  100-137  140-177  CLUSTERS  LIMITS

CPU  hartBEAT    wF EF idleSTAT
00   044007 000010 00  000
01   000000 000000 00  000
02   000000 000000 00  000
03   000000 000000 00  000

ERROR – Error information. (Idle status)
```

```
                         Memory – Absolute
ADX  00000000002000
P    0000012110b  A0  177777  177777  177777  177777  S0  000000  000000  000000  000000
PN   000          A1  177777  177777  177777  177777  S1  177777  177777  177777  177777
XA   0002000      A2  000000  000000  000000  000000  S2  000000  000000  000000  000001
EA0  0002000      A3  000000  000000  000000  000000  S3  000000  000000  000000  000001
EA1  0006000      A4  000000  000000  000000  000000  S4  000000  000000  000000  000000
EA2  0006000      A5  000000  000000  000000  000000  S5  000000  000000  000000  000000
EA3  0006000      A6  000000  000000  000000  000000  S6  000000  000000  000000  000000
EA4  0006000      A7  000000  000000  000000  000000  S7  000000  000000  000000  000000

CN 001      MODES 016  BR RST EBM     IM 03403200  II  III FII III III IFI III III
VL 200                 DM MCR SDM                  RU  FOP EBC MRI IPD MNA XNU ODN
                       D1 OEI LM                   PM  PRR XPM CTP OCL IXM IXX VVV

STATUS 00 XNU ODN 002 SS **V FWB    IF 00003000 RM FOP EBM MRI IPD MNA XNU ODN
          IXN VVV     BB **N PSM                PE PRR EPE CTC OCL IEM IXN VIV
          SSS SSS     MU **U S L                EU EEE XIC UIP II  IXI   F FVI

LAT0 RWXC 16 RWXD 16 PB 0000000140000 LB 00000000000000 LL 00000000340000
LAT1 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT2 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT3 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT4 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT5 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT6 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT7 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
```

```
                         Memory (0140000)
ADX  00000000002000
P    0000012110b  A0  177777  177777  177777  177777  S0  000000  000000  000000  000000
PN   000          A1  177777  177777  177777  177777  S1  177777  177777  177777  177777
XA   0002000      A2  000000  000000  000000  000000  S2  000000  000000  000000  000001
EA0  0002000      A3  000000  000000  000000  000000  S3  000000  000000  000000  000001
EA1  0006000      A4  000000  000000  000000  000000  S4  000000  000000  000000  000000
EA2  0006000      A5  000000  000000  000000  000000  S5  000000  000000  000000  000000
EA3  0006000      A6  000000  000000  000000  000000  S6  000000  000000  000000  000000
EA4  0006000      A7  000000  000000  000000  000000  S7  000000  000000  000000  000000

CN 001      MODES 016  BR RST EBM     IM 03403200  II  III FII III III IFI III III
VL 200                 DM MCR SDM                  RU  FOP EBC MRI IPD MNA XNU ODN
                       D1 OEI LM                   PM  PRR XPM CTP OCL IXM IXX VVV

STATUS 00 XNU ODN 002 SS **V FWB    IF 00003000 RM FOP EBM MRI IPD MNA XNU ODN
          IXN VVV     BB **N PSM                PE PRR EPE CTC OCL IEM IXN VIV
          SSS SSS     MU **U S L                EU EEE XIC UIP II  IXI   F FVI

LAT0 RWXC 16 RWXD 16 PB 0000000140000 LB 00000000000000 LL 00000000340000
LAT1 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT2 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT3 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT4 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT5 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT6 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
LAT7 RWXC 00 RWXD 00 PB 0000000000000 LB 00000000000000 LL 00000000000000
```

Figure 44. Intolerable Interrupt with No Handler in the Controller or Control Point

Exchange from within the Standard Code (Controller or Control Point)

When an interrupt occurs while the controller or control point standard code is executing, the CPU exchanges using tEXP. The CPU executes the code at iTRAPDC, which writes A0 (always 3) at the memory location idleSTAT+CPU and writes A2 (address of the current exchange package) at trapADDR; then, the CPU hangs in a loop. After the exchange, tEXP contains the exchange package that was running when the exchange in the standard code occurred.

For an exchange within the standard code, the MME base window shows that CPU 0 has a controller error code of TRP [invalid exchange (trap)]. The ERROR runtime information display for the controller indicates that the controller had an invalid exchange. The P register in the tEXP table of the EXCHANGE runtime information display for the controller shows the code that was executing when the exchange occurred. View the exchange package at the P register value to verify the interrupt flag. View the trapSTAT and idleSTAT locations to verify the values.

Exchange with an Invalid Exchange Address Handler

An invalid exchange occurs when the CPU exchanges to the wrong exchange package and that exchange package is not an exchange package for some other control point (tEXP, dEXP or sEXP). When this happens, the CPU exchanges to iTRAPXA, which traps the CPU.

For an invalid exchange, the MME base window indicates that the controller has some trap condition. The ERROR runtime information display for the controller shows what caused the trap condition and indicates that you should view the EXCHANGE runtime information display for the controller. The EXCHANGE runtime information display shows that an invalid exchange took place at 3700_8. The P register value for this exchange package indicates which code the CPU was executing when the exchange took place.

Click on Halt

Click on Halt ▷ to issue the Halt –> Register Dump option, which is the only halt option that is available in environment 2. The following sequence of events occurs for each executing control point:

1. MME sets the A7 registers for the deadstart exchange package (DEXP) and the starting exchange package (SEXP) for the control point to the CPU number of the control point:

   DEXP A7 = CPU number
   SEXP A7 = CPU number

2. MME sets the SEXP exchange address to its original value plus the CPU number (in A7) multiplied by 40_8 [SEXP XA = XA + (A7 * 40_8)]. This makes the XA point to the WEXP for the CPU.

   MME also sets exit address 0 (EA0) through EA4 to the original value plus A7 multiplied by 40_8:

   EA0 = EA0 + (A7 * 40_8)
   EA1 = EA1 + (A7 * 40_8)
   EA2 = EA2 + (A7 * 40_8)
   EA3 = EA3 + (A7 * 40_8)
   EA4 = EA4 + (A7 * 40_8)

3. MME copies SEXP to FEXP.

4. MME writes the data to mainframe addresses 0_8 through 200_8, which contain the first four exchange packages for the controller.

5. MME writes the TEXP for the current CPU.

6. MME writes a halt command in the controller communications port. Refer again to Table 14 for more information about the HALT command.

7. MME waits for the command to clear.

8. The CPU stops executing control point code.