Solbourne and Series4/600, and Series4/500 are trademarks of Solbourne Computer, Inc.

Part Number: 101686-AC

October 1989

Copyright © 1989 by Solbourne Computer, Inc. All rights reserved. No part of this publication may be reproduced, stored in any media or in any type of retrieval system, transmitted in any form (e.g., electronic, mechanical, photocopying, recording) or translated into any language or computer language without the prior written permission of Solbourne Computer, Inc., 1900 Pike Road, Longmont, Colorado 80501. There is no right to reverse engineer, decompile, or disassemble the information contained herein or in the accompanying software.

Solbourne Computer, Inc. reserves the right to revise this publication and to make changes from time to time without obligation to notify any person of such revisions or changes.
Preface

This manual describes mdg, the Solbourne Computer, Inc., standalone multiprocessor test controller for the Solbourne systems. This manual contains four sections, as follows:

Section 1 - Introduction
   This section introduces the Bootable/Standalone Multiprocessor Diagnostics program mdg.

Section 2 - Getting Started with mdg
   This section explains how to begin using mdg.

Section 3 - mdg Tests
   This section presents the tests currently available using mdg.

Section 4 - Commands
   This section gives the user commands available when using mdg.
# Table of Contents

Section 1: Introduction ........................................................................................................ 1-1
  1.1 Introduction ................................................................................................................. 1-1
  1.2 Related Documentation ............................................................................................ 1-1

Section 2: Getting Started with mdg ............................................................................... 2-1
  2.1 Introduction .................................................................................................................. 2-1
  2.2 Invoking mdg ................................................................................................................. 2-1
  2.3 The Prompt ................................................................................................................... 2-2
  2.4 Entering Commands .................................................................................................... 2-2
  2.5 Using mdg Commands ............................................................................................... 2-3
    2.5.1 Test Control Commands ...................................................................................... 2-3
  2.6 Selecting Processors for Testing ................................................................................ 2-4
  2.7 Starting Test Execution .............................................................................................. 2-5
    2.7.1 Variations of Test Execution ................................................................................ 2-6
  2.8 Handling Test Failures ............................................................................................... 2-6
  2.9 Removing and Adding Processors .............................................................................. 2-7
  2.10 Exiting mdg ................................................................................................................. 2-8

Section 3: mdg Tests ........................................................................................................... 3-1
  3.1 Introduction .................................................................................................................. 3-1
  3.2 Test 01: Atomic Load-Store Test ................................................................................. 3-1
  3.3 Test 02: Memory Data RAM Test ............................................................................... 3-2
  3.4 Test 03: Shared-Memory Pattern Test ......................................................................... 3-2
  3.5 Test 04: Cache Block Alias Test ................................................................................ 3-3
  3.6 Test 05: Floating Point Store Test .............................................................................. 3-4
  3.7 Test 06: Cache Data Request Test .............................................................................. 3-4
  3.8 Test 07: Cache Data Bus Pattern Test ....................................................................... 3-5
  3.9 Test 08: Interrupt Test .................................................................................................. 3-6

Section 4: Commands ......................................................................................................... 4-1
  4.1 Introduction .................................................................................................................. 4-1
Section 1: Introduction

1.1 Introduction

mdg is a standalone test controller for use on Solbourne multiprocessor systems. This program is used by design, manufacturing, and field engineering personnel to help in determining defective boards and in diagnosing these failures. The intended primary user of this program is the manufacturing organization.

The software for mdg includes:

- the mdg standalone test controller (mdg(1))
- test control commands
- mdg tests

1.2 Related Documentation

Information that may be useful while using the mdg program is available in the following documentation:

- Series4/600 Service Manual, Part number 101249-AA
- Series4/600 Theory Manual, Part number 101250-AA
- Series4/500 Service Manual, Part number 102161-AA
- Extended ROM Resident Diagnostics Manual, Part number 101489-AB
- System Power On Self Test Manual, Part number 101486-AB
Section 2: Getting Started with mdg

2.1 Introduction

This section gives step-by-step instructions and examples for getting started using mdg.

In this section, commands you enter are given in boldface type. Command parameters for which you substitute a value are given in italic.

2.2 Invoking mdg

The steps to follow the first time mdg is invoked are given below.

The user must first bring the Solbourne system to the ROM> prompt. If UNIX is running, it must be shutdown using the halt (1) command.

1. At the ROM> prompt, type:
   \[ \text{ROM} > \text{mdg} \]

2. When mdg starts up, the following message is displayed:

   \[
   \text{MDG - Multiprocessor Diagnostic Test Controller}
   \text{Version 1.1 September 25, 1989}
   \text{Copyright (c) 1989 Solbourne Computer, Inc.}
   \]

3. As mdg starts up, the following steps are undertaken by the MASTER processor:
   - Obtain the number of processors in the system and the results of power-up diagnostics from the diagnostic RAM.
   - Calculate the system-wide (shared memory) and CPU-specific (private memory) test limits.
   - Configure the memory configuration table with the number of memory boards in the system as well as their addressing range.
   - Configure the frame buffer configuration table with the values found during power-up.
   - Initialize the VMEbus configuration table as empty.
   - Awake each SLAVE processor in the system that passed the power-up diagnostics. Each SLAVE processor will register with the MASTER processor in order for the MASTER to include it as part of the selected list of available system processors that mdg maintains.
   - By default, all available tests are selected and all the available processors are included for testing.

4. Upon completion of the previous setup, mdg will display the following message:
CPU Configuration:

2 CPU boards:

<table>
<thead>
<tr>
<th>Slot#</th>
<th>Power-Up-State</th>
<th>Selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>M 5</td>
<td>PASS</td>
<td>YES</td>
</tr>
<tr>
<td>6</td>
<td>PASS</td>
<td>YES</td>
</tr>
</tbody>
</table>

In this example, mdg found two processors in the system, both passed power-up diagnostics, and as a result both were selected for inclusion in the list of available processors. In the case of a processor failing power-up diagnostics, mdg will not include it as one of the SELECTED processors. However, mdg provides to the user the capability to attempt to include a processor that failed power-up diagnostics at any time.

2.3 The Prompt

The mdg prompt follows the following format:

```
{ CPUs not included during test / CPUs included during test } <Pass limit> =>
```

For example: In a system with two processors (in slots 5 and 6), with only the processor in slot 6 to be included during testing, and the pass limit set to 1 the prompt to be displayed will be as follows:

```
{ 5/6 } <1> =>
```

2.4 Entering Commands

mdg commands and parameters are case insensitive and mdg accepts input only when the prompt is displayed.

The rules for entering commands include:

- In general more than one command can be entered in a single command line to the prompt at the same time.

```
{ /5 6 } <1> => tests 1 2 3 names on passlim 0 between 5 run
```

The above command line selects tests 1, 2, and 3, turns the printing of test names on, sets the pass limit to 0 (no passlim), the between count is set to 5, and begins test execution with the run command.

- Commands that process user input in an interactive mode, such as the vmeconf(1) and fbconfig(1), cause commands that follow on the command line to be ignored.

- Commands must be separated by white space(s), including tabs or spaces. (Semicolons are not recognized by mdg as spaces.)
• If any of the command(s) entered return an error condition, all following commands are ignored and the prompt is redisplayed.

• If a command is unrecognized by mdg, the following is displayed:

  Unknown command (command name)

• All command lines are terminated by a Return.

• Some commands may display additional error messages if numeric values are entered incorrectly or if the numeric values are not legal. These messages identify the value that is out of range, for example

  illegal address (value given)

If an illegal value is given, additional information may be displayed that identifies the legal range of values.

• Memory and I/O addresses and contents must be entered in hexadecimal format. Any value that has to do with hardware must also be entered in hexadecimal (e.g., register data, memory address, or memory data).

• Counters and test numbers should be entered in decimal format (e.g., counts and limits).

• The mdg help (1) command can be used any time the prompt is displayed. A summary of the command given as an argument to help will be displayed.

2.5 Using mdg Commands

Example usage of each mdg command is given in Section 4 of this manual. All commands can be used with any other commands. All the mdg commands are for test control.

The test control commands are commands so categorized because they cause execution or alter the execution of the test programs.

2.5.1 Test Control Commands

The test control commands allow users to control tests run by the mdg debugger. The command names and their functions follow:

• between (1) - Set or display between count
• config (1) - Display system processor configuration
• continue (1) - Set or display continue on error flag
• cpus (1) - Select or display processors included in tests
• cpu lim (1) - Display or set processor specific memory test limits
• deposit (1) - Deposit data at specified address
• err lim (1) - Set or display error limit
• errors (1) - Display error count
• examine (1) - Examine contents of memory
• fbconfig (1) - Generates or display the frame buffer configuration
• halt (1) - Remove processors from mdg environment
2.6 Selecting Processors for Testing

By default, when mdg is started all the processors that passed power-up diagnostics are selected for testing. Processors are selected/deselected for testing with the cpus (1) command. If the cpus command is entered without an argument, all the selected processors are displayed. For example:

```
{ 5/6 } <1> => cpus
  selected cpus:
    6
{ 5/6 } <1> =>
```

The processor selection can be modified at any time the prompt is displayed. For example:
Processors that failed power-up diagnostics, are not automatically included for testing, however by using the `wake(1)` command, it may be possible to include processors that failed. The ability for a processor to start `mdg` depends heavily on the type of failure it had during power-up. In that case, where a slave processor that is requested to start `mdg`, is unable to do so, the master processor will timeout after a given time period.

For additional information on processor selection, see the `cpus` and `wake` commands in section 4.

### 2.7 Starting Test Execution

When `mdg` is initially started, all the tests are selected. Tests are executed when the `run(1)` command is entered at the command line. For example:

```plaintext
{ /5 6 } <1> => tests run
```

If the `tests` command is entered without an argument, all the selected tests are displayed. For example:

```plaintext
{ /5 6 } <1> => tests
selected tests:
  1  2  3
```

The test selection can be modified at any time the prompt is displayed. For example:

```plaintext
{ /5 6 } <1> => tests 1
{ /5 6 } <1> => tests
selected tests:
  1
{ /5 6 } <1> =>
```

The `menu` command identifies the test names or their functions. For example
Menu of installed test programs:
Test 01: Atomic Load-Store Test
Test 02: Memory Data RAM Test
Test 03: Shared-Memory Pattern Test
Test 04: Cache Block Alias Test
Test 05: Floating Point Store Test
Test 06: Cache Data Request Test
Test 07: Cache Data Bus Pattern Test
Test 08: Interrupt Test

For additional information on test execution, see the tests (1), run (1), and menu (1) commands in Section 4.

2.7.1 Variations of Test Execution

This section discusses some of the basic variations that can be applied to test commands. There are other variations than those given here.

Two results can occur during test execution. The test can pass or the test can fail.

If the test passes, the user can do any of the following:
- Tell the controller how many iterations to run using the passlim (1) command
- Controls whether the test names are printed using the names (1) command
- The user can also stop test execution at any time by entering a Control-C (\(\text{Ctrl-C}\))

Because of the difficulty controlling multi-asynchronous CPUs, mdg tests restart from the beginning when the loop (1) command is used. Therefore, even though the loop command is supported, it is not practical for setting up a scope loop. Instead, it is recommended that a logic analyzer be used for evaluating failures.

If the test fails, the user can do any of the following:
- If required, remove other processors from the test.
- Skip to the next test in the selected sequence of tests using the next (1) command
- Restart the entire sequence using the restart (1) command.

2.8 Handling Test Failures

Several of the commands given in Section 4 that are used for test control can be used when test failures occur. In the following example, test 3 detects a failure and the loop and quiet commands are used to set up a scope loop.
Note that test 3 has displayed its error message which identified the failing test case and returned to the prompt. If the user wishes to evaluate this test failure by setting up a logic analyzer, the sequence of commands shown in the following illustration may be entered.

Note that test 3 has been re-executed and has redisplayed the same error message. This suggests the presence of a solid failure. To speed up the loop and avoid having to reenter the run command, the sequence of commands in the following illustration may be entered.

A Control-C must be entered to halt the program and return to the prompt.
2.9 Removing and Adding Processors

It may be desirable in many occasions to completely remove a processor from mdg. The halt (1) command provides a mechanism to do this. The effect of the halt command is to put the target processor in an idle loop at the ROM level.

☆ ☆ ☆ NOTE ☆ ☆ ☆

If the target processor happens to be the master processor, it will first tell all of the available processors to exit mdg, and then it will exit from mdg itself.

In the other hand, the wake command provides the user with the ability to tell a processor, which is idle at the ROM level, to start mdg as a slave processor and be part of the mdg environment.

See the config (1), halt, and wake commands in Section 4 for additional information.

2.10 Exiting mdg

To exit mdg use the quit (1) command.
Section 3: mdg Tests

3.1 Introduction

This section explains the functionality of the tests shipped by Solbourne Computer for use with the mdg (1) debugger. These tests include:

1. Atomic Load-Store Test
2. Memory Data RAM Test
3. Shared-Memory Pattern Test
4. Cache Block Alias Test
5. Floating Point Store Test
6. Cache Data Request Test
7. Cache Data Bus Pattern Test
8. Interrupt Test

☆ ☆ ☆ NOTE ☆ ☆ ☆
Error messages from one test are not valid, if failures have occurred during previous tests. The errors from a test must be corrected before advancing to the next test.

3.2 Test 01: Atomic Load-Store Test

This test verifies the logic related to atomic load-store unsigned byte instructions by having all processors involved in the test attempt to access and own a resource (byte) in memory for a predetermined number of iterations.

In a multiprocessor system, two or more processors executing atomic load-store instructions addressing the same byte simultaneously are guaranteed to execute them in some serial order.

Each processor attempts to lock this resource and assign a unique ownership identification (its slot number) to the locked resource. Upon successfully locking the given byte, the processor will assign its unique identification to this lock. Other processors should not be able to lock this byte until it has been unlocked by this processor.

If after attempting to lock this byte, a processor finds that it is its owner but the identification pattern is not its pattern, an error condition is detected and reported. For example:
Upon test initialization, the master processor ensures that all involved processors register in order to proceed with the test. If for some reason a processor is unable to register to the master processor, the master reports this as an error condition as follows:

<table>
<thead>
<tr>
<th>Processor does not register</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor = 5</td>
</tr>
</tbody>
</table>

If the prompt flag for this test is set, the test prints out messages indicating the status of the test (however this may slow down execution).

### 3.3 Test 02: Memory Data RAM Test

This test is similar to the Memory Data RAM Test in mdg except that each installed cpu card accesses only a portion of each memory block. The test performs a movin inverse test algorithm but “shares” each tested memory block with all other installed processors.

An example of a Data RAM Test failure follows:

<table>
<thead>
<tr>
<th>Error occurred in data RAM memory test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error code = 0xe0 Virtual addr = 0x008000000 Physical addr = 0x00ea0000 Board slot = 2</td>
</tr>
<tr>
<td>A data failure was found in the second read on the reverse pass</td>
</tr>
<tr>
<td>exp = 0x55555555 act = 0x55555555d xor = 0x00000008</td>
</tr>
</tbody>
</table>

### 3.4 Test 03: Shared-Memory Pattern Test

This test verifies basic cache consistency by having all processors involved in the test work in sequence during access to the same block of memory in FF space.

During test initialization, the master processor assigns each processor a unique pattern that each processor writes to memory after waiting for another processor to write its pattern. In this manner, the entire block of memory is addressed and results in an environment upon where each processor is constantly validating/invalidating its cache.

If after a predetermined number of retries, a processor fails to match the expected pattern from memory, an error condition is detected and reported. For example:
Upon test initialization, the master processor insures that all involved processors register in order to proceed with the test. If for some reason a processor is unable to register to the master processor, the master reports this as an error condition as follows:

**Processor does not register**

**Processor = 5**

If the prompt flag for this test is set, the test prints out messages indicating the status of the test (however this may slow down execution).

### 3.5 Test 04: Cache Block Alias Test

This test verifies that the cache tags will reference the correct entry in the cache rams. It verifies that a reference to the same physical location through different virtual addresses works differently.

In a multiprocessor environment, this test performs a series of memory page write and reads in which all physical page addresses from \texttt{XXXX000Y} to \texttt{XXXXe00Y} (hex) are written and read using all combinations of virtual page addresses from \texttt{XXXX000Y} to \texttt{XXXXe00Y} including FF space addresses. Each processor involved in this test will have a different starting base address from the other processors (Y).

**For Series4**

A write to a physical block using a virtual block address creates a unique physical-to-virtual mapping within the MMU. When the physical block is accessed using a different virtual address, the MMU must break the existing physical-to-virtual mapping, write the data block to its physical location in memory, and re-read it into the cache at the new virtual index. This creates the new physical-to-virtual mapping.

This test insures that the MMU logic which detects the purge condition is operational and that the data (unique for each processor and for each physical page) is correctly transferred between the cache and the memory system among all the processors.

**For Series5**

Since the Series5 processor does not have a virtual/physical cache this test simply exercises the TLB and cache.

Possible error message:
Data error at vaddr 0x8e000, paddr 0xff08e000
exp = 0x058e000
act = 0x0a21000

Upon test initialization, the master processor insures that all involved processors register in order to proceed with the test. If for some reason a processor is unable to register to the master processor, the master will report this as an error condition as follows:

Processor does not register
Processor = 5

If the prompt flag for this test is set, the test prints out status messages that indicate which physical and logical addresses are being used (however this slows down execution).

3.6 Test 05: Floating Point Store Test
This test is executed on all installed processors and is designed to exercise Kbus cache consistency protocols when each processor's floating point unit is busy doing store operations to its cache.
Each active processor is given a unique 64 Kbyte region of memory by the master processor. This 64 Kbyte region is then divided into two 32 Kbyte regions by each processor. Each processor then tags the first 32 Kbyte region with the floating point representation of its BID pattern and the integer representation of its BID pattern into the second 32 Kbyte region. Each processor, then begins a loop in which it alternates between writing its own memory regions and verifying the two regions for all other processors.
An example error message is show below:

Data error in non-FP store region of CPU in slot 2
FP store region = 0xff800000:0xff807fff
non-FP store region = 0xff808000:0xff80ffff
failing address = 0xff807002
exp = 0x22222222
act = 0x20222222

3.7 Test 06: Cache Data Request Test
This test is executed on all installed processors and is designed to exercise each processors ability to supply data in response to a Kbus data request (cacheable read) while busily performing cache/memory operations.
Each processor begins by initializing memory with a sequence of patterns. 32-byte memory blocks are allocated to each processor (modulo the number of processors) with each memory
block containing a unique data pattern for the processor it is allocated to. The data pattern consists of an address tag in words 0, 2, 4 and 6 of the memory block and the BID of the processors which owns the memory block distributed across each nibble of words 1 and 5. Words 3 and 7 are initialized with the complement of the pattern in words 1 and 5.

After memory has been initialized by all the processors, each processor gets synchronized with the other processors and begins to read and check each the contents of the other processor's caches for the correct data. This creates the desired kbus data request traffic.

An example error message is show below:

<table>
<thead>
<tr>
<th>Data error detected by cpu X</th>
</tr>
</thead>
<tbody>
<tr>
<td>block address = 0xbbbbbbb</td>
</tr>
<tr>
<td>word address = 0xwwwwwwww</td>
</tr>
<tr>
<td>exp = 0xeeeeeeee</td>
</tr>
<tr>
<td>act = 0xaaaaaaaa</td>
</tr>
<tr>
<td>block contents:</td>
</tr>
<tr>
<td>11111111 22222222</td>
</tr>
<tr>
<td>33333333 44444444</td>
</tr>
<tr>
<td>55555555 66666666</td>
</tr>
<tr>
<td>77777777 88888888</td>
</tr>
<tr>
<td>Data belonged to cpu Y</td>
</tr>
</tbody>
</table>

3.8 Test 07: Cache Data Bus Pattern Test

This test is similar to the Cache Data Request Test except that when each processor reads data from another processors cache, a dirty cache block must first be flushed from the cache of the processor initiating the data request.

Each processor begins by initializing its allocated memory segment (64 Kbytes for Series4, 128 Kbytes for Series5) with alternating walking ones and walking zeroes patterns in successive cache lines. The memory segments allocated for each processor are equal in size to the cache size and segments are contiguous within the physical address space. This is done so that blocks within the cache of one processor must be flushed out to memory when the corresponding block within the cache of another processor is read.

After memory has been initialized by all the processors, each processor gets synchronized with the other processors and reads the contents of the other processor's caches. The data read is not checked. The read operations cause data within the processor performing the read to be flushed out to memory. When all blocks from the other processors cache have been read, ECC is enabled and the original data which was flushed out to memory is re-read and checked to be correct.

If the test fails, one of the following error messages will be displayed:

<table>
<thead>
<tr>
<th>Data fault occurred accessing block at address 0xaaaaaaaaa</th>
</tr>
</thead>
<tbody>
<tr>
<td>FVAR = 0xbbbbbbb</td>
</tr>
</tbody>
</table>

This indicates that a data fault exception occurred when the cpu accessed address "0xaaaaaaaaa"
ECCS fault occurred accessing block at address 0xaaaaaaaa
FPAR = 0xbbbbbbbb
FES = 0xcc

This indicates that an ECC single bit exception occurred when the cpu accessed the cache block at address "0xaaaaaaaa"

Data error occurred at address 0xaaaaaaaa
Pass N
exp = 0xeeeeeeee
act = 0xaaaaaaaa
block contents:
11111111 22222222
33333333 44444444
55555555 66666666
77777777 88888888

This indicates that the cpu read incorrect data at the specified address. "Pass" indicates how many repetitions were completed when the error occurred.

3.9 Test 08: Interrupt Test

This test verifies that each processor can send directed interrupts to all other processors, and that each processor receives an interrupt from all others.

For Series5, the global interrupt capability is verified in the same manner as for directed interrupts.

Each processor starts by getting synchronized with all other processors, then all processors simultaneously begin sending interrupts to another processor. 10,000 iterations of the test are performed.

The following error may occur:

Never received directed interrupt from cpu in slot X
passes completed = YY

This indicates that all processors finished sending directed interrupts but one processor failed to receive it.

If executed on a Series5 machine, the following error could also occur:
| Never received global interrupt from cpu in slot X |
| passes completed = YY |

This indicates that all processors finished sending global interrupts but one processor failed to receive it.
4.1 Introduction

This section offers printed copies of man pages for all commands associated with mdg(1). The commands are presented in the UNIX man page reference format.

A summary of command usage is displayed on-line when mdg is running by typing:

\{ /5 6 \} <1> => ? .

The following is a listing of the mdg commands available in this section:

- between (1)
- config (1)
- continue (1)
- cpus (1)
- cpulim (1)
- deposit (1)
- errlim (1)
- errors (1)
- examine (1)
- fbconfig (1)
- halt (1)
- help (1)
- limit (1)
- loop (1)
- master (1)
- mdg (1)
- memconfig (1)
- menu (1)
- names (1)
- next (1)
- passes (1)
- passlim (1)
- prompt (1)
- quiet (1)
- quit (1)
- restart (1)
- run (1)
- status (1)
- time (1)
- tests (1)
- vmeconf (1)
- wake (1)
NAME
between - Set or display between count

SYNOPSIS
between [ count ]

DESCRIPTION
between sets or displays the current setting of the between count. between suppresses printing test completed messages to the screen until count passes have completed. When the status (1) reset command is used, the between count is reset to 1.

OPTION
count Specifies the number of test passes that must be completed before a completion message is displayed. By default the between count is always set to 1.

EXAMPLE
User input in the example is shown in boldface type.
The following example illustrates how to set and redisplay the between count.

{ /5 6 } <1> => between 4
{ /5 6 } <1> => between
   Between count = 4
{ /5 6 } <1> =>

SEE ALSO
mdg (1), passlim (1), status (1)
NAME
continue - Set or display continue on error flag

SYNOPSIS
continue [ on | off ]

DESCRIPTION
continue sets or displays the continue-on-error flag. If no parameters are specified, continue displays the current setting of the continue-on-error flag.
The continue flag commands tests to continue executing after a test failure occurs. Tests are designed to check the continue flag to determine if test execution should be halted (the default condition) or if the next test case should be executed.

OPTIONS
on         Turns on the continue-on-error flag.
off        Turns off the continue-on-error flag.

EXAMPLES
User input in the examples is shown in boldface type.
The following example causes the current error message enable flag to be displayed.

    { /5 6 } <1> => continue
    continue = off
    { /5 6 } <1> =>

The following example illustrates how the continue flag is changed and redisplayed.

    { /5 6 } <1> => continue on
    { /5 6 } <1> => continue
        continue = on
    { /5 6 } <1> =>

SEE ALSO
mdg (1), status (1)
NAME
cpus - Select or display processors included in testing

SYNOPSIS
cpus [ all | cpu ... | cpus cpu ... ]

DESCRIPTION
cpus select the processors that are to be tested by the selected tests. By default, all pro­cessors are selected for testing when the program is initialized.

Single processors or a range of processors may be selected by specifying the processor numbers or range of processors number.

OPTIONS
all Select all the available processors. all can be specified at any time to reselect all processors.
cpu Select specified cpu. If cpu is not specified, the cpus command displays the current processor selection.

EXAMPLES
User input in the examples is shown in boldface type.
The following example illustrates how to display the processor selection.

{ /5 6 } <1> => cpus
selected cpus:
  5  6
{ /5 6 } <1> =>

In the following example, processor 5 is selected and then displayed.

{ /5 6 } <1> => cpus 5
{ 6/5 } <1> => cpus
selected cpus:
  5
{ 6/5 } <1> =>

In the following example, processors 6 and 5 are selected and displayed. Note that pro­cessors may be selected in any order.

{ 6/5 } <1> => cpus 6:5
{ /5 6 } <1> => cpus
selected cpus:
  6  5
{ /5 6 } <1> =>

In the following example, all processors are selected and displayed.

{ /5 6 } <1> => cpus all
{ /5 6 } <1> => cpus
selected cpus:
  5  6
{ /5 6 } <1> =>
SEE ALSO

mdg(1)
NAME
cpulim - Display or set processor specific memory test limits

SYNOPSIS
cpulim [ cpu | [ low high | reset ] ]

DESCRIPTION
cpulim displays or sets the processor specific (private) memory test limits. By default, cpulim displays all memory limits for all of the processors in the system.
cpulim is calculated using the amount of free memory and the number of processors in the system.
Some test programs examine the private memory limits to determine how much memory to test.

OPTION
reset Resets the limits back to the default settings. The default settings are determined by the amount of free memory and the number of processors in the system.
low high low is the first address and high is the last address to test, inclusive.

EXAMPLE
User input in the example is shown in boldface type.
The following example displays the current limit settings for all the processors in the system.

```bash
{ /5 6 } <1> => cpulim
CPU Specific Memory limits:
Slot#   LOW     HIGH
   5   df000   4a3fff
   6   4c4000  82bfff
```
The following example resets the memory limit to their default values.

```bash
{ /5 6 } <1> => cpulim reset
{ /5 6 } <1> => cpulim
CPU Specific Memory limits:
Slot#   LOW     HIGH
   5   dc000   4a3fff
   6   4a4000  86bfff
```
The following example sets the memory limit for processor 5 to the range ef000 through 400000 hex, inclusive.

```bash
{ /5 6 } <1> => cpulim 5 ef000 400000
{ /5 6 } <1> => cpulim 5
CPU Specific Memory limits:
Slot#   LOW     HIGH
   5   ef000   400000
```
SEE ALSO
config(1), limit(1), mdg(1)
NAME
    deposit - Deposit data at specified address

SYNOPSIS
    deposit [-b | h | w] [addr_range] = value

DESCRIPTION
    deposit writes data to an address or range of addresses.

OPTIONS
    [-b | h | w]
    Specifies the width of the data to be examined.
    -b - byte (8 bits)
    -h - half word (16 bits)
    -w - word (32 bits)
    If the width is not specified, a width of -b (1 byte) is assumed.

addr_range
    One of the following forms:
    addr - the location addr
    addr #count - count locations starting from addr
    addr1 addr2 - all locations from addr1 to addr2.

=value    Value to be written to the specified address.

EXAMPLES
    The following example writes 32 bits of data (zero) to address ff000000 hex.

    { /5 6 } <1> => deposit -w 0xff000000=0
    { /5 6 } <1> =>

SEE ALSO
    mdg(1), examine(1)
NAME
errlim - Set or display error limit

SYNOPSIS
errlim [ limit ]

DESCRIPTION
errlim sets or displays the current setting of the test error limit.

OPTION
limit Specifies the number of test errors that can occur before test execution is halted. By default, the limit is set to zero (no error limit). However, the error limit may be changed by specifying a new limit value. The limit value must be entered in unsigned decimal format and be between 0 and 2,147,483,647, inclusive.

EXAMPLES
User input in the examples is shown in boldface type.
The following example illustrates how to display the current error limit.
{ /5 6 } <1> => errlim
   Error limit = 0
{ /5 6 } <1> =>
The following example illustrates how to change and re-display the error limit.
{ /5 6 } <1> => errlim 100
{ /5 6 } <1> => errlim
   Error limit = 100
{ /5 6 } <1> =>

SEE ALSO
errors (1), mdg (1), status (1)
NAME
errors - Display error count

SYNOPSIS
errors

DESCRIPTION
errors displays the number of test errors that have occurred since the last run(1) command.

EXAMPLE
User input in the example is shown in boldface type.
The following example illustrates how to display the error count.

{ /5 6 } <1> => errors
   Total test errors = 0
{ /5 6 } <1> =>

SEE ALSO
errlim(1), mdg(1), status(1)
NAME
examine - Examine contents of memory

SYNOPSIS
examine [ -b | h | w ] [ addr_range ]

DESCRIPTION
examine reads data from the specified address or addresses.

OPTIONS
[ -b | h | w ]
  Specifies the width of the data to be examined.
  -b - byte (8 bits)
  -h - half word (16 bits)
  -w - word (32 bits)

addr_range
  One of the following forms:
  addr - the location addr
  addr #count - count locations starting from addr
  addr1 addr2 - all locations from addr1 to addr2.
  If range is not specified, the address range used on the previous examine command is used.

EXAMPLES
The following example shows how to examine a byte from location 17000000 hex.

{ /5 6 } <1> => examine -b 0x17000000
(0x17000000): 0x3d
{ /5 6 } <1> =>

SEE ALSO
mdg(1), deposit(1)
NAME
  fbconfig - displays the frame buffer configuration file

SYNOPSIS
  fbconfig

DESCRIPTION
  The frame buffer configuration is read from the diagnostic RAM when MDG is invoked.
  The board must be configured in descending slot order.

EXAMPLE
  User input in the example is shown in boldface type.

    { /5 6 } <1> => fbconfig

    Frame Buffer Configuration:

      1 graphics board(s):
      Slot  IO address  Board Type  Resolution
         1      a1000000    monochrome    low
    Slot number of default board to test: 1
    { /5 6 } <1> =>

SEE ALSO
  mdg(1)
NAME
halt - Remove processors from mdg environment

SYNOPSIS
halt [ all | cpu ... | cpu:cpu ... ]

DESCRIPTION
halt removes the specified processors from the mdg environment. The effect of removing
a processor from mdg is to put the specified processor in an idle loop at the ROM level,
thus exiting from mdg.

If the specified processor is the master processor, the master processor will first tell all of
the available processors in the system to exit mdg, and then it will exit mdg itself.

OPTIONS
all Halt all the available processors. all can be specified at any time to halt all pro­
cessors.
cpu Halt specified cpu.

EXAMPLES
User input in the examples is shown in boldface type.

In the following example, processor 6 is halted and removed from mdg.

{ /5 6 } <1> => halt 6
{ /5 } <1> =>

In the following example, processors 6 and 5 are both halted, thus in effect both proces­
sors exiting from mdg. Note that processors may be selected in any order.

{ 6/5 } <1> => halt 6:5

In the following example, all processors are halted. This results in both processors exit­
ing from mdg.

{ /5 6 } <1> => halt all

SEE ALSO
config(1), mdg(1), wake(1)
NAME
help - Display command list or information on a specific command

SYNOPSIS
help [command ...]

DESCRIPTION
The help command with no arguments causes a list of command and command usages to be displayed. This is equivalent to the? command.
The help command with an argument causes the command usage for the specified command to be displayed.

OPTIONS
command
name of command for which help is desired.

EXAMPLE
The following example causes the command usage for the tests command to be displayed:

{ /5 6 }<1> => help tests
   Usage: tests [ all | test ... | test:test ... ]
{ /5 6 }<1> =>

SEE ALSO
mdg(1)
NAME  
limit - Display or set memory test limits

SYNOPSIS  
limit [ reset | memname [ low high | reset ] ]

DESCRIPTION  
limit displays or sets the memory test limits of the system. By default, limit displays all  
the memory limits.

limit is set to the amount of installed memory for each memory devices in the system.  
Memory devices include physical shared memory, VMEbus address map memory, and  
VMEbus resident memory boards.

The test programs examine the memory limits to determine how much memory to test.

OPTION  
reset   Resets the limits back to the default settings. The default settings are deter­  
        mined by the amount of installed memory and the number of processors in the  
        system.

low high  
low is the first address and high is the last address to test, inclusive.

EXAMPLE  
User input in the example is shown in boldface type.

The following example displays the current limit settings for all the memory devices.

{ /5 6 } <1> => limit  
System Memory limits:  LOW      HIGH  
sysmem = 87c000  affffff  
vmemap = 20 7ff  
vmemem = 3 5ff

{ /5 6 } <1> =>

The following example resets the memory limits to their default values.

{ /5 6 } <1> => limit reset  
{ /5 6 } <1> => limit  
System Memory limits:  LOW      HIGH  
sysmem = 86e000  fffffff  
vmemap = 20 7ff  
vmemem = 0 7ffffff

{ /5 6 } <1> =>

The following example sets the memory limits for physical shared memory to the range  
86e000 through cffffff hex and set the VMEbus address map limits to 20 through ff hex,  
inclusive.
{ /5 6 } <1> => limit mem 86e000 cfffff
{ /5 6 } <1> => limit vmemap 20 ff
{ /5 6 } <1> => limit

System Memory limits: LOW    HIGH
               sysmem =  86e000  cfffff
             vmemap =       20    ff
            vmemem =         0   7ffff
{ /5 6 } <1> =>

The following example resets only the VMEbus address map limits to their default values. The physical shared memory values are not modified.

{ /5 6 } <1> => limit vmemap reset
{ /5 6 } <1> => limit

System Memory limits: LOW    HIGH
               sysmem =  86e000  cfffff
             vmemap =       20    7ff
            vmemem =         0   7ffff
{ /5 6 } <1> =>

SEE ALSO
  config(1), mdg(1)
NAME
loop - Set or display loop on test flag

SYNOPSIS
loop [ on | off ]

DESCRIPTION
loop sets or displays the loop on error flag. If no parameters are specified, loop displays the current setting of the loop flag.

The loop flag commands tests to loop on the failing test case in the event a test error occurs. Tests are designed to halt when errors occur so that the loop command may be entered.

OPTIONS
  on    Turns on the loop flag.
  off   Turns off the loop flag.

EXAMPLES
User input in the examples is shown in boldface type.
The following example causes the current loop flag do be displayed.

  { /5 6 } <1> => loop
  loop = off
  { /5 6 } <1> =>

The following example illustrates how the loop flag is changed and re-displayed.

  { /5 6 } <1> => loop on
  { /5 6 } <1> => loop
      loop = on
  { /5 6 } <1> =>

SEE ALSO
mdg(1), status(1)
NAME
master - Set or display master processor

SYNOPSIS
master [ cpu ]

DESCRIPTION
master sets or displays the current master cpu of the system. The master cpu is the processor that is responsible for controlling all of mdg. This processor is in charge of monitoring the other processors, as well as handling any requests for service initiated by the these.

This command should be used when it is desired to have a specific processor control the mdg environment.

OPTION
cpu Select specified cpu to be the master. If cpu is not specified, the master command displays the current master processor.

EXAMPLES
User input in the examples is shown in boldface type.
The following example illustrates how to display the current master processor.

{ /5 6 } <1> => master
     Master CPU = 5
{ /5 6 } <1> =>

The following example illustrates how to change and re-display the master processor.

{ /5 6 } <1> => master 6
{ /5 6 } <1> => master
     Master CPU = 6
{ /5 6 } <1> =>

SEE ALSO
mdg(1)
NAME

mdg - description of the standalone multiprocessor diagnostic test controller

SYNOPSIS

mdg

DESCRIPTION

mdg is a standalone multiprocessor test controller. The test controller provides the commands necessary to randomly select and execute any of the available test programs on any or all of the processors in the system. The operator has control over test execution and can command test programs to loop on error or repeat execution indefinitely.

The following is a list of the mdg commands with the shortest possible abbreviation in capital letters. Command names and abbreviations are not case sensitive.

The acceptable commands follow (bold, uppercase letters represent the abbreviated usage of the command name):

? Display summary of mdg commands
between Set or display between count
config Display system processor configuration
continue Set or display continue on error flag
cpus Select or display processors included in test
cpulim Set or display processor specific memory test limits
deposit Deposit data at specified address
errlim Set or display error limit
errors Display error count
examine Examine contents of memory
fbconfig Displays the frame buffer configuration
halt Remove processor from mdg environment
help Display summary of mdg commands
limit Display or set system memory test limits
loop Set or display loop on test flag
master Set or display master processor
memconfig Display system memory configuration
menu Display listing of the available tests
names Enable or disable printing of test names during test execution
next Execute next selected test
passes Display pass count
passlim Set or display pass limit
prompt Set or display prompt flags
quiet  Set or display error message enable flag
quit   Exit from mdg program
restart Restart execution of selected tests
run    Start execution of selected tests
status Display or reset state of modes, flags and counts
tests  Select or display tests to be executed
time   Set or display print time flag and display current date and time
vmeconf Configure VMEbus devices
wake   Add processor to mdg environment
NAME  
memconfig - Display memory configuration file

SYNOPSIS  
memconfig

DESCRIPTION  
memconfig displays the memory configuration. When MDG is invoked it creates a memory configuration table based on the memory configuration information saved in the diagnostic RAM during the power-up self-tests.

EXAMPLES  
User input in the examples is shown in boldface type.
In the following example, memconfig is entered at the prompt. The contents of the memory configuration table is displayed.

{ /5 6 } <1> => memconfig  
Memory Configuration:

2 boards totaling 32 Mbytes  
Slot 1  16 Mbytes  Base address = 00000000  
Slot 2  16 Mbytes  Base address = 01000000

{ /5 6 } <1> =>

SEE ALSO  
mdg(1)
NAME
   menu - Display listing of available tests

SYNOPSIS
   menu

DESCRIPTION
   menu lists the names of all available tests in the default order of execution. menu displays tests in the default order of execution.

EXAMPLE
   User input in the example is shown in boldface type.
   The following example displays the list of installed tests.

   { /5 6 } <1> => menu
   Menu of installed test programs:
   Test 01: Atomic Load-Store Test
   Test 02: Memory Data RAM Test
   Test 03: Shared-Memory Pattern Test
   Test 04: Cache Block Alias Test
   Test 05: Floating Point Store Test
   Test 06: Cache Data Request Test
   Test 07: Cache Data Bus Pattern Test
   Test 08: Interrupt Test

   { /5 6 } <1> =>

SEE ALSO
   mdg(1), tests(1)
NAME

names - Enable or disable printing of test names during test execution

SYNOPSIS

names [ on | off ]

DESCRIPTION

names enables or disables the printing of test names during test execution.

OPTIONS

on    Enables the printing of the test names during test execution. This is the default setting.
off   Disables the printing of the test names during test execution.

EXAMPLES

User input in the examples is shown in boldface type.

The following example causes the state of the name flag to be displayed.

    { /5 6 } <1> => names
    names = on
    { /5 6 } <1> =>

The following example illustrates how the names flag is changed and redisplayed.

    { /5 6 } <1> => names off
    { /5 6 } <1> => names
    names = off
    { /5 6 } <1> =>

SEE ALSO

mdg(1), status(1)
NAME
next - Execute next selected test

SYNOPSIS
next

DESCRIPTION
next causes the test sequence to be continued, starting with the next selected test. It is
used when a test halts on an error and the user wishes to continue test execution with the
next test in the sequence.

EXAMPLE
User input in the example is shown in boldface type.
In the following example run was entered to begin test execution. The current test selec­
tion was executed until an error was encountered in test 1. next was entered to continue
the test sequence starting with the next test in the sequence.

```plaintext
{ /5 6 } <1> => run
Starting Test 1: (testname)
Test 1 error: (error message)

{ /5 6 } <1> => next
Starting Test 2: (testname)
.
.
Starting Test n: (testname)

Tests completed: Passes = 1  Errors = 1  Tue Nov 22 14:58:04 1988
{ /5 6 } <1> =>
```

SEE ALSO
between(1), errlim(1), mdg(1), passlim(1), restart(1), run(1)
NAME
  passes - Display pass count

SYNOPSIS
  passes

DESCRIPTION
  passes displays the number of complete test passes that have made since the last run command.

EXAMPLE
  User input in the example is shown in boldface type.
  The following example illustrates how to use the passes command.
  { /5 6 }  1 => passes
  Total passes = 0
  { /5 6 }  1 =>

SEE ALSO
  m4g(1), passlim(1)
NAME
  passlim - Set or display pass limit

SYNOPSIS
  passlim [ limit ]

DESCRIPTION
  passlim sets or displays the current setting of the test pass limit. passlim specifies the number of test passes that can occur before test execution is halted.
  This command should be used when it is desired to execute numerous passes of the test sequence.

OPTION
  limit  Sets the number of test passes that will be run. By default, limit is set to one. Limit must be entered in unsigned decimal format in the range 0-to-2,147,483,647, inclusive. A limit of 0 specifies that tests execute continuously until a Control-C is entered.

EXAMPLES
  User input in the examples is shown in boldface type.
  The following example illustrates how to display the current pass limit.

  { /5 6 } <1> => passlim
      Pass limit = 1
  { /5 6 } <1> =>

  The following example illustrates how to change and re-display the pass limit.

  { /5 6 } <1> => passlim 0
  { /5 6 } <0> => passlim
      Pass limit = 0
  { /5 6 } <0> =>

SEE ALSO
  mdg (1), passes (1)
NAME
prompt - Set or display prompt flags

SYNOPSIS
prompt [ all | off | test test ... | test: test ... ]

DESCRIPTION
prompt sets or displays the prompt flag for each test program. The command allows the
user to selectively alter the default behavior of the test programs by turning the flag for
the specified tests on or off.

Only a few of the mdg tests use the prompt flag. The behavior of the test depends on
what the test is attempting to accomplish. In some case, if a test isn't prompted it does
not execute. In others, it modifies the test algorithm or enables the printing of informa­
tional messages.

Single tests or a range of tests may be prompted by specifying the test numbers or range
of tests number.

The menu (1) command indicates which tests examine their prompt flags.

OPTIONS
all Set prompt flags for all tests. all can be specified at any time to prompt all tests.
off Turns prompt flags for all tests off. off can be specified at any time to turn off
prompts for all tests.
test Prompt specified test. If test is not specified, the prompt command displays the
current status of the prompt flags.

EXAMPLES
User input in the examples is shown in boldface type.

The following example illustrates how to display the prompt flags.

```
{ /5 6 } <1> => prompt
no prompted tests
{ /5 6 } <1> => prompt all
{ /5 6 } <1> => prompt
prompted tests:
  1   2   3
{ /5 6 } <1> => prompt off
{ /5 6 } <1> => prompt 2 3
{ /5 6 } <1> => prompt
prompted tests:
  2   3
{ /5 6 } <1> =>
```

SEE ALSO
- mdg (1), menu (1), tests (1)
NAME
quiet - Set or display error message enable flag

SYNOPSIS
quiet [ on | off ]

DESCRIPTION
quiet sets or displays the error message enable flag. If no parameters are specified, quiet
displays the current setting of the flag.
The error message enable flag prevents error messages from being displayed on test
failures. This feature should be used to create the tightest possible loop when the loop
flag is on. A Control-C must be entered to stop the loop and return to the prompt.

OPTIONS
on       Turns on the quiet flag.
off      Turns off the quiet flag.

EXAMPLES
User input in the examples is shown in boldface type.
The following example causes the current error message enable flag to be displayed.

{ /5 6 } <1> => quiet
quiet = off
{ /5 6 } <1> =>

The following example illustrates how the quiet flag is changed and redisplayed.

{ /5 6 } <1> => quiet on
{ /5 6 } <1> => quiet
quiet = on
{ /5 6 } <1> =>

SEE ALSO
mdg (1), status (1)
NAME
   quit - Exit from mdg

SYNOPSIS
   quit

DESCRIPTION
   quit exits from the mdg debugger program and returns the user to the ROM> prompt.

SEE ALSO
   mdg(1)
NAME
restart - Restart execution of selected tests

SYNOPSIS
restart

DESCRIPTION
restart causes the current test(1) selection to be executed beginning with the first test
current test selection. The major difference between restart and run(1) is that restart goes
back to the first test in the sequence, while run continues execution with the next selected
test.
The number of times the test selection is executed depends on the value of the passlim(1)
limit.

EXAMPLE
User input in the example is shown in boldface type.
In the following example run was entered to begin test execution. The current test selec-
tion were executed until an error was encountered in test 1. restart was entered to start
the test sequence again from the beginning:

\{ /5 6 \} <1> => run
Starting Test 1: (testname)
Test 1 error: (error message)

\{ /5 6 \} <1> => restart
Starting Test 1: (testname)
  
Starting Test n: (testname)

Tests completed: Passes = 1 Errors = 0 Tue Nov 22 14:58:04 1988
\{ /5 6 \} <1> =>

SEE ALSO
mdg(1), next(1), passlim(1), run(1)
NAME
run - Start execution of selected tests

SYNOPSIS
run

DESCRIPTION
run causes the current test(1) selection to be executed. The number of times the test selection is executed depends on the value of the passlim(1) limit.

EXAMPLE
User input in the example is shown in boldface type.
In the following example run was entered to begin test execution. The current test selection was executed once (passlim = 1) followed by a tests completed message. If passlim's limit is set to a value other than one, the complete test sequence would be repeatedly executed until limit is reached, at which time the program would return to the prompt. The test completed message is displayed after each pass.

{ /5 6 } <1> => run
Starting Test 1: (testname)
  
Starting Test n: (testname)

Tests completed:  Passes = 1  Errors = 0  Tue Nov 22 14:58:04 1988
{ /5 6 } <1> =>

SEE ALSO
mdg(1), next(1), passlim(1), restart(1)
NAME
status - Display or reset state of modes, flags, and counts

SYNOPSIS
status [ reset ] [ flags ]

DESCRIPTION
status displays the current state of all modes, program flags, and counters. flags resets all
the flags, which includes names, continue, loop, quiet, and xbuf.

OPTION
reset Resets the status of flags, counts, and limits to the default setting. reset also
resets the test selection back to default values.
flags Resets the status of flags to the default settings.

EXAMPLE
User input in the example is shown in boldface type.

{ /5 6 } <1> => status
Tue Nov 22 12:45:20 1988
Names = on
Continue = off
Loop = off
Quiet = off
Time = off
Pass count = 0 Pass limit = 1
Error count = 0 Error limit = 0
Between count = 1

{ /5 6 } <1> =>

SEE ALSO
between(1), continue(1), ecc(1), errlim(1), errors(1), loop(1), mdg(1), names(1),
passes(1), passlim(1), quiet(1), time(1)
NAME
tests - Select or display tests to be executed

SYNOPSIS
tests [ all | test test ... | test:test ... ]

DESCRIPTION
tests select the tests for execution by the run(1) command. By default, all tests are selected for execution when the program is initialized.

Single tests or a range of tests may be selected by specifying the test numbers or range of tests number.

OPTIONS
all Execute all the tests. all can be specified at any time to reselect all tests.
test Execute specified test. If test is not specified, the tests command displays the current test selection.

EXAMPLES
User input in the examples is shown in boldface type.
The following example illustrates how to display the test selection.

```
{ /5 6 } <1> => tests
   selected tests:
   1 2 3
{ /5 6 } <1> =>
```

In the following example, tests 1 and 2 are selected and then displayed.

```
{ /5 6 } <1> => tests 1 2
{ /5 6 } <1> => tests
   selected tests:
   1 2
{ /5 6 } <1> =>
```

In the following example, tests 3 through 1 are selected and displayed. Note that tests may be selected to run in any order.

```
{ /5 6 } <1> => tests 3:1
{ /5 6 } <1> => tests
   selected tests:
   3 2 1
{ /5 6 } <1> =>
```

In the following example, all installed tests are selected and displayed.

```
{ /5 6 } <1> => tests all
{ /5 6 } <1> => tests
   selected tests:
   1 2 3 4 5 6 7 8
{ /5 6 } <1> =>
```
SEE ALSO

mdg(1), next(1), restart(1), run(1)
NAME

time - Set or display print time flag

SYNOPSIS

time [ on | off ]

DESCRIPTION

time sets or displays the print-time flag. If no parameters are specified, time displays the current setting of the print-time flag and the current time and date. The print-time flag controls whether the current time and date is printed when test names are displayed during test execution. The default state of the print-time flag is off (no time printed). If both the names flag and print-time flag are on, the time and date is printed on the line following the test name during test execution.

OPTIONS

on  Turns on the print-time flag.
off  Turns off the print-time flag.

EXAMPLES

The following example causes the current print-time flag to be displayed:

{ /5 6 } <1> => time
Tue Nov 22 14:20:00 1988
  Time = off
{ /5 6 } <1> =>

The following example illustrates how the print-time flag is changed and redisplayed.

{ /5 6 } <1> => time on
{ /5 6 } <1> => time
Tue Nov 22 14:20:00 1988
  Time = on
{ /5 6 } <1> =>

SEE ALSO

mdg(1), names(1), status(1)
NAME
vmeconf - Configure VMEbus devices

SYNOPSIS
vmeconf

DESCRIPTION
vmeconf generates or displays the VMEbus configuration table.

When mdg is invoked, it does not ask the user to generate a VMEbus configuration table. Therefore, if the user wishes to perform tests of the VMEbus chassis, they must first execute this command.

vmeconf prompts for all user input. It accepts no options or arguments at the command line.

Currently, vmeconf supports the Ciprico Rimfire, Excelan Ethernet, and Plessy RAM boards.

EXAMPLE
User input in the example is shown in boldface type.

The following example shows how vmeconf is used to remove an Excelan Ethernet VMEbus board from the configuration, then how the program would be used to put the board back into the configuration table.

```
{ /5 6 } <1> => vmeconf

VMEbus Configuration consists of four boards
(0) Ciprico Rimfire 3500 VMEbus-to-SCSI
    Am = 0x2d  Addr = 0x5000  Physaddr = 0x85ff5000
(1) Excelan Ethernet
    Am = 0x3d  Addr = 0x000000  Physaddr = 0x87d00000
(2) Plessey RAM (512K)
    Am = 0x3d  Addr = 0x100000  Physaddr = 0x87100000

Do you wish to change this configuration? (y/n) y
Do you want the default configuration? (y/n) n
Do you want to delete any entries? (y/n) y
Entry number to delete (q to quit)? 1
Entry number to delete (q to quit)? q
Do you want to add any entries? (y/n) n

(0) Ciprico Rimfire 3500 VMEbus-to-SCSI
    Am = 0x2d  Addr = 0x5000  Physaddr = 0x85ff5000
(2) Plessey RAM (512K)
    Am = 0x3d  Addr = 0x100000  Physaddr = 0x87100000

Do you wish to change this configuration? (y/n) y
Do you want the default configuration? (y/n) n
Do you want to delete any entries? (y/n) n
Do you want to add any entries? (y/n) n
How many vme boards are to be added? (0-5) 1

Enter information for board 1:
```
Valid VME board types are:
0: none
1: Ciprico Rimfire 3500 VMEbus-to-SCSI
2: Excelan Ethernet
3: Plessey RAM (512K)

Type of board? 2

Valid address modifiers are:
9: extended user data access
d: extended supervisor data access
39: standard user data access
3d: standard supervisor data access
29: short user data access
2d: short supervisor data access

Address modifier? 3d
Address? d00000

VMEbus Configuration consists of 3 boards
(0) Ciprico Rimfire 3500 VMEbus-to-SCSI
   Am = 0x2d   Addr = 0x5000   Physaddr = 0x85ff5000
(1) Excelan Ethernet
   Am = 0x3d   Addr = 0xd00000  Physaddr = 0x87d00000
(2) Plessey RAM (512K)
   Am = 0x3d   Addr = 0x100000  Physaddr = 0x87100000

Do you wish to change this configuration? (y/n) n
{ /5 6 } <1> =

SEE ALSO
   mdg(1)
NAME
wake - Add processor to mdg environment

SYNOPSIS
wake [cpu]

DESCRIPTION
wake provides a method to tell a processor, which is idle at the ROM level, to start mdg as a slave processor. The specified processor must be recognized by the master as a valid processor in the system.

OPTIONS
cpu Add specified cpu. The specified processor is told to start mdg as a slave processor.

EXAMPLES
User input in the examples is shown in boldface type.
In the following example, processor 6 is awaken and added to mdg.

{ /5 6 } <1> => wake 6
{ /5 6 } <1> =>

SEE ALSO
config(1), halt(1), mdg(1)
NAME
pdelstruct - delete structure

SYNOPSIS
#include "phigs.h"

void
pdelstruct(struct_id)
Pint  struct_id;  /* structure identifier */

DESCRIPTION
Call pdelstruct (3P) to delete a structure and its contents.

OPERATING STATES
(PHOP,*,*,*)

EFFECT
The specified structure is deleted; its identifier, its contents and all references to it are removed from PHIGS. It is unposted from all workstations to which it is posted. In the event the specified structure is the open structure, the resulting functionality is equivalent to the following sequence:

    CLOSE STRUCTURE
    DELETE STRUCTURE (structure identifier)
    OPEN STRUCTURE (structure identifier)

If the specified structure does not exist, no action is taken.

SEE ALSO
pdelallstruct (3P), pdelstructnet (3P)

DIAGNOSTICS
002  Ignoring function, function requires state (PHOP,*,*,*)
Index

A
Awake SLAVE processor, 2-1

Using commands, 2-3
Memory configuration table, 2-1

C
Calculate CPU test limits, 2-1
Calculate shared memory, 2-1

Removing and adding processors, 2-8

F
Frame buffer configuration table, 2-1

Starting text execution, 2-5

H
Handling test failures, 2-6

Variations of test execution, 2-6
VMEbus configuration table, 2-1

M
mdg tests, 3-1
Test 01, -3
Test 02, -3
Test 03, -3
Test 04, -3
Test 05, -3
Test 06, -3
Test 07, -3
Test 08, -3
mdg:
Available commands, 4-1
Definition, 1-1
Entering commands, 2-2
Error messages, 2-3
Exiting, 2-8
help command, 2-3
Instructions, 2-1
Invoking, 2-1
Prompt, 2-2
Related documentation, 1-1
Selecting processors for testing, 2-4
Test control commands, 2-3