MemTest-86 User Manual

Version 4.3

Table of Contents

1 Introduction

1.1 Memory Reliability

Properly functioning memory is critical for reliable operation of a PC or laptop. Few computer users fully understand the risks associated with memory errors. Because PCs typically do not have any mechanisms for detecting memory errors, confusing and potentially disastrous consequences can result from these undetected memory problems. Memory errors will often cause erratic behavior with software applications that can mysteriously fail. The most serious risk from memory errors, however, is corruption of data that manages how information is stored on disk. In most cases, this type of corruption will cause one or more files to be lost. There are cases where a memory error can cause the loss of the entire contents of your hard disk. Periodic testing of memory with a rigorous and thorough memory test will greatly reduce the risk of problems and data loss due to memory errors.

1.2 Memtest86 Overview

Memory errors are often pattern sensitive and may be very intermittent. Detecting these errors is technically challenging and is an imperfect science. Memtest86 uses advanced algorithms that have been refined for more than 20 years. These testing techniques are highly effective at detecting difficult to find memory errors. In addition, Memtest86 has the capability to test all available memory. Memory testing programs execute from memory and therefore are not able to test the memory that is occupied by the test program itself. Memtest86 is designed to move itself to a different portion of memory and then tests the memory that it previously occupied.

1.3 Compatibility

Memtest86 is designed to work with all processors using the Intel/AMD x86 and X86_64 architecture. For 64 bit CPUs, Memtest86 currently executes in 32 bit mode using PAE. For 32 bit CPUs, testing is limited to 64 GB. 64 bit CPUs running MemTest86 executes in “long” mode which allows for testing of up to 8 TB of memory. CPUs executing in 32 bit mode can test a maximum of 2 GB of memory at a time. This 2 GB window is then advanced, allowing for all of memory to be tested. A native 64 bit implementation is planned.

Starting with version 4.0 Memtest86 is multi-threaded and is able to concurrently use multiple CPUs to test memory. It will function properly with any number of CPUs but is currently configured to use a maximum of 32 CPUs for testing.

Memtest86 is able to test all types of memory. There is no need for Memtest86 to know what type of memory it is testing. Memtest86 attempts to detect and display information about the hardware it is testing but this information is not used during testing.

Since Memtest86 is a standalone program it does not require any operating system support for execution. It can be used with any PC regardless of what operating system, if any, is installed.

2 Setup and Use

Memtest86 is loaded from a floppy disk, CD-ROM, USB flash drive or, with Linux systems, by the boot loader (for example, LILO or Grub). To use Memtest86 on a Windows computer either a 3.5-inch floppy drive or CD-ROM drive, or USB flash drive is required. Any Unix or Windows system may be used to create a floppy drive or CD-ROM. Once a Memtest86 boot disk has been created, it may be used on any computer. If you already have a Memtest86 floppy disk, CD-ROM or USB flash drive please skip to section 2.3.

2.1 Boot-disk Creation using Windows

Before you can use Mestest86 on a Windows system it must first be installed on a CD-ROM, USB flash drive or 3.5-inch floppy disk. To create a CD-ROM a system with the capability of creating a CD-Image from an ISO file is required.

Create a boot-able CD-ROM:

1. Download the Windows Memtest86 ISO image.
2. Right click on the downloaded file and select the "Extract to Here” option. This places the CD-ROM ISO image in the current folder.
3. Use the CD burning software available on your system to create a CD-ROM using the extracted ISO image. Be sure that you create a CD image from the ISO file rather than placing a copy of the ISO file onto a data CD. Look for “Burn Image from File” or similar option under the File menu of your CD burning software.

Create a boot-able USB Flash drive:

1. Download the Windows Memtest86 USB image zip file.
2. Extract the contents of the zip file to a directory
3. Plug in the USB drive
4. Launch the ImageUSB application that was included in the zip file
5. Select the USB drive from the list
6. If it is not already selected, select the image file included in the zip file
7. Click 'Write to UFD'
8. After accepting a few more prompts this should give you a working bootable USB drive

Create a boot-able floppy:

1. Download the Windows Memtest86 floppy image.
2. Right click on the downloaded file and select the "Extract to Here” option. This places the floppy disk image in the current folder.
3. Creating a boot-able floppy disk requires use of a third party program to write the floppy disk image to a disk. A number of programs are available to write the disk image. Rawwrite is a recommended free program available at: http://www.chrysocome.net/rawwrite. A more robust solution is WinImage, available at http://www.winimage.com/download.htm.

2.2 Boot-disk Creation Using Linux

Memtest86 is a stand alone program and can be loaded from a bootable CD-ROM, USB disk, floppy disk or a Linux disk partition. It is recommended that Linux users download and install pre-compiled packages to create boot-able media. Advanced users may wish to build from source and optionally make source code changes.

Create a boot-able CD-ROM:

1. Download the Linux Memtest86 ISO image.
2. Uncompress the ISO image (gunzip memtest86-iso.gz).
3. Use the CD burning software available on your system to create a CD-ROM using the uncompressed ISO image. Be sure that you create a CD image from the ISO file rather than placing a copy of the ISO file onto a data CD. Look for “Burn Image for File” or similar option under the File menu of your CD burning software.

Create a boot-able USB Flash drive:

1. Download the Linux Memtest86 USB image.
2. Untar the package (tar xvzf memtest86-usb.tar.gz). An image file and a README file will be created in the current directory.
3. Follow instructions in the README to write the USB flash disk.

Create a boot-able floppy:

1. Download the Linux floppy disk image.
2. Untar the package (tar xvzf memtest86-floppy.tar.gz). An image file and a README file will be created in the current directory.
3. Follow instructions in the README to write the floppy disk.

2.3 Building Memtest86 from source

1. Review the Makefile and adjust options as needed.
2. Type "make"

This creates a file named "memtest.bin" which is a bootable image. If you encounter build problems a pre-compiled binary image has been included (precomp.bin). This image file may be copied to a floppy disk or may be loaded from a disk partition by Lilo or Grub boot loaders from a hard disk partition. The Makefile creates bootable images that may be written directly to a floppy disk or CD. A method is not provided for creating a bootable USB flash drive from source.

Create a Memtest86 floppy disk:

1. Insert a blank write-enabled floppy disk.
2. As root, type "make install"

Create a bootable CD-ROM

1. type “make iso”
2. An ISO image file (memtest86.iso) is created in the source directory. Use CD burning software available on your system to create a CD-ROM using the extracted ISO image.

Boot from a disk partition using Grub:

1. Copy memtest.bin to a permanent location (for example: /boot/memtest.bin).
2. Add an entry in the Grub config file (/boot/grub/menu.lst) to boot memtest86. Only the title and kernel fields need to be specified. The following is a sample Grub entry for booting memtest86:

   ```
   title Memtest86
   linux16 /boot/memtest.bin
   ```

Boot from a disk partition using Lilo:

1. Copy memtest.bin to a permanent location (ie. /boot/memtest.bin).
2. Add an entry in the lilo config file (usually /etc/lilo.conf) to boot memtest86. Only the image and label fields need to be specified. The following is a sample Lilo entry for booting memtest86:

   ```
   image = /boot/memtest.bin
   label = memtest86
   ```

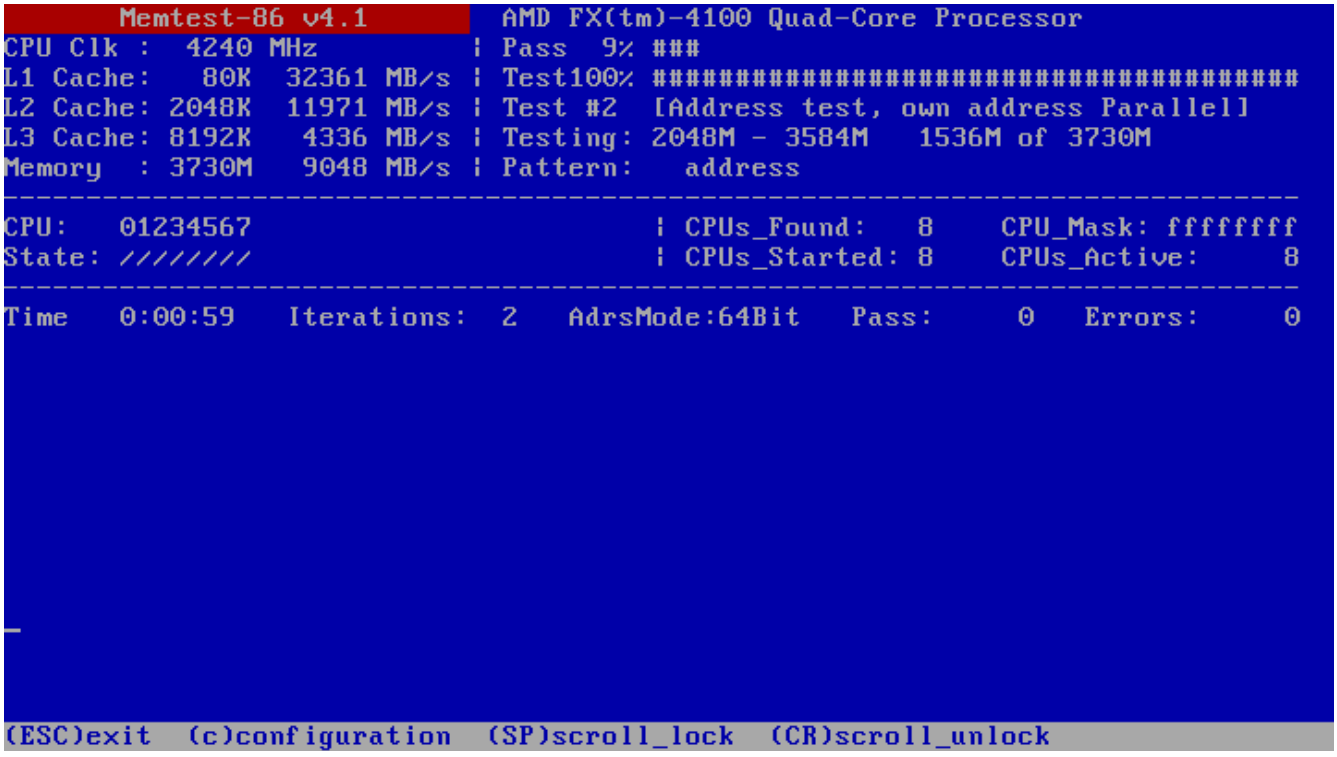
3. As root, type "lilo"

2.4 Using Memtest86

To start Memtest86 insert the Memtest86 floppy disk, CD-ROM or USB flash drive into the appropriate drive and restart your computer.

> **Note:** The BIOS must be configured to boot from the device that Memtest86 is installed on. Newer computers have an optional boot menu that is enabled be pressing a key at startup (often ESC, F9, F11 or F12). If available use the boot menu to select the correct drive. With older computers the boot sequence must list the drive used to load Memest86 before any other bootable media (i.e. a hard disk where Windows is installed). Most systems will be configured with a suitable boot sequence. If not you can reconfigure the boot sequence using the BIOS settings. Please consult your motherboard documentation for details.

When Memtest86 starts it displays details about the system configuration and then begins testing. Memtest86 executes a repeating cycle of tests. Testing will continue to run until the program execution is interrupted (by pressing the ESC key or pressing the reset button). There is no set time for how long the test should be run. The following is an example of the Memtest86 screen.



Memtest86 executes a series of numbered test sections to check for errors. The execution order for these tests has been arranged so that errors will be detected as rapidly as possible. The pass counter increments each time that all of the tests have been run. The first pass is abbreviated to find errors more quickly. Subsequent passes execute longer and will be more thorough. Generally a single pass is sufficient to detect all but the most obscure errors. For complete confidence in cases where intermittent errors are suspected, testing for a longer period is advised. The time required for a complete pass of Memtest86 will vary greatly

depending on CPU speed, memory speed and memory size.

If memory errors are detected they will be displayed on the lower half of the screen. The default error reporting mode will display a detailed summary of all errors.

If Memtest86 runs multiple passes without errors you can be certain that your memory is functioning properly. Other problems (hardware or software) may exist but at least you can eliminate memory as a culprit. In addition the CPU must work properly to run Memtest86 so successful execution implicitly assures that your CPU is also functioning properly.

Memtest86 can not diagnose many types of PC failures. For example a faulty CPU that causes Windows to crash will most likely just cause Memtest86 to crash in the same way.

2.5 Runtime Configuration Options

Memtest86 may be configured during operation via runtime configuration commands. Pressing the “C” key at anytime will display the runtime command menu. The following figure shows an example of the Configuration command menu.

Settings:

(1) Test Selection

(2) Address Range

(3) Error Report Mode

(4) CPU Selection Mode

(5) Refresh Screen

(6) Restart Test

(7) Miscellaneous Options

(0) Continue

The runtime configuration commands allow the user to adjust the following settings.

(1) **Test Selection** - Individual tests may be selected for execution or the current test may be skipped. When a failure has been identified it is much faster to troubleshoot by running only the failing test.

(2) **Address Range** - With this option memory testing may be restricted to a selected portion of memory. This is also useful to speed up the troubleshooting process by testing only the failing portion of memory.

(3) **Error Report Mode** - The error report mode may be changed to provide different types of error information. The default “Error Summary” mode provides all of the details required for diagnosing memory problems. However, other error reporting modes are available. "BadRAM” patterns may be used with Linux systems to map out bad memory blocks.

(4) **CPU Selection Mode** – This option controls how CPUs are selected for testing. The options are All, Round Robin and Sequential. With Round Robin CPUs are selected in round robin fashion for each test. With the Sequential option all available CPUs are

used for each test.

(5) **Refresh Screen** - Redraws the screen when the display becomes garbled.

(6) **Restart Test** – Restarts test with default settings.

(7) **Miscellaneous Options –** Enable and disable runtime options. Currently only two options are available, One Pass and Boot trace. The One Pass feature runs the complete test once and then exits, but only if there were no errors. This provides a convenient method for unattended testing. The Boot Trace option is a debugging tool that is primarily designed for troubleshooting startup problems, but may be enabled at anytime.

A help bar is displayed at the bottom of the screen with the following options.

Keyboard Assignment	***Description***
ESC	Exits the test and does a warm restart via the BIOS
C	Enter the configuration menu
SP (Spacebar)	Set scroll lock (Stops scrolling of error messages) **Note:** Memory testing is suspended when the scroll lock option is set and the scrolling display region is full.
CR (Enter)	Clear scroll lock (Enables error message scrolling)

2.6 Startup (boot) Options

A number of options may be specified at boot time. The following options are available:

◆One Pass – Enables the One Pass option. This feature runs the complete test once and then exits, but only if there were no errors. This provides a convenient method for unattended testing. One Pass may also be enabled via an on-line command. The syntax is “onepass” and there are no additional options.

◆Btrace – Enables the boot trace option and is used to diagnose test failures please see section 2.7 for details. The syntax is “btrace” and there are no additional options.

◆Maxcpus – Set the maximum number of CPU's to start. This is primarily a troubleshooting option. Setting maxcpus to one skips all code related to finding and starting additional CPUs. The syntax is “maxcpus=N” where N = 1 to 31.

◆Test List – Allows for execution on a subset of the normal test sequence. The syntax is “tstlist=list” where list is a comma separated list of test numbers to run (do not include spaces). Example: tstlist=1,4,5,7

◆CPU Mask – A mask of CPU numbers that are started and enables. A 32 bit mask is specified with a 1 set for each CPU that will be enabled. CPU 0 cannot be disabled and will be enabled regardless of the mask. The syntax is “cpumask=N” where N is a 32 bit hexadecimal number with or without a 0x prefix. Example: cpumask=0x55555555 enables all

even numbered CPUs.

◆Console – Used to setup serial console parameters. The syntax is “console=ttySn,baudPL” where n = serial port number (0 or 1), baud = baud rate, P = parity setting (N, O or E) and L is the number of bits (7 or 8). Example: “console=ttyS0,9600e8 uses port 0 at 9600 baud, even parity and 8 bits.

The standard Memtest86 media images provided use Syslinux for booting and allow for boot options to be specified interactively. At the “boot:” prompt type “memtest” followed by any of the above options. Multiple options may be specified separated by spaces. For example “memtest onepass tstlist=1,6,7,8” will start the test with the One Pass option enabled and execute tests 1,6,7 and 8.

2.7 Troubleshooting Memtest86 Problems with Boot Trace

This is section is targeted at troubleshooting problems with execution of the test code and not for diagnosing reported memory errors. In Version 4.1 a Boot Trace feature was added that provides a simple mechanism for troubleshooting of test failures by both technical and non-technical users. With the very large variety of computer hardware available it is impossible to test Memtest86 an all platforms and some incompatibilities exist causing failures. In the past determining the cause of these failures has been difficult to impossible. With trace information many of these faults may now be addressed.

Enabling Boot Trace - The boot trace feature is enabled with either a boot time option or an on-line command. However, most problems occur during startup so the boot option is the preferred method. All of the released images have a boot option to start the test with boot tracing enabled. When booting from a Linux system with LILO or GRUB add “btrace” to the boot options line.

When Boot Trace is enabled two columns of trace information will be displayed one the bottom two thirds of the screen. The CPU, line number from the source file, a short message and two parameters are displayed in each trace point. A “>” character denotes the current trace point. A total of 26 trace points are displayed and older trace points are lost as execution progresses. Pressing any key will advance to the next trace point. As initialization proceeds other portions of the screen will be filled in with information and the header lines will be erased.

Gathering Trace Information – There are two types of failures where trace information is needed to help diagnose the problem, hangs and crashes. Gathering traces for a situation when the test hangs is very simple. Just continue to press return until the test stops. At minimum email the last 5 trace points. Much more useful would be a digital picture of the screen. For cases where the test crashes we need to see the trace points just before the crash. This requires slowly stepping through the traces and identifying the point where the test reboots. Then run the test again and stop at the trace point just before the crash and report the information. Again, email at least the last 5 trace points or better a picture of the screen. Please email failure information to bugs@memtest86.com.

Using Trace Information - Technical users will be able to diagnose problems using trace data that previously would have required detailed understanding of Memtest86's internal workings. By following the trace points in the source you will be able to follow the path of execution and identify problems. As needed new trace points may be added to the code to provide more detail when diagnosing a problem.

3 Repairing Memory Faults

When Memtest86 detects errors the error count will be incremented and the error details will be displayed on the lower half of the screen. The key information needed to diagnose errors are the “Error Confidence Value” and the “Errors per Memory Slot” information. Error confidence values over 100 should always be considered to be legitimate. The errors per memory slot indicate both the number of memory modules present in your PC and the number of errors for each memory module. This information may not be available for older PCs. The remaining error details may be ignored.

To diagnose and repair a memory fault requires you to open your computer case and handle sensitive electronic components. With proper handling procedures this is not difficult. If you do not want repair your own hardware you can use Memtest86 to test your RAM, but rely on a third party to do the actual repair.

3.1 Anti-Static Handling Procedures

Electronic components may be damaged by static electricity. The key to proper handling is to simply avoid causing a static discharge though the component that is being handled. This is done by discharging static buildup before a component comes in contact with another surface. For example when installing a component into a computer, first touch a bare metal part of the computer case. If you want to set a component on a table, touch the table first and then set down the component. If you take a step or even shuffle your feet you will need to re-discharge any static buildup.

3.2 Re-Seating Memory Modules

In many cases memory problems are caused by a poor connection. This can be resolved by simply removing and reinstalling the memory module(s) using the following procedure.

1. Using the documentation for your motherboard locate the memory module slots and identify the memory module(s).
2. Using proper anti-static handing procedures remove the memory module(s). In most cases this is done by pressing down on the locking tabs at the end of the module slot.
3. Carefully clean any dust or debris from the module and the motherboard memory module slots.
4. Reinstall the memory module(s) into their original slot on the motherboard.
5. Rerun Memtest86.

3.3 Replacing Modules

If re-seating memory modules does not resolve the problem then a memory module will generally need to be replaced.

When more than one module is installed you will need to determine which module is failing.

First consult your motherboard documentation to determine if memory modules may be used individually or if they must be used in pairs. If your motherboard is configured with the minimum number of memory modules you will need to purchase a replacement memory

module in order to locate the failing one. Using the guidelines in your motherboard manual to maintain a working configuration selectively remove modules from the system and rerun Memtest86 to find the bad module(s). Be sure to carefully note which modules are in the system when the test passes and fails.

In most cases memory failures will be due to a faulty memory module and replacing the module will resolve the problem. However, the motherboard affects memory operation and may also cause memory errors. There are instances where only a particular combination of memory and motherboard will cause errors. This is typically due to the memory not being of sufficient quality and speed to keep up with the motherboard. This memory may function properly when installed in a less demanding motherboard. When errors are only detected by test #5 it is usually due to memory not being fast enough to work properly with the motherboard. More conservative memory timing BIOS settings (if supported) may resolve these problems. Otherwise higher quality memory may be required, or possibly the motherboard may need to be replaced.

3.4 Error Validity

There are many users who question if errors detected by Memtest86 are valid. In at least 99.9% of cases reported errors will be legitimate and must be corrected. Memtest86 simply exercises all available RAM looking for errors. If any error is detected in RAM, regardless of how or where, it is a legitimate failure that needs to be corrected. There are no known issues regarding compatibility. It is possible, but unlikely, that a particular error will never show up in normal operation. However, operating with marginal memory is risky and can result in data loss and disk corruption. Even if there is no overt indication of problems you cannot assume that your system will be unaffected.

There are some rare cases where motherboard manufacturers will map hardware registers into space normally occupied by memory. When these locations are tested errors are reported. In this case, the reported errors will be invalid. To better identify these rare cases where invalid errors are reported an error confidence value is created by Memtest86. The program tracks failures and then does some simple analysis to determine the probability of invalid errors. When the confidence value exceeds 100 it is nearly impossible that the reported errors will be invalid.

4 Over Clocking

4.1 Background

Memtest86 is an invaluable tool for over clocking and may be used to increase your systems performance and reliability. Many shy away from over clocking fearing that it will make their PC less reliable. With a proper procedure, fine tuning your system timings is safe and in some cases will even improve reliability.

Before embarking on over clocking one must understand the concept of margins or margin for error.

4.2 Operating Margins

To achieve high reliability computer systems are designed and tested under conditions that are more strenuous than those expected in normal use. Take for example a computer that is designed to operate with a bus frequency of 133 MHz. A quality manufacturer will design and test for operation at perhaps 140 MHz or higher. This margin for error provides confidence that even with inevitable manufacturing variances and changing conditions operation will be reliable. When components from different manufacturers are combined an even greater margin for error is required since the exact characteristics of the associated components are not known. The result is the majority of computers will end up having a much larger margin for error than is needed. This means that a lot of performance is being wasted. On the other hand there will be a small number of systems that do not have enough margin of error and will have poor reliability even without over clocking.

Many think of over clocking as simply making a computer operate as fast as possible. This approach is unwise and will often result in unreliable operation. A much better philosophy is to adjust and fine tune computer timings to maintain an appropriate margin for error. An appropriate margin includes not leaving too much margin, wasting performance.

For example let say we have a computer that will operate properly with a system clock of 189 MHz. Obviously a lot of performance will be wasted if we operate the computer at the standard 133 MHz. In addition if the computer fails at 190 MHz it would be unwise to operate at 189 MHz since there is little margin for error and operation will likely not be reliable. An operating margin of 3% to 6% is sufficient to insure good reliability. For this example a system clock of 178 to 183 would be ideal. There may be cases where it will be advisable to under clock. If a system with a normal clock rate of 133 MHz does not operate properly at 136 MHz or more then there is not enough margin and under clocking is required to ensure reliability.

To summarize, over clocking should be thought of as fine tuning a computer to ensure reliability first and secondarily maximizing performance.

4.3 Using Memtest86 for Over Clocking

The first step in a proper over clocking procedure is to determine the operational limits of your computer. After the operational limits have been identified the system settings may be adjusted to provide enough margin to ensure high reliability. Memtest86 is an ideal tool to accurately determine the operation limits of your memory and CPU. Using the guidelines below, experiment with the settings available for your motherboard to find settings that result in the highest clock rate combined with the highest reported memory bandwidth.

1. Before attempting to over clock you need to know what system parameters your motherboard will allow you to adjust and how they are adjusted. Some motherboards will allow all useful parameters to be adjusted while others do not allow for any adjustment. Consult your motherboard manual for details. Some of the parameters that may be available are:
    - System clock rate
    - CPU clock multiplier
    - Memory timing
    - Memory clock multiplier
    - CPU voltage
    - Memory voltage
2. You need to know how to reset the CMOS to factory settings. It is common when over clocking to end up with settings that will not run the BIOS. When the parameters are set via CMOS the only way to recover is to reset the CMOS to the factory configuration. For some motherboards this is accomplished by removing a jumper. Other will require removal of the CMOS backup battery. Make sure that you know how to recover before starting.
3. Before adjusting any parameters, run Memtest86 for a full pass to establish a baseline. Record the memory bandwidth and CPU clock frequency for the default configuration.
4. Typically the best place to start is with the system clock rate. Increase the clock rate in fairly small increments (2-4 MHz) and then run at least a partial pass of Memtest86. Running at least 15% of tests 1-5 should be the minimum amount of testing for each iteration. Continue increasing the clock rate until you get a failure. Take your time and take good notes. For each step be sure that you record the memory bandwidth reported by Memtest86. Some BIOS's automatically adjust memory timings according to clock rate. You may find that by increasing the clock rate, memory performance will decrease. Once you find a failure back off on the clock rate until you find the point at which you get errors. Once you find the point at which memory errors occur back the clock off one step and run a full pass of Memtest86 to confirm the operational limit. In some cases the CPU will hang (stop responding) before memory errors are detected.
5. Some motherboards will allow you to adjust memory timing. Memory timings are typically listed as 4 values separated by hyphens. However, some motherboards only offer choices like fast, faster and fastest. There are two strategies for adjusting memory timing. If memory errors are detected before the CPU exhibits problems then slower memory timing may be used to allow a higher system clock rate to be used. The second strategy is to use faster memory timings to get more memory bandwidth without increasing the system clock rate. It is impossible to know which values will effect errors. Some of the memory timings will affect memory bandwidth and others will not. Be sure to record the reported memory bandwidth for each parameter change.
6. If in step 4 the CPU hangs before memory errors appear then the CPU has less margin for error than the memory. If available you may want to try reducing the CPU multiplier and then continue to increase the system clock until either the CPU stops or memory errors occur. This is helpful if memory timings are not adjustable or are ineffective.

7. CPU and memory voltages are adjustable on some motherboards and increasing them may allow you to run at higher speeds. In particular higher CPU voltages tend to be quite effective for over clocking. However, higher voltages also mean higher temperatures so be sure that you have plenty of case cooling and an effective CPU cooler. Use carefully.
8. Some motherboards allow you to use a system clock multiplier for memory. The default is usually 1:1, or in other words the system and memory clock rates will be the same. This setting is only useful when the memory and CPU operational limits are significantly different and can not be brought into balance using the techniques listed above.

Once you have established the operational limits of your system then you need to select settings that allow for a reasonable margin for error. Do not be tempted to use the maximum settings! A margin of 3% to 6% is recommended for reliable operation. The easiest way to add operational margin is to simply reduce the system clock rate by 3% to 6% from the maximum setting that functioned properly.

In many cases the operational limits for the CPU and memory will be different. For example you can get more CPU speed by reducing memory settings. Or you can get more memory performance by reducing the CPU multiplier. For these cases you will need to choose a compromise. Both memory bandwidth and CPU clock rate are important so don't be tempted to only optimize for one or the other.

Memtest86 provides good assurance of reliable memory operation when over clocking. Even when a dramatic increase in memory bandwidth is achieved. As long as Memtest86 does not report errors and appropriate margins have been applied then you should not hesitate to fully maximize memory timings. However, some caution must be exercised for system clock rate increases. With most motherboards the clock rate for the PCI and AGP busses are based on the system clock. Generally these busses will have no problem running at somewhat higher rates. Memtest86 does not test PCI or AGP and do not provide any assurance that anything other than the CPU and memory are working properly. Sadly there is currently no safe way to determine the operational limits for PCI and AGP and therefor there is no way to assure that there are appropriate margins. Unless your motherboard is able to independently establish the frequency of PCI and AGP busses you should be careful about running with large (more than 15%) increases in the system clock. In addition running your CPU at higher frequencies will generate more heat. Small frequency increases will generally be fine with the installed CPU cooler. Larger increases in the system clock rate may necessitate a larger, more effective CPU cooler.

Appendices

Appendix A.Technical Information

Appendix A contains technical information from the Memtest86 README file that was previously released under the Gnu Public License (GPL). This section provides additional background information and technical information that may be useful for advanced users.

A.1 Memory Testing Philosophy

There are many approaches for testing memory. However, many tests simply throw some patterns at memory without much thought or knowledge of memory architecture or how errors can best be detected. This works fine for hard memory failures but does little to find intermittent errors. BIOS based memory tests are useless for finding intermittent memory errors.

Memory chips consist of a large array of tightly packed memory cells, one for each bit of data. The vast majority of the intermittent failures are a result of interaction between these memory cells. Often writing a memory cell can cause one of the adjacent cells to be written with the same data. An effective memory test attempts to test for this condition. Therefore, an ideal strategy for testing memory would be the following:

1. write a cell with a zero
2. write all of the adjacent cells with a one, one or more times
3. check that the first cell still has a zero

It should be obvious that this strategy requires an exact knowledge of how the memory cells are laid out on the chip. In addition there is a never ending number of possible chip layouts for different chip types and manufacturers making this strategy impractical. However, there are testing algorithms that can approximate this ideal strategy.

A.2 Memtest86 Test Algorithms

Memtest86 uses two algorithms that provide a reasonable approximation of the ideal test strategy above. The first of these strategies is called moving inversions. The moving inversion test works as follows:

1. Fill memory with a pattern
2. Starting at the lowest address
   2a. Check that the pattern has not changed
   2b. Write the patterns complement
   2c. Increment the address
   Repeat 2a - 2c
3. Starting at the highest address
   3a. Check that the pattern has not changed
   3b. Write the patterns complement
   3c. Decrement the address
   Repeat 3a - 3c

This algorithm is a good approximation of an ideal memory test but there are some limitations. Most high density chips today store data 4 to 16 bits wide. With chips that are more than one bit wide it is impossible to selectively read or write just one bit. This means that we cannot guarantee that all adjacent cells have been tested for interaction. In this case the best we can do is to use some patterns to insure that all adjacent cells have at least been written with all possible one and zero combinations.

It can also be seen that caching, buffering and out of order execution will interfere with the moving inversions algorithm and make less effective. It is possible to turn off cache but the memory buffering in new high performance chips can not be disabled. To address this limitation a new algorithm called Modulo-X was created. This algorithm is not affected by cache or buffering. The algorithm works as follows:

1. For starting offsets of 0 - 20 do steps 2-5
2. write every 20th location with a pattern
3. write all other locations with the patterns complement
4. repeat 1b one or more times
5. check every 20th location for the pattern

This algorithm accomplishes nearly the same level of adjacency testing as moving inversions but is not affected by caching or buffering. Since separate write passes (1a, 1b) and the read pass (1c) are done for all of memory we can be assured that all of the buffers and cache have been flushed between passes. The selection of 20 as the stride size was somewhat arbitrary. Larger strides may be more effective but would take longer to execute. The choice of 20 seemed to be a reasonable compromise between speed and thoroughness.

A.3 Individual Test Descriptions

Memtest86 executes a series of numbered test sections to check for errors. These test sections consist of a combination of test algorithm, data pattern and caching. The execution order for these tests were arranged so that errors will be detected as rapidly as possible. A description of each of the test sections follows:

Test 0 [Address test, walking ones, no cache]

Tests all address bits in all memory banks by using a walking ones address pattern. Errors from this test are not used to calculate BadRAM patterns.

Test 1 [Address test, own address Sequential]

Each address is written with its own address and then is checked for consistency. In theory previous tests should have caught any memory addressing problems. This test should catch any addressing errors that somehow were not previously detected. This test is done sequentially with each available CPU.

Test 2 [Address test, own address Parallel]

Same as test 1 but the testing is done in parallel using all CPUs using overlapping addresses.

Test 3 [Moving inversions, ones & zeros Parallel]

This test uses the moving inversions algorithm with patterns of all ones and zeros. Cache is enabled even though it interferes to some degree with the test algorithm. With cache enabled this test does not take long and should quickly find all "hard" errors and some more subtle errors.

Test 4 [Moving inversions, 8 bit pattern]

This is the same as test 3 but uses a 8 bit wide pattern of "walking" ones and zeros. This test will better detect subtle errors in "wide" memory chips. A total of 20 data patterns are used.

Test 5 [Moving inversions, random pattern]

This test uses the same algorithm as test 3 but the data pattern is a random number and it's complement. This test is particularly effective in finding difficult to detect data sensitive errors. A total of 60 patterns are used. The random number sequence is different with each pass so multiple passes increase effectiveness.

Test 6 [Block move]

This test stresses memory by using block move (movsl) instructions and is based on Robert Redelmeier's burnBX test. Memory is initialized with shifting patterns that are inverted every 8 bytes. Then 4MB blocks of memory are moved around using the movsl instruction. After the moves are completed the data patterns are checked. Because the data is checked only after the memory moves are completed it is not possible to know where the error occurred. The addresses reported are only for where the bad pattern was found. Since the moves are constrained to an 8MB segment of memory the failing address will always be lest than 8MB away from the reported address. Errors from this test are not used to calculate BadRAM patterns.

Test 7 [Moving inversions, 32 bit pattern]

This is a variation of the moving inversions algorithm that shifts the data pattern left one bit for each successive address. The starting bit position is shifted left for each pass. To use all possible data patterns 32 passes are required. This test is quite effective at detecting data sensitive errors but the execution time is long.

Test 8 [Random number sequence]

This test writes a series of random numbers into memory. By resetting the seed for the random number generator the same sequence of numbers are created for a reference. The initial pattern is checked and then complemented and checked again on the next pass. However, unlike the moving inversions test writing and checking can only be done in the forward direction.

Test 9 [Modulo 20, random pattern]

Using the Modulo-X algorithm should uncover errors that are not detected by moving inversions due to cache and buffering interference with the algorithm. A sequence of 6 32 bit random patterns are used.

Test 10 [Bit fade test, 2 patterns]

The bit fade test initializes all of memory with a pattern and then sleeps for 5 minutes. Then memory is examined to see if any memory bits have changed. All ones and all zero patterns are used.

A.4 Error Display Information

Memtest86 has 4 options for reporting errors. The default is error summary mode where key failure information is displayed along with an error confidence value. Reporting of individual errors is also available. In BadRAM Patterns mode patterns are created for use with the Linux BadRAM feature. This feature allows Linux to avoid bad memory pages. Details about the BadRAM feature can be found at:

http://home.zonnet.nl/vanrein/badram

The error summary mode reports the following data:

Error Confidence Value:

A value that indicates the validity of the errors being reported with larger values indicating greater validity. There is a high probability that all errors reported are valid regardless of this value. However, when this value exceeds 100 it is nearly impossible that the reported errors will be invalid.

Lowest Error Address:

The lowest address that where an error has been reported.

Highest Error Address:

The highest address that where an error has been reported.

Bits in Error Mask:

A mask of all bits that have been in error (hexadecimal).

Bits in Error:

Total bit in error for all error instances and the min, max and average bit in error of each individual occurrence.

Max Contiguous Errors:

The maximum of contiguous addresses with errors.

ECC Correctable Errors:

The number of errors that have been corrected by ECC hardware.

Test Errors:

On the right hand side of the screen the number of errors for each test are displayed.

When the individual error reporting mode is selected the following information is displayed. An error message is only displayed for errors with a different address or failing bit pattern. All displayed values are in hexadecimal.

Label	***Description***
Tst:	Test number
Failing Address :	Failing memory address
Good:	Expected data pattern
Bad:	Failing data pattern
Err-Bits:	Exclusive or of good and bad data (this shows the position of the failing bit(s))
Count:	Number of consecutive errors with the same address and failing bits

A.5 Serial Console

Memtest86 can be used on PC's equipped with a serial port for the console. By default serial port console support is not enabled since it slows down testing. To enable the serial console requires that Memtest86 be recompiled on a Linux platform. This is done by changing the SERIAL_CONSOLE_DEFAULT define in config.h from a zero to a one and then recompiling. The serial console baud rate may also be set in config.h with the SERIAL_BAUD_RATE define. The other serial port settings are no parity, 8 data bits, 1 stop bit. All of the features used by Memtest86 are accessible via the serial console. However, the screen sometimes is garbled when the online commands are used.

Appendix B.Licensing

Memtest86 is released under the terms of the Gnu Public License (GPL). Other than the provisions of the GPL there are no restrictions for use, private or commercial. See: http://www.gnu.org/licenses/gpl.html for details.

Appendix C.Product Support

Please report problems to:

Email: help@passmark.com

Please include:

- The software version.
- A detailed description of the problem and how to reproduce it.
- A copy of the result files / log files (if available).
- Details of how we may be able to contact you.

C.1 Known Problems

Sometimes when booting from a floppy disk the following messages scroll up on the screen:

X:8000

AX:0212

BX:8600

CX:0201

DX:0000

This is the BIOS reporting floppy disk read errors. Either re-write or toss the floppy disk.

Memtest86 can not diagnose many types of PC failures. For example a faulty CPU that causes Windows to crash will most likely just cause Memtest86 to crash in the same way.

Before submitting a problem report please check the Known Problems section to see if this problem has already been reported. Be sure to include the version number and also any details that may be relevant.

With some PC's Memtest86 will just die with no hints as to what went wrong. Without any details it is impossible to fix these failures. Fixing these problems will require debugging on your part. There is no point in reporting these failures unless you have a Linux system and would be willing to debug the failure.

Memtest86 supports all types of memory. If fact the test has absolutely no knowledge of the memory type nor does it need to. This not a problem or bug but is listed here due to the many questions about this issue.

C.2 Enhancements

Please send enhancement requests to:

info@passmark.com

All requests will be considered, but not all can be implemented

Appendix D.Change Log

Enhancements in v4.3.3 (Sept/2013)

- Fixed incorrect progress calculation for test 4
- Fixed potential false positives in parallel mode caused by overlapped/unaligned memory chunk allocations per CPU
- Fixed program freeze when selecting test 0 or 1 when running in non-parallel mode

Enhancements in v4.3.2 (Aug/2013)

- Memory bandwidth is now measured for one CPU (as opposed to a total for all CPUs)
- Fixed crash when attempting to boot a hyperthread reported by MADT
- Restored the “Start only one CPU” boot option

Enhancements in v4.3.1 (Aug/2013)

- Fixed bug with Test 6 (Block Move Test) not testing the end of a memory segment correctly
- Removed unnecessary boot options in menu

Enhancements in v4.3 (Jul/2013)

- Changed default CPU selection mode to round robin. Running all CPUs at once has been shown to cause false positives on a number of systems.
- Fixed a bug that could cause the program to go into a tight loop that could not be escaped when setting certain memory ranges to test.
- Fixed a bug displaying the memory location of individual errors. The values after the decimal point in the MB readout were incorrect.
- Fixed a bug in configuring upper and lower memory limits, previously lower limits equal or grater than 2gb would not work, as well as some other more obsucre configurations.
- Added a misc option to display the systems memory map.
- Fixed a bug that would cause the number of passes to not correctly reset after changing the selected tests.
- Added missing source code to some of the download packages.
- Fixed a bug in test 8 causing a single error to cascade into multiple errors.
- Fixed a bug causing the average error bits to be incorrect once the errors had maxed out at 65k
- Fixeda bug preventing test 10 to be selected as a single test to run.
- Fixed bug displaying individual test error counts.
- Fixed bug making overall errors 10x what they should be.

Enhancements in v4.2 (Mar/2013)

- Fixed issues with USB keyboards. The USB keyboard functionality is memory mapped into a portion of low memory on some (maybe many) machines, typing on a USB keyboard changes some values in RAM as the key presses are stored in memory as you type. This can cause the keyboard to become unresponsive during testing or input from the keyboard to generate errors in the tests.
- Fixed crash when configuring memory ranges. Changing the meory range during the first test, or changing the memory range multiple times during later tests could cause the current test number to become negative, triggering a crash.
- Fixed highest error address not reporting correctly on error.
- Fixed error counters overflowing and reseting to 0 after too many errors.
- Fixed max contiguous error reporting.
- Cleaned up some UI text.
- The Windows USB package now includes ImageUSB to make creating Memtest86 USB drives easier.

Enhancements in v4.1 (Jan/2013)

- Added a new boot trace option that single steps through the testing process and displays messages and data that is valuable in diagnosing problems with test execution. A large number of trace points have been added in key portions of the code (in particular SMP startup routines) to provide visibility of obscure failures. This feature will allow non-technical users to provide troubleshooting data for better test stability.
- Added a new One Pass feature. This feature runs the complete test once and then exits, but only if there were no errors. This provides a convenient method for unattended testing. One Pass may be enabled via a boot option or via an on-line command.
- Images for CD, USB key and Floppy disks now use Syslinux for booting and include a variety of standard options and two previous versions of Memtest86. The new boot time options may be specified at the boot prompt.
- A feature has been added to allow customization of the list of tests to be run. The test list may be specified via a boot option or via an on-line command.
- A feature has been added to restrict specific CPUs that are to be used for testing. The maximum number of CPUs may be specified or a 32 bit CPU mask may be specified. These are enabled with boot options.
- A number of problem with use of on-line commands when testing with more than one CPU have been fixed.
- A selection of boot time parameters are were added. These options enable boot tracing, the One Pass feature, limit the maximum number of CPUs to use, specify a CPU mask to select CPUs to be used and setup serial console parameters.
- Improved and extended CPU identification routines. Newer CPUID based method is now used to determine cache sizes for Intel CPUs for better accuracy and supportability.

- Routines for calculating cache and memory speeds have been reworked for better accuracy. An overflow problem has been fixed that resulted in no memory speed being reported for CPUs with large L3 caches.
- Fixed some errors in the crash reporting routines.
- Misc minor fixes and code cleanup.

Enhancements in v4.0 (28/Mar/2011)

- Full support for testing with multiple CPUs. All tests except for #11 (Bit Fade) have been multithreaded. A maximum of 16 CPUs will be used for testing.
- CPU detection has been completely re-written to use the brand ID string rather than the cumbersome, difficult to maintain and often out of date CPUID family information. All new processors will now be correctly identified without requiring code support.
- All code related to controller identification, PCI and DMI has been removed.
- This may be a controversial decision and was not made lightly. The following are justifications for the decision:
    1. Controller identification has nothing to do with actual testing ofmemory, the core purpose of Memtest86.
    2. This code needed to be updated with every new chipset. With the ever growing number of chipsets it is not possible to keep up with the changes. The result is that new chipsets were more often than not reported in-correctly. In the authors opinion incorrect information is worse than no information.
    3. Probing for chipset information carries the risk of making the program crash.
    4. The amount of code involved with controller identification was quite large, making support more difficult.
- Removing this code also had the unfortunate effect of removing reporting of correctable ECC errors. The code to support ECC was hopelessly intertwined the controller identification code. A fresh, streamlined implementation of ECC reporting is planned for a future release.
- A surprising number of conditions existed that potentially cause problems when testing more than 4 GB of memory. Most if not all of these conditions have been identified and corrected.
- A number of cases were corrected where not all of memory was being tested.
- For most tests the last word of each test block was not tested. In addition an error in the paging code was fixed that omitted from testing the last 256 bytes of each block above 2 GB.
- The information display has been simplified and a number of details that were not relevant to testing were removed.
- Memory speed reporting has been parallelized for more accurate reporting for multi channel memory controllers.
- This is a major re-write of the Memtest86 with a large number of minor bugfixes and

substantial cleanup and re-organization of the code.

Enhancements in v3.5 (3/Jan/2008)

- Limited support for execution with multiple CPUs. CPUs are selected round-robin or sequential for each test.
- Support for additional chipsets. (from Memtest86+ v2.11).
- Additions and corrections for CPU detection including reporting of L3 cache.
- Reworked information display for better readability and new information.
- Abbreviated iterations for first pass.
- Enhancements to memory sizing.
- Misc fixes and code cleanup

Enhancements in v3.4 (8/Sep/2007)

- A new error summary display with error confidence analysis
- Display of memory module information (from Memtest86+ v1.70)
- Relocated testing reworked to overlap main testing for better error detection
- Support for additional chipsets. (from Memtest86+ v1.70)
- Additions and corrections for CPU identification
- Misc bug fixes and code cleanup

Enhancements in v3.3 (12/Jan/2007)

- Added support for additional chipsets. (from Memtest86+ v1.60)
- Changed Modulo 20 test (#8) to use a more effective random pattern rather than simple ones and zeros.
- Fixed a bug that prevented testing of low memory.
- Added an advanced menu option to display SPD info (only for selected chipsets).
- Updated CPU detection for new CPUs and corrected some bugs.
- Reworked online command text for better clarity.
- Added a fix to correct a Badram pattern bug.

Appendix E.Acknowledgments

Memtest86 was developed by Chris Brady with the resources and generous assistance of individuals and sources listed below:

The initial versions of the source files bootsect.S, setup.S, head.S and build.c are from the Linux 1.2.1 kernel and have been heavily modified.

Doug Sisk provided code to support a console connected via a serial port.

Code to create BadRAM patterns was provided by Rick van Rein.

Tests 5 is based on Robert Redelmeier's burnBX test.

Screen buffer code was provided by Jani Averbach.

Eric Biederman provided all of the feature content for version 3.0 plus many bugfixes and significant code cleanup.

Major enhancements to hardware detection and reporting in version 3.2, 3.3 and 3.4 provided by Samuel Demeulemeester (from Memtest86+ v1.11, v1.60 and v1.70).