MuForce

Thomas Richter

COLLABORATORS			
	TITLE : MuForce		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Thomas Richter	August 25, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 MuForce **1**

1.1 MuForce Guide 1

1.2 The THOR-Software Licence 2

1.3 What's the MMU.library? 3

1.4 What's the job of MuForce? 3

1.5 Installation of MuForce 4

1.6 Command line options and tooltypes 4

1.7 Credits : Thank you goes to... 4

1.8 Writing debuggers that support MuForce 5

1.9 Debuggers: Trapping a hit 5

1.10 Debuggers: Not causing a hit 6

1.11 Processor specific notes 6

1.12 System patches installed by the library 7

1.13 Optimizing the MMU tables 8

1.14 68020 Notes 8

1.15 68030 Notes 8

1.16 68040 Notes 8

1.17 68060 Notes 10

1.18 Option: QUIET 10

1.19 Option: TINY 10

1.20 Option: SMALL 11

1.21 Option: SHOWPC 11

1.22 Option: STACKLINES 11

1.23 Option: STACKCHECK 11

1.24 Option: AREGCHECK 11

1.25 Option: CAPTURESUPER 11

1.26 Option: DREGCHECK 12

1.27 Option: DATESTAMP 12

1.28 Option: DEADLY 12

1.29 Option: FSPACE 12

1.30 Option: VALIDZERO . . . 12
1.31 Option: VERBOSE . . . 12
1.32 Option: LED . . . 13
1.33 Option: PARALLEL . . . 13
1.34 Option: RAWIO . . . 13
1.35 Option: FILE . . . 13
1.36 Option: STDIO . . . 13
1.37 Option: BUFFERSIZE . . . 13
1.38 Option: INTRO . . . 14
1.39 Option: PRIORITY . . . 14
1.40 Option: NOALERTPATCH . . . 14
1.41 Option: NOGURUPATCH . . . 14
1.42 Option: ON . . . 14
1.43 Option: QUIT . . . 14
1.44 Option: NEWVBR . . . 15
1.45 Option: FATALHITS . . . 15
1.46 Option: DISABLEBELL . . . 15
1.47 Option: DISPC . . . 15
1.48 Option: DISRANGE . . . 15
1.49 Option: PREPMUNGWALL . . . 15
1.50 Example MuForce output . . . 16
1.51 Detail Example Hit . . . 17
1.52 Output: Date Stamp . . . 18
1.53 Output: Write Hit . . . 18
1.54 Output: Address hit . . . 19
1.55 Output: Data Write . . . 19
1.56 Output: Program Counter . . . 19
1.57 Output: CPU Status Register . . . 19
1.58 Output: Special information . . . 19
1.59 Output: Task Control Block . . . 19
1.60 Output: MMU Context . . . 19
1.61 Output: Data Register Dump . . . 19
1.62 Output: D0 Register . . . 20
1.63 Output: D1 Register . . . 20
1.64 Output: D2 Register . . . 20
1.65 Output: D3 Register . . . 20
1.66 Output: D4 Register . . . 20
1.67 Output: D5 Register . . . 20
1.68 Output: D6 Register . . . 20

1.69 Output: D7 Register 20
1.70 Output: Address Register Dump 21
1.71 Output: A0 Register 21
1.72 Output: A1 Register 21
1.73 Output: A2 Register 21
1.74 Output: A3 Register 21
1.75 Output: A4 Register 21
1.76 Output: A5 Register 21
1.77 Output: A6 Register 21
1.78 Output: A7 Register 22
1.79 Output: Stack Dump 22
1.80 Output: Stack Word 22
1.81 Output: SegTracker 22
1.82 Output: SegTracker Address 22
1.83 Output: SegTracker Name 22
1.84 Output: Hunk 22
1.85 Output: Offset 22
1.86 Output: Task/Process Name 23
1.87 Output: Task Name 23
1.88 Output: CLI Command Name 23
1.89 Output: Alerts 23
1.90 Output: Alert Number 23
1.91 Output: Show PC 23
1.92 Output: Show PC-$20 23
1.93 Output: Show PC-$1C 23
1.94 Output: Show PC-$18 23
1.95 Output: Show PC-$14 24
1.96 Output: Show PC-$10 24
1.97 Output: Show PC-$0C 24
1.98 Output: Show PC-$08 24
1.99 Output: Show PC-$04 24
1.100 Output: Show PC+$00 24
1.101 Output: Show PC+$04 24
1.102 Output: Show PC+$08 24
1.103 Output: Show PC+$0C 24
1.104 Output: Show PC+$10 24
1.105 Output: Show PC+$14 25
1.106 Output: Show PC+$18 25
1.107 Output: Show PC+$1C 25
1.108 Output: Disassembler output 25
1.109 Output: Disassembler output with PC 25
1.110 Index 25
1.111 History 27

Chapter 1

MuForce

1.1 MuForce Guide

´### ,#. ,#### ,### ####' #### ,#### ,###' ####' #### __________ ,#### ,###' | | ####' #### | ___ ___ | ___ ,#### ,###' ---- | | | | | | |___ ####' #### | | | | | | | ,#####. ,###' . | |___| |___| |_ ___| ####'##. ,#### ,## ,#### ########' # ,##' ####' `####' `###' ,#### ##### © 1999 THOR - Software, ####' Thomas Richter `##'

MuForce Guide

Guide Version 1.11 MuForce Version 40.21

MuForce is based on the Enforcer © 1992-1998 Mike Sinz, check

http://www.users.fast.net/~michael_sinz/Enforcer/index.html

The Licence : Legal restrictions

MuTools : What is this all about, and what's the MMU library?

Credits : Thanks goes to...

What is it : Overview

Installation : How to install MuForce

Synopsis : The command line options and tool types

Output : A tiny example output

Detailed Ex. : A very detailed example output

Notes : Processor specific information

Debugging : Writing debuggers that support MuForce

Optimizing : Change the MMU setup to optimize the performance

History : What happened before

MuForce is provided for software testing. In this role it is vital. Software that causes MuForce hits may not be able to run on newer hardware. (Hits of high addresses on systems not running MuForce but with a 68040 or 68060 will most likely crash the system). Future systems and hardware will make this even more important. The system can NOT survive software that causes hits.

However, MuForce is NOT a system protector. As a side effect, it may well keep a system from crashing when MuForce hits happen, but it may just as well make the software crash earlier. MuForce is mainly a development and testing tool.

MuForce causes no ill effects with correctly working software. If a program fails to work while MuForce is active, you should contact the developer of that program.

When an access violation happens, a report such as the following is output:

Output Example Detail Example

© THOR-Software

Thomas Richter

Rühmkorffstraße 10A

12209 Berlin

Germany

EMail: thor@einstein.math.tu-berlin.de

WWW: http://www.math.tu-berlin.de/~thor/thor/index.html

1.2 The THOR-Software Licence

1.3 What's the MMU.library?

All "modern" Amiga computers come with a special hardware component called the "MMU" for short, "Memory Management Unit". The MMU is a very powerful piece of hardware that can be seen as a translator between the CPU that carries out the actual calculation, and the surrounding hardware: Memory and IO devices. Each external access of the CPU is filtered by the MMU, checked whether the memory region is available, write protected, can be hold in the CPU internal cache and more. The MMU can be told to translate the addresses as seen from the CPU to different addresses, hence it can be used to "re-map" parts of the memory without actually touching the memory itself.

A series of programs is available that make use of the MMU: First of all, it's needed by the operating system to tell the CPU not to hold "chip memory", used by the Amiga custom chips, in its cache; second, several tools re-map the Kickstart ROM to faster 32Bit RAM by using the MMU to translate the ROM addresses - as seen from the CPU - to the RAM addresses where the image of the ROM is kept. Third, a number of debugging tools make use of it to detect accesses to physically unavailable memory regions, and hence to find bugs in programs; amongst them is the "Enforcer" by Michael Sinz. Fourth, the MMU can be used to create the illusion of "almost infinite memory", with so-called "virtual memory systems". Last but not least, a number of miscellaneous applications have been found for the MMU as well, for example for display drivers of emulators.

Unfortunately, the Amiga Os does not provide ANY interface to the MMU, everything boils down to hardware hacking and every program hacks the MMU table as it wishes. Needless to say this prevents program A from working nicely together with program B, Enforcer with FastROM or VMM, and other combinations have been impossible up to now.

THIS HAS TO CHANGE! There has to be a documented interface to the MMU that makes accesses transparent, easy and compatible. This is the goal of the "mmu.library". In one word, COMPATIBILITY.

Unfortunately, old programs won't use this library automatically, so things have to be rewritten. The "MuTools" are a collection of programs that take over the job of older applications that hit the hardware directly. The result are programs that operate hardware independent, without any CPU or MMU specific parts, no matter what kind of MMU is available, and programs that nicely co-exist with each other.

I hope other program authors choose to make use of the library in the future and provide powerful tools without the compatibility headache. The MuTools are just a tiny start, more has to follow.

1.4 What's the job of MuForce?

MuForce is a mmu.library compatible "Enforcer" type debugging tool. MuForce watches illegal memory accesses and reports them over the serial port or any other output stream.

MuForce makes use of the mmu.library to setup the MMU to watch illegal accesses to the zero page or non-existent memory.

If SegTracker is running in the system when MuForce is started, MuForce will use the public SegTracker seglist tracking for identifying the hits.

When an access violation happens, a report such as the following is output:

Output Example Detail Example

1.5 Installation of MuForce

Installation is pretty simple:

- First, install the "mmu.library": Copy this library to your LIBS: drawer if you haven't installed it yet. It's contained in this archive.

- Copy "MuForce" wherever you want.

- Remove other Enforcer tools like the "Enforcer" or the "CyberGuard" and detach MuForce *after* SetPatch, and possibly after SegTracker. Just start it like:

run >NIL: <NIL: MuForce

plus all the options you wish.

You can also run "MuForce" from the workbench, it will read its options from the tooltypes in this case.

1.6 Command line options and tooltypes

MuForce can be started either from the workbench or from the shell. In the first case, it reads its arguments from the "tooltypes" of its icon; you may alter these settings by selecting the "MuForce" icon and choosing "Information..." from the workbench "Icon" menu. In the second case, the arguments are taken from the command line. No matter how the program is run, the arguments are identically.

The options for MuForce are as follows:

QUIET DATESTAMP STDIO TINY DEADLY BUFFERSIZE SMALL FSPACE INTRO SHOWPC VERBOSE PRIORITY STACKLINES LED NOALERTPATCH STACKCHECK PARALLEL ON AREGCHECK RAWIO QUIT DREGCHECK FILE NEWVBR VALIDZERO FATALHITS DISABLEBELL DISPC DISRANGE PREPMUNGWALL NOGURUPATCH CAPTURE-SUPER

When started from the workbench, MuForce knows one additional tooltype, namely:

WINDOW=<path>

where <path> is a file name path where the program should print its output. This should be a console window specification, i.e. something like

CON:0/0/640/100/MuForce

This argument defaults to NIL:, i.e. all output will be thrown away.

It is possibly a good idea, in case no serial terminal is available, to set the window to a console like in the example above, and use the tooltype

FILE=*

to redirect the MuForce output to that window.

1.7 Credits : Thank you goes to...

Michael Sinz (a real *BIG* thank you!) for discussing a lot of details of CachePreDMA/CachePostDMA, for sending me the sources of these functions in his 68040, and especially - and that's really great - for making the Enforcer sources available.

Ralph Babel for giving information about the CachePreDMA/CachePostDMA functions and for taking the time and helping me a bit with the messy details.

Bjoern Schmidt for allowing me to run some tests on his 060.

Werner Müller for his 040 based system.

All the testers for running tests and sending me detailed information about their systems. Thank you to all of you, this project won't clearly possible without your support!

1.8 Writing debuggers that support MuForce

If you wish to make a debugger that works with MuForce to help pinpoint MuForce hits in the application and not cause hits itself, here are some simple tips and a bit of code:

Debuggers: Trapping a hit

Debuggers: Not causing a hit

Note that none of the above is identically to the "Enforcer" hints, with the mmu.library in your system, more powerful and compatible methods are available.

1.9 Debuggers: Trapping a hit

To trap a hit requires a number of things to work.

First, the debugger itself must never cause a hit, see Debuggers: Not causing a hit.

Second, the debugger must start *AFTER* MuForce starts. If MuForce is detached from the shell, give it a few seconds to setup the MMU tables for you. If it is started before MuForce, the hits will not be trapped. Note that this is not a problem, but is at best confusing.

The preferred method of writing a debugger is by using the mmu.library exception handler mechanism. It can be setup to trap hits of a special task only, or for all tasks at once.

The following code shows how to setup a global exception handler:

/* Setup the exception handler */ MuForceHook=AddContextHook(MADTAG_CONTEXT,ctx, MADTAG_TYPE,MMUEH_SEGFAU MADTAG_CODE,&NewException, MADTAG_DATA,getreg(REG_A4), /* MADTAG_TASK, mytask for non-global debuggers */ MADTAG_NAME,muname, MADTAG_PRI,-64, TAG_DONE);

if (MuForceHook) { ActivateException(MuForceHook);

/* we're ready to receive hits... */

}

To remove the exception hook, first call "DeactivateException()", then "RemContextHook()". For details about these functions, read the autodocs of the mmu.library.

"NewException" should be preferably an assembly language routine that is called as soon as an exception is reported. Here's the MuForce exception handler as an example:

;** ;** _NewExcept ** ;** ** ;** the new exception handler ** ;** We are called like this: ** ;** *a0 = ExceptionData ** ;** *a4 = Data segment ** ;** ** ;** we must return with d0=0 and an eq-flag ** ;** d0-d1/a0-a1/a4-a5 can be used ** ;** xdef _NewException _NewException: move.l a0,a5 ;keep *ExceptData in a5 (a5 is a scratch)

bsr FlashLED ;flash the power LED

bsr Main_Disp ;display the intro bsr Date_Disp ;and the date bsr Info_Disp ;display the main information ;note that physical faults ;are never shown here ;because they aren't handled ;by the library at all... tst.l _Tiny_Flag(a4) ;if tiny, this is all bne.s .exit

;** ;** now display the SR. We don't display the SSW because this ;** is processor dependent and not available. All information we ;** need is here. ;**

bsr USP_Disp ;display this information

movec.l usp,a0 lea exd_DataRegs(a5),a1 move.l exd_ReturnPC(a5),d0 bsr RegDisp ;display registers & stack

move.l exd_ReturnPC(a5),d0 bsr PCDisp ;display PC

move.l exd_ReturnPC(a5),a1 bsr NameDisp ;display hunk & name

.exit: ori.w #$C000,$124(a6) ;fake scheduling. This is a hack! move.w #INTF_SETCLR!INTF_SOFTINT,_intreq ; softint move.l _Bad_ReadValue(a4),exd_Data(a5) ;set the return data bsetm EXDB_READBACK,exd_Flags(a5) ;read it back, or declare the write for done

move.l exd_MMUBase(a5),a6 ;restore MMUBase for the handler (a6 is not a scratch) moveq #0,d0 ;the handler is done and handled the fault rts

This code prints debugging information and sets the EXDB_READBACK flag to continue the faulty program. A different approach would be to set the EXDB_CALL flag, provide a function pointer in exd_ReturnPC and to let the library call a user level routine for you. This routine could, for example, sent out a signal and wait for a return signal. For details how this is done, consider the "Exception.doc" file in the autodocs drawer of the "mmu.library" distribution.

1.10 Debuggers: Not causing a hit

In order not to cause hits, you can do a number of things. The easiest is to test the address with the TypeOfMem() EXEC function. If TypeOfMem() returns 0, the address is not in the memory lists. However, this does not mean it is not a valid address in all cases. (ROM, chip registers, I/O boards). Since a number of boards are available that add hardware and bypass the expansion library AutoConfig mechanism (bah!), it is no longer safe to check the expansion library like proposed in the enforcer documentation. Instead, consider using the functions of the MMU library to check out whether an access is legal or not. The code below is more or less taken from the "MuForce" sources. It requires a pointer to a MMU library "context"; for "standard" Amiga applications, the default context is good enough, it is returned by the "DefaultContext()" call of the library. If you're catching a hit from an external task, you should call "CurrentContext()" to get the currently active one, or read the context field from exception data structure if you call this within an exception hook (note that this is perfectly legal, unlike other calls).

;** ;** IsValidPi ** ;** check whether the address *a0 is valid ** ;** returns NE if this is not so. ** ;** *a5=ExceptionData ** ;** IsValidPi: saveregs d0-d2/a0-a3/a6

move.l a0,a3 ;keep it move.l _MMUBase(a4),a6 move.l exd_Context(a5),a0 jsr GetPageSize(a6) ;read the page size lea -1+(4*8)(a3),a1 ;end of range to check move.l a3,d1 move.l a1,d2 neg.l d0 ;get page mask eor.l d1,d2 ;which bits differ ? and.l d0,d2 ;and page mask: Is this misaligned ? beq.s .aligned

move.l exd_Context(a5),a0 ;check end address first deftag endtag GetPagePropertiesA(a6),a2

and.l #MAPP_IO!MAPP_SWAPPED!MAPP_INVALID,d0 ;anything which would bus-error us? bne.s .buserror

.aligned: move.l a3,a1 move.l exd_Context(a5),a0 ;read the context deftag endtag GetPagePropertiesA(a6),a2

and.l #MAPP_IO!MAPP_SWAPPED!MAPP_INVALID,d0 ;anything which would bus-error us?

.buserror: loadregs rts

Note that this code is completely CPU independent and callable even from interrupts or exceptions. It will respect even weird hardware that is linked into the system without using the AutoConfig mechanism.

1.11 Processor specific notes

General Notes about the MMU library

System patches installed

The following notes discuss processor specific features of the mmu.library and its exception handling. Remember that MuForce itself does not contain *any* CPU specific code at all.

68020 Notes

68030 Notes

68040 Notes

68060 Notes

Bridge-Board

1.12 System patches installed by the library

This is the MuForce release V40. You may think of it as a V40 release of Michael Sinz's Enforcer V37.

Let's see what Mike has to say about the history of the Enforcer. What is said here about the Enforcer goes for "MuForce" as well:

Bryce Nesbitt came up with the original "Enforcer" that has been instrumental to the improvement in the quality of software on the Amiga. The Amiga users and developers owe him a great deal for this. Thank you Bryce! Enforcer V37, however, is a greatly enhanced and more advanced tool.

Enforcer V37 came about due to a number of needs. These included the need for more output options and better performance. It also marks the removal of all kludges that were in the older versions. Also, some future plans required some of these changes...

...

Since AbsExecBase is in low memory, reads of this address are slower with Enforcer running. Caching AbsExecBase locally is highly recommended since it is in CHIP memory and on systems with FAST memory, it will be faster to access the local cached value. (In addition to the performance increase when running Enforcer) Note that doing many reads of location 4 will hurt interrupt performance.

I consider MuForce as the next step in the development history. It is compatible to the mmu.library and all programs that make use of it to access the MMU. It is fully CPU independent, no processor specific code is required.

The mmu.library patches the EXEC function ColdReboot() in an attempt to "get out of the way" when someone tries to reboot the system. It will clean up as much as possible the MMU tables and then call the original LVO. It will also patch the EXEC Alert() function and remove itself in case a dead-end alert is created. This will allow the system to reboot smoothly. Unfortunately, the original Alert() routine does *not* call the ColdReboot() vector itself, hence the need for this kludge. The library will refuse to get flushed if it can't remove its patches.

MuForce will patch the EXEC Alert() function, too, attempting to provide tracking of other alert events in the system. The output of the Alert() patch looks now very like the typical "hit" output, and is as configurable as it. See Michael Sinz's "LawBreaker" notes for a more complete example of this. This can be disabled by the NOALERTPATCH command line argument.

Starting with release 40.21, MuForce will also patch AddTask(), and replaces the default exception handlers of exec and the dos.library by its own exception handler. This means that all MC68K exceptions offending tasks can't handle themselves will be routed to MuForce. It will, however, not stop the faulty task but run afterwards into the usual "Software error" requester where a post-mortem debugger might be able to catch the task. This function can be disabled with the NOGURUPATCH option.

If the CAPTURESUPER option is found on the command line, MuForce will also replace some exception handlers by its own handler to be able to capture supervisor exceptions. The installed exception handler first tries to re-direct the exception to the task-specific trap code, and runs only into the MuForce handler in case the exception was generated in supervisor mode. The following exception handlers are replaced:

003 0x00c Address Error 004 0x010 Illegal Instruction 005 0x014 Integer Divide by Zero 006 0x018 CHK,CHK2 Instruction 007 0x01c FTRAPcc, TRAPcc, TRAPV instruction 010 0x028 Line A emulator 013 0x034 Coprocessor Protocol Violation 014 0x038 Format Error 024 0x060 Spurious Interrupt 032 0x080 Trap #0 : : : 047 0x0bc Trap #15 056 0x0e0 MMU Configuration Error 057 0x0e4 MMU Illegal Operation Error 058 0x0e8 MMU Access Level Violation Error

and finally, the following entry is always patched for the MuForce output handler:

025 0x064 Level 1 Interrupt Autovector

Other notes - these are about the library since MuForce itself does not depend on the CPU it is running on:

68020 Notes

68030 Notes

68040 Notes

68060 Notes

Bridge-Board

1.13 Optimizing the MMU tables

The MMU table setup by the mmu.library or 68040/68060.library should work fine for your system, but need not to be optimal for your hardware, which is in worst case not recognizable by them. Well designed extension hardware should be "auto-configuring", meaning - roughly speaking - it should make the Os know where it is. Unfortunately, some manufacturers decided not to implement this feature, either requiring a special version of the 68040/68060.library or the need to customize the MMU table with MuSetCacheMode.

To give one example, some hardware shows up in the so-called "F space" which was reserved for internal use of Commodore. To include this hardware, the following command would be enough:

MuSetCacheMode from 0x00f00000 size 0x00080000 CACHEINHIBIT IO

Another (mis-)used place for this hardware is in locations 0xfff00000 and above.

If you use a third party 68040 or 68060.library, this library might build MMU tables specialized to the hardware of this manufacturer. In case this hardware is not available for your system, you should tell MuForce so, to be able to detect - then illegal - accesses to this unused address space. The following lines give an example how to define memory regions as invalid. Use them only in case you installed MuForce before, or failed accesses into these regions will cause a guru!

MuSetCacheMode from 0xfff00000 size 0x00100000 invalid MuSetCacheMode from 0x00f00000 size 0x00080000 invalid

In case you know precisely which address in your machine is used for what, check the MuScan output for memory regions that are erroneously marked as "valid", and add lines like the above to the startup-sequence to be able to detect accesses into these regions as well.

1.14 68020 Notes

The 68020 does not have a built-in MMU but has a co-processor feature that lets an external MMU be connected. The library comes with a module for the 68851 specific calls, but since the 68030 MMU and 68851 are very alike, 68020/68851 users should see the 68030 NOTES section.

1.15 68030 Notes

The 68030 uses cycle/instruction continuation and will supply the data on reads and ignore writes during an access fault rather than let the real bus cycle happen. This means that on a fault caused by MMU tables, no bus cycle to the fault address will be generated. (For those of you with analyzers)

In some cases, the 68030 will have advanced the Program Counter past the instruction by the time the access fault happens. This is usually only on WRITE faults. For this reason, the PC may either point at the instruction that caused the fault or just after the instruction that caused the fault.

Note that there is a processor called 68EC030. This processor has a disabled or defective MMU. The library is smart enough to check for a true 68030, though, and MuForce will refuse to get installed if no MMU is available.

1.16 68040 Notes

The mmu.library, on the 68040, *requires* that the "68040.library" is installed and it requires an MMU 68040 CPU. The 68EC040 does not have a MMU and the mmu.library is smart enough to detect this processor and tell MuForce not to install on this system; however, an 040 based Amiga system will *NOT* be able to operate reliable if any DMA device has been added to the system. I doubt any Amiga model makes use of this processor for that reason. The 68LC040 does have an MMU and is supported.

Due to the design of the 68040, the mmu.library exception handler and the general MMU handling is much trickier for the 68040 than for the 68030. For example, the MMU page size can only be either 8K or 4K. This means that to protect the low 1K of memory, MuForce will end up having to mark the first 4K of memory as invalid. It is the job of the library exception handler to emulate the access to the 3K of that memory that is valid. For this reason MuForce *tries* to move a number of possible structures from the first 4K of memory to higher addresses. This means that the system will continue to run at a reasonable speed. The first time MuForce is run it may need to allocate memory for these structures that it will move, this memory is released, though, when MuForce quits. Unfortunately, not all structures can be relocated, like the "copper list" of the graphics library which is build very early on startup.

In addition to the fact that the 68040 MMU table size is different, the address fault handling is also different. Namely, the 68040 can only rerun the cycle and not continue it like the 68030. This means that on a 68040, the page must be made available first and then made unavailable. To make this work, the mmu.lib exception handler will switch the instruction that caused the error into trace mode and let it run with a special MMU setup. When the trace exception comes in, the MMU is set back to the way it was. MuForce does its best to keep debuggers working. Note, however, that the interrupt level during a trace of a hit will end up being set to 7. This is to prevent interrupts from changing the order of trace/MMU table execution. The level will be restored to the original state before continuing. Since T0 mode tracing is also supported, there are also some changes in the way it operates. T0 mode tracing is defined, on the 68040, to cause a trace whenever the instruction pipeline needed to be reloaded. While on the 68020/030 processors this was normally only for the branch instructions, in the 68040 this includes a large number of other instructions, including a NOP! Anyway, if an hit happens while in T0 tracing mode, the trace will happen even on instructions that normally would not cause a T0 mode trace. Since this may actually help in debugging and because it was not possible to do anything else, this method of operation is deemed acceptable.

Another issue with the 68040 is that WRITE faults happen *after* the instruction has executed, except for MOVEMs, and a number of floating point instructions that access more than two longwords at once, e.g. fmove of an extended precision number, and fmovem. In fact, it is common for the 68040 to execute one or more extra instructions before the WRITE fault is executed. This design makes the 68040 much faster, but it also makes the Program Counter value that MuForce, or the library exception vector in general, can report for the fault much less likely to be pointing to the instruction that caused it. The worst cases are sequences such as a write fault followed by a branch or "jump subroutine" instruction. In these cases, the branch is usually already executed before the write fault happens and thus the PC will be pointing to the target of the branch. There is nothing that can be done to help out here. You will just need to be aware of this and deal with it as best as possible.

Along with the above issue, is the fact that since a write fault may be delayed, a read fault may happen before the write fault shows up. Internally, the mmu.library calls its exception handlers twice (or even thrice) to report all hits; note that this is different to how the Enforcer worked. The library provides service for virtual memory system and has therefore to care about each single access fault. Along the same lines, the hit generated from a MOVEM instruction may only show as a single hit rather than one for each register moved, same goes for fmove.x and fmovem, even though the last two are even more tricky; the MuForce handler will not print any information about the size of the access in this case, but the library provides the necessary information. It was felt that this is a minor problem. Even more problematic are access faults by FPU instructions that read or write more than two longwords at once. They only show up as a single (or at most two, for misaligned accesses) hit, and there's nothing the library could do for you to provide more detailed information; they will show up as a single long word move, even though they might access more bytes at once; the 040 will call the exception handler again in case the instruction was misaligned, even though the "misaligned" flag is not set correctly. Call this a "feature" (if not a bug) of the 040 design. Decoding the instruction at the fault PC is unfortunately not an option because the PC might have advanced already. However, all this should not be a problem for virtual memory systems because each single access will cause the correct page to be swapped in.

On the Amiga, MOVE16 is not supported 100%. Causing an access fault with a MOVE16 will cause major problems. Since the library can't handle these, a real GURU will be generated. Same goes for true hardware bus errors, BTW.

Another rather problematic issue are instructions that make use of double indirection, as for example "move.l ([a4]),d0". Due to a bug in the firmware of the 68040 (and the 68060), only one access fault is reported, and continuing (or restarting, on the 68060) this instruction will make the CPU to forget one indirection level; hence, it will return the wrong data. There's nothing I can do against this, just avoid these instructions. *Sigh*

The functions CachePreDMA(), CachePostDMA(), and CacheControl() are patched by the mmu.library, first to handle the cache of the 68040, and secondly to provide the logical to physical translation required by DMA devices. The library refuses to get flushed when the patches can't be removed safely. Note that not all DMA devices call these functions as they should, and therefore memory re-mapping is not as reliable as it should. I will try to make patches for certain third party devices available. Thanks to Ralph Babel, the "omniscsi.device" is already supported.

1.17 68060 Notes

The mmu.library, on the 68060, *requires* that the "68060.library" be installed. The mmu.library works even with broken 68060 libraries that do not provide all the functions they should. The CachePre/PostDMA() functions - missing for some of the libraries - are replaced by smarter functions of the mmu.library anyway.

The 68060 exception model is full-restart, which means that all instructions are re-run. Both reads *and* writes. This means that the mmu.library exception handler has to do heavy trickery to provide the data that was written out on the fault. This means that faults happen before the instruction is executed (usually) and thus the reported PC will be more exact. This restart model also means that if an real bus-fault happens, the mmu.library will be unable to do much other than let it happen, it is simply not handled and passed over to the GURU, as for all other CPUs. The library is supposed to be MMU library, and not a general "fix all my bus error problems" library. In case you need to handle physical bus errors, please install your own handler before the library is loaded. MuForce maps all addresses as either valid based on system configuration or invalid. This is so that no address should cause a bus fault unless the system configuration is incorrect and an address that was marked valid actually causes a fault, or a part of the hardware is defective.

Be sure to read the 68040 notes as the 68060 is a superset of much of these notes.

Due to the complexity of emulating access to lower memory and the fact that the 68060 was introduced well after V39 kickstart, it is highly recommended that you use V39 or better with 68060 CPUs. This mainly has to deal with lower 4K of memory. As of V38 of exec.library, 68040/68060 processors would map out the lower 4K of RAM rather than just 1K. This was required since the newer CPUs did not have page sizes less than 4K.

It turns out that some 68060 CPU cards also have other hardware on them. This is not a problem, even if this hardware does not auto-configure provided the 68060.library MMU table is correct. Nevertheless, I would call these designs as broken. A common location for such hardware control registers to be placed is in the reserved $00F00000 address range (known as F-Space). This 512K space was reserved for future Kickstart growth. It also has some magic in it so that you can wedge special startup routines there for things like 68060 cards. At least one vendor is doing all of this correctly and has even made sure that expansion.library knows about the hardware that is located in the $00F00000 address range.

If, when running MuForce, your machine does lock up *and* when you run MuForce with the VERBOSE option it does not say that the $00F00000 address range is a "board address" then you may wish to try the FSPACE option to see if this is the reason. If this does not fix the problem, you more than likely have either a real bug or some other non-AutoConfig hardware in your system. First try to use the 68060.library that came with your board to fix this problem, in case that doesn't help, you should contact the board vendor.

1.18 Option: QUIET

QUIET/S

This tells MuForce not to complain about any invalid access and to just open and lock the mmu.library in memory. It does nothing else. This option is mainly provided for Amiga Bridge-Board users in a 68030 environment so that the system can run with the data cache turned on. In this case,

```
RUN >NIL: <NIL: MuForce QUIET
```

should be placed into the startup-sequence right after SetPatch.

1.19 Option: TINY

TINY/S

This tells MuForce to output a minimal hit. The output is basically the first line of the hit.

1.20 Option: SMALL

SMALL/S

This tells MuForce to output the hit line, the USP: line, and the Name: line. This means that no register or stack display will be output.

1.21 Option: SHOWPC

SHOWPC/S

This tells MuForce to also output the two lines that contain the memory area around the PC where the hit happened. Useful for disassembly. This option will not do anything if QUIET, SMALL or TINY output modes are selected.

1.22 Option: STACKLINES

STACKLINES/K/N

This lets you pick the number of lines of stack back-trace to display. The default is 2. If set to 0, no stack back-trace will be displayed. There is NO ENFORCED LIMIT on the number of lines.

NOTE: Unlike for the enforcer, this works, too, for the lines printed for alerts.

1.23 Option: STACKCHECK

STACKCHECK/S

This option tells MuForce that you wish all of the long words displayed in the stack to be checked against the global seglists via "SegTracker". This will tell you what seglist various return addresses are on the stack. If you are not displaying stack information in the hit then STACKCHECK will have nothing to check. If you are displaying stack information, then each long word will be check and only those which are in one of the tracked seglists will be displayed in a "SegTracker" line. The output will show the PC address first and then work its way back on the stack such that you can read it from bottom up as the order of calling or from top down as the stack-frame back-trace.

NOTE: Unlike for the enforcer, this works, too, for the lines printed for alerts.

1.24 Option: AREGCHECK

AREGCHECK/S

This option tells MuForce that you wish all of the values in the Address Registers checked via "SegTracker", much like STACKCHECK.

1.25 Option: CAPTURESUPER

CAPTURESUPER/S

This option tells MuForce that it should even try to capture exceptions that are caused in supervisor mode. In case this exception is part of an interrupt procedure, MuForce might be able to print the exception message, but it won't be able to stop the interrupt. Supervisor code called from within a task context will, however, halt the task.

1.26 Option: DREGCHECK

DREGCHECK/S

This option tells MuForce that you wish all of the values in the Data Registers checked via "SegTracker", much like STACKCHECK.

1.27 Option: DATESTAMP

DATESTAMP/S

This makes MuForce output a date and time with each hit. Due to the nature of the way MuForce must work, the time can not be read during the hit itself so the time output will be the last time value the main MuForce task set up. MuForce will update this value every second as to try to not use any real CPU time. The time displayed in the hit will thus be exact. (Assuming the system clock is correct.) The date is output before anything from the hit other than the optional introduction string.

1.28 Option: DEADLY

DEADLY/S

This makes MuForce a bit nasty. Normally, when an illegal read happens, MuForce returns 0 as the result of this read. With this option, MuForce will return $ABADFEED as the read data. This option can make programs with hits cause even more hits.

1.29 Option: FSPACE

FSPACE/S

This option will make the special $00F00000 address space available for writing to. This is useful for those people with $00F00000 boards. Mainly Commodore internal development work -- should only be used in that environment.

1.30 Option: VALIDZERO

VALIDZERO/S

If this command line switch is given, MuForce will not restrict accesses to the zero page. Note that this will make MuForce unable to catch most hits - since most bad programs will access the zero page erroneously. However, this option is required to be able to run Mac emulators like the ShapeShifter happy since the MacOs keeps global data in this area.

You *should not* use this option unless you really MUST run ShapeShifter with MuForce. This option weakens the power of MuForce considerably.

1.31 Option: VERBOSE

VERBOSE/S

This option will make MuForce print the MMU output much like MuScan. Since MuForce does not build the MMU tables itself, no board information can be printed. For details about the meaning of this output, please check the "MuScan" guide. This information maybe useful in specialized debugging.

1.32 Option: LED

LED/K/N

This option lets you specify the speed at which the LED will be toggled for each MuForce hit. The default is 1 (which is like it always was). Setting it to 0 will make MuForce not touch the LED. Using a larger value will make the flash take longer (such that it can be noticed when doing I/O models other than the default serial output) The time that the flash will take is a bit more than 1.3 microseconds times the number. So 1000 will be a bit more than 1.3 milliseconds. (Or 1000000 is a bit more than 1.3 seconds.)

1.33 Option: PARALLEL

PARALLEL/S

This option will make MuForce use the parallel port hardware rather than the serial port for output.

1.34 Option: RAWIO

RAWIO/S

This option will make MuForce stuff the hit report into an internal buffer and then from the main MuForce process output the results via the RawPutChar() EXEC debugging LVO. Since the output happens on the MuForce task it is possible for a hit that ends in a system crash to not be able to be reported. This option is here such that tools which can redirect debugging output, like Sushi or Sashimi, can redirect the output too.

1.35 Option: FILE

FILE/K

This option will make MuForce output the hit report but to a file instead of sending it to the hardware directly or using the RAWIO LVO. A good example of such a file is CON:0/0/640/100/HIT/AUTO/WAIT or just * to print the output to the console MuForce was started at. Another thing that can be done is to have a program sit on a named pipe and have MuForce output to it. This program can then do whatever it feels like with the hits. (Such as decode them, etc.)

NOTE: It is not a good idea to have MuForce hits go to a file on a disk as if the system crashes during/after the MuForce hit, the disk may become corrupt.

1.36 Option: STDIO

STDIO/S

This option will make MuForce output the hit report to STDOUT. This option only works from the CLI as it requires STDOUT. It is best used with redirection or pipes.

1.37 Option: BUFFERSIZE

BUFFERSIZE/K/N

This lets you set MuForce's internal output buffer for the special I/O options. This option is only valid with the RAWIO, FILE, or STDIO options. The minimum setting is 8192, which is the default, too. Having the right amount of buffer is rather important for the special I/O modes. The reason is due to the fact that no operating system calls can be made from a bus error. Thus, in the special I/O mode, MuForce must store the output in this buffer and, via some special magic, wake up the MuForce task to read the buffer and write it out as needed. However, if a task is in Forbid() or Disable() when the MuForce hit happens, the MuForce task will not be able to output the results of the hit. This buffer lets a number of hits happen even if the MuForce task was unable to do the I/O. If the number of hits that happen before the I/O was able to run gets too large, the last few hits will either be cut off completely or contain only partial information.

1.38 Option: INTRO

INTRO/K

This optional introduction string will be output at the start of every hit. For example: INTRO="*NBad Program!" The default is no string.

1.39 Option: PRIORITY

PRIORITY/K/N

This lets you set MuForce's I/O task priority. The default for this priority is 99. In some special cases, you may wish to adjust this. It is, however, recommended that if you are using one of the special I/O options (RAWIO, FILE, or STDIO) that you keep the priority rather high. If the priority you supply is outside of the valid task priority range (-127 to 127) MuForce will use the default priority.

1.40 Option: NOALERTPATCH

NOALERTPATCH/S

This option disables the patching of the EXEC Alert() function. Normally MuForce will patch this function to provide information as to what called Alert() and to prevent the MuForce hits that a call to Alert() would cause.

1.41 Option: NOGURUPATCH

NOGURUPATCH/S

This option tells MuForce not to try to capture all other MC68K exceptions, known as "hardware" gurus. The most important exceptions of this kind are the "illegal instruction", the "line a", the "line f" and the "privilege violation" exception.

After having printed the exception, MuForce will either start the DOS exception handler which will show up the "Software Alert" requester, or it will halt the task by running into a Wait(SIGBREAKF_CTRL_E) where a postmortem debugger might capture it. At this time, the register d2 contains the guru number, register a2 is the user stack pointer, and the user registers d0-d7/a0-a6 are pushed to the stack.

1.42 Option: ON

ON/S

Mainly for completeness. If not specified, it is assumed you want to turn ON MuForce.

1.43 Option: QUIT

QUIT=OFF/S

Tells MuForce to quit. MuForce can also be stopped by sending a CTRL-C to its process.

1.44 Option: NEWVBR

NEWVBR/S

With this option given, MuForce will always relocate and rebuild the interrupt vector base. Without this option, MuForce tries to re-use the vector base already present, if possible.

1.45 Option: FATALHITS

FATALHITS/S

Tells MuForce to dump the hit, but not to continue execution of the faulty program. MuForce will instead pass the error thru to the default exception handler, which is usually the system alert requester. This might be useful in conjunction with a debugger.

1.46 Option: DISABLEBELL

DISABLEBELL/S

Disables printing of the ASCII bell (BEL 0x07) that beeps the display on external terminals or on Sushi output.

1.47 Option: DISPC

DISPC/S

Enables disassembly of the code around the fault, which is most useful for debugging and for having an immediate impression what the bug might be.

This option requires the disassembler.library which is included in the distribution.

1.48 Option: DISRANGE

DISRANGE/K/N

Specifies the approximate number of bytes to disassemble around the fault if the DISPC option is turned on. This number is by default 32, i.e. MuForce will disassemble at least 32 bytes ahead and below of the fault.

1.49 Option: PREPMUNGWALL

PREPMUNGWALL/S

Enables a backwards compatibility hack for the "MungWall" tool, making MungWall believe that the old "Enforcer" is running. This hack should be used with care! First of all, MungWall is obsolete and has been replaced by the more powerful "MuGuardianAngel". Second, this hack is not even needed provided you start MungWall in front of MuForce. Third, it may well be that this hack causes problems with Sushi. If that turns out to be a problem, use the more advanced "Sashimi" tool instead, or even better, do not use this hack and run "MuGuardianAngel".

1.50 Example MuForce output

Example MuForce output

```
03-Jul-99 14:12:23 WORD WRITE to 00000000 data=0000 PC: 01044774 USP : 010D07B8 SR: 0004 (U0)(-)(-) TCB: 0101A8C8
Data: DDDD0000 DDDD1111 DDDD2222 DDDD3333 01044722 DDDD5555 DDDD6666 DDDD7777 Addr: AAAA0000
AAAA1111 AAAA2222 AAAA3333 AAAA4444 01044722 00002920 001FFFC4 Stck: 00000000 00FA06D6 00001000 0101B2AC
01F80000 00E9A8F0 01077004 0A0A0104 Stck: 30E80000 010D09F0 010D07F0 010D0BF0 00000200 000000BC 00000020
010D0814 PC-8: 2222263C DDDD3333 280D2A3C DDDD5555 2C3CDDDD 66662E3C DDDD7777 31C00000 PC *: 4EA-
EFF7C 20144EAE FF8811C1 01014EAE FF7621C0 01024EAE FF822E3C 35000000 01044764 : 2c3c dddd 6666 move.l
#-$2222999a,d6 0104476a : 2e3c dddd 7777 move.l #-$22228889,d7 01044770 : 31c0 0000 move.w d0,$0.w 01044774 : *4eae
ff7c jsr -$84(a6) 01044778 : 2014 move.l (a4),d0 0104477a : 4eae ff88 jsr -$78(a6) 0104477e : 11c1 0101 move.b d1,$101.w
01044782 : 4eae ff76 jsr -$8a(a6) Name: "Initial CLI" CLI: "lawbreaker" Hunk 0000 Offset 00000074
```

```
03-Jul-99 14:12:23 LONG READ from AAAA4444 PC: 01044778 USP : 010D07B8 SR: 0015 (U0)(F)(-) TCB: 0101A8C8
Data: DDDD0000 DDDD1111 DDDD2222 DDDD3333 01044722 DDDD5555 DDDD6666 DDDD7777 Addr: AAAA0000
AAAA1111 AAAA2222 AAAA3333 AAAA4444 01044722 00002920 001FFFC4 Stck: 00000000 00FA06D6 00001000 0101B2AC
01F80000 00E9A8F0 01077004 0A0A0104 Stck: 30E80000 010D09F0 010D07F0 010D0BF0 00000200 000000BC 00000020
010D0814 PC-8: DDDD3333 280D2A3C DDDD5555 2C3CDDDD 66662E3C DDDD7777 31C00000 4EAEFF7C PC *:
20144EAE FF8811C1 01014EAE FF7621C0 01024EAE FF822E3C 35000000 4EAEFF94 01044768 : 6666 bne.s $10447d0
0104476a : 2e3c dddd 7777 move.l #-$22228889,d7 01044770 : 31c0 0000 move.w d0,$0.w 01044774 : 4eae ff7c jsr -$84(a6)
01044778 : *2014 move.l (a4),d0 0104477a : 4eae ff88 jsr -$78(a6) 0104477e : 11c1 0101 move.b d1,$101.w 01044782 : 4eae
ff76 jsr -$8a(a6) 01044786 : 21c0 0102 move.l d0,$102.w Name: "Initial CLI" CLI: "lawbreaker" Hunk 0000 Offset 00000078
```

```
03-Jul-99 14:12:23 BYTE WRITE to 00000101 data=11 PC: 00002896 USP : 010D07B4 SR: 0010 (U0)(F)(D) TCB: 0101A8C8
Data: 00000000 DDDD1111 DDDD2222 DDDD3333 01044722 DDDD5555 DDDD6666 DDDD7777 Addr: AAAA0000
AAAA1111 AAAA2222 AAAA3333 AAAA4444 01044722 00002920 001FFFC4 Stck: 01044786 00000000 00FA06D6 00001000
0101B2AC 01F80000 00E9A8F0 01077004 Stck: 0A0A0104 30E80000 010D09F0 010D07F0 010D0BF0 00000200 000000BC
00000020 PC-8: 17004EF9 0105A1A0 4EF90105 A1B44EF9 00F81674 4EF900F8 164A4EF9 00F8162E PC *: 4EF900F8
29344EF9 00F8292C 4EF900F8 18A04EF9 00F81892 4EF901F7 541A4EF9 00002884 : 4ef9 00f8 1674 jmp $f81674 0000288a
: 4ef9 00f8 164a jmp $f8164a 00002890 : 4ef9 00f8 162e jmp $f8162e 00002896 : *4ef9 00f8 2934 jmp $f82934 0000289c :
4ef9 00f8 292c jmp $f8292c 000028a2 : 4ef9 00f8 18a0 jmp $f818a0 Name: "Initial CLI" CLI: "lawbreaker"
```

```
03-Jul-99 14:12:23 LONG WRITE to 00000102 data=00000000 PC: 0104478A USP : 010D07B8 SR: 0014 (U0)(-)(D) TCB:
0101A8C8 Data: 00000000 DDDD1111 DDDD2222 DDDD3333 01044722 DDDD5555 DDDD6666 DDDD7777 Addr: AAAA0000
AAAA1111 AAAA2222 AAAA3333 AAAA4444 01044722 00002920 001FFFC4 Stck: 00000000 00FA06D6 00001000 0101B2AC
01F80000 00E9A8F0 01077004 0A0A0104 Stck: 30E80000 010D09F0 010D07F0 010D0BF0 00000200 000000BC 00000020
010D0814 PC-8: 2E3CDDDD 777731C0 00004EAE FF7C2014 4EAEFF88 11C10101 4EAEFF76 21C00102 PC *: 4EAEFF82
2E3C3500 00004EAE FF94201F 670A4EAE FF7C2240 4EAEFE86 4E750024 0104477a : 4eae ff88 jsr -$78(a6) 0104477e :
11c1 0101 move.b d1,$101.w 01044782 : 4eae ff76 jsr -$8a(a6) 01044786 : 21c0 0102 move.l d0,$102.w 0104478a : *4eae
ff82 jsr -$7e(a6) 0104478e : 2e3c 3500 0000 move.l #$35000000,d7 01044794 : 4eae ff94 jsr -$6c(a6) 01044798 : 201f move.l
(a7)+,d0 Name: "Initial CLI" CLI: "lawbreaker" Hunk 0000 Offset 0000008A
```

```
03-Jul-99 14:12:23 Alert !! Alert 35000000 TCB: 0101A8C8 CTX: 01043B88 USP: 010D07B4 Data: 00000000 DDDD1111
DDDD2222 DDDD3333 01044722 DDDD5555 DDDD6666 35000000 Addr: AAAA0000 AAAA1111 AAAA2222 AAAA3333
AAAA4444 01044722 00002920 010D07B0 Stck: 01044798 00000000 00FA06D6 00001000 0101B2AC 01F80000 00E9A8F0
01077004 Stck: 0A0A0104 30E80000 010D09F0 010D07F0 010D0BF0 00000200 000000BC 00000020 PC-8: 20144EAE
FF8811C1 01014EAE FF7621C0 01024EAE FF822E3C 35000000 4EAEFF94 PC *: 201F670A 4EAEFF7C 22404EAE FE864E75
00245645 523A204C 61774272 65616B65 01044788 : 0102 btst.l d0,d2 0104478a : 4eae ff82 jsr -$7e(a6) 0104478e : 2e3c
3500 0000 move.l #$35000000,d7 01044794 : 4eae ff94 jsr -$6c(a6) 01044798 : *201f move.l (a7)+,d0 0104479a : 670a beq.s
$10447a6 0104479c : 4eae ff7c jsr -$84(a6) 010447a0 : 2240 movea.l d0,a1 010447a2 : 4eae fe86 jsr -$17a(a6) 010447a6 : 4e75
rts Name: "Initial CLI" CLI: "lawbreaker" Hunk 0000 Offset 00000098
```

Here is a breakdown of what these reports are saying:

In the first report, the first line is the date stamp.

The first line of each report describes the access violation and where it happened from. In the case of a WRITE, the data that was being written will be displayed as well. If an instruction mode access caused the fault, there will be an (INST) in the line.

The second line (starts USP:) displays the USER stack pointer (USP), and the status register (SR:). It then displays the supervisor/user state and the interrupt level. This will be from (U0) to (U7) or (S0) to (S7) (S=Supervisor) Next is the forbid state

(F=forbid, -=not) and the disable state (D or -) of the task that was running when the access fault took place. Finally, the task control block address is displayed (TCB:)

The next two lines contain the data and address register dumps from when the access fault happened. Note that A7 is not listed here. It is the stack pointer and is listed as USP: in the line above.

Then come the lines of stack back-trace. These lines show the data on the stack. If the stack is in invalid memory, MuForce will display a message to that fact.

If "SegTracker" was installed before MuForce, the "---->" lines will display in which seglist the given addresses are in based on the global tracking that "SegTracker" does. If no seglist match is found, no lines will be displayed. One line will be displayed for each of the stack longwords asked for (see the STACKCHECK option) and one line for the PC address of the hit. (The PC line is always checked for is "SegTracker" is installed.) The lines are in order: hit, first stack find, second stack find, etc. This is useful for tracking down who called the routine that caused the MuForce hit.

Next, optionally, comes the data around the program counter when the access fault happened. The first line (PC-8:) is the 8 long-words before the program counter. The second line starts at the program counter and goes for 8 long words.

The next optional section is the disassembly around the faulty access. This output is enabled with the DISPC option, and requires the disassembler.library in the MuForce distribution. Note that, as show here on a 68040 based system, the PC detecting the fault might not be the PC of the faulty instruction. This happens due to the push line of the microprocessor which delays write accesses in case the bus is needed for other purposes. In worst case, as seen here for the third fault, the CPU might have jumped to a completely different part of the code, making the disassembly rather useless.

The last line displays the name of the task that was running when the access fault took place. If the task was a CLI, it will display the name of the CLI command that was running. If the access fault was found to have happened within the seglist of a loaded program, the segment number and the offset from the start of the segment will be displayed. (Note that this works for any LoadSeg()'ed process)

Note that the name will display as "Processor Interrupt Level x" if the access happened in an interrupt.

07-Feb-99 20:29:42 Alert !! Alert 35000000 TCB: 00218EE0 CTX: 0022D128 USP: 0024AFF4 Data: 00000000 DDDD1111 DDDD2222 DDDD3333 0023043A DDDD5555 DDDD6666 35000000 Addr: AAAA0000 AAAA1111 AAAA2222 AAAA3333 AAAA4444 0023043A 00200830 0024AFF0 Stck: 002304B0 00000000 00FA06D6 00001000 002198C4 00268000 00004FF8 00000000 Stck: 00008000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 ----> 002304B0 - "LawBreaker" Hunk 0000 Offset 00000098 PC-8: 20144EAE FF8811C1 01014EAE FF7621C0 01024EAE FF822E3C 35000000 4EAEFF94 PC *: 201F670A 4EAEFF7C 22404EAE FE864E75 00245645 523A204C 61774272 65616B65 Name: "Initial CLI" CLI: "LawBreaker" Hunk 0000 Offset 00000098

This output happens when a program or the OS calls the EXEC Alert function. MuForce catches these calls and will display the alert information as seen above. CTX is the address of the mmu.library "MMU Context".

See also the Detail Example for information.

1.51 Detail Example Hit

Example "MuForce" Hit: Click on the field for explanation.

07-Feb-99 20:29:42

WORD WRITE to 00000000 data=0000 PC: 01044774 USP : 0024AFF8 SR: 0004 (U0)(-)(-) TCB: 00218EE0 Data: DDDD0000 DDDD1111 DDDD2222 DDDD3333 0023043A DDDD5555 DDDD6666 DDDD7777

Addr: AAAA0000 AAAA1111 AAAA2222 AAAA3333 AAAA4444 0023043A 00200830 00202288 Stck: 00000000 00FA06D6 00001000 002198C4 00268000 00004FF8 00000000 00008000

Stck: 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 00000000 ----> 0023048C - "LawBreaker" Hunk 0000 Offset 00000074 PC-8: 2222263C DDDD3333 280D2A3C DDDD5555 2C3CDDDD 66662E3C DDDD7777 31C00000

PC *: 4EAEFF7C 20144EAE FF8811C1 01014EAE FF7621C0 01024EAE FF822E3C 35000000 01044764 : 2c3c dddd 6666 move.l #-$2222999a,d6

0104476a : 2e3c dddd 7777 move.l #-$22228889,d7 01044770 : 31c0 0000 move.w d0,$0.w

01044774 : *4eae ff7c jsr -$84(a6) 01044778 : 2014 move.l (a4),d0

0104477a : 4eae ff88 jsr -$78(a6) 0104477e : 11c1 0101 move.b d1,$101.w

01044782 : 4eae ff76 jsr -$8a(a6) Name:"Initial CLI" CLI: "LawBreaker Hunk 0000 Offset 00000074

And, for Alert hits:

07-Feb-99 20:29:42

Alert !! Alert 35000000 TCB: 00218EE0 CTX: 0022D128 USP: 0024AFF4 Data: 00000000 DDDD1111 DDDD2222 DDDD3333 0023043A DDDD5555 DDDD6666 35000000

Addr: AAAA0000 AAAA1111 AAAA2222 AAAA3333 AAAA4444 0023043A 00200830 0024AFF0 Stck: 01044798 00000000 00FA06D6 00001000 002198C4 00268000 00004FF8 00000000

Stck: 00008000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 ----> 01044798 - "LawBreaker" Hunk 0000 Offset 00000098 PC-8: 20144EAE FF8811C1 01014EAE FF7621C0 01024EAE FF822E3C 35000000 4EAEFF94

PC *: 201F670A 4EAEFF7C 22404EAE FE864E75 00245645 523A204C 61774272 65616B65 01044788 : 0102 btst.l d0,d2

0104478a : 4eae ff82 jsr -$7e(a6) 0104478e : 2e3c 3500 0000 move.l #$35000000,d7

01044794 : 4eae ff94 jsr -$6c(a6) 01044798 : *201f move.l (a7)+,d0

0104479a : 670a beq.s $10447a6 0104479c : 4eae ff7c jsr -$84(a6)

010447a0 : 2240 movea.l d0,a1 010447a2 : 4eae fe86 jsr -$17a(a6)

010447a6 : 4e75 rts Name:"Initial CLI" CLI: "LawBreaker Hunk 0000 Offset 00000098

Note that "MuForce" hit output is very configurable. The above example hit was produced with options: SHOWPC DATESTAMP STACKCHECK STACKLINES=2 DISPC DISRANGE=16

Here are some examples of different output configurations:

Output with the TINY option: (Commandline: MuForce TINY)

WORD WRITE to 00000000 data=0000 PC: 0763857C

Output with the SMALL option: (Commandline: MuForce SMALL)

WORD WRITE to 00000000 data=0000 PC: 0023048C USP : 0024AFF8 SR: 0004 (U0)(-)(-) TCB: 00218EE0 Name:"Initial CLI" CLI: "LawBreaker

Output with DEFAULT options: (Commandline: MuForce)

WORD WRITE to 00000000 data=0000 PC: 0023048C USP : 0024AFF8 SR: 0004 (U0)(-)(-) TCB: 00218EE0 Data: DDDD0000 DDDD1111 DDDD2222 DDDD3333 0023043A DDDD5555 DDDD6666 DDDD7777

Addr: AAAA0000 AAAA1111 AAAA2222 AAAA3333 AAAA4444 0023043A 00200830 00202288 Stck: 00000000 00FA06D6 00001000 002198C4 00268000 00004FF8 00000000 00008000

Stck: 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 00000000 ----> 0023048C - "LawBreaker"

1.52 Output: Date Stamp

The date stamp field, if enabled, is at the start of the "MuForce" hit. The time is only exact to +/- 1 second.

1.53 Output: Write Hit

This tells you that the "MuForce" hit was a READ from or WRITE to memory. The possible writes are:

-- WRITE - - READ -- BYTE WRITE - 8-bit write - BYTE READ WORD WRITE - 16-bit write - WORD READ LONG WRITE - 32-bit write - LONG READ WRITE - all remaining- READ

"all remaining" are movems, double or extended precision fmoves and fmovems.

1.54 Output: Address hit

This field in the output shows the illegal address that was accessed which triggered the "MuForce" report.

1.55 Output: Data Write

On an illegal WRITE to memory, the value that was attempted to be written will be displayed here. The size of this field changes to match the size of the write.

1.56 Output: Program Counter

This field displays the program counter at the time of the MMU trap of the invalid access. Note that this address is not always the exact instruction that caused the hit. See the various notes for your processor for more details.

General Notes

68020 Notes

68030 Notes

68040 Notes

68060 Notes

1.57 Output: CPU Status Register

This is the CPU status register as found on the MMU trap stack frame. It contains the condition flags and the current mode/etc.

1.58 Output: Special information

This field contains special task information. This is useful for determining what is going on at the time of the hit.

(U0)(-)(-) ^^ ^ ^ || | | || | +-- This will have a D if the task is DISABLE state || +------ This will have a F if the task is FORBID state |+-------- This is the processor IPL level (0 is normal code) +--------- This is the processor state: U=user, S=supervisor

1.59 Output: Task Control Block

This is the address of the Task Control Block, also known as the task structure. (See exec/tasks.h) This is used by "MuForce" to tell you who caused the hit.

1.60 Output: MMU Context

This is the address of the MMU Context the faulty task was attached to, see "mmu/contexts.h".

1.61 Output: Data Register Dump

This line contains a dump of the data registers at the time of the "MuForce" hit.

1.62 Output: D0 Register

The D0 register of the 680x0 CPU.

See Data:

1.63 Output: D1 Register

The D1 register of the 680x0 CPU.

See Data:

1.64 Output: D2 Register

The D2 register of the 680x0 CPU.

See Data:

1.65 Output: D3 Register

The D3 register of the 680x0 CPU.

See Data:

1.66 Output: D4 Register

The D4 register of the 680x0 CPU.

See Data:

1.67 Output: D5 Register

The D5 register of the 680x0 CPU.

See Data:

1.68 Output: D6 Register

The D6 register of the 680x0 CPU.

See Data:

1.69 Output: D7 Register

The D7 register of the 680x0 CPU.

See Data:

1.70 Output: Address Register Dump

This line contains a dump of the address register at the time of the "MuForce" hit.

1.71 Output: A0 Register

The A0 register of the 680x0 CPU.

See Addr:

1.72 Output: A1 Register

The A1 register of the 680x0 CPU.

See Addr:

1.73 Output: A2 Register

The A2 register of the 680x0 CPU.

See Addr:

1.74 Output: A3 Register

The A3 register of the 680x0 CPU.

See Addr:

1.75 Output: A4 Register

The A4 register of the 680x0 CPU.

See Addr:

1.76 Output: A5 Register

The A5 register of the 680x0 CPU.

See Addr:

1.77 Output: A6 Register

The A6 register of the 680x0 CPU.

See Addr:

1.78 Output: A7 Register

The A7 register of the 680x0 CPU is also known as the Stack Pointer or SP. In the "MuForce" hit, the USER SP (the stack of the task that caused the hit) is displayed in the USP: field.

See Addr:

1.79 Output: Stack Dump

These lines contain stack dumps from the task that caused the "MuForce" hit. It can be used to figure out what the program was doing and what routines called the current routine by looking at the values on the stack.

1.80 Output: Stack Word

This is a longword on the stack of the task that caused the hit See Stck: for more details.

1.81 Output: SegTracker

This symbol "---->" identifies a line produced via the "SegTracker" utility.

See the "FindHit" documentation for details as to how to use this information.

1.82 Output: SegTracker Address

This is the address that the hunk/offset describes. This is here such that you can cross-reference it with a value on the stack, in a register, or the program counter. The hunk/offset on the same line are produced when this address is processed via "SegTracker". See the "FindHit" documentation for details as to how to use this information.

1.83 Output: SegTracker Name

This is the name of the file, as passed to LoadSeg, which was found to be loaded around the address given. See the "FindHit" documentation for details as to how to use this information.

1.84 Output: Hunk

This is the hunk in the load file that was loaded around the given address. See the "FindHit" documentation for details as to how to use this information.

1.85 Output: Offset

This is the offset from the start of the hunk that this address is at within the given load file. See the "FindHit" documentation for details as to how to use this information.

1.86 Output: Task/Process Name

This line contains the decoding of the TCB into the TASK name, the CLI command (if a CLI), and if the hit happened in the SegList attached to the process, the hunk and offset for the hit. Note that this hunk/offset is not produced by "SegTracker".

1.87 Output: Task Name

This field contains the task name as stored in the TCB of the task that caused the "MuForce" hit. If the TCB is invalid, it will say so.

1.88 Output: CLI Command Name

This field will contain the name of the CLI command that caused the hit if the TCB is a CLI process and there was a command loaded. If the task is not a CLI process or no command is loaded, this field will not be displayed.

1.89 Output: Alerts

This output happens when a program or the OS calls the EXEC Alert function. "MuForce" catches these calls and will display the alert information as seen above. (With the data and time as needed)

1.90 Output: Alert Number

This field contains the alert number that was generated. Check the include file exec/alerts.h or exec/alerts.i for details as to how to decode this number.

1.91 Output: Show PC

If the SHOWPC option is turned on, "MuForce" will dump the 8 longwords before the program counter and the 8 longwords starting at the PC.

This can be used to help debug programs by being able to look at the code around the hit by disassembling it.

1.92 Output: Show PC-$20

This is the longword at the memory address (PC - $20) where PC is the Program Counter of the "MuForce" hit.

1.93 Output: Show PC-$1C

This is the longword at the memory address (PC - $1C) where PC is the Program Counter of the "MuForce" hit.

1.94 Output: Show PC-$18

This is the longword at the memory address (PC - $18) where PC is the Program Counter of the "MuForce" hit.

1.95 Output: Show PC-$14

This is the longword at the memory address (PC - $14) where PC is the Program Counter of the "MuForce" hit.

1.96 Output: Show PC-$10

This is the longword at the memory address (PC - $10) where PC is the Program Counter of the "MuForce" hit.

1.97 Output: Show PC-$0C

This is the longword at the memory address (PC - $0C) where PC is the Program Counter of the "MuForce" hit.

1.98 Output: Show PC-$08

This is the longword at the memory address (PC - $08) where PC is the Program Counter of the "MuForce" hit.

1.99 Output: Show PC-$04

This is the longword at the memory address (PC - $04) where PC is the Program Counter of the "MuForce" hit.

1.100 Output: Show PC+$00

This is the longword at the memory address (PC) where PC is the Program Counter of the "MuForce" hit.

1.101 Output: Show PC+$04

This is the longword at the memory address (PC + $04) where PC is the Program Counter of the "MuForce" hit.

1.102 Output: Show PC+$08

This is the longword at the memory address (PC + $08) where PC is the Program Counter of the "MuForce" hit.

1.103 Output: Show PC+$0C

This is the longword at the memory address (PC + $0C) where PC is the Program Counter of the "MuForce" hit.

1.104 Output: Show PC+$10

This is the longword at the memory address (PC + $10) where PC is the Program Counter of the "MuForce" hit.

1.105 Output: Show PC+$14

This is the longword at the memory address (PC + $14) where PC is the Program Counter of the "MuForce" hit.

1.106 Output: Show PC+$18

This is the longword at the memory address (PC + $18) where PC is the Program Counter of the "MuForce" hit.

1.107 Output: Show PC+$1C

This is the longword at the memory address (PC + $1C) where PC is the Program Counter of the "MuForce" hit.

1.108 Output: Disassembler output

This line is generated by the disassembler, it presents the code in Motorola opcode notation around the PC where the fault was detected. Note that due to the push line of some processors, this might be different to the location of the instruction that actually caused the fault.

1.109 Output: Disassembler output with PC

This line is the disassembly of the instruction where the fault was detected. It is marked by an asterisk "*" in front of the opcodes. Note that due to the push line of some processors, the "*" might not indicate the actual instruction that caused the problem, but will usually be "near" that instruction.

1.110 Index

68020 Notes 68030 Notes 68040 Notes 68060 Notes

A...

Option: AREGCHECK Output: A0 Register Output: A1 Register Output: A2 Register Output: A3 Register Output: A4 Register Output: A5 Register Output: A6 Register Output: A7 Register Output: Address hit Output: Address Register Dump Output: Alert Number Output: Alerts

B...

Option: BUFFERSIZE

C...

Option: CAPTURESUPER Output: CLI Command Name Command line options and tooltypes Output: CPU Status Register Credits : Thank you goes to...

D...

Output: D0 Register Output: D1 Register Output: D2 Register Output: D3 Register Output: D4 Register Output: D5 Register Output: D6 Register Output: D7 Register Output: Data Register Dump Output: Data Write Output: Date Stamp Option: DATESTAMP Option: DEADLY Debuggers: Not causing a hit Debuggers: Trapping a hit Detail Example Hit Option: DISABLEBELL Output: Disassembler output Output: Disassembler output with PC Option: DISPC Option: DISRANGE Option: DREGCHECK

E...

Example MuForce output

F...

Option: FATALHITS Option: FILE Option: FSPACE

H...

History Output: Hunk

I...

Installation of MuForce Option: INTRO

L...

Option: LED The THOR-Software Licence

M...

Output: MMU Context MuForce Guide MuSetCacheMode

N...

Option: NOALERTPATCH Option: NOGURUPATCH

O...

Optimizing the MMU table setup Option: AREGCHECK Option: BUFFERSIZE Option: CAPTURESUPER Option: DATESTAMP Option: DEADLY Option: DISABLEBELL Output: Disassembler output Output: Disassembler output with PC Option: DISPC Option: DISRANGE Option: DREGCHECK Option: FATALHITS Option: FILE Option: FSPACE Option: INTRO Option: LED Option: NOALERTPATCH Option: NOGURUPATCH Option: ON Option: PARALLEL Option: PREPMUNGWALL Option: PRIORITY Option: QUIET Option: QUIT Option: RAWIO Option: SHOWPC Option: SMALL Option: STACKCHECK Option: STACKLINES Option: STDIO Option: TINY Option: VALIDZERO Option: VERBOSE Output: A0 Register Output: A1 Register Output: A2 Register Output: A3 Register Output: A4 Register Output: A5 Register Output: A6 Register Output: A7 Register Output: Address hit Output: Address Register Dump Output: Alert Number Output: Alerts Output: CLI Command Name Output: CPU Status Register Output: D0 Register Output: D1 Register Output: D2 Register Output: D3 Register Output: D4 Register Output: D5 Register Output: D6 Register Output: D7 Register Output: Data Register Dump Output: Data Write Output: Date Stamp Output: Hunk Output: MMU Context Output: Offset Output: Program Counter Output: SegTracker Output: SegTracker Address Output: SegTracker Name Output: Show PC Output: Show PC+$00 Output: Show PC+$04 Output: Show PC+$08 Output: Show PC+$0C Output: Show PC+$10 Output: Show PC+$14 Output: Show PC+$18 Output: Show PC+$1C Output: Show PC-$04 Output: Show PC-$08 Output: Show PC-$0C Output: Show PC-$10 Output: Show PC-$14 Output: Show PC-$18 Output: Show PC-$1C Output: Show PC-$20 Output: Special information Output: Stack Dump Output: Stack Word Output: Task Control Block Output: Task Name Output: Task/Process Name Output: Write Hit

P...

Option: PARALLEL System patches installed by the library Option: PREPMUNGWALL Option: PRIORITY Processor specific notes Output: Program Counter

Q...

Option: QUIET Option: QUIT

R...

Option: RAWIO

S...

Output: SegTracker Output: SegTracker Address Output: SegTracker Name Option: SHOWPC Output: Show PC Output: Show PC+$00 Output: Show PC+$04 Output: Show PC+$08 Output: Show PC+$0C Output: Show PC+$10 Output: Show PC+$14 Output: Show PC+$18 Output: Show PC+$1C Output: Show PC-$04 Output: Show PC-$08 Output: Show PC-$0C Output: Show PC-$10 Output: Show PC-$14 Output: Show PC-$18 Output: Show PC-$1C Output: Show PC-$20 Option: SMALL Output: Special information Option: STACKCHECK Output: Stack Dump Option: STACKLINES Output: Stack Word Option: STDIO System patches installed by the library

T...

Output: Task Control Block Output: Task Name Output: Task/Process Name Option: TINY The THOR-Software Licence

V...

Option: VALIDZERO Option: VERBOSE

W...

What's the job of MuForce? What's the MMU.library? Writing debuggers that support MuForce Output: Write Hit

1.111 History

Release 40.1:

This is the first official release.

Release 40.2:

The INTRO keyword was broken. Fixed.

Release 40.7:

MuForce did not respect a ROM to RAM re-mapping, fixed. NEWVBR keyword added, MuForce tries now to re-use an already relocated vector base.

Release 40.10:

Added the "FATALHITS" option on request by Sam Jordan.

Release 40.13:

Fixed allocation of "invalid" low memory. The last releases disabled ROM re-mapping, fixed! Thanks to Nils Goers for testing and finding the problems. Added the DISABLEBELL option.

Releases 40.14 and 40.15:

Internal releases.

Release 40.16:

Made the segmentation fault handler of MuForce global, it remains now valid for all MMMContexts. Fixed a bug in the Exception hander which did not check correctly for available memory. Added an explicit check for I/O hardware, MuForce will no longer try to read or dump it, at least for the hardware that behaves "correctly".

Release 40.17:

Added an API for MuGuardianAngel.

Release 40.18:

Changed the SegTracker output again to the old Enforcer style to be backwards compatible.

Release 40.20:

Added the disassembler, the DISPC, the DISRANGE and the PREPMUNGWALL options.

Release 40.21:

MuForce tries now, too, to capture all other MC68K exceptions if possible, and if the task that caused the exception can't capture them itself. This new feature can be disabled by the new "NOGURUPATCH" option.

Release 40.22:

The MuForce exec default exception handler runs now into a Wait(SIGBREAKF_CTRL_E). Postmortem debuggers may capture the task from there. Register d2 is the guru number, register a2 the user stack pointer. The user registers d0-d7/a0-a6 at the position of the fault are pushed to the user stack. Added the CAPTURESUPER option and included a new exception vector management. MuForce marks now invalid pages explicitly as non-copyback, repairable invalid on startup.