MuSetCacheMode

Thomas Richter

COLLABORATORS			
	TITLE : MuSetCacheMode		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Thomas Richter	August 27, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

MuSetCacheMode

1.1 MuSetCacheMode Guide

´### ,#. ,#### ,### ####’ #### ,#### ,###’ ####’ #### _____ ,#### ,###’ || ####’ #### | ___ ___ | ___ ,#### ,###’ ---- | | | | | | |___ ####’ #### | | | | | | | ,#####. ,###’ . ||___||___||_ ___| ####’##. ,#### ,## ,#### ########’ # ,##’ ####’ `####’ `###’ ,#### ##### © 1999 THOR - Software, ####’ Thomas Richter `##’

MuSetCacheMode Guide

Guide Version 1.04 MuSetCacheMode Version 40.7

The Licence : Legal restrictions

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

What is it : Overview

Installation : How to install MuSetCacheMode

Synopsis : The command line options and tool types

Examples : How to use MuSetCacheMode to optimize MuForce

History : What happened before

© 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

The THOR-Software Licence (v2, 24th June 1998)

This License applies to the computer programs known as "MuSetCacheMode" and the "MuSetCacheMode.guide". The "Program", below, refers to such program. The "Archive" refers to the package of distribution, as prepared by the author of the Program, Thomas Richter. Each licensee is addressed as "you".

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 MuSetCacheMode?

MuSetCacheMode allows you to specify how the CPU should use its internal buffers, the so-called caches , in a mmu.library compatible way. There's usually no need to change the settings the library selected for you, but special hardware or software might require special treatment. It is a replacement for the P5 "SetCacheMode" program that hacks on the MMU tables directly without any access or protection mechanism. It can be used, too, to customize the MMU setup for your machine and therefore improve the performance of debugging tools like MuForce. Check the examples section .

This program should not be run unless you really know what you're doing.

1.5 What's a cache?

All members of the Motorola family, starting with the 68020, come with one or two build in data buffers, the so-called "caches". A cache is a small buffer within the CPU itself that keeps frequently accessed memory locations; hence, it avoids unnecessary bus accesses and memory reads and writes of data recently accessed and therefore speeds up the CPU operation. However, special care must be taken in case the "memory location" is in fact a hardware component, e.g. an I/O port that might change its contents without the knowledge of the CPU. If this port would be buffered, a program trying to read from the port would access the internal CPU caches instead of trying to re-read the up-to-date data from the port. It is therefore important to tell the CPU not to buffer I/O ports. Since the chip-memory is accessed by the blitter bypassing the CPU, it must not be buffered as well.

The following cache modes are available:

CACHEINHIBIT:

The page must not be hold in the cache.

NONSERIAL:

The page must not be hold in the cache, but the CPU might re-order the accesses to this memory region to speed up access. Hence, external bus requests to the page might show up in a different order compared to what was written in the code. This is harmless in case of chip memory, but most I/O device drivers want the accesses just in the order intended. I/O ports should therefore NOT be marked as "NONSERIAL".

This flag is 68040 specific and will be ignored on other processors.

IMPRECISE:

The page must not be hold in the cache, but the CPU might speed up accesses by using a "sloppy" access mechanism not allowing to fix bus errors in all circumstances. I/O ports not generating bus errors can be set to IMPRECISE without any risk; this holds, too, for chip memory. Accesses remain "serialized", i.e. will be performed in the order encoded in the program.

This flag is 68060 specific and will be ignored if no '060 is available.

WRITETHROUGH:

The page will be cached, but writes to the memory will not only update the cache, but will be written to memory immediately as well. Reads, however, will read the data from the cache if available. I/O ports and chip memory MUST NOT be set to WRITETHROUGH.

COPYBACK:

The page will be cached. Writes into the memory will not be performed immediately but will be hold in the cache. The data will be written out as soon as the cache entry is required for other data. This is clearly the most efficient cache mode and will be selected by default for "ordinary" fast mem. I/O ports and chip memory MUST NOT be set to COPYBACK.

This flag is not available on a 68020 or a 68030. If it is selected, the MMU library will fall back to the WRITETHROUGH mode instead.

1.6 Installation of MuSetCacheMode

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 "MuSetCacheMode" wherever you want.

- Remove the obsolete "SetCacheMode" program. Replace the calls to "SetCacheMode" in the startup-sequence by "MuSetCacheMode".

- If you run MuForce, check the examples section how to optimize it for your machine.

1.7 Command line options and tooltypes

MuSetCacheMode 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 "MuFastRom" 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.

MuSetCacheMode ADDRESS=FROM/A,SIZE/A, COPYBACK/S,WRITETHROUGH/S, NOCACHESERIALIZED=CACHEINHIBIT
NONSERIAL/S,NOCACHE=IMPRECISE/S, IO=IOSPACE/S,NOIO=NOIOSPACE/S, ROM/S,NOROM/S,WRITEPROTECTED/S,N
VALID/S,INVALID/S,BLANK/S, USERONLY/S,SUPERVISORONLY/S,VERBOSE/S

ADDRESS=FROM

The base address of the memory region to set the cache mode for. This address should be given in hexadecimal notation, using a leading "$" or "0x", as in "0x00f80000". This base address must be divisible by the page size selected by the MMU library, which is usually 0x0400 (1K) on a 68020/68030 and 0x1000 (4K) on a 68040 or 68060. This command line option *MUST* be given.

Unless like the original P5 program SetCacheMode, the address is not restricted at all, but it might make no sense to set the cache mode of anything but certain I/O devices or bridge-boards.

SIZE

The size of the memory block whose cache modes should be adjusted. As before, this should be given in hexadecimal notation with a leading "$" or "0x", and it must be divisible by the page size.

COPYBACK

Sets the cache mode to copyback.

This cache mode is only available for the 68040 and 68060.

WRITETHROUGH

Sets the cache mode to writethrough.

CACHEINHIBIT

Sets the cache mode to cache inhibited.

NONSERIAL

Sets the cache mode to cache inhibited, non-serialized.

This cache mode is only available for the 68040.

IMPRECISE

Sets the cache mode to cache inhibited, imprecise exception processing.

This cache mode is only available for the 68060.

IO=IOSPACE

Sets the memory region to "IO space". This is a pure software flag and does not change the cache modes at all. It is used by debugging utilities like MuGuardianAngel and MuForce to avoid accessing this address space because it might toggle certain hardware.

NOIO=NOIOSPACE

Sets the memory region to "memory space". This is a pure software flag and does not change the cache modes at all.

ROM

Enables the defensive write protection for the indicated memory region. All writes will be ignored silently.

NOROM

Removes the defensive write protection again.

WRITEPROTECTED

Enables the aggressive write protection. In case a write to the memory region is detected, a bus error will be generated. In case MuForce is running, you'll see a "hit".

NOTWRITEPROTECTED

Removes the aggressive write protection and makes the memory writable again.

VALID

Validate the memory region, i.e. allow accesses, even if it has been marked as blank or invalid before. This keyword extends the function of the cache-mode specific key words above; it may not be specified without them.

INVALID

Mark the specified memory region as invalid, reads or writes to this area will cause MuForce hits. BE WARNED! This option should be used *only* if MuForce is already running, or programs accessing this memory area will run into a Guru. It is recommended to use the "BLANK" keyword instead if MuForce is not running.

BLANK

Mark the specified memory region as blank. Accesses of this memory region will read dummy data, and writes into this memory domain will be ignored. This option should be used to work around bad programs that write into non-existent memory regions, i.e. to allow the access but to ignore it.

USERONLY

Changes the cache mode only for user level accesses. All ordinary Amiga programs run in user mode.

The default is to set the cache mode for both, user and supervisor mode accesses.

SUPERVISORONLY

Changes the cache mode only for supervisor level accesses. For example, all interrupts are processed in supervisor mode.

When started from the workbench, MuSetCacheMode 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/MuSetCacheMode/AUTO/CLOSE/WAIT

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

1.8 Examples how to use MuSetCacheMode

One application for MuSetCacheMode is to setup or modify the MMU table build by the mmu.library or 68040/68060.library to include special hardware that is not recognized 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 this tool.

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.

1.9 History

Release 40.3:

This is the first official release.

Release 40.4:

Added the VALID, INVALID, IO, NOIO and BLANK command line options.

Release 40.5:

Internal release.

Release 40.6:

Made the information messages more useful. Added the write protection keywords. IOSPACE will now set the memory region to non-cacheable implicitly. This can be overridden with additional keywords.

Release 40.7:

MuSetCacheMode did not unload in case it was run from the workbench. Fixed. The code is now a bit more error tolerant. "IMPRECISE" and "NONSERIAL" interacted and could not be specified both at once. Fixed.