MuScan

Thomas Richter

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

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

MuScan

1.1 MuScan Guide

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

MuScan Guide

Guide Version 1.01 MuScan Version 40.2

© 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 "MuScan" and the "MuScan.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 to work 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 MuScan?

MuScan prints the MMU setup for the so called "public context" in a compact and mmu.library compatible way. It's output is mainly interesting to experts for debugging reasons.

1.5 Installation of MuScan

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 "MuScan" wherever you want.

1.6 Command line options and tooltypes

MuScan 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 "MuScan" 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.

MuScan TO/K

TO

Takes a path name where MuScan should print its output to. This defaults to the console, or wherever the output was redirect to.

When started from the workbench, MuFastRom 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/MuScan.

This argument defaults to NIL:, i.e. all output will be thrown away. This is, admittedly, not very useful at all.

1.7 What does the output mean?

A typical printout of the MuScan program looks like this (here taken from my running system):

MuScan 40.2 (11.5.99) © THOR

68040 MMU detected. MMU page size is 0x1000 bytes.

Memory map: 0x00000000 - 0x00000FFF Repairable Invalid (0x00000000) 0x00001000 - 0x00007FFF CopyBack Single Remapped to 0x010D9000 0x00008000 - 0x001FFFFF CacheInhibit NonSerial 0x00200000 - 0x003FFFFF CacheInhibit NonSerial I/O space 0x00400000 - 0x00BBFFFF CacheInhibit Repairable Invalid (0x00000000) 0x00BC0000 - 0x00BFFFFF CacheInhibit I/O space 0x00C00000 - 0x00D7FFFF CacheInhibit Repairable Invalid (0x00000000) 0x00D80000 - 0x00DFFFFF CacheInhibit I/O space 0x00E00000 - 0x00E8FFFF CacheInhibit Repairable Invalid (0x00000000) 0x00E90000 - 0x00EAFFFF CacheInhibit I/O space 0x00EB0000 - 0x00EFFFFF CacheInhibit Repairable Invalid (0x00000000) 0x00F00000 - 0x00F7FFFF Repairable Invalid (0x00000000) 0x00F80000 - 0x00FFFFFF WriteProtected CopyBack Repairable 0x01000000 - 0x01FFFFFF CopyBack 0x02000000 - 0xFFFFFFFF Repairable Invalid (0x00000000)

The header prints the release information, below that the MMU type installed in your system. Four different MMU types have been used in the Amiga computer, and MuScan (or more precisely, the mmu.library) knows all of them:

The 68851 (external) MMU of the 68020 processor, the 68030 (internal) MMU, the 68040 (internal) MMU and the 68060 (internal) MMU.

Below that, the page size gets printed, in hexadecimal notation. The MMU organizes memory in pages of the printed size, each of the pages can be given attributes. A "page" is the smallest possible memory chunk the MMU can manage. Page sizes range from 256 bytes to 32K for the 68851 and 68030 to only 4K and 8K for the 68040 and 68060. The "mmu.library" will typically select either 1K or 4K pages, whatever is available.

Below that point, MuScan prints the memory map. The number on the left hand side is the memory range, i.e. the logical addresses as seen from the CPU. The right hand side shows the properties of this memory block.

The following properties are known:

WriteProtected: The memory range is protected against overwriting. A write into this region will cause a bus error exception.

The following two flags are not the "true" hardware flags but only presets selected on a software level:

U: The memory range is marked to have been touched by the CPU. "U" stands for "used".

M: The memory range has been modified by the CPU.

Global: The "global" attribute of the memory block is set. This means that special instructions will not flush these entries from the cache of the MMU. (Very technically.)

TT: Transparently translated. This memory range is not under control of the MMU, but of the "transparent translation registers". All other attributes will be ignored. The "mmu.library" will try to disable all "TT" registers if possible to avoid this loss of control.

ROM: Almost identically to "WriteProtected" except that a write into this domain is silently tolerated and no exception is generated.

UP0: User page attribute 0. This is a user selectable attribute that appears on a special pin outside of the CPU. Special hardware may make use of it. Do not touch this attribute without good reason.

UP1: User page attribute 1. A second user hardware flag.

CacheInhibit: The CPU is told not to hold the memory range in its cache. This flag should be set for IO spaces and memory ranges other hardware has access to that may modify the memory without notice of the CPU. Chip memory has to be marked as "non cachable" as well as the IO hardware and graphics board RAM.

Imprecise: Imprecise exception model. This is a special flag available for the 060 only (it will be ignored by all other MMUs). It tells the CPU to react a bit "sloppy" on bus error faults to speed up memory access.

NonSerial: Non-serialized access. This flag is only available for the 040 and will be ignored otherwise. It allows the CPU to re-order memory accesses to speed up transfer rates.

CopyBack: The copyback cache is enabled. This flag is only available for the 040 and 060 MMU and is ignored otherwise. It tells the CPU to keep writes in its cache without writing the data back into the memory immediately. The selected cache entry is written back as soon as it is needed for a different address or it is flushed explicitly.

SuperOnly: The page is only available in the "supervisor" mode of the CPU. An access from within user mode will cause an exception.

Blank: This memory range doesn't address any memory or hardware, it is simply blank. Reading from or writing to this range will be ignored silently, but the result of reads are unpredictable, even though they are tolerated.

Shared: This flag is currently not yet used by the library. It is reserved for private contexts sharing parts of their memory layout with the public context.

Single: Single page layout. The library is told to use one descriptor per page for this memory region, even though a more efficient layout would be possible. Using single pages enables the use of the hardware based calls of the library, i.e. of "GetPageProperties", "SetPageProperties" and "PhysicalPageLocation".

Repairable: In case of access faults, these faults are repairable by software and the program causing a fault by accessing these pages can be continued. Allowing repair by software means more work for the library, hence the MMU table layout may be less efficient and exception handling will be slower. NOT setting this flag means that some exceptions might be fatal and cause a guru.

I/O space: The MMU pages belongs to I/O hardware and not to RAM or ROM. Reading or writing these addresses might imply I/O operations and should be handled with care.

U0: User software flag 0. This is a pure software flag, ignored, but kept by the library.

U1..U3: User software flags 1 to 3.

Invalid (xxxxxx): The range is marked as invalid. Accessing this range by either reads or writes will cause an exception. In case this range is not marked as "repairable" as well, a user attribute may be assigned to the pages which is printed in the brackets.

Swapped (xxxxxx): The range is marked as swapped out. The contents of this page are saved to a swap medium, for example a HD. Accessing this memory cause a possible swapper daemon to get run that loads the memory contents back from the HD.

Re-mapped to xxxxxx: The logical address printed in the left column is different from the physical address, the MMU has been used to mirror this memory range to elsewhere. The true physical location gets printed.

Bundled to xxxxxx: All pages within this memory range are "bundled" to one single page. Any access of memory within this region will be re-routed to a single memory page.

Indirect at xxxxxx: The user provided a MMU table descriptor itself which is kept at the address printed out. Remember that doing so will break the compatibility between different types of MMUs.

1.8 History

Release 40.1:

This is the first official release. I would call "MuScan" pretty final because there's not much this program has to do. All the tough tricks are run by the mmu.library anyways. However, the "mmu.library" is still in beta state.

Release 40.2:

Added the MAPP_IO flag for the 0.30 library.