MMU

Thomas Richter

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

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

MMU

1.1 MMU Guide

´### ,#. ,#### ,### / ####’ #### / / ,#### ,###’ / / ####’ #### / / ,#### ,###’ / / ####’ #### / / ,#### ,###’ / ____ /_____ ####’ #### / / / \ ,#####. ,###’ . / / / / ####’##. ,#### ,## / / / / ,#### ########’ # ,##’ / / / / ####’ `####’ `###’ /_____ __/__ /_____/ ,#### ##### © 1999 THOR - Software, ####’ Thomas Richter `##’

---

mmu.library Master Guide

Guide Version 1.04 MMU Library Version 40.51

Thanks a lot, Mike!

An additional note: The complete effort of the mmu.library is published here under my "Freeware Licence" terms because I think the Amiga deserves better software and I hope the library and the tools using this library will give you the power to write this software. It took over a year and endless hours of thinking, writing, debugging to obtain this result. I hope this effort is not wasted, neither for you nor for me. I would highly appreciate some feedback for that reason; in case you’re using these tools in software development, or the library itself in your software, consider offering me a licence of your program as I offered you all this for free.

Thanks a lot, and happy programming,

Thomas

© 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 "mmu.library", the "disassembler.library", the version 40 releases of the "68040.library", the "MuTools", including "MuForce", "MuGuardianAngel", "MuSetCacheMode", "MuScan", "MuLink", "MuMove4K", "FixCybAccess", "CheckFPU", "MuOVLYMGR", "MuLockLib", "MuOmniSCSIPatch" and the corresponding documentation, known as ".guide" files. 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".

The Program and the data in the archive are freely distributable under the restrictions stated below, but are also Copyright (c) Thomas Richter.

Distribution of the Program, the Archive and the data in the Archive by a commercial organization without written permission from the author to any third party is prohibited if any payment is made in connection with such distribution, whether directly (as in payment for a copy of the Program) or indirectly (as in payment for some service related to the Program, or payment for some product or service that includes a copy of the Program "without charge"; these are only examples, and not an exhaustive enumeration of prohibited activities).

However, the following methods of distribution involving payment shall not in and of themselves be a violation of this restriction:

(i) Posting the Program on a public access information storage and retrieval service for which a fee is received for retrieving information (such as an on-line service), provided that the fee is not content-dependent (i.e., the fee would be the same for retrieving the same volume of information consisting of random data).

(ii) Distributing the Program on a CD-ROM, provided that

a) the Archive is reproduced entirely and verbatim on such CD-ROM, including especially this licence agreement;

b) the CD-ROM is made available to the public for a nominal fee only,

c) a copy of the CD is made available to the author for free except for shipment costs, and

d) provided further that all information on such CD-ROM is re-distributable for non-commercial purposes without charge.

Redistribution of a modified version of the Archive, the Program or the contents of the Archive is prohibited in any way, by any organization, regardless whether commercial or non-commercial. Everything must be kept together, in original and unmodified form.

Limitations.

THE PROGRAM IS PROVIDED TO YOU "AS IS", WITHOUT WARRANTY. THERE IS NO WARRANTY FOR THE PROGRAM, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IF YOU DO NOT ACCEPT THIS LICENCE, YOU MUST DELETE THE PROGRAM, THE ARCHIVE AND ALL DATA OF THIS ARCHIVE FROM YOUR STORAGE SYSTEM. YOU ACCEPT THIS LICENCE BY USING OR REDISTRIBUTING THE PROGRAM.

Thomas Richter

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.

See also: Logical vs. physical addresses

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 Logical vs. Physical Addresses and the DMA problem

Each byte, that is, each memory cell, in the computer memory has its address, something like a "name" by which it is known to the system. Each address is just a number, something like you might imagine as the house number in a long street. And now I'm telling you that there are actually two numbers for the same house?

Yes, there are two, because of some "cheating" the memory management unit does. The "physical addresses" are easier to understand: These are the addresses that are really physical available in the wiring of your computer. There are a couple of wires - the "address bus" - taking the addresses from the microprocessor, which does the actual computation, to the surrounding chips which are responsible for graphics, sound, for the disk drives, for the HD and so on. These wires carry the physical addresses.

Besides the address bus, there's a second set of wires, the "data bus", which carries the actual contents of the memory cells to be read and written. Hence, while the "address bus" works like the labeling on an envelope of a letter, the "data bus" carries the contents of the letter.

However, these "physical addresses" are not the addresses seen by programs running on your computer! The program execution, the interpretation of the program code, is done within the microprocessor. Programs have of course to deal with addresses as well, as they contain instructions like "please tell me the contents of the storage cell xy". The addresses known by programs are the "logical addresses". These addresses have to pass thru a specific part of the microprocessor before they are made available thru the outside world, the memory management unit. The MMU is usually integrated in the same chip as the actual processing logic, with the only exception of the 68020 which has a separate external chip, the 68851. Still, the physical address lines from "outside chips" are connected to the 68851 and not to the 68020.

What's now the job of the MMU: It takes the logical addresses specified by the program, and translates it to the actual physical addresses, and makes only the translated physical addresses available to the outside world. Hence, it translates the labeling on the letter.

This is as if programs specifies street addresses only in - say - French, but the outside world reads only English labels. The MMU is the translator between the French and English street labeling system, but it does not care about the contents of the letter, hence it does not care about the "data bus". One might now think that the physical addresses are "more important" because they are "more real". This is not the case. The most important part of the computer is its "software", the programs, and these are speaking "French", as far as addresses are concerned. Hence, all addresses programs will print our your screen will be "French", i.e. "logical" addresses - unless the program is written smart enough to speak a non-native language.

Now, this memory management unit was not available on all Amiga systems from the beginning. The 68000 does not have a MMU, and back at that time, French and English street names used the same words. Things changed with the introduction of the 68020, but to remain backwards compatible, the MMU was most of the time disabled and had nothing to do except for a few "special" programs.

Where's now the problem with these two languages? If all programs speak French, there shouldn't be any confusion? Now, there is a problem: The microprocessor, the execution unit, is not necessarily the only part of the computer system that has access to the "address bus". For example, the microprocessor doesn't usually load data from the hard-disk itself and places it into memory. This is usually done by "asking a specialist", a so called "DMA controller", to read data from the hard disk and put it into the memory at location "xy". As long as the "specialist" is busy, the CPU is free for other operations - which is the reason why "DMA access" is so fast: It works parallel to the computing.

See also: What is DMA, please?

Did you notice the problem? Obviously, the program has to tell the specialist where to put the data to be read, and hence has to ship a letter from the program to the specialist, with the contents of were the data to be read should be put. The letter will arrive fine, even if it is labeled in French, because of the job of the MMU. It will translate the address for the program. But the MMU translates only the address label, and not the contents of the mail - which contains a French "logical" address and not an English "physical" address. Hence, for these special operations, the program, in this case the disk driver, has to carry out the translation manually - a translator is required. This is the matter of the operating system.

Unfortunately, due to the history of the Amiga, some disk drivers do not consult the operating system for translations and write French letters instead of English letters, without knowing that there's a difference.

1.5 Compatibility Guidelines

The mmu.library and its tools are written in a very careful way, to ensure maximal compatibility. However, some manufactures choose not to follow the CBM development guidelines too closely, leaving some gaps.

First here's a list of known restrictions, sorted by CPU type:

68000 and 68010 based systems

68020 based systems

68030 based systems

68040 based systems

68060 based systems

Systems with a PPC processor

Additional problems may arise for certain hard-disk and other DMA related controllers:

Programmed I/O interfaces

These components do not use DMA to transfer data to and from the storage medium to the memory. This holds for most IDE controllers as for example the A4000 "scsi.device" (which is truly not SCSI, but IDE), and for some SCSI host adapters like the "oktagon" controller.

DMA based interfaces

Most SCSI interfaces belong to this category, the "gvpscsi.device", the "omniscsi.device" found on GVP boards, the "cybscsi.device" and the "cybppc.device", the A3000 and the A4000T (not the A4000) "scsi.device", and lots of others.

Non-auto-configuring memory is not really a problem for the mmu.library, but it can be added to the system in a more elegant way without third-party tools:

See here: Non-auto-configuring memory

1.6 What is DMA, please?

DMA is short for "direct memory access". This is a mechanism for performing fast input/output operations, as reading data from a storage system like a hard-disk.

On a "programmed I/O" system, the CPU reads the data byte by byte from the controller chip connecting the HD to the computer system, and places the data byte by byte into the memory where it should be put. Needless to say that this keeps the CPU busy, especially if a lot of data has to be read in a fast way.

A "DMA" system works differently: The CPU tells the controller chip which data it should read, and where the controller should put this data into memory. The CPU then just starts the transfer and leaves it to the controller chip to carry out the actual work. In the meantime, the CPU is free to perform other computations. Hence, DMA is usually faster, because it works in parallel to other operations, but it is not without problems.

See also: Logical vs. Physical Addresses and the DMA problem

1.7 What is AutoConfig, please?

"AutoConfig" is a combined hardware/software protocol developed by Commodore to make expansion hardware known to the system. The operating system checks board expansions within the boot process, using this protocol, and links the found expansions into a list which is available to the MMU library and other tools. Especially, it ensures that this hardware is made visible by the CPU, using the MMU. Therefore, it has been highly encouraged by CBM to use this protocol for hardware expansions. Nevertheless, some board manufacturers choose not to follow these guidelines, to allow "cost efficient solutions".

Hardware which is not auto-configuring and which is neither standard-hardware will require some special versions of the 68040 or 68060 library, provided by the manufacturer. This is not a problem for the mmu.library, but it means that some memory is wasted because the MMU configuration built by the manufacturer supplied library will remain in memory, even though it is no loaded into the MMU. Hence,

It will work, but it will require more memory than necessary.

Special problems arise with the 68060: Most 68060.libraries will not allow 100% reliable virtual memory applications, this is only possible with the 68060.library in this archive. This is because the 68060 and the 68040 do not implement all instructions in hardware and use a tricky mechanism to emulate these in software. Due to how this emulation works internally, the 68040 does not, but the 68060 does require certain care about situations where the emulation accesses virtual memory. In short,

Virtual memory on a 68060 will not work 100% reliable on a system without using the 68060.library in this archive. This is NOT a problem for the 68020/68851, the 68030 nor the 68040. Failure situations will be rare, but possible.

Experts will be able to run the "generic" 68040/68060.library versions in this archive anyways by editing the MMU-Configuration file, but except for that, the mmu.library is very tolerant.

1.8 What is virtual memory, please?

Virtual memory is the emulation of memory which is physically not available. This sounds like something impossible, but due to the MMU tricks, it is not.

It means in detail that memory, which is currently allocated by programs, but which is not made use of is stored on the hard disk and made available for other programs. As soon as a program tries to access memory which isn't really there, the parts of the operating system intercept and load the data back from the hard-disk to the main memory, possibly making room by storing other program data currently not in use on the hard disk.

Hence, virtual memory simulates more main memory by making use of the storage capacity of a hard-disk. Up to 1GB of memory can be emulated by this method, but for the price that accessing memory might be slow: Simply because it is possible that this memory has to be "swapped in" from an external storage medium.

Special problems arise with the 68060: Most 68060.libraries will not allow 100% reliable virtual memory applications, this is only possible with the 68060.library in this archive. This is because the 68060 and the 68040 do not implement all instructions in

hardware and use a tricky mechanism to emulate these in software. Due to how this emulation works internally, the 68040 does not, but the 68060 does require certain care about situations where the emulation accesses virtual memory. In short,

Virtual memory on a 68060 will not work 100% reliable on a system without using the 68060.library in this archive. This is NOT a problem for the 68020/68851, the 68030 nor the 68040. Failure situations will be rare, but possible.

1.9 What is Zorro-II memory, please?

Zorro-II memory is memory on expansion cards designed for the A2000 and A500. It uses a less effective protocol for accesses because the faster Zorro-III protocol was not yet invented at the time of the A2000 and A500, and Zorro-II was sufficient for the slower 68000 used at that time.

Zorro-II expansions still work in the more advanced Amiga models, but they will be as slow as in the A2000 and A500. The special problem with this type of expansion is that some logic near the CPU has to split up the 32-bit wide accesses of the CPU into smaller 16-bit wide accesses to fit into the Zorro-II protocol. This does not work correctly for all types of boards under all circumstances.

In case you encounter problems with that, see also: Zorro-II 16-bit Memory Problems to find information on how to fix this.

For the experts: Zorro-II memory is found in the address range 0x00200000 to 0x00a00000.

1.10 68000 and 68010 based systems

Systems based on a 68000 or 68010 do not cause any trouble with the mmu.library because they lack a MMU. The mmu.library and its system components can be installed on these systems, but they won't be able to do much work for you. Of the mmu.library, only the administration calls will work, but all calls making use of the MMU - and these are most, as this is the job of the library - will return a failure code.

Hence, even though installation is possible, this doesn't make any sense to do so.

1.11 68020 based systems

Systems with a 68020 CPU may or may not have a MMU. Since the 68020 does not come with a built-in MMU, an external chip, the "68851 MMU" is required. If this chip is not available and not installed on your system, it doesn't make much sense to install the mmu.library. It won't fail, but it just wouldn't be able to do much for you, see also: 68000 and 68010 based systems.

If this IC is available, the 68020/68851 combo behaves very much like the 68030 and you should check the 68030 compatibility notes. The only difference between the 68020/68851 and the 68030 is that enabling the 68851 MMU costs more speed than on a 68030 system.

1.12 68030 based systems

Two types of 68030 CPUs are available and have been used in the Amiga hardware: True 68030's and the 68EC030. The 68EC030 is a cheaper variant of the 68030 without a MMU, but the mmu.library will be smart enough to detect this chip and will fail on a 68EC030. If you aren't sure about whether you own a true 68030 or just an "EC", just install the package and try, it will tell you. Using the mmu.library on systems without a MMU is possible, but doesn't make much sense, see 68000 and 68010 based systems.

On a true 68030 system, there are little problems, except for those caused by some DMA controllers you usually won't run into. In the case you make use of any tools that take over the MMU, as for example "CPU FastRom" or "SetCPU FastROM", or Michael Sinz "Enforcer", you should either start the MMU library after these tools, or use the corresponding MuTools in the first place.

See here: Software replacement list

1.13 68040 based systems

Systems based on a 68040 usually have a MMU; systems with an 680EC40 processor without an MMU are rare, but can be found. Installation of the mmu.library on these systems doesn't make much sense, of course.

See also: 68000 and 68010 based systems

All other 68040 based systems fall into two categories: Those that follow the guidelines and those that don't. The MMU library will work on both systems, but systems following the guidelines will get a special bonus, namely a new 68040.library. Those that don't will require running the 68040.library provided by the board manufacturer. This will work, but at the cost of higher memory consumption.

The rule whether you can use the V40 68040 library is in fact very simple: In case your system runs fine with the original Commodore 37.30 68040.library, it will continue to work fine with the V40 edition. The advantage of the V40 68040 lib is that it is shorter and uses less memory because it makes use of the mmu.library for all the tricky details. It is also slightly, but unnoticably faster than the 37.30 and uses better math support code, the latest version available by Motorola.

In case the 37.30 does not work, you might make the V40 68040 lib working by tweaking the MMU-Configuration file, but this should be done by experts only. Installation notes and an half-automated setup procedure are described in the following section:

Installation of the MMU-Configuration

In case you make use of any tools that take over the MMU, as for example "CPU FastRom" or "SetCPU FastROM", Michael Sinz "Enforcer" or the "OxyPatcher", you should either start the MMU library after these tools, or use the corresponding MuTools in first place.

See here: Software replacement list

1.14 68060 based systems

Systems based on a 68060 usually have a MMU; 68060 based systems fall into two categories: Those that follow the guidelines and those that don't. The MMU library will work on both systems, but systems following the guidelines will get a special bonus, namely a new 68060.library. Those that don't will require running the 68060.library provided by the board manufacturer.

There is no easy rule which systems do follow the guidelines, but in case of doubt, you should try to install the V40 68060 library and check whether the system remains working. The advantage of the V40 68060 lib is that it is shorter and uses less memory because it makes use of the mmu.library for all the tricky details. Moreover, 100% reliable virtual memory is not possible except when using this special edition.

See also here: What is virtual memory, please?

In case the V40 does not work immediately, you might make the V40 68060 lib working by tweaking the MMU-Configuration file, but this should be done by experts only. Installation notes and an half-automated setup procedure are described in the following section:

Installation of the MMU-Configuration

In case you make use of any tools that take over the MMU, as for example "CPU FastRom" or "SetCPU FastROM", Michael Sinz "Enforcer" or the "OxyPatcher", you should either start the MMU library after these tools, or use the corresponding MuTools in first place.

See here: Software replacement list

1.15 Software replacement list

In the case you make use of any tools that take over the MMU, as for example "CPU FastRom" or "SetCPU FastROM", Michael Sinz "Enforcer" or the "OxyPatcher", you should either start the MMU library after these tools, or use the corresponding MuTools in first place, here's a list:

ROM remappers like "CPU FastROM", "SetCPU FastROM" and "QuickROM" are replaced by MuFastROM.

"The Enforcer", and the "CyberGuard" is replaced by "MuForce", using basically the "Enforcer" sources by Michael Sinz. See also: Credits.

"GuardianAngel", "MemSniff" and the "GUARD" option of the CyberGuard are replaced by MuGuardianAngel

"PrepareEmul" is replaced by MuMove4K PrepareEmul

The "MoveSSP" argument of MuFastZero is an extra that allows you to re-map the supervisor stack as well. Details are in the program documentation.

"SetCacheMode" is replaced by MuSetCacheMode

"SpeedyChip" is replaced by MuFastChip

Some special display drivers for the Macintosh "ShapeShifter" emulator shouldn't be run with the MMU.library active. There's not yet a clean replacement.

Tools like "FastExec" placing the exec.library in fast memory or mounting memory should be replaced by a combination of "MuMove4K" and "MoveFastZero", and by editing the MMU-Configuration file. The "MuMove4K" program should go in place of "FastExec", "MuFastZero FORCENATIVE FASTEXEC ON" should go somewhere behind "SetPatch".

For how to mount memory, please check this: Non-Auto-configuring Memory

Very important: All other programs that modify the MMU table and do not use the MMU library must be loaded IN FRONT OF all the "MuTools" with the exception of MuMove4K which can be put in front of SetPatch.

1.16 Boards with a PPC processor

The PPC processor itself doesn't cause any problems, but the system software controlling the processor might. Two different software drivers are available for the PPC. The first one is "PowerUp" and the "ppc.library", the second is "WarpUp" and its "WarpOs". For short: The mmu.library will only work with "WarpUp". But this shouldn't be a loss since an emulation version for the ppc.library is available under WarpOs, written by Frank Wille.

The details are that the original "ppc.library" hacks the MMU itself without giving other programs *any* control over it and without documenting how it makes use of it. I would be able to offer support for the ppc.library provided I would get documents about its internals, but I don't. The manufacturer refused to publish these internals. Since I regard the construction of the ppc.library as rather "hacky", this is in my opinion not really a loss.

The mmu.library might be able to work on a PPC emulated 68040, but these are future plans because no such board is yet available, even though the software emulation is.

1.17 Programmed I/O interfaces

Firmware of programmed I/O interface hardware does not cause *ANY* trouble with the mmu.library at all. This includes for example the popular "oktagon" SCSI host-adapter, as well as most IDE interface adapters like the A4000 "scsi.device" and others.

See also: What is DMA, please?

1.18 CybSCSI problems

Due to a firmware "feature" of the "cybscsi.device", "MuGuardianAngel" will detect an illegal memory access of the device driver each second. This is because the device accesses memory it never allocated, and because the job of MuGuardianAngel is just to find these accesses, it will complain about the device. To fix this problem, run the "FixCybAccess" program before launching the "MuGuardianAngel". I do not plan to integrate workarounds for firmware using questionable techniques that could have been avoided easily into "MuGuardianAngel" itself.

1.19 DMA based interfaces

DMA based interfaces might cause problems with some system software making use of the mmu.library. The current collection of the "MuTools" does, however, not cause problems at all but "MuGuardianAngel" due to a firmware "feature" of one popular device. And even that problem is not truly DMA related.

See here: CybSCSI problems

To guarantee compatibility with future applications of the MMU, the use of two operating system functions have been encouraged by CBM, named "CachePreDMA" and "CachePostDMA". Various authors of DMA device drivers decided, however, not to use them, for various reasons: First, the interface provided by these functions is clumsy and not very well thought about, leaving lots to be desired, and not really providing the power these functions have been designed for in the first place. And second because they slow down the device access a bit. There exist currently a work-around for Ralph Babel's "omniscsi.device" alone.

See also : MuOmniScsiPatch and Credits.

This patch should be used whenever a "omniscsi.device" is in the system, or can be installed in the system. If possible, the older "gvpscsi.device" should be replaced by the "omniscsi". For details, check the MuOmniSCSIPatch guide

Device drivers that use these functions correctly are rare, the "scsi.device" of the A3000 and A4000T is an example.

See also: Logical vs. physical addresses for more insight into the DMA problem.

To say that again: There is currently no problem with that, but there might be a problem for future applications.

1.20 Non-Auto-configuring Memory

Some hardware manufactures require their customers to run special tools to make on-board memory available to the system. While this is clearly clumsy and a break of the AutoConfig concept, this does not mean trouble for the mmu.library. It is, however, recommended to run the third-party tool of the board manufacturer in front of the mmu.library and its tools, even though this is not strictly required. However, the MMU.library has to offer better methods to fix this. Here's one method, in a step-by-step procedure:

o) Hold both mouse buttons simulatenously and reboot your computer with Ctrl Amiga Amiga. Keep both mouse buttons hold until the boot menu shows up.

o) In the boot menu, choose "Boot without Startup-Sequence".

o) In the shell, type

"Tools/ShowConfig" Return

to run the ShowConfig program. It is part of the 3.1 workbench distribution.

o) Write down or save the paragraph describing the memory setup, starting at the line "RAM:" up to the line "BOARDS:". This describes all "auto-configuring" memory of your system which does not require special treatment.

o) Run the memory mounter provided by the board manufacturer and wait until it is done.

o) Run the "ShowConfig" tool again.

Compare its output with the first output. You should see now one additional entry in the "RAM:" section, describing the memory added by the tool. This output might look like follows, for example:

Node type $A, Attributes $5 (FAST), at $7000000-$7ffffff (16384 K)

The important information is here the rightmost number and the third number from the right. In this example, the "16384 K" describing the size of the memory, and "$7000000" describing its start address. Write down both numbers, check especially that you get the correct number of zeros!

The second number is the size of the memory in K-bytes. Use a calculator and multiply this number by 1024, write down the result. For the above example, this would be 16777216.

o) Now, edit the file "ENVARC:MMU-Configuration". For example, using the system editor "Ed". Hence, type the following command on the shell:

Ed ENVARC:MMU-Configuration" Return

o) Enter the following lines:

SetCacheMode from $7000000 size #16777216 valid copyback

AddMem from $7000000 size #1677216

where you've to replace the numbers in this example by what you got from "ShowConfig" and your calculation, of course. Make sure that you didn't forget the "$" and the "#" sign in the lines. These two lines tell the mmu.library to mount the memory on startup, and to make it visible to the CPU.

o) Save the results by pressing Esc x and then Return.

o) Copy the MuLockLib tool from the archive to the C: directory. In case you unpacked the archive to "SYS:", this would be

copy SYS:MuTools/MuLockLib to C:" Return.

o) You've now to modify your startup-sequence and to remove the third-party tool that used to mount the memory. To do this, enter the following command on the shell: "Ed S:Startup-Sequence" Return

o) Locate the line which invokes the third-party program. Place a semicolon in front of it to disable it. The mmu.library will take over to mount this memory.

o) Go back to the start of the line, in front of the semicolon. Press Return once to make some room in the file. Move upwards one line and insert the following command to load the mmu.library:

MuLockLib

o) Save the results with Esc x and again Return.

o) Wait until the HD busy light turns off and reboot the machine.

1.21 Zorro-II 16-bit Memory Problems

Do you use a memory expansion board in an A2000 or A500 with a 68040 or 68060 based environment? If this is the case, then it might be possible that enabling the CPU cache for this type of memory is fatal and causes a very unstable system. This can be worked around by disabling the CPU caches explicitly for Zorro-II memory. Here's a step-by-step solution:

o) In case your computer doesn't start before you're able to enter any command, hold both mouse buttons, and reboot it with the three-key-sequence. In the boot menu, disable the startup-sequence and wait for the shell. Do NOT run SetPatch.

o) Edit or create the MMU-Configuration file with an editor of your choice. Hence, enter in the shell

Ed ENVARC:MMU-Configuration Return

o) Add the following line to the file:

SetCacheMode from 0x00200000 size 0x00800000 cacheinhibit

This will prevent all caching for Z-II memory.

The default cache mode is set by the 68040.library or 68060.library you use. In case you're running the libraries contained in this distribution, the default is set by the KickStart ROM on startup, which again disables caching for the Z-II address space. This might slow down your system unnecessarily, though.

See also: What is Zorro-II memory, please?

See also: How to speed up the computer

About the background: The Zorro-II bus is only 16 bit wide, whereas the 68040 and 68060 have a pure 32 bit bus. This means specifically that the 32 bit accesses of the microprocessor have to be broken up into 16 bit accesses for the Zorro-II bus. This works usually fine on all boards, but with one exception: If caching is enabled, the CPU will try so called "burst accesses" to this memory, reading four longwords rapidly one after another. Some boards are not able to break up burst accesses into 16 bit memory accesses and hence will read non-sense data instead. One board that suffers from this bug is for example my GVP Combo 040/33 board.

1.22 Contents of the Archive

The mmu.library comes not alone: Quite a number of tools are provided, to make it useful and to offer an "all in one" solution. Here are the contents of the archive:

MuTools Directory. This directory contains various tool programs making use of the MMU library, for customers as well as for developers.

MuForce MuGuardianAngel MuSetCacheMode MuMove4K MuLink and MuOVLYMGR

MuFastROM MuFastZero MuFastChip MuLockLib MuScan MuOmniScsiPatch

Autodocs This directory contains the documentation of the mmu.library in the "autodocs" format. This is for developers only.

"mmu.doc" Autodocs of the mmu.library "exceptions.doc" Details about the exception handling mechanism of the mmu.library. "disassembler.doc" Autodocs of the disassembler.library. "680x0.doc" Autodocs of the 680x0.library. "68040.doc" Autodocs of the 68040.library.

Include This directory contains the include files for most C compilers and assemblers to make use of the mmu.library and the disassembler.library. This is provided for developers only.

vbcc Include files especially for the "vbcc" C compiler, contributed by Michaela Prüß. See also: Credits.

ReadMe Latest news and changes made to the contents.

History Version information.

LIBS Contains various system libraries used by the "MuTools" and related programs. These should be copied to "LIBS:" on startup.

mmu.library This is the library what all text is about. Should be copied to the "LIBS:" directory on installation.

disassembler.library This library is used by MuForce and MuGuardianAngel to provide disassembled outputs of the binary code. It should be copied either to LIBS: or to the "MuTools" directory on installation or disassembly won't be possible.

68040.library This is the V40 edition of the 68040.library. Some care has to be taken before installing it, more on this is in the 68040 compatibility chapter.

68060.library This is the V40 edition of the 68060.library. Some care has to be taken before installing it, more on this is in the 68060 compatibility chapter.

680x0.library This is the master CPU dependent library. If the SetPatch update has been installed, SetPatch will load this library on startup. It's then the matter of the 680x0.library to check for the CPU type and load the correct processor support library, e.g. the 68040 or the 68060.library.

This library is for the CPU and FPU what the mmu.library is for the MMU: The abstraction level for CPU and FPU control. For example, it is required by the "FPU" command to setup the exceptions the FPU might generate.

Shell_Only Various tiny tools that can be invoked from the shell only and which are not directly related to the MMU.library.

PrintTTX Prints the setup of the transparent translation registers. Sometimes useful for debugging. For experts only.

ClearTTX Clears the transparent translation registers, similar to the "ClearTTX" command in the MMU-Configuration file.

PrintBusError Prints the location of the "access error handler" currently installed. This is for experts only.

ResetBusError Resets the "access error handler" to its default location. Should NOT be run after the mmu.library has been loaded. Again, for experts only.

CheckFPU Prints the internal release number of the 68040 FPU. Again, only for experts.

FPU Enables and disables exceptions the FPU might generate. Disabling them is usually not required unless to work around broken software. This command requires the 680x0.library and the V40 edition of the processor libraries or it will either fail or do nothing.

C_Sources This directory contains the source code of various "MuTools" and other demonstration sources for documentation purposes. The sources demonstrate the developer how the MMU library can be made use of in own programs.

Fixes Various fixes for system software. At the current time, this directory contains the following files:

SPatch A patch program, used to update and patch in the fixes.

mathieedoubbas.pch The fix for a bug in the V38 mathieeedoubbas.library, it fails to compare some floating point numbers correctly.

SetPatch.pch Patches "SetPatch" to load the 680x0.library instead of the 68040.library on startup, regardless of the processor available.

SetPatch_44.pch Patches the V44 edition of "SetPatch" to load the 680x0.library instead of the 68040.library.

FixCybAccess Works around a firmware "feature" of the cybscsi.device and is required for owners of this device to be able to run MuGuardianAngel without receiving a hit every second. More is here:

FixCybAccess

ConsoleFix Fixes a bug on window resize of the console.device. Not really required to run the MuTools, but good to have. More's in the ReadMe for the program.

PatchRAM Increases the stack size of the RAM disk to avoid justified "MuGuardianAngel" hits from RAM: More's in its ReadMe.

narrator.device.pch Patches the narrator.device to increase the stack size of the narrator main task, which is really a bit low on stack.

Fixes.ReadMe More about the "Fixes" directory.

Install Several tools required for "automated" generation of the MMU-Configuration file.

FindPort A tiny program that scans for a named public port. This program is currently not made use of.

FindResident Another program that scans for a resident tag in the Os. Again, this program is currently not required by the installation procedure.

ShowBoards This program lists all autoconfiguring hardware expansions installed in the system. It is required by the automated setup procedure.

P5Identify This hack scans for non-autoconfiguring P5 hardware. It is required for the setup-script to recognize P5 hardware correctly.

PPCIdentify Just another hack that scans for P5 PPC implementations which are again (*sigh*) not autoconfiguring. Required by the setup script.

ScanMMUPort This is an external command of the mmu.library which is made use of to load the (undocumented) P5 MMU caching database into the mmu.library on startup. It will be copied to LIBS:mmu by the setup script if P5 hardware is found, and will be run as part of the MMU-Configuration script afterwards.

BuildMMUConfig.rexx The setup script. This script scans for all hardware, compares the hardware with its database and generates a setup file for the mmu.library automagically. It tries to set the optimal caching modes for graphic cards, too.

This script must be run WITHOUT the 68040 or 68060.library active or the result might be unaccurate.

It takes one argument, a file name, and will generate an MMU-Configuration file of this name. The argument should be "ENVARC:MMU-Configuration", of course.

More details are here:

Installation of the MMU-Configuration

1.23 MuForce

MuForce is a MMU library and option-compatible replacement for Michael Sinz's famous "Enforcer" tool. It is mainly based on Michaels sources, with some adjustments and enhancements. More about it is found in its own guide:

The MuForce guide

1.24 The disassembler.library

The disassembler.library is a system library offering functions to disassemble 680x0-code into human-readable form. It is optionally used by MuForce and MuGuadianAngel for its printouts. The library need not to be available system-wide, it is enough to have it in the "MuTools" directory such that these two tools have it available. Use by other programs is, however, welcome as the autodocs are available.

It is based on the disassembler by the debugger "COP" of the same author.

1.25 MuGuardianAngel

MuGuardianAngel is a debugging tool much like the original "GuardianAngel" or the "GUARD" option of "CyberGuard". It is a completely new mmu lib conforming rewrite of this tool. It checks programs for illegal memory accesses, either read or write accesses, to memory regions that haven't been allocated correctly, and it will warn you in case the memory list is corrupt. This tool must be run on top of MuForce, i.e. MuForce must be run first, then MuGuardianAngel. More about the tool is it's guide:

The MuGuardianAngel guide

Because the cybscsi.device does an illegal memory access every second, FixCybAccess must be run by owners of the cybscsi.device in order to fix this.

See also: CybSCSI problems

1.26 FixCybAccess

FixCybAccess is a tiny fix for a firmware "feature" of the cybscsi.device. This device accesses a chip-mem location every second without having it allocated correctly, causing MuGuardianAngel to warn you about this problem. To avoid these error messages, run this tool in front of MuGuardianAngel. I do not plan to include this fix in MuGuardianAngel since this is really a problem of the "cybscsi".

1.27 MuSetCacheMode

MuSetCacheMode is a tool for "fine-tuning" the MMU tables built by the mmu.library. It can be used to make some non-auto-configuring hardware working or to disable or enable certain caching modes for memory regions in case the memory doesn't allow faster or cached accesses. This is definitely an expert tool and replaces the "SetCacheMode" tool of some third-party developers.

Anyways, unless dynamic changes of the MMU tables are not required, it is recommended to edit the MMU-Configuration file instead.

More details are in the MuSetCacheMode guide

1.28 MuMove4K

MuMove4K is a memory preparation tool for the MuFastZero "FastExec" option, and for the "Shapeshifter" Macintosh emulator. If used, it should be placed very early in the startup-sequence as it will reboot the computer on its first invocation.

More here: MuMove4K guide

This tool is a bit misnamed because it moves actually 32K and not 4K, but is named as such for traditional reasons.

1.29 MuLink

MuLink is a software post-processing tool for developers. It enables memory-protection for selected parts of an executable and hence protects parts of the binary from getting overwritten. It requires the "overlay manager" MuOVLYMGR for its job, this part is linked to the executable to provide the protection.

See also: MuLink documentation

1.30 MuOVLYMGR

The MuOVLYMGR is not a self-running program but a binary file containing the code for memory protection, used by MuLink. This code is linked to executables instead of executed alone.

MuLink documentation

1.31 MuFastROM

MuFastROM is a mmu.library conforming ROM remapper. It creates a mirror image of the ROM in faster RAM, hence speeding up ROM accesses noticeably. It is a replacement for various hacks like "CPU FastROM", "SetCPU FastROM" and others.

Details: The MuFastROM guide

1.32 MuFastZero

MuFastROM is a mmu.library conforming vector table, execbase and supervisor stack remapper. It creates a mirror image of the vector table of the MC68K CPUs in fast RAM, speeding up all interrupts in a very compatible way. Additionally, if MuMove4K is run as well, it may even remap important system libraries to Fast RAM in case no auto-configuring RAM is available. This will speed up most programs a bit. Finally, it may remap the supervisor stack of the MC68K to faster memory if required. This helps again in interrupt performance.

Details: The MuFastZero guide

1.33 MuFastChip

MuFastChip boosts the chip memory on 68040 or 68060 based systems where the supplied 68040 or 68060.libraries can't be used. In case the MMU library has to build MMU tables from scratch, the boost will be activated automatically, as is the case if you use the V40 68040.library or 68060.library. Furthermore, this patch does nothing useful on a 68030 or 68020, even though it won't hurt.

Details: The MuFastChip guide

Just the same function can be obtained by editing the MMU-Configuration file by hand or by manually adjusting the cache mode by MuSetCacheMode.

1.34 MuLockLib

MuLockLib does nothing more but loads the mmu.library on its start and locks it in memory so it can't go away. Running the program again releases the lock. It should be called in the startup-sequence to ensure that the mmu.library remains available all the time.

Details: MuLockLib documentation

This tool was improved and enhanced by Gunther Nikl, see also: Credits

1.35 MuScan

The purpose of MuScan is to print the MMU table layout, i.e. the current configuration of the MMU. It can be used to check the function of various other tools like MuFastROM or MuFastZero and is usually not required but by experts. This is a replacement of Michael Sinz's "MMU" tool.

See also: The MuScan guide

1.36 MuOmniScsiPatch

The MuOmniScsiPatch makes Ralph Babel's "omniscsi.device" MMU library aware. That is, with this patch installed, the device will recognize the MMU table setup correctly and will perform a logical to physical address translation. Even though not required by the current tools using the mmu.library, it is recommended to install this patch.

More: MuOmniSCSIPatch guide

Thanks goes to Ralph Babel for providing documentation about the internals of the omnisci. Check the Credits section.

See also: Logical vs. physical addresses and More on DMA interface controllers.

1.37 How to speed up the computer

Here's the collection of some useful tips to get the best from your system:

Some boards setup the hardware in way that leaves the CPU caches disabled as soon as the MMU library is loaded. Even though this is justified for some boards, for example for those with broken Zorro-II memory, this is usually not required and caches can be enabled for all memory types. Hence, you should try the following:

o) Open a shell,

o) edit the file "ENVARC:MMU-Configuration" with an editor of your choice. The editor "Ed" is good enough for that and available on all systems. Hence, enter the command

Ed ENVARC:MMU-Configuration Return

o) Add the following line to the file

ClearTTx

and press Return.

o) Now save the file. In case you use the editor "Ed", press Esc, then x and Return.

o) Now reboot the system if you like.

This may, however, result in hangs and crashes for very few boards, related to a Zorro-II bus interface problem. In case you should encounter these problems, you've to edit the MMU-Configuration again. Here's the fix: Zorro-II 16-bit Memory Problems

Memory accesses for the "problematic" part will be slower, but at least working.

Additional speedup tips and tricks:

First, you might want to run MuFastZero. Other CPU libraries might enable this by default, but since this might potentially cause compatibility problems, I decided to leave it to you to turn it on explicitly. Add this line to the startup-sequence, somewhere behind "SetPatch", i.e. SetPatch must be run first, MuFastZero afterwards:

MuFastZero ForceNative MoveSSP On

In case important system libraries end up in slower memory than desired, there's also a fix for this. Add the following line before SetPatch:

MuMove4

and use the following options of "MuFastZero" instead to move the system libraries to faster memory:

MuFastZero ForceNative MoveSSP FastExec On

In case you can spare 512KB of memory, you also might want to remap the ROM to RAM where the CPU will access it more quickly. This is done by the following command, which should go somewhere in the startup-sequence below SetPatch:

MuFastROM On

In case you can't use the supplied 68040 or 68060.libraries, the chip memory might be accessed slower than necessary. The following line will speed up chip memory a bit for 68040 and 68060 based machines:

MuFastChip On

This line will do nothing on a 68030 or 68020 system, but it won't harm either. In case the MMU library build its MMU setup from scratch, faster chipmem is turned on anyways.

1.38 The V40 68040.library

The 68040.library found in this archive is NOT a CBM or Amiga provided version. It was entirely written by the author as part of the "mmu.library" project. It replaces the older "generic" 37.30 edition of Michael Sinz, providing the latest Motorola FPU "FPSP040" emulation sources for 68040 based systems. It makes use of the mmu.library to setup the MMU tree for the processor, reducing the overhead of having to build the MMU tree twice. Due to this, the "mmu.library" must be installed as well.

However, this library might not work on certain third-party products not following the AutoConfig standard for expansion hardware. Either, the manufacturer's 68040.library must be used, or special editing of the MMU-Configuration file is required.

For more insight, study the 68040 compatibility notes.

1.39 The V40 68060.library

The 68060.library found in this archive is NOT a CBM or Amiga provided version. It is an updated version of the 40.15 68060 library of Carsten Schlote, replacing other "generic" editions of the same library and providing a streamlined version of Motorola's "FPSP060" emulation package. This library guarantees correct emulation of certain unimplemented floating point instructions in a virtual memory environment, unlike other libraries. Furthermore, it makes use of the mmu.library to setup the MMU tree for the processor, reducing the overhead of having to build the MMU tree twice. Due to this, the "mmu.library" must be installed as well.

However, this library might not work on certain third-party products not following the AutoConfig standard for expansion hardware. Either, the manufacturer's 68060.library must be used, leaving VMM compatibility behind, or special editing of the MMU-Configuration file is required.

For more insight, study the 68060 compatibility notes.

See also: Credits

1.40 The MMU-Configuration File

The MMU parses its preferences, the file "ENV:MMU-Configuration" or "ENVARC:MMU-Configuration", on startup, provided it is available. This file can be used for experts to fine-tune the MMU tables built by the library, for example to disable or enable caching for Zorro-II memory if required, to map in third-party, non-auto-configuring hardware or memory and hence to obtain compatibility to products that do not follow the CBM guidelines too closely.

This file should not be touched unless you really, really know what you're doing. Modifying this file may easily cause crashes and hangs, and might result in certain "surprise" moments. This is definitely an "advanced feature".

The file is a pure ASCII file and can be edited with each text editor. For example, the system editor "Ed" is clearly good enough.

The syntax of the file is very much like the syntax of a shell script: All characters behind a semicolon ";" are considered to be a comment and are hence ignored. Everything else is a command and executed by the library. However, unlike the shell, some commands are build in the library, and external commands are not searched in the C: logical device.

The library knows currently only four build-in commands, everything else is considered an external file and tried to be loaded as external module from the "LIBS:mmu" directory. In case the library is, too, unable to locate an external command, the command is ignored and no error condition is generated. The syntax of the arguments follows the syntax of the shell, with the restriction that the input and output redirection characters ">" and "<" are not available because there is neither an input nor an output stream.

The following internal commands are available:

ClearTTX

AddMem

SetCacheMode

DescriptorCacheInhibit

The following external commands are available:

ScanMMUPort

For example, to add non-auto-configuring fast cacheable memory from 0x02000000 to 0x03ffffff, setup your "MMU-Configuration" as follows:

SetCacheMode from 0x02000000 size 0x01000000 valid copyback AddMem from 0x02000000 size 0x01000000

To mark Zorro-II memory as non-cacheable explicitly, use

SetCacheMode from 0x00200000 size 0x00800000 cacheinhibit

This will prevent "burst accesses" to Zorro-II 16Bit memory which is not possible for some Turbo-Boards, for example for the GVP040 Combo.

To enable Zorro-II caching even though it is disabled by the Os in the boot process if the "generic" version of the 68040 or 68060 library is used:

ClearTTX

1.41 ClearTTX

ClearTTX ITT0/S,ITT1/S,DTT0=TT0/S,DTT1=TT1/S,ALL/S

This command controls how the mmu.library uses the transparent translation registers. By default, they are considered, their setup is included in the MMU tree layout and they are cleared afterwards. Using this command, several TTx registers can be made to be ignored, even though they are still cleared because they are "in the way". It is usually a good idea to clear them or, in the default setup, the ROM will disable all caches for Zorro-II memory.

ITT0 Ignore the instruction transparent translation register 0. (68040 and 68060 only).

ITT1 Ignore ITT1 (68040 and 68060 only).

DTT0 Ignore the data transparent translation register number 0 (68040 and 68060) or the transparent translation register 0 (68030).

DTT1 Ignore DTT1 (68040 and 68060) or TT1 (68030)

DTT0 Ignore DTT0

ALL Ignore all TTx registers. This is the default if no other options are given.

This command does nothing on a 68851 because this MMU doesn't offer transparent translation registers at all.

1.42 AddMem

AddMem FROM=ADDRESS/A,LENGTH=SIZE/A,ATTR=FLAGS/K,PRI/K,NAME/K

Adds memory to the exec memory pool. Note that THIS DOES NOT make the memory visible, i.e. it is NOT AUTOMATICALLY marked as "valid". Hence, an "AddMem" command requires a "SetCacheMode" for the same memory region or the library will crash on startup.

FROM=ADDRESS Base address of the memory to be added, in hex notation. A leading $ or 0x is allowed. This address must be aligned to a 64K boundary.

LENGTH=SIZE Size of the memory to be added in bytes, in hex. Again, must be divisible by 64K.

ATTR=FLAGS Memory attribute flags in hex. Defaults to 0x05 which is MEMF_PUBLIC|MEMF_FAST. More flags are documented in exec/memory.h. The library *does not* know the mnemonics for the memory types, though, only numerical values must be used.

PRI Priority of the memory pool to be added, in decimal notation. This must be a number between -128 and 127. Defaults to 6.

NAME Name of the memory pool. Defaults to "MMU expansion memory".

This command *does not* make the memory visible to the processor, an additional SetCacheMode is required.

1.43 SetCacheMode

SetCacheMode FROM=ADDRESS/A,LENGTH=SIZE/A, COPYBACK/S,WRITETHROUGH/S,CACHEINHIBIT/S, NONSERIAL/S,IMPRECISE/S, VALID/S,BLANK/S, IO=IOSPACE/S,NOIO=NOIOSPACE/S, ROM/S,NOROM/S

This is a "cut down" version of the MuSetCacheMode command, it supports a sub-set of its cache control commands. Options that modify memory in a way that could result in access errors are not supported and must be setup by hand with "MuSetCacheMode".

FROM=ADDRESS The base address of the memory region whose cache mode shall be changed. This is in hex notation, a leading 0x or $ is allowed. The library will round this down to the next page size boundary, which is usually 4K or 1K.

LENGTH=SIZE The size of the memory region in bytes, in hex notation. The library will round this up to the next page size if required.

COPYBACK Enables the copyback cache mode. This is the fastest cache mode available, reads and writes are cached if the CPU allows this. This option will fall back to WRITETHROUGH on a 68030 or 68851.

WRITETHROUGH Enables the writethrough cache mode. Reads will be cached, writes will enter the cache but will be written out to memory as well.

CACHEINHIBIT Disables the cache completely.

NONSERIAL Disables the cache, but allows the CPU to reorganize accesses to speed up memory transfers a bit. (68040 only, does nothing on all others).

IMPRECISE Disables the cache, but allows the CPU to handle access errors a bit "sloppy" to speed up access a bit. (68060 only, does nothing on all others).

VALID Validates the memory region, i.e. makes it visible. This option is required to enable a memory region that should be added to the exec memory pool by the "AddMem" command.

BLANK Invalidates the memory region, all accesses will be remapped to a "dummy" page.

IO=IOSPACE Marks the memory region as mapped IO space, and sets the cache mode to CACHEINHIBIT. The cache mode can be overridden by using the cache control options.

This is a pure software flag which is, however, used by MuForce and friends. Memory marked as IOSPACE is never disassembled or dumped because of undesirable side effects that might result.

NOIO=NOIOSPACE Marks the memory as plain RAM space, negative form of IO=IOSPACE.

ROM Marks the memory as ROM space and enables the defensive write protection. Writes to this area will be ignored silently.

NOROM Marks the memory as RAM space, writes are allowed. Negative form of ROM.

1.44 DescriptorCacheInhibit

DescriptorCacheInhibit ON/S,OFF/S

This command is used to control the cache mode of the memory where the library will keep its descriptors. By default, the library will take no precautions about the cache type of the MMU descriptors because it is written carefully enough to read and modify descriptors even with the copyback cache mode enabled. However, some old programs may bypass the mmu.library and may want to hack the MMU themselves. This technique IS, AND HAS NEVER BEEN documented AND MUST BE AVOIDED. There is no, and never has been any guarantee that MMU tables are in cache-inhibited memory. Some 68040.libraries do this, others don't. Some hacks do, others don't. The MMU library usually DOES NOT, this option is OFF by default.

You may turn it ON as a workaround. This means that the MMU library will put the descriptors of each MMU table into cache-inhibit memory, AS SEEN FROM THE TABLE ALONE. If another MMU table becomes active, might it be because of a context switch or might it be because you switch from user to supervisor mode, all inactive tables will not be cache-inhibited. This means, specifically,

User descriptors are non-cacheable for accesses from user mode,

Supervisor descriptors are non-cacheable for accesses from supervisor mode.

But, User descriptors are not cache-inhibited from supervisor mode, for example.

1.45 ScanMMUPort

ScanMMUPort (does not take arguments)

Scans an undocumented database which is build by some P5 boards on power-up and adjusts the MMU library database accordingly. This database contains memory areas whose cache mode must be adjusted because it is used, for example, as buffer for the DMA controller.

1.46 Installation of the mmu.library

This library is the heart of the system, it doesn't make sense not to install it because all the tools in this archive require it.

o) This first step is simple:

Copy the "mmu.library" to LIBS:. This is the head of the system, nothing will work without it. Hence, enter:

Copy LIBS/mmu.library to LIBS: Return

1.47 Installation of the debugging tools

o) In case you want to run the debugging tools MuForce, MuGuardianAngel or PatchWork and want a disassembly of faulty code, the disassembler.library is required as well:

Copy LIBS/disassembler.library to LIBS: Return

o) If you want to keep the "MuTools", just drag the "MuTools" drawer wherever you want it.

o) PatchWork is copyrighted by Richard Körber and can be found in the "Contributions" directory. If you want to use it, copy it whereever you want.

1.48 Installation of the SetPatch upgrade

The SetPatch upgrade makes SetPatch loading the 680x0.library instead of the 68040.library, without looking at the processor. This makes it possible to boot from the same partition from different machines and different CPUs without having to re-install the correct processor driver library.

Hence, SetPatch will load the 680x0.library in first place which will then pick the correct processor driver for you. This is not just another dummy library. The 680x0.library offers, too, function entries to identify the processor, the FPU, the MMU type and to control the FPU exception. Hence, this library is also required by the "FPU" command.

The patched 43.7 or 44.3 edition does not contain new fixes, neither is it an official Amiga release, but it will load the 680x0.library on startup instead the 68040.library. It's then the matter of the 680x0.library to check for the CPU and load the correct CPU library. Additionally, this trick will work even for the 68020 or the 68010, it then checks for a "68020.library" etc. These libraries are not yet available, but might for example patch in a software FPU emulation if no FPU is available. The 680x0.library is for the CPU what the mmu.library is for the MMU, it comes with a set of user-callable LVOs that allow an abstraction from the available hardware.

Hence, this trick replaces the need for 68040 dummy libraries, and the 680x0.library is required for the "FPU" command anyways.

This patch is independent of the mmu.library or the 68040.library or the 68060.library in this archive, even though their installation is recommended if this is possible.

o) In case you want to install the "SetPatch" update or the "FPU" command, you've to install the 680x0.library as well. The 680x0.library can be used with ANY third party 68040 or 68060.library and is independent of the mmu.library, but the V40 editions are recommended. "FPU" won't work with the correct libraries installed, either.

Copy LIBS/680x0.library to LIBS: Return

o) Copy "C:SetPatch" to a safe place.

o) Copy "C:SetPatch" to RAM: to make it available for the patch.

Copy C:SetPatch to RAM: Return

o) Dependent on the SetPatch version installed on your system, copy the file Fixes/SetPatch.pch or Fixes/SetPatch_44.pch to the RAM disk:

Copy Fixes/SetPatch.pch to RAM:SetPatch.pch Return

or, for Os 3.5

Copy Fixes/SetPatch_44.pch to RAM:SetPatch.pch Return

o) Copy the file Fixes/SPatch to the Ram Disk:

Copy Fixes/SPatch to RAM: Return

o) Apply the patch:

ram:spatch -pRAM:SetPatch.pch -oRAM:SetPatch.new RAM:SetPatch Return

o) Copy new version of SetPatch back to C:

Copy RAM:SetPatch.new to C:SetPatch Return

In case you applied the patch make sure that you installed the 680x0.library or the system will start in slow-motion!

In case SPatch should warn you that it couldn't apply the patch, make sure that you really tried to patch the 43.6 version of SetPatch. It won't work with other releases.

o) In case you're interested, you might want to keep some of the tools in the Shell_Only directory. Especially, the "FPU" program might be of some use, it controls FPU exceptions and is able to disable or enable them. This requires, however, the 680x0.library and the V40 editions of the processor libraries.

1.49 Installation of developer material

In case you're a developer:

o) Copy the contents of the "Autodocs" directory to where you want to keep it, and copy the "Include" directory to the include directory of your C compiler. It contains all the material for writing programs using the library. In case you use the "vbcc" compiler, you need the material in the "vbcc" directory as well.

o) In case you want to keep the example sources, copy the "C_Sources" directory. This is again for developers only and contains the sources of some of the MuTools for demonstrational purposes.

o) You might want to keep the MuLink and the "MuOVLYMGR" files and their documentation to write programs which protect themselves from getting overwritten.

1.50 Installation of the 68040 Library

In case you own a 68040 based system and you're able to run the 37.30 version or the generic Os 3.5 edition of the 68040.library, you should update this library to the V40 release. This special 68040.library is not required in case you just want to install the 680x0.library and the "SetPatch" update, but it is recommended since it leaves the MMU setup to the mmu.library and is therefore shorter and wastes less memory.

copy LIBS/68040.library to LIBS: Return

In case you're not sure whether you can run this release, check the 68040 compatibility notes. In case of doubt, backup your old library to a safe place and install the 40.2 on top of it.

o) The default MMU-Configuration should now be setup. Later fine-tuning is of course possible:

copy ENVARC/MMU-Configuration to ENVARC: Return

Note: A higher version number does not always mean a better library. I'm very conservative about version numbering, so this library could have possibly been called a V50 release by other people.

1.51 Installation of the 68060 Library

In case you own a 68060 based system and you're able to run Carten Schlote's 40.15 68060.library, you should update this library to the V40 release. This special 68060.library is not required in case you just want to install the 680x0.library and the "SetPatch" update, but it is recommended since it leaves the MMU setup to the mmu.library and is therefore shorter and wastes less memory. Furthermore, it supports virtual memory correctly, unlike all other 68060.libraries I've seen.

copy LIBS/68060.library to LIBS:

In case you're not sure whether you can run this release, check the 68060 compatibility notes. In case of doubt, backup your old library to a safe place and install the new release on top of it.

o) The default MMU-Configuration should now be setup. Later fine-tuning is of course possible:

copy ENVARC/MMU-Configuration to ENVARC: Return

Note: A higher version number does not always mean a better library. I'm very conservative about version numbering, so this library could have possibly been called a V50 release by other people. All other 68060.libraries are not VMM compatible.

1.52 Installation of Software Fixes

The Os and some Os libraries contain unfortunately some bugs which may or may not conflict with the mmu.library and which have not been fixed for Os 3.5 (*sigh*).

o) The V38 mathieeedoubbas.library has a bug in the floating point "compare" routine that should be fixed. I ship this fix separately and did not include it in the 68040 library because I hoped (*sigh*) this bug would be get fixed by Amiga with the Os 3.5 release. Furthermore, I don't think that the 68040.library should be a replacement for SetPatch. Here's how to apply the fix:

o) Copy the file LIBS:mathieeedoubbas.library to RAM:

copy LIBS:mathieeedoubbas.library to RAM: Return

o) Copy the file Fixes/mathieeedoubbas.pch to RAM:

copy Fixes/mathieeedoubbas.pch to RAM: Return

o) Copy the file Fixes/SPatch to ram:

copy Fixes/SPatch to RAM: Return

o) Apply the patch:

ram:spatch -oram:new -pram:mathieeedoubbas.pch ram:mathieeedoubbas.library Return

o) Copy the new library back to LIBS:

copy RAM:new to LIBS:mathieeedoubbas.library Return

o) The "narrator.device" is a bit too low on stack and will generate justified warnings if MuGuardianAngel is run. The following patch will increase its stack size:

o) Copy the file "DEVS:narrator.device" to RAM: if you have it installed:

copy DEVS:narrator.device to RAM: Return

o) Copy the fix to RAM: as well:

copy Fixes/narrator.device.pch to RAM: Return

o) Apply the patch with

ram:spatch -oram:new -pram:narrator.device.pch ram:narrator.device Return

o) Copy the fixed binary back to DEVS:

copy RAM:new to DEVS:narrator.device Return

o) The console device window resizing procedure contains a bug which may occasionally cause problems if the console.device gets low on stack. To fix this bug, copy "Fixes/ConsoleFix" to C:

copy Fixes/ConsoleFix to C: Return

o) edit the Startup-Sequence with an editor of your choice and run "ConsoleFix" somewhere behind the "SetPatch" command.

Ed S:Startup-Sequence Return

o) Save the modified Startup-Sequence back. In case you're using "Ed", this would be Esc-X and Return.

o) The system RAM disk is a bit low on stack and MuGuardianAngel will complain about it if it is run. To fix the RAM disk, "PatchRAM" must be run in the Startup-Sequence before the RAM disk is used for the first time.

copy Fixes/PatchRAM to C: Return

Ed S:Startup-Sequence Return

o) Now check your startup-sequence for the first command that makes use of the Ram-Disk. This is typically something like "makedir RAM:T" to build the temporary files directory. Run "PatchRAM" in front of this command. Then save the startup-sequence back. In case of "Ed", this is Esc-X and Return.

o) In case you own a cyberscsi.device and want to use the "MuGuardianAngel", one "Firmware Feature" of the device must be fixed with the FixCybAccess program which should be run in front of the MuGuardianAngel program. If you want to run it in the startup-sequence, you've to edit it accordingly:

copy Fixes/FixCybAccess to C: Return

Ed S:Startup-Sequence Return

1.53 Installation of MuLockLib

MuLockLib is a tiny program that loads the mmu.library in background and locks it in memory so it can't be flushed any more. This step is not really required, but it reduces the loading time of the other "MuTools".

Furthermore, it is superfluous if you installed the 680x0.library or the 68040.library or 68060.library in this archive because these libraries already lock the mmu.library in memory and therefore perform the "MuLockLib" functionality automatically.

o) In case you want to install "MuLockLib", copy it to C: and run it in the startup-sequence. In case you really have to run MMU hacks - not the MuTools of course, MuLockLib must be run after these hacks, but should be run in front of all other "MuTools" or it is simply wasted since the first "MuTool" will load the mmu.library anyhow.

copy MuTools/MuLockLib to C: Return

o) The command to be inserted is "MuLockLib >NIL:"

Ed S:Startup-Sequence Return

o) Check now your Startup-Sequence for MMU-Hacks, and replace them by the MuTools whenever possible. In case of doubt, run "MuLockLib" behind these tools.

1.54 Installation of MuOmniSCSIPatch

In case you own an "omniscsi.device" driven SCSI adapter, for example an upgraded GVP host adapter, you might want to run the "MuOmniSCSIPatch". This little program makes the device "MMU-aware", i.e. it will be notified correctly about remapped memory, write protection, etc, and will use the correct phyiscal addresses for its DMA operation.

o) In case you want to install the patch: Copy the MuOmniScsiPatch to C:

copy MuTools/MuOmniSCSIPatch to C: Return

o) and run this patch in the startup-sequence whereever you want. If MuLockLib is installed, the recommended place is below this command. The command to be inserted is simply "MuOmniSCSIPatch >NIL:"

Ed S:Startup-Sequence Return

o) Save the changes back to disk. For "Ed", this is Esc-X and Return.

1.55 Installation of MMU-Hack replacements

Some MMU hacks should be better replaced by mmu.library and therefore compatible counterparts. This requires editing the startup-sequence manually.

o) Edit the startup-sequence with an editor of your choice:

Ed S:Startup-Sequence Return

o) Now check the Software replacement list for mmu.library compatible tools. Use the MuTools whenever possible. If some non mmu.lib aware tools could not be replaced, then re-organize your startup sequence in a way that the mmu.library gets loaded after these hacks have been run so the library will be able to consider the hacks.

o) Save the startup-sequence back. For "Ed", this is Esc-X and Return.

1.56 Installation of the MMU-Configuration

The "ENVARC:MMU-Configuration" file is the "preferences" file for the mmu.library. It is scanned on startup of the library and controlls the way how the library builds the MMU tables. The default file or even no configuration file at all will be usually good enough as the library uses a very conservative setup anyhow, but special adjustments must be made in case you run the 68040 or 68060.library in this archive with P5 hardware that does not make use of the AutoConfig standard to identify itself.

Unexperienced users should not try the steps descriped here but should rather use the board-specific 68040/68060.libraries that came with their hardware, and should not try to install the 68040 and 68060.library in this archive.

o) You might either want to try to edit this file manually yourself. This step should be left, however, to the experts.

More on ENVARC:MMU-Configuration

o) Another way of getting the new 68040/68060.libraries to work with the P5 hardware is to run an automated setup script. Perform the following operations only after all other installation steps have been performed. Open a shell. On the shell, first change to the "Install" directory of this archive, and then run the script:

cd MMULib/Install Return

SYS:Rexxc/rx BuildMMUConfig.rexx ENVARC:MMU-Configuration Return

o) This type of installation is still in an experimental state, so the advanced user might want to edit this file manually afterwards:

ed ENVARC:MMU-Configuration Return

1.57 Installation of non-autoconfiguring memory

For those of you with non-auto-configuring memory: It is possible to replace various third-party "AddMem" tools by the mmu.library. For first, leave the system like it is, the steps required to make the mmu.library mount this memory require a reboot and editing the MMU-Configuration file by hand. Therefore, take some time and read the Non-Auto-configuring Memory section for details, possibly print it out and then start the installation of this feature after all other modifications have been performed.

1.58 Installation of Zorro-II memory fixes

In case you've Zorro-II 16 bit memory in your system, it might be required to turn off caching for this memory explicitly: If your system behaives very unstable, or crashes almost immediately after booting, you might own a board or a memory expansion that does not handle "Burst Access" correctly and makes it impossible to leave some memory blocks cacheable. This will slow down the system, but it will run stable, at least.

o) Usually, configuration of this feature is not required, though. In case of doubt, open or create the file ENVARC:MMU-Configuration with an editor

Ed ENVARC:MMU-Configuration Return

o) and add the following line:

SetCacheMode from 0x00200000 size 0x00800000 cacheinhibit Return

o) This will disable caching for the full Zorro-II area. Now save the file back. In case of "Ed", this is done by Esc-X and Return.

1.59 Installation

Installation depends heavily on what of the MuTools and of the mmu.library is required by your system. Moreover, the supplied 68040 and 68060 libraries will not work on every board. Here's a step-by-step installation procedure:

o) Open a shell. This goes for all installation steps below.

Highly recommended: Installation of the mmu.library

Recommended: Installation of the MuTools

Optional: Upgrade SetPatch, install the 680x0.library

Optional: Installation the AutoDocs and Example Sources

Optional: Installation of the new 68040.library

Optional: Installation of the new 68060.library

Highly recommended: Installation of Software Fixes

Optional: Installation of MuLockLib

Optional: Installation of the MuOmniScsiPatch

Highly recommended: Replace MMU hacks

For experts, and for P5 compatibility: MMU-Configuration adjustments

For systems with non-auto-configuring memory: Mount memory by the mmu.library on startup

Solving Zorro-II 16 bit memory crashes: Memory caching adjustments

In case you're in an experimental mood: How to speed up the computer

More details about the configuration of the debugging tools MuForce and MuGuardianAngel is available separately in their documentation.

1.60 Future plans about the mmu.library

Even though the basis is set, I've quite a lot of plans with this library. Here are some that come to my mind:

o) A VMM like virtual memory manager. This is planned as three level design. Level 1 is the mmu.library. Level 2 is another system library, the memory.library. It will be used to handle memory pools and to provide functions for swapping memory in and out. Programs making use of this library will be able to ask for virtual memory. Level 3 is a patch that assigns virtual memory to certain "well-behaving" programs, much like VMM.

o) A MMU based Kickstart ROM replacer very much like SKick that really works, unlike most other constructions that I've seen. This will however require the special 68040 and 68060 libraries in this distribution.

o) A OxyPatcher replacement that makes use of the memory reserved by MuMove4K and remapped to fast memory by MuFastZero.

o) mmu.library compatible ShapeShifter display drivers.

In case you're interested in writing code for these projects, please contact me, I'll provide help and the background information in case you need more than documented in the autodocs.

1.61 Frequently asked questions, did you check these?

Here's the collection of some questions that I have been asked quite frequently, so in case you encounter a problem, it's a good idea to check this list first.

Q: How can I speed up the V40 68040 or 68060 library?

A: You'll find some tips here: How to speed up the system

Q: I'm running MuFastZero with the arguments "FORCENATIVE ON FASTEXEC" as described in the manual, and I also installed "MuMove4K" correctly, but some system tools tell me that ExecBase is still in Chip memory. What's wrong?

A: The system tools. "MuFastZero" will not touch ExecBase, the logical address will remain the same. However, due to the MMU magic, this logical address is no longer chip memory, but true fast memory. To be on the safe side, please run "MuScan" and check the memory range containing ExecBase. It should say "remapped to $xxxxxxxx" where $xxxxxxxx is the true physical address.

See also: Logical vs. Physical Addresses and the DMA problem

Q: MuFastROM complains that "The ROM is already remapped to RAM".

A: You're already using some kind of MMU tool that mirrors a RAM area to the ROM address space. Hence, some other tool is using the MMU for the same purpose as MuFastROM. In that case, you should remove this tool from your startup-sequence and replace it by MuFastROM.

Q: MuFastZero complains that "The zero page is already remapped", but I am not using any other tool for this purpose.

A: Some third-party 68040 or 68060 libraries remap the zero page for you without asking you or giving you control about this procedure. While this is a good idea in general, it might break certain applications. However, to be able to use "MuFastZero" even with these libraries, add the command line option "FORCENATIVE" to MuFastZero.

Q: The machine hangs while booting, having installed some Zorro-II 16 bit memory in my computer. What to do?

A: Some boards are not able to access Zorro-II 16 bit memory with the caches enabled. To fix this problem, the MMU-Configuration file must be edited by hand.

More details can be found here: Zorro-II 16-bit Memory Problems

---

Q: Once MuForce is installed, I can't quit it.

A: Before exiting, MuForce checks whether it can remove it's patches safely, and it will refuse to exit if it can't. Unfortunately, some patch-"improver" programs like "SetMan" do not support the protocol used for this query function. Either do not run SetMan, or replace it by more compatible tools, for example "TRSaferPatches" by the same author.

Q: I want to redirect the MuForce output to a console window, but it should not open unless a hit occurs. Especially, I do not want to see the "Welcome!" message.

A: Use the following tooltypes: "WINDOW=NIL:" and, additionally, "FILE=CON:////MuForceHit/AUTO/WAIT/CLOSE". This will redirect the ordinary output to "NIL:", i.e. will throw it away, and failure messages will be redirected to the specified console stream.

Q: When running MuGuardianAngel, the program keeps complaining about a nearly out of stack condition of the harddisk, the narrator.device or the RAM disk.

A: This hit is real, the FFS stack is truely too small, and so is the stack of the narrator.device and the RAM disk. The narrator device should have been fixed by the installation procedure by applying the "narrator.device.pch". The RAM disk can be fixed by running "PatchRAM" in the startup sequence. The FFS stack must be unfortunately adjusted manually, there's no tool that does this automatically. Several RDB preparation utilities will be able to help you out here, adjust the stack of the FFS to be at least 1024 bytes large.

Q: I'm using the ShapeShifter emulator, but MuMove4K and PrepareEmul don't seem to like each other. I can't get them working together.

A: This might be true, MuMove4K uses quite similar techniques for it's job. However, you don't need PrepareEmul anymore in first place, just replace it by MuMove4K and its "PREPAREEMUL" option. In case this option doesn't work, add the "A1200" switch as well.

More is here: The MuMove4K guide

Q: I need to run a special setup program of the hardware manufacturer in my startup-sequence to add the board memory. Can you help me?

A: Yes. Check this chapter: Non-Auto-configuring Memory.

Q: After having installed the mmu.library, some special debugger tools of my hardware manufacturer will no longer work correctly.

A: You won't need them anymore, replacement is available. MuForce replaces the Enforcer and similar tools, MuGuardianAngel is an even more picky memory guard that checks for illegal memory accesses.

Q: When running MuGuardianAngel, I get a hit every second. I'm using the cybscsi.device.

A: This is a firmware "feature" of the device. You can work around this problem by running "FixCybAccess" manually before running MuGuardianAngel.

See here: CybSCSI problems

Q: MuGuardianAngel and PatchWork don't seem to like each other, all I see is a guru or a warning when booting with both tools active.

A: Please ensure that you run PatchWork first, and MuGuardianAngel afterwards. The PatchWork memory patches will conflict with the MuGuardianAngel patches of the same functions.

Q: Should I install the 68040 and 68060 library provided in this archive?

A: If it is possible for you to use it, then yes. Both libraries require less memory than other releases because they don't have to build MMU tables on their own. However, both libraries will only work on "generic" systems, they won't support hardware that does not follow the autoconfig-standard. The "Install" directory contains a script to adjust the 68040/68060 library even to these non-standard boards, but the MMU configuration generated by this installation procedure might still require some manual adjustment. Moreover, the 68060 library in this archive will be able to support virtual memory correctly, something which is usually not considered by other products.

More is here: Compatibility Guidelines

Q: The library crashes. I own a P5 board and installed the 68040 or 68060.library in this archive.

A: Unfortunately, P5 hardware does not follow the CBM "AutoConfig" standard and therefore is not supported by a "generic" 68040 or 68060 library. However, both libraries are adjustable by the ENVARC:MMU-Configuration script to make them P5 aware. The "Install" directory contains an ARexx script that tries to setup the config file even for these non-standard boards correctly. More on this is in the installation notes, specifically in the MMU-Configuration setup

---

Q: Why are you doing this? This is senseless!

A: Why are you using an Amiga? This is senseless, a PC has much more to offer for you. In case you don't agree with this statement, then I'll give you my motivation: First, the mmu.library fills a gap in the Amiga Operating System. Second, it is an interesting project to work on with many tricky details I like to solve. I like these riddles, you know.

Q: Could you support the ppc.library, please?

A: I'd like to, but how? I don't get the required information from its manufacturer. Just run WarpOs, and on top of that the ppc.library emulation by Frank Wille. This combination is compatible to the mmu.library.

Q: Does the MMU library support WarpOs?

A: Yes in the sense that it works on a WarpOs driven mixed PPC/68K system. WarpOs cooperates nicely with the mmu.library. It might work even under an emulation of the 68K by the PPC, but this has to be tested and worked out. It does not run as native on a PPC because WarpOs itself provides functions to control the PPC MMU, so the library is not really required for WarpOs in first place.

Q: What's the meaning of life?

A: 41.999999, computed by a Pentium XXXVI.

1.62 Credits: People I'd like to thank.

The MMU.library project wouldn't be possible without the nice help of a lot of friendly folks!

Thanks a lot! The project wouldn't have been possible without your support, help, background information, testing, providing sources! Thanks for your time and for all the friendly EMail!

Special thanks goes to:

Michael Sinz: For explaining me a lot of internals of exec, for detailed information about the Amiga hardware, for providing the sources of his "Enforcer", for answering all my stupid questions and for being a nice guy.

Ralph Babel: For answering my questions about caching and the CachePre/PostDMA() questions, for documenting some of the internals of the omniscsi.device and hence allowing me to make his "GuruROM" software mmu.library aware.

Sam Jordan: For providing me some insight into WarpOs.

Simon N. Goodwin: For running a lot of tests on various systems, especially for some insider information about the Motorola 68060.

Michaela Prüß: For making the VBCC includes available for all my frequent updates and bugfixes.

Carsten Schlote: For preparing a MMU.library aware 68060 library.

Bjoern Schmidt: For allowing me to run some tests on his 060 and for taking time for me several afternoons.

Werner Müller: For providing a 68040 system for a bargain price which finally allowed me to write support code for the most tricky member of the motorola family, and to get rid of my 68030 based system and its "creative" chip memory.

Olaf Barthel: For allowing me to include his program "Sashimi" in this distribution, for all his great Amiga Software and for building the DevCD 2.1.

Richard Körber: For his great "PatchWork" tool, and for allowing me to include it in this archive.

Motorola: For providing the service of shipping the MC68K manual to my place for free.

I wish to thank all beta testers, for risking to run the beta software on their systems and for providing all the information I required to make the mmu.library great: (in alphabetical order by last name)

Stephen Brookes Carl Drougge Gaelan Griffin (with special thanks for testing this on his 68020/68851) Gene Heskett Andreas R. Kleinert Stéphane PAYET, 29 years and 13 with Amiga. Jon Peterson - Avid shareware supporter. Raphael 'PIK' Pilarczyk: "Murphy hates me" Thomas Pucyk (aMiGaFAN) on #AmIRC at IRCNet Hynek Schlawack Nicholas Stallard Flemming Steffensen Geoffrey Taylor Smyrna, TN USA Carlos A. Tirado Maure Vaughn Etienne Vogt

1.63 Glossary

Address : A unique "ID" where information in the computer memory is kept. See also: "Physical Address" and "Logical Address"

Address Bus : The wires in the computer that carry the address information where to get or put data to. See also: Data Bus

AutoConfig : A hardware/software protocol developed by CBM to integrate expansion hardware easily into the system. See also: What is AutoConfig, please?

CPU : The central processing unit, the chip that actually carries out all computations. Amiga models are equipped with a 68000, 68010, 68020, 68030, 68040, 68060 or a PPC.

Data Bus: : The wires in the computer that carry data to be read from or to be written to memory and I/O chips. See also: Address Bus.

DMA : Direct memory access, an I/O mechanism that bypasses the CPU for higher transfer rates. See also: What is DMA, please?

FPU : The floating point unit. Either provided as external chip or built into the CPU, this part carries out floating point arithmetics faster than the CPU could. The 68040 and 68060 FPUs are not complete and require external support by the 68040 or 68060.library.

Hofstadter, D. : Author of the Book "Gödel, Escher, Bach", physicist and mathematician. If you see this book in a library, get it and read it. (-:

I/O : Input and output: Data that enters the computer memory system, coming from devices like a harddisk, the mouse, the keyboard or the floppy, and going into devices like the screen, the harddisk or a printer. I/O is done by specialized chips in the computer.

Logical Address : The storage address as used by programs, the software address. See also: Logical vs. Physical Addresses and the DMA problem

MMU : The memory management unit. This is either an external chip or integrated into the CPU and controls all memory accesses done by programs. It furthermore translates logical addresses to physical addresses. See also: Logical vs. Physical Addresses and the DMA problem

Mu : Greek letter for "M" which looks like a "u" with an additional tail. Forms the unpronounceable word "MMU" with a second "M" in front. Also used by Douglas Hofstadter in "Gödel, Escher, Bach" to "un-ask" (sic) questions.

Physical Address: The storage address used by the hardware available as electric signals on the address bus, and the address used by DMA controllers. See also: Logical vs. Physical Addresses and the DMA problem

PIO : Programmed Input/Output: Making use of the CPU to transfer data from I/O circuits to the main memory.

PPC : PowerPC. A microprocessor family developed and produced by Motorola and IBM, more advanced than the 68000 series found in older Amigas.

Quantum Physics : "The dreams stuff is made of". (M. Sinz) A common reason of death for cats in experiments run by physicists that fail to understand how reality works.

U : See X.

Virtual Memory : The simulation of memory by harddisk space, due to some tricks the MMU can play. See also: What is virtual memory, please?

X : See U.

Zorro : The name of the Amiga Expansion bus which electrically connects the CPU to the expansions, also used for the hardware protocol that defines how the CPU has to communicate with the expansions. The original version Zorro-II is 16 bits wide, the enhanced Zorro-III is 32 bits wide and faster.

1.64 History: What happened before?

Release 40.51 :

The 68060 setup logic was slightly broken and left the MMU disabled in case it was disabled before the mmu.library has been loaded. Shouldn't have been a problem because the MMU based 68060.library hasn't been shipped with the 40.50.

Release 40.50 :

This is the first public Aminet upload of the final library.

1.65 Index