

# **MMU**

Thomas Richter

**COLLABORATORS**

	<i>TITLE :</i> MMU		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Thomas Richter	August 24, 2022	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>MMU</b>	<b>1</b>
1.1	MMU Guide . . . . .	1
1.2	The THOR-Software Licence . . . . .	2
1.3	What's the MMU.library? . . . . .	3
1.4	Logical vs. Physical Addresses and the DMA problem . . . . .	3
1.5	Compatibility Guidelines . . . . .	4
1.6	What is DMA, please? . . . . .	5
1.7	What is AutoConfig, please? . . . . .	5
1.8	What is virtual memory, please? . . . . .	5
1.9	What is Zorro-II memory, please? . . . . .	6
1.10	68000 and 68010 based systems . . . . .	6
1.11	68020 based systems . . . . .	6
1.12	68030 based systems . . . . .	6
1.13	68040 based systems . . . . .	7
1.14	68060 based systems . . . . .	7
1.15	P5 System compatibility problems . . . . .	8
1.16	Software replacement list . . . . .	9
1.17	Boards with a PPC processor . . . . .	9
1.18	Programmed I/O interfaces . . . . .	9
1.19	CybSCSI/z3scsi problems . . . . .	10
1.20	DMA based interfaces . . . . .	10
1.21	Non-Auto-configuring Memory . . . . .	10
1.22	Zorro-II 16-bit Memory Problems . . . . .	11
1.23	Contents of the Archive . . . . .	12
1.24	MuForce . . . . .	14
1.25	The disassembler.library . . . . .	15
1.26	MuGuardianAngel . . . . .	15
1.27	FixCybAccess . . . . .	15
1.28	MuSetCacheMode . . . . .	15
1.29	MuMove4K . . . . .	15

---

---

1.30 MuLink . . . . .	16
1.31 MuOVLVYMGGR . . . . .	16
1.32 MuFastROM . . . . .	16
1.33 MuFastZero . . . . .	16
1.34 MuFastChip . . . . .	16
1.35 MuLockLib . . . . .	16
1.36 MuScan . . . . .	17
1.37 MuOmniScsiPatch . . . . .	17
1.38 MuEVD . . . . .	17
1.39 FastIEEE . . . . .	17
1.40 MuProtectModules . . . . .	17
1.41 How to speed up the computer . . . . .	18
1.42 The V40 68020.library . . . . .	19
1.43 The V40 68030.library . . . . .	19
1.44 The V40 68040.library . . . . .	19
1.45 The V40 68060.library . . . . .	20
1.46 The MMU-Configuration File . . . . .	20
1.47 ClearTTX . . . . .	21
1.48 AddMem . . . . .	22
1.49 SetCacheMode . . . . .	22
1.50 DescriptorCacheInhibit . . . . .	23
1.51 For . . . . .	23
1.52 ClearMMU . . . . .	23
1.53 ScanMMUPort . . . . .	24
1.54 P5Init . . . . .	24
1.55 Installation of the mmu.library . . . . .	24
1.56 Installation of the debugging tools . . . . .	24
1.57 Installation of the SetPatch upgrade . . . . .	25
1.58 Installation of developer material . . . . .	26
1.59 Installation of the 68020 Library . . . . .	26
1.60 Installation of the 68030 Library . . . . .	26
1.61 Installation of the 68040 Library . . . . .	26
1.62 Installation of the 68060 Library . . . . .	27
1.63 Installation of Software Fixes . . . . .	27
1.64 Installation of MuLockLib . . . . .	28
1.65 Installation of MuOmniSCSIPatch . . . . .	29
1.66 Installation of MMU-Hack replacements . . . . .	29
1.67 Installation of the MMU-Configuration . . . . .	29
1.68 Installation of non-autoconfiguring memory . . . . .	30

---

---

1.69	Installation of Zorro-II memory fixes . . . . .	30
1.70	Installation of the BVisionPPC fixes . . . . .	30
1.71	Installation . . . . .	31
1.72	Future plans about the mmu.library . . . . .	33
1.73	Frequently asked questions, did you check these? . . . . .	33
1.74	Credits: People I'd like to thank. . . . .	40
1.75	Glossary . . . . .	41
1.76	History: What happened before? . . . . .	42
1.77	Index . . . . .	48

---

# Chapter 1

## MMU

### 1.1 MMU Guide

'### ,#. ,#### ,### / #####' ##### // ,#### ,###' // #####' ##### // ,#### ,###' // #####' ##### // ,#### ,###' / \_\_\_\_ / \_\_\_\_\_  
 #####' ##### // \ ,#####. ,###' . // // #####' ##. ,#### ,## // // ,#### #####' # ,##' // // #####' `#####' `###' / \_\_\_\_\_  
 \_ / \_ / \_\_\_\_\_ / ,#### ##### © 1999-2002 THOR - Software, #####' Thomas Richter `##'

---

mmu.library Master Guide

Guide Version 1.20 MMU Library Version 43.6

[The Licence : Legal restrictions](#)

[What is it : What is this all about, and what's the MMU library?](#)

[Compatibility: The compatibility guideline](#)

[Contents : What's all in this stuff in this archive](#)

[Installation : How to install the MMU library and its tools](#)

[FAQ : Frequently asked questions. Did you check these?](#)

[Future : What other plans exist for the mmu.library?](#)

[Credits : People I want to thank](#)

[Glossary : What do all these technocratic words mean?](#)

[History : What happened before](#)

---

"MuForce" contains code which is copyrighted © by Michael Sinz and which was originally written for the "Enforcer". This code is re-published here in the form of "MuForce" with the explicit permission of Michael Sinz.

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

E-Mail: thor@math.tu-berlin.de

## 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](#)

Manufacturer related problems:

[P5 based accelerators](#)

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 "cyb-scsi.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.

If your 68020 comes with a FPU - the 68881 or the 68882 - and you want make use of the "FPU" control command, you might want to **install the 68020.library** and the 680x0.library which are required by the FPU command.

## 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](#)

If your 68030 comes with a FPU - the 68881 or the 68882 - and you want make use of the "FPU" control command, you might want to [install the 68030.library](#) and the 680x0.library which are required by the FPU command.

## 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 unnoticeably 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 P5 System compatibility problems

Unfortunately, some members of the P5 product family do not follow the CBM **AutoConfig** guidelines which means that these hardware components do not make "themselves known to the Os". In particular, they are not, and cannot be recognized by the `mmu.library` to provide the proper MMU setup. This is not a problem, though, as long as the MMU-library isn't used to build the initial MMU setup, but it will be problematic as soon as you want to replace the P5 specific 68040 or 68060.libraries by the versions in this archive.

The well known accelerators "MK-I" and "MK-II" are, compatible, but some members of the "Blizzard"-family, especially those equipped with a **PPC processor** are not. I've no overview about the P5 product family, though, and you might be on your own to check this out.

In case you own a non-compatible board, it is easiest to use the P5 68040 or 68060.library, and the `mmu.library` on top of these libraries. This will cost somewhat more memory, but it will work at least, in any case.

In case you want to use the `mmu.library` specific 68040 or 68060.library in this archive, proceed as follows:

o) Make a backup of your current processor library, i.e. copy it to a safe place:

```
Mkdir SYS:Backup copy LIBS:68040.library TO SYS:Backup/ copy LIBS:68060.library TO SYS:Backup/
```

o) Unpack this archive if not already done.

o) Open a shell.

o) "CD" into the "Install" directory of this archive, e.g. by

```
cd MMULib/Install
```

o) **Install** the `mmu.library` if not already done:

```
copy /LIBS/mmu.library to LIBS:
```

o) In case you own a 68040 based board, copy the `mmu.library` specific 68040.library to LIBS: In case you own a 68060 driven board, copy the 68060.library to LIBS: **DO NOT REPLACE A DUMMY 68040.LIBRARY by the `mmu.library`. You can get rid of the dummy library by other means.**

```
copy /LIBS/68060.library to LIBS:
```

o) Run the installation script "BuildMMUConfig.rexx" by entering

```
rx BuildMMUConfig.rexx ENVARC:MMU-Configuration
```

o) This script might copy some additional software to LIBS:mmu/ and C:. In case it prints a warning about the "ppc.library", make sure that you do not use ppc.library driven programs.

In case the script recommends using the "BPPCFix" program, it will have copied it already to C:, but some manual work is required, too. You have to adjust your "Startup-Sequence" to run BPPCFix to remove some ROM resident libraries:

```
Ed S:Startup-Sequence
```

o) Locate the "SetPatch" command in this file. Please add the following line in front of, i.e. above this line:

```
BPPCFix 040 install reboot
```

o) In case you have been using "BPPCFix" before, ensure that you run it with the "040" option as this will remove the ROM resident 68040 and 68060.libraries. In case you're using the Os 3.5 "SetPatch", it will reset your computer anyhow a second time while booting - you will not need the "reboot" option then.

o) Save the file and leave the editor. In case you used the system editor "Ed", this is

```
Esc X and Return
```

o) Turn off your computer, wait a minute, turn it on and hope the best...

Please remember, I do not own a P5 board, I've no documentation about their hardware whatsoever. In case something goes wrong, you possibly have to fine-tune the **ENVARC:MMU-Configuration** file to adapt or optimize it for your needs.

## 1.16 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". The "FORCENATIVE" option should be dropped in case you're using the supplied 68040 or 68060 libraries because it is not required in this case.

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.

## 1.17 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.

## 1.18 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?](#)

---



## 1.19 CybSCSI/z3scsi problems

Due to a firmware "feature" of the "z3scsi.device" and 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.

## 1.20 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 "gvp SCSI.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.

## 1.21 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) Copy the "MemModes" program from the "Install" directory of the distribution to a safe place to make it available for the following steps.

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, run the "MemModes" program

```
"MemModes PREP" Return
```

to generate the memory MMU setup.

o) Write down or save the output of this program. This describes all "auto-configuring" memory of your system which does not require special treatment. Each memory node will generate one block of data, separated from each other by comments and blank lines. Possibly, print the output out.

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

o) Run the "MemModes" tool again with the same arguments.

"MemModes PREP" Return

Compare its output with the first output. You should see now one or more additional entries, describing the memory added by the tool. This output contains paragraphs which look like the following:

```
SetCacheMode from 0x07000000 size 0x01000000 Copyback Valid AddMem from 0x07000000 size 0x01000000
```

One or more of these paragraphs must be added to the ENVARC:MMU-Configuration, namely those that did not appear in the first call.

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) One paragraph of the "MemModes" output contains two commands, one line to mark the memory as valid, and a second line to add it to the system memory pool. Now copy exactly those paragraphs from the second "MemModes" output literally into the ENVARC:Configuration file that did not show up in the first output you wrote down or printed out before. In our example, this could be

```
SetCacheMode from 0x07000000 size 0x01000000 Copyback Valid AddMem from 0x07000000 size 0x01000000
```

The comment in front of these lines is unimportant and need not to be inserted.

Make sure that you haven't forgotten the "Valid" keyword for the first command. 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.

## 1.22 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.

## 1.23 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](#) [MuEVD](#) [MuProtectModules](#)

ReadMe Latest news and changes made to the contents.

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.

68020.library This is a new system library, the [68020.library](#) . It is a CPU driver for the 68020 chip. It is however not really required for the mmu.library itself.

68030.library This is a new system library, the [68030.library](#) . It is a CPU driver for the 68020 chip. It is however not really required for the mmu.library itself.

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.

mmu This directory contains external modules which can be loaded by the mmu.library on startup as part of the MMU-Configuration file. Currently, only one external module is available, P5Init. It will initialize P5 board hardware and setup the MMU accordingly. This is unfortunately necessary since some P5 hardware isn't auto-configurable.

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 mathieedoubbas.library, it fails to compare some floating point numbers correctly. Only required for pre-Os 3.9.

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.

SetPatch\_44\_6.pch Similar, for the 44.6 edition of SetPatch.

SetPatch\_44\_13.pch The same again for SetPatch 44.13.

SetPatch\_44\_16.pch And again the same for SetPatch 44.16.

SetPatch\_44\_17.pch And again the same for SetPatch 44.16.

FixCybAccess Works around a firmware "feature" of the cybscsi.device and the z3scsi.device and is required for owners of these device to be able to run **MuGuardianAngel** without receiving a hit every second. More is here:

#### FixCybAccess

FixP5Scsi A similar fix, comparable to the above, but works by different means. More is here:

#### FixP5Scsi

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. Only required for pre-Os 3.9

PatchRAM Increases the stack size of the RAM disk to avoid justified "MuGuardianAngel" hits from RAM: More's in its ReadMe. Only for pre-Os 3.9

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.

mfmdev.pch Patches the mfm.device of CrossDos to increase the stack size and to avoid warning messages of MuGuardianAngel. Provided by Gene Heskett.

IPrefs.pch Increases the stack size of the 3.1 (40.7) IPrefs release to avoid stack overflows when loading workbench backdrop pictures. Only required for pre Os 3.5

RamLibFix Fixes two bugs of the "ramlib" process which loads external libraries. First, it increases the stack size of ramlib to 4K to avoid stack overflows. Second, it changes the signal bit of its communications message port to avoid some race conditions caused by semaphores in library startup code. This program **\*MUST\*** be copied into C:, and its name **\*MUST NOT\*** be changed. Only required for pre Os 3.9

FixLongMult Fixes a bug in the utility.library 64 bit math functions if no 68020 or higher processor is available. The result is then forwarded in the wrong registers. This fix is not really part of the MuLib archive, and it is only worth using, in fact, if you do not own a MMU, on the 68000 and 68010, namely. It is included in SetPatch 44.16 anyhow so you don't need it with the latested BoingBag installed. Only required for pre Os 3.9

BVisionPPC\_42r7.pch Fixes an undocumented 68040/060 library call in the BVisionPPC driver which, therefore, depends on the P5 libraries without this patch. This patch requires running P5Init in the MMU-Configuration. This is the release for the v42.7 (CGfx 4) edition.

BVisionPPC\_28.pch Fixes the same problem for the v28 (CGfx 3) edition.

PatchString Allows to fix the same problem on any BVisionPPC driver by running it twice like this:

PatchString BVisionPPC 68040.library p5emu.library PatchString BVisionPPC 68060.library p5emu.library

P5Integrate A third alternative for the same problem. This will supply the undocumented routines of the P5 library for the MuLib 68040 or 68060.library. It will, however, make these libraries "lie" about their version number, but it doesn't require a permanent patch of the BVision driver.

Fixes.ReadMe More about the "Fixes" directory.

Install Several installation related tools.

Install The MuLib installation script, requires the Installer software found on the official workbench.

FindPort A tiny program that scans for a named public port.

FindDevice A similar program that scans the system list of available device drivers. Required by the installation script.

**FindResident** Another program that scans for a resident tag in the Os. Again, this program is required by the installation script.

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

**FastIEEE** A speedup patch for the mathieeexxxx.libraries provided the fsp.resource is available. This patch will redirect the unimplemented math instructions within the math libraries directly to the emulation routines, hence avoiding the overhead of going thru an emulator trap. The fsp.resource is build by the MuLib processor libraries. Only required for pre Os 3.9

**MemModes** Another setup related tool. This tool prints the memory setup in a way that is suitable for the Rexx setup scripts, used for automated setup of the MMU-Configuration.

**MuScan** Identical to the **MuScan** program of the MuTools directory. It is used by the Rexx setup scripts.

**PrintMMUPort** Prints the contents of the MMU-Config port which is build and required by some P5 hardware. Included for internal debugging purposes only.

**Sed** A powerful pattern matching program in the spirit of the standard but less powerful "Search" command. It is required by the "Installer" script to parse and modify the Startup-Sequence.

**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. This module is obsolete.

**P5Init** An improved version of ScanMMUPort which should keep care about all special P5 init kludges at once, thus avoiding some manual adjustments that had to go into the MMU-Configuration otherwise. The new setup scripts use this command now. It will be copied into LIBS:mmu by the setup script if P5 hardware has been detected.

**BuildMMUConfig.rexx** **ScanToConfig.rexx** Some manual Rexx based setup scripts. These scripts scans for all hardware, compares the hardware with its database and generates a setup file for the mmu.library automagically. They try to set the optimal caching modes for graphic cards, too.

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

The difference between the two scripts is that BuildMMUConfig.rexx uses the information provided by the expansion.library about system expansions to build its configuration, plus some P5 setup magic, whereas ScanToConfig.rexx scans the currently active MMU table. In most cases, ScanToConfig will cause a more accurate, but possibly over-complicated MMU table, whereas BuildMMUConfig.rexx should be more canonical. Try BuildMMUConfig.rexx first, then ScanToConfig.rexx if it doesn't work.

More details are here:

### **Installation of the MMU-Configuration**

Where are the includes, autodocs and sources?

To keep the user archive small, they moved to a different archive, together with all the remaining development material. Check the MuManual archive, you'll find them there.

## **1.24 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

---

## 1.25 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 [MuGuardianAngel](#) 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.

## 1.26 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 its guide:

The MuGuardianAngel guide

Because the z3scsi.device and the cybscsi.device do an illegal memory access every second, [FixCybAccess](#) must be run by owners of the cybscsi.device in order to fix this.

See also: [CybSCSI/z3scsi problems](#)

## 1.27 FixCybAccess

FixCybAccess is a tiny fix for a firmware "feature" of the z3scsi.device and the cybscsi.device. These devices 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" and the "z3scsi.device".

An alternative fix for the same problem is available by the "FixP5Scsi" program. Instead of pre-allocating the memory for the un-allocated memory access, this program will try to patch the device driver code directly. Therefore, it will not fragmentize the chip memory, but it will work only for some hard-coded devices, namely the cybscsi.device, the z3scsi.device, the 2060scsi.device, the 2040scsi.device, the 1260scsi.device and the 1240scsi.device. Should you become aware of more P5 device drivers that suffer from this feature, please let me know.

## 1.28 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

## 1.29 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.

---

### 1.30 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](#)

### 1.31 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](#)

### 1.32 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](#)

### 1.33 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](#)

### 1.34 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](#).

### 1.35 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](#)

---

## 1.36 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](#)

## 1.37 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 omniscsi. Check the [Credits](#) section.

See also: [Logical vs. physical addresses](#) and [More on DMA interface controllers](#) .

## 1.38 MuEVD

MuEVD is an enhanced MMU driven external video driver for the ShapeShifter Macintosh emulator. It supports all MMUs, starting at the 68020/68851 up to the 68060 and is capable to drive ECS, AGA and P96/CGfx screens in 1, 2, 4, 8, 15 and 24 bit video modes. It comes also with a HAM8 emulation to display the 15 bit HiColor modes on AGA screens. MuEVD replaces various other video drivers that use similar MMU techniques.

See also: [The MuEVD guide](#)

## 1.39 FastIEEE

FastIEEE is a speedup patch for the mathieexxxx.libraries provided the fsp.resource is available. This patch will redirect the unimplemented math instructions within the math libraries directly to the emulation routines, hence avoiding the overhead of going thru an emulator trap. The fsp.resource is build by the MuLib processor libraries.

The speedup factor of the transcendental math functions compared to a non-patched system using the V38 math libraries is about three to six, as measured on a 68040 system.

Note: FastIEEE is obsolete starting with the Os 3.9 (V45) math libraries because these libraries are smart enough to detect the fsp.resource themselves. It is not recommended to install FastIEEE on top of these libraries, even though it won't hurt. It will just waste some memory.

See also: [How to speed up the computer](#)

## 1.40 MuProtectModules

MuProtectModules write-protects disk-based resident modules that entered the system by means of the "LoadModule" command of the same author. It is useful to prevent Os 3.5 or Os 3.9 specific modules from getting overwritten accidentally if these are made resident by means of "LoadModule". The program does not work for modules loaded by "SetPatch" and the "AmigaOs ROM Updates", sorry.

Details: [MuProtectModules documentation](#)

---

## 1.41 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
```

The "ForceNative" option is not required if you run the MuLib on top of one of the supplied "processor"-libraries in this archive. "MuFastZero" will inform you about this situation anyhow.

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:

```
MuMove4K
```

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

```
MuFastZero ForceNative MoveSSP FastExec On
```

Again, the "ForceNative" switch might not be necessary.

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 behind 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.

More hints for 68040 or 68060 based systems

The 68040 and the 68060 do not come with a full floating point coprocessor. Instead, some of the FPU instructions have to be emulated by means of CPU "exception processing"; going thru an exception is both slow and not very multitasking friendly since multitasking is disabled as long as the instruction is emulated.



Therefore,

### FastIEEE

patches the mathieee system libraries to use the emulation routines directly instead of going thru the unnecessary exception processing.

You find FastIEEE in the "Install" directory of this distribution. It can be copied to "C:" and run as part of the startup-sequence.

Note: FastIEEE requires that the 68040 or 68060.library make their FPU emulation routines available to the public by means of the "fppsp.resource". This resource is of course not build on 68020 or 68030 systems since IF they come with a FPU, their FPU doesn't require any kind of emulation. Furthermore, the resource is, unfortunately, not build by the 40.17 68060.library by Carsten Schlote.

## 1.42 The V40 68020.library

This archive contains a new system library, the 68020.library. This library is not really required by the mmu.library, but it is provided for consistency as a specialized "CPU driver" for the 68020 processor. It contains specific 68020 software fixes for the Os, as well as the FPU control logic for the "FPU" command. The 680x0.library and the mmu.library are then required as well and should be installed, too.

These libraries should work on all 68020 based systems, regardless of whether a FPU is available - even though its FPU logic and the "FPU" command are in this case of course not available.

The 68020.library will always open the mmu.library and build an MMU tree, as if the old "Enforcer QUIET" command has been used. This is usually not required for 68020 based Amigas with an MMU as all it will do is that it will slow down the system a bit due to the way how the 68851 MMU works.

For more insight, study the [68020 compatibility notes](#) .

## 1.43 The V40 68030.library

This archive contains a new system library, the 68030.library. This library is not really required by the mmu.library, but it is provided for consistency as a specialized "CPU driver" for the 68030 processor. It contains specific 68030 software fixes for the Os, as well as the FPU control logic for the "FPU" command. The 680x0.library and the mmu.library are then required as well and should be installed, too.

These libraries should work on all 68030 based systems, regardless of whether a FPU is available - even though its FPU logic and the "FPU" command are in this case of course not available.

The 68030.library will always open the mmu.library and build an MMU tree, as if the old "Enforcer QUIET" command has been used. Hence, it helps to work around a hardware bug of the 68030 which might cause problems with certain bridge board hardware or graphics software if the MMU is not enabled. Therefore, the mmu.library must be installed together with this library.

For more insight, study the [68030 compatibility notes](#) .

## 1.44 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.

In case you own a 68060 based board, you MUST NOT install this library. Either, keep the 68040 "dummy" library which was provided by the board manufacturer, or install the 680x0.library and apply the [patch to SetPatch](#) .

For more insight, study the [68040 compatibility notes](#) .

---



## 1.45 The V40 68060.library

The 68060.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 other 68060.libraries, as for example the P5 68060.libraries or Carsten Schlote's 40.15 release. 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.

You MUST NOT install the 68040.library for an 68060 based board. Either, keep the 68040 "dummy" library which was provided by the board manufacturer, or install the 680x0.library and apply the [patch to SetPatch](#) .

For more insight, study the [68060 compatibility notes](#) .

See also: [Credits](#)

## 1.46 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](#)

[For](#)

[ClearMMU](#)

The following external commands are available:

[ScanMMUPort](#)

[P5Init](#)

---

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

---

To re-run the setup of the memory, ignoring any previous memory configuration:

ClearMMU Memory

This command overrides the MMU setup that might have been active before the mmu.library has been loaded. The boot ROMs of some boards, for example the Apollo boards, might be told to remap the ROM to RAM before booting and hence leave the MMU active. However, the caching mode selected for the memory is sub-optimal and will slow down the system. The following command will help in this case:

ClearMMU Memory Blank Expansion Motherboard

This will reset the MMU settings to default except for the ROM configuration.

---

To mark the full area of the board(s) of manufacturer ID 2193, product ID 1 as "CacheInhibit NonSerial Imprecise":

For 2193 1 SetCacheMode {base} {size} CacheInhibit NonSerial Imprecise

This board is the "GVP Spectrum" graphics card, or to be precise, its video RAM. Video RAM must be cache-inhibited - which is also the default for expansion boards, but it can be set to the faster "NonSerial Imprecise" caching mode as well. Note that this step should be left to the graphics card software, but only Picasso96 is currently able to adapt and optimize the MMU setup.

## 1.47 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.

---

## 1.48 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.

## 1.49 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.

## 1.50 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.

## 1.51 For

For MANUFACTURER/A,PRODUCT/A,ZORRO2=Z2/S,ZORRO3=Z3/S,BIG/S,CMD/F

This command provides some kind of "conditional" execution of other commands in the MMU-Configuration. It checks the list of installed hardware expansions for a specific hardware board, as given by the manufacturer ID and the product ID, and will then execute the command following the For command on the same line. If more than one board of the same manufacturer or the same product ID is found, this command is executed several times, once for each board. On execution of the command, For substitutes several variables on the argument line; all of them have to be enclosed in curly brackets "{" resp. "}"

{base} The base address of the board in hex notation {size} The size of the memory range occupied by the board in bytes, as required by SetCacheMode. {tail} The end address of the board, exclusively.

Simple mathematics is possible by adding or subtracting a constant offset from these expressions. This is done by specifying the offset in the curly brackets, e.g.

{base+0x8000} 32K above the base address {size-0x10000} The size minus 64K {tail-0x40000} The start address of the last 256K of the board

The For command will accept customary hex notations for the numbers, i.e. by specifying "0x" or "\$". The offsets default to hex, either. Decimal notation can be enforced by a leading hash-mark "#".

Additional options for For:

ZORRO2=Z2 The command is only executed if the corresponding board is in Z2 autoconfig space. Useful since some boards behave differently whether they are configured as Z2 or as Z3 boards.

ZORRO3=Z3 The command is only executed if the corresponding board is in Z3 autoconfig space.

BIG The command is only executed if the autoconfig area is larger than one megabyte. This is useful for boards that require two autoconfig areas, but specify the same product ID for both ranges. With this identifier, the larger range can be selected, which is typically some kind of board RAM.

## 1.52 ClearMMU

ClearMMU MEMORY/S,MOTHERBOARD/S,EXPANSION/S,ROM/S,BLANK/S,ALL/S

Resets or overrides a previously made MMU setup completely or partially. This is helpful in case the MMU library has to be loaded on top of an already active MMU setup which is sub-optimal, and adjustment of this configuration by hand would be too

painful. This command will therefore ignore parts of the MMU setup which was found active before and will re-run parts of the automatic setup procedure of the mmu.library.

MEMORY/S Reconfigure MMU setup for the memory in the system. This setup marks FAST memory as "copyback" and CHIP memory as "cacheinhibit imprecise nonserial".

NOTE: Do not try this on certain P5 boards as some on-board hardware will expect that parts of the memory are not in "copyback" mode.

MOTHERBOARD/S Reconfigure the address space of all motherboard resources, as for example the CIAs, the custom chips, the IDE and SCSI ports, the PCMCIA slot etc... This should hardly be necessary.

EXPANSION/S Reconfigure the address space of all expansion boards. Their caching mode is set to "cacheinhibit" completely.

ROM/S Reconfigure the Kickstart ROM space and the "Debug ROM" area. The ROM area is set to "ROM", the "Debug ROM" or "F-Space" area is set to "cacheinhibit" since some boards provide non-autoconfiguring resources here.

NOTE: Do not use this setting in case your board depends on a MMU-remapped ROM for booting, i.e. some Apollo boards.

BLANK/S Reconfigure all blank areas to "blank". Some boards as for example the Apollo might leave these blank areas as "invalid" which means that faulty software will cause a crash immediately. This can be worked around by re-configuring unused areas of the memory map by this option.

ALL/S Ignore the previous setup completely and re-run the automatic configuration from start.

NOTE: This will cause heavy problems on boards that require a custom setup, as for example some P5 boards.

## 1.53 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.

## 1.54 P5Init

P5Init (does not take arguments)

This is an advanced version of [ScanMMUPort](#) which should take all steps required to configure the undocumented P5 hardware, including PPC boards. It sets all caching modes for this hardware automatically and includes a scan of the P5 MMU setup kludge. It replaces "ScanMMUPort" plus all former P5 setup lines in the MMU-Configuration and should be used instead.

## 1.55 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

## 1.56 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 wherever you want.

## 1.57 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.

## 1.58 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.

## 1.59 Installation of the 68020 Library

In case you own a 68020 based system and you want to make use of the FPU command, you need a specific CPU driver which offers the required control functions. The FPU command requires the 680x0.library as well, so both should be installed in this case. Moreover, you might want to launch the 68020.library already on startup, in which case **SetPatch must be modified** .

Installation of the library itself is simple: Just copy it to LIBS:

copy LIBS/68020.library to LIBS: Return

It is not required to edit or to build a **MMU-Configuration file** for the 68020, unless you want to fine-tune your setup. In this case, start from the default configuration in the "ENVARC" drawer of this archive and copy it to ENVARC:

copy ENVARC/MMU-Configuration to ENVARC: Return

## 1.60 Installation of the 68030 Library

In case you own a 68030 based system and you want to make use of the FPU command, you need a specific CPU driver which offers the required control functions. The FPU command requires the 680x0.library as well, so both should be installed in this case. Moreover, you might want to launch the 68030.library already on startup, in which case **SetPatch must be modified** .

Another advantage of running the 68030.library is that it programs the MMU to work around a hardware bug of the 68030 which could cause problems with some bridge-board or older graphics card software. It replaces the older "Enforcer QUIET" command you would have to run in this case.

Installation of the library itself is simple: Just copy it to LIBS:

copy LIBS/68030.library to LIBS: Return

It is not required to edit or to build a **MMU-Configuration file** for the 68030, unless you want to fine-tune your setup. In this case, start from the default configuration in the "ENVARC" drawer of this archive and copy it to ENVARC:

copy ENVARC/MMU-Configuration to ENVARC: Return

## 1.61 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.

In case you own a 68060 based board, you **MUST NOT** install this library. Either, keep the 68040 "dummy" library which was provided by the board manufacturer, or install the 680x0.library and apply the **patch to SetPatch** .

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.

## 1.62 Installation of the 68060 Library

In case you own a 68060 based system and you're able to run the mmu.library based 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.

You MUST NOT install the 68040.library for an 68060 based board. Either, keep the 68040 "dummy" library which was provided by the board manufacturer, or install the 680x0.library and apply the [patch to SetPatch](#) .

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.

## 1.63 Installation of Software Fixes

The Os and some Os libraries contain unfortunately some bugs which may or may not conflict with the mmu.library. This directory may change a bit from distribution to distribution, the latest details about the programs in this directory can be found in the "ReadMe" file.

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 this bug was fixed with the Os 3.9 release, finally. This fix is no longer required for Os 3.9 for this reason. Here's how to apply the fix otherwise:

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. This bug has been fixed with the Os release 3.9. Otherwise, if you don't run 3.9, 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. This bug has been fixed with Os 3.9 as well. To fix the RAM disk otherwise, "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 "mkdir 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 P5 based SCSI device as the cybscsi, the 2060scsi or the z3scsi.device and want to use the "MuGuardianAngel", one "Firmware Feature" of the device must be fixed with the **FixCybAccess** or **FixP5Scsi** 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

or, alternatively,

copy Fixes/FixP5Scsi to C: Return

Ed S:Startup-Sequence Return

Now move the cursor below "SetPatch", press Return to insert a blank line and add the following command:

Run <>NIL: FixCybAccess <>NIL:

or

FixP5Scsi

Press then Esc and x to save the changes back.

## 1.64 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.

## 1.65 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 physical 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 wherever you want. If **MuLockLib** is installed, the recommended place is behind 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.

## 1.66 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.

Furthermore, MMU driven ShapeShifter drivers should be replaced by the **MuEVD** video driver. For that, drag the "MuEVD" icon into the "ShapeShifter/Video Drivers" directory and edit the ShapeShifter preferences accordingly. For details, read the MuEVD.guide.

## 1.67 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 controls 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 described 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. In this case, the configuration file is NOT required. In other words, ENVARC:MMU-Configuration is, except for some special cases, only required if you install the 68040 or 68060.library contained in this distribution.

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
```

o) If the MMU-Configuration generated by BuidMMUConfig.rexx did not work, please try the second setup script, ScanToConfig.rexx. It might create a more complicated MMU table, but might be more accurate:

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

o) You might want to hand-edit this file as well; especially, you may remove the "P5Init" command if you do not own a P5 board.

## 1.68 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.

## 1.69 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 behaves 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.

## 1.70 Installation of the BVisionPPC fixes

The CyberGraphics BVision PPC graphics card driver software depends unfortunately on undocumented internals of the P5 68040.library or 68060.library and will not work under any other library. This means, too, that this fix is not required if you continue using the P5 processor libraries.

No matter what, this strange behaviour can be fixed, though, by one of two methods:

The first method, and IMHO the more canonical, is to patch the driver software to avoid this. The second method is to integrate a minimal subset of the required but undocumented 68040 or 68060.library functions by means of a patch.

First step: (Goes for both methods)

Make sure you run P5Init in the ENVARC:MMU-Configuration. Make also sure that you installed at least the 40.3 version of P5Init or the required resources for the fix will not be build. In case you're not sure which version you use, copy "Libs/MMU/P5Init" from this archive into "Libs:MMU/P5Init" manually.

copy Libs/MMU/P5Init to LIBS:MMU/P5Init

First alternative: Patch the Cybergraphics drivers.

Before you start, make a backup of the original driver!

Now patch the driver as follows: Go into the "Fixes" directory of the archive, and enter the following two commands from the shell:

```
PatchString DEVS:Monitors/BVisionPPC REPLACE 68040.library BY p5emu.library
```

```
PatchString DEVS:Monitors/BVisionPPC REPLACE 68060.library BY p5emu.library
```

For experts: You might want to patch the BVisionPPC driver manually as well. Use a binary file editor, search for the strings "68040.library" and "68060.library" and replace both by "p5emu.library". Then save the changes back.

Second alternative: Patch the 68040/68060.library

The second alternative is to apply an on-the-fly patch to the 68040 or 68060.library within this archive. For that, please copy the file "Fixes/P5Integrate" within this archive to C: and run it in the startup sequence somewhere between SetPatch and CyberGraphics.

copy Fixes/P5Integrate to C:

Then open S:Startup-Sequence with an editor of your choice:

```
Ed S:Startup-Sequence
```

Locate the line saying "SetPatch". Move the cursor to the end of the line and press Return. This should insert a new blank line below SetPatch. Move the cursor there, and enter

```
P5Integrate
```

Then save the changes back.

## 1.71 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, at least not without the proper setup.

The simplest method to install the MuLib and its components is, however, to run the supplied installation script in the "Install" directory. It will keep care about most of the technicalities mentioned below and will carry out the same installation steps as the manual installation.

For experts: Don't worry, the script will really ask about every single operation it performs. It will also make a backup of the former setup, nothing will get lost.

Anyhow, 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`**

This is the heart of the MuLib archive. Won't work without.

Recommended:

**Installation of the `MuTools`**

A lot of tools that re-implement common MMU-hacks in a clean way.

Bonus feature:

### Upgrade SetPatch, install the 680x0.library

Not really required and independent of the rest of the project, this makes the startup mechanism a bit more canonical.

Optional:

### Installation the AutoDocs and Example Sources

Not required for the average user, useful for the programmer.

Bonus feature:

### Installation of the new 68040.library

Not really required to make the MuLib working, but this will save some memory and will make the startup mechanism more canonical. Installation recommended on 68040 machines, but not required.

Bonus feature:

### Installation of the new 68060.library

Not really required to make the MuLib working, but this will save some memory and will make the startup mechanism more canonical. Will NOT work on ill-behaving boards, most likely, but recommended otherwise.

Possibly required:

In case you're using the 68040 or 68060.library contained in this archive, and you own a BVisionPPC graphics card under CyberGraphics, you have unfortunately to patch the CyberGraphics drivers to support it.

### Fixing the BVision CyberGraphics drivers

Unfortunately, the CyberGraphics drivers depend on some undocumented internals of the P5 68040 or 68060.libraries.

Bonus feature:

### Installation of the 68030.library

Not really required, but useful for 68030 based machines to have the "FPU" control program available.

Bonus feature:

### Installation of the 68020.library

Not really required, but useful for 68020/68881 combinations to allow FPU finetuning.

Highly recommended:

### Installation of Software Fixes

Some third-party fixes for common firmware bugs, mainly stack overflows. Not required, but recommended. Bonus feature

Bonus feature:

### Installation of MuLockLib

Locks the MMULib in memory on startup to avoid unnecessary loading delays afterwards.

Bonus feature:

### Installation of the MuOmniScsiPatch

Makes the omniscsi.device ("Guru-ROM") MMU aware.

Highly recommended:

### Replace MMU hacks

Replace popular MMU hacks by better behaving software. Highly recommended.

Bonus feature for experts, and for P5 compatibility:

### MMU-Configuration adjustments

This is definitely for experts only if you want to try the MuLib specific 68040/68060.libraries. Not required otherwise.

See also: More on [P5 related problems](#) including an experimental MuLib setup.

---

Bonus feature for experts:

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

Only for some third-party boards that require manual memory installation without using the autoconfig mechanism.

Bonus feature for experts:

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

Only required for some boards/expansions with broken Zorro bridges.

Bonus feature for experts:

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

Bonus feature for experts:

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

## 1.72 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.

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.

## 1.73 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: Various MuFastZero options do not work on my setup.

A: Please do not redirect the MuFastZero output to NIL: and check what the command has to say. Study the error message. MuFastZero will abort its installation process if one of its options failed.

---

Q: MuFastZero FASTEXEC fails on my setup.

A: MuFastZero FASTEXEC requires MuMove4K to run prior its setup.

---

Q: MuFastZero FASTEXEC does not remap ExecBase at all. Various tools say that it is still in chip memory here.

A: Ignore the output of these tools. MuFastZero uses the MMU to remap the memory the exec library is located in. The address of ExecBase does not change, indeed, but the very same addresses represent now fast memory. Use "MuScan" to check successful relocation.

---

Q: MuFastZero FASTEXEC does not remap ExecBase at all. My board provides auto-configuring fast memory.

A: ExecBase is then already in fast memory. Why do you want to remap it then at all? Your setup is already optimal.

---

Q: What's the fuzz with the FASTVBR option, do I need it?

A: No, you don't. The "VBR" specifies where the CPU shall find its interrupt vectors - for optimal performance, these vectors should sit in fast memory, even though the system places the VBR by default at address zero, in chip memory. Now, there are two methods to move the auto vectors out of chip memory: Method one is to remap the entire zero page containing the vectors into fast memory, using the MMU. This is what "MuFastZero ON" does by default, and the recommended operation as it won't break some games that expect the VBR fixed at address zero. FASTVBR, however, will move the auto vectors elsewhere by loading a CPU register some old games are not aware of. Both methods are valid, the first is typically less cumbersome, but you surely do not need both.

---

Q: I'm using "MuFastZero ON", but various tools indicate that the interrupt vectors are still in chip memory.

A: No, they don't, ignore these tools. The point is that the MMU remapped the memory region erroneously identified as "chip memory" by the MMU, as your tools are not aware of the MMU. The indicated memory region is now, thanks to MMU magic, indeed fast memory. To remind you, you really do not need the "FASTVBR" option.

---

Q: Could you please provide a mmu.library based Kickstart ROM replacement patch, similar to SKick?

A: No, I cannot, sorry. The reason is that I cannot guarantee that a program like this will work correctly on all available boards. Some boards will re-program the MMU already on startup (some P5 boards, you guessed right) and would therefore break this tool. Furthermore, it would also require a patch attack of the exec kernal.

---

Q: I own a 68060 board and installed your library, but my board won't boot anymore.

A: Please check the following:

- You must not install the 68040.library in this archive on top of the tiny 68040 "dummy" library that came with your board. This won't work. Either, keep this dummy library, or install the 680x0.library and apply the [SetPatch patch](#) .

- 68060 boards are critical in the sense that most manufacturers did not follow the CBM guidelines how hardware should behave. Therefore, if you want to use this library, you must [build an MMU-Configuration](#) file. In most cases, the pre-defined sample files within the "ENVARC" drawer of the distribution might work. For non-P5 boards, copy the file "ENVARC/MMU-Configuration" from this archive to "ENVARC:MMU-Configuration". For P5-boards, try to copy "ENVARC/MMU-Configuration.P5" to "ENVARC:MMU-Configuration"; note that there is no ".P5" in the destination file. You further need to copy "LIBS/MMU/P5Init" of this archive to "LIBS:MMU/P5Init".

If this sounds too complicated, please run the automated installer script. You might also want to check the [P5 related problems](#) section which includes some hints and tricks how to setup this file.

- The mmu.library will not run if the ppc.library is installed. Sorry for this stupid restriction, it's not my fault as I don't get the documentation which would be required to support this library. WarpOs will work.

- Please check the [software replacement list](#) . Some old software will try to hack the MMU, something that won't work anymore.

- In case nothing helps, please sent me a detailed description of your board, the software you're using, the MMU-Configuration file that was build by the setup ARexx script, a scan of the public port list and a scan of the resource list. The latter two are available thru system monitors, e.g. "Scout" or "XOper". In the meantime, do not run this 68060.library but install the mmu.library on top of the 68060.library that came with your board. It is not required to install the 68060.library that is in this archive.

---

Q: I own a 68040 board and installed your library, but my board won't boot anymore.

A: Please check the following:

---



- Some 68040 boards are critical in the sense that their manufacturers did not follow the CBM guidelines how hardware should behave. Therefore, if you want to use this library on your board, you might have to **build an MMU-Configuration** file.

See also: More on **P5 related problems** , and the paragraph concerning the MMU-Configuration in the FAQ entry above.

- The mmu.library will not run if the ppc.library is installed. Sorry for this stupid restriction, it's not my fault as I don't get the documentation which would be required to support this library. WarpOs will work.

- Please check the **software replacement list** . Some old software will try to hack the MMU, something that won't work anymore.

- In case nothing helps, please sent me a detailed description of your board, the software you're using, the MMU-Configuration file that was build by the setup ARexx script, a scan of the public port list and a scan of the resource list. The latter two are available thru system monitors, e.g. "Scout" or "XOper". In the meantime, do not run this 68040.library but install the mmu.library on top of the 68040.library that came with your board. It is not required to install the 68040.library that is in this archive.

Q: I own a 68040 or 68060 board and installed your library, but the system is now slower than usual.

A: Please check whether you installed the **MMU-Configuration** file, to be found in the ENVARC drawer of the archive, in ENVARC: Make sure this assign is really available at the time SetPatch is executed. To be on the safe side, boot the computer with the startup-sequence disabled, then enter in the command

```
type ENVARC:MMU-Configuration
```

This file should contain at least one command, namely "ClearTTX". This command is required to override the settings the exec boot code left in the system - namely, which left the copyback caches disabled.

Q: I own an Apollo 68040 or 68060 board and installed your library, followed the above question but the system is still slower than usual.

A: The Apollo boot ROMs seem to provide an option to remap the Kickstart ROM to RAM, using the MMU. The MMU setup used for this procedure will, by the very design of the mmu.library, remain used even after the library has been loaded, even though this setup is sub-optimal.

To avoid these problems, please add the following line to the "ENVARC:MMU-Configuration" file:

```
ClearMMU Memory Blank Motherboard Expansion
```

This will re-initialize and override the sub-optimal settings of the Apollo boot ROM.

Q: I own an Apollo 68040 or 68060 board and installed your library, but some strange software seems now to crash with a "Bus Error", the Guru number 80000002.

A: Please use the same procedure described in the FAQ entry above.

Q: I'm using the 68040 or 68060 library in the MMU archive and the MMU library together with a CyberVision graphics board. SetPatch requires now several seconds to initialize and requires a lot of memory as well. Is there something I can do against it?

A: The CyberVision board is to blame here. It allocates 64 MEGABYTES of Zorro-III address space even though it requires only a minor part of it. To avoid this, several lines have to be added to the ENVARC:MMU-Configuration file. Open the ENVARC:MMU-Configuration file with an editor of your choice, e.g.

```
ed ENVARC:MMU-Configuration
```

and then insert the following lines from the sample MMU-Configuration found in ENVARC/MMU-Configuration in this archive:

```
;Cybervision Z3 For 8512 34 Z3 SetCacheMode {base} {size} Blank IOSpace For 8512 34 Z3 SetCacheMode {base} 0x01400000
Valid IOSpace CacheInhibit For 8512 34 Z3 SetCacheMode {base+0x01400000} 0x00c00000 Valid IOSpace CacheInhibit Non-
Serial Imprecise For 8512 34 Z3 SetCacheMode {base+0x02000000} 0x02000000 Valid IOSpace CacheInhibit
```

```
;CyberVision 3D For 8512 67 SetCacheMode {base} {size} Blank IOSpace For 8512 67 Z2 SetCacheMode {base} 0x00380000
Valid IOSpace CacheInhibit NonSerial Imprecise For 8512 67 Z2 SetCacheMode {base+0x00380000} 0x00080000 Valid IOSpace
```



```
CacheInhibit For 8512 67 Z3 SetCacheMode {base+0x04000000} 0x01000000 Valid IOSpace CacheInhibit NonSerial Imprecise
For 8512 67 Z3 SetCacheMode {base+0x05000000} 0x00010000 Valid IOSpace CacheInhibit For 8512 67 Z3 SetCacheMode
{base+0x05800000} 0x00008000 Valid IOSpace CacheInhibit For 8512 67 Z3 SetCacheMode {base+0x07000000} 0x00008000
Valid IOSpace CacheInhibit For 8512 67 Z3 SetCacheMode {base+0x08000000} 0x00001000 Valid IOSpace CacheInhibit
For 8512 67 Z3 SetCacheMode {base+0x0c000000} 0x00010000 Valid IOSpace CacheInhibit For 8512 67 Z3 SetCacheMode
{base+0x0c0e0000} 0x00001000 Valid IOSpace CacheInhibit
```

Now save the changes and reboot. This should solve the problem.

Q: I'm using the 68040 or 68060 library in the MMU archive and the MMU library together with a BVisionPPC or CyberVisionPPC graphics board. The machine does not boot anymore, or CyberGraphics aborts with an error requester.

A: The BVisionPPC/CyberVisionPPC CyberGraphics driver is to blame here. It depends on some undocumented functions in the P5 68040 or 68060.library which are not supported by the 68040 or 68060.library contained in this archive. The problem can be solved, though. Please read:

#### Installation of the BVisionPPC fixes

Q: I'm using your 68040 or 68060 library together with the ShapeShifter Macintosh emulator, and I get crashes when starting Netscape or opening a drawer.

A: First of all, the MacOs has never been adapted to the 68060 processor, which means that not all software - initially written for the 68040 - will work correctly. This goes especially for programs using the FPU. Additionally, the MacOs and some of its applications are not written in a very clean way and will cause the FPU to generate exceptions on "operand errors". A true Macintosh will ignore these exceptions, but AmigaOs does not. However, you can change this behaviour: Please install the "FPU" program, to be found in the MMULib.lha distribution. Run "FPU" as follows in the Startup-Sequence, or before running the ShapeShifter:

```
FPU NoOpErr
```

This should fix the problem.

Q: I set the caching mode of my chip memory to "cacheable", and my system is now much faster! What do you say?

A: Arrrgghghhh! There are good reasons why chip memory must not be cached. The first reason is that the blitter coprocessor may modify the memory "under the feet" of the CPU. If the CPU caches memory accesses, it won't be aware of these modifications. The second problem is completely independent on whether you use the blitter or not: All members of the 680x0-family may try to run "burst accesses" into cache-able memory, but these advanced cycles are not supported by the processor to chip ram interface. It might seem to work most the time, but it may even though break down in critical moments. It is simply not recommended.

Q: I'm using a Kickstart replacement tool to reboot my system from an alternate ROM image, but the system locks up as soon as I load your system libraries.

A: The mmu.library makes all the memory within the exec free memory list and the ROM space available to the system, but if your Kickstart replacement tool modifies the memory layout such that the image itself is outside of the memory regions the system knows about, the mmu.library will mark it as un-mapped and not accessible. Unfortunately, there is no fool-proof algorithm to determinate where the ROM image went, and hence the mmu.library is not able to make this memory available automatically. You have to do this yourself. Some Kickstart replacement tools put the ROM image into auto-config memory of 0x00200000 and above. If you use such a tool, the following line with the "ENVARC:MMU-Configuration" file will help:

```
SetCacheMode 0x00200000 0x00080000 Valid Copyback
```

If the Kickstart ROM image went into a different memory area, this line must be modified accordingly, i.e. 0x00200000 must be replaced by the start address of the image. Unfortunately, I cannot present an easy and bullet-proof algorithm how to get this information.

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 \$xxxxxxx" where \$xxxxxxx 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: [MuFastZero](#) complains "FORCENATIVE option is not required" on startup.

A: You switched from a vendor-specific 68040 or 68060.library to the MuLib adapted versions in this distribution, or are using a processor library which does not touch the zero page on startup. In this case, just drop the "FORCENATIVE" option of MuFastZero and everything will be fine.

---

Q: ShapeShifter hangs on startup.

A: You might be using a MMU hacking video driver for ShapeShifter. In this case, replace it by [MuEVD](#), disable the REMAP8K tooltype of ShapeShifter and try again.

---

Q: I'm using MuEVD, but ShapeShifter still hangs on startup.

A: Make sure that you really disabled all MMU hacks the ShapeShifter might want to try. This is for first the REMAP8K tooltype of ShapeShifter, which should be replaced by [MuFastZero](#). Then, the "Write-protect ROM" feature should be disabled as well. This option can be found in the "Miscellaneous" control window of the ShapeShifter preferences.

---

Q: The MuEVD video driver tells me that it runs out of memory on startup.

A: Reduce the amount of memory for the ShapeShifter emulator. Especially, you should not use the "Allocate largest free block" option or no memory will be left for MuEVD itself.

---

Q: Does my setup require an ENVARC:MMU-Configuration file or not?

A: This depends on how you use the mmu.library. If you run it on top of a "standard" 68040 or 68060.library, you don't. Please erase this file in this case. The only case where a ENVARC:MMU-Configuration file is required is if you use the 68040.library or the 68060.library in this distribution. Even then, the default MMU-Configuration file in the ENVARC drawer of this archive is good enough, unless you own a P5 board. In this case, some [fine tuning](#) is required. Possibly, you might want to try the experimental [P5 setup procedure](#) to generate the required configuration file.

---

68020 and 68030 owners usually don't need this file because these boards are typically well-behaving.

---

Q: The ENVARC:MMU-Configuration file contains a "SetCacheMode" command, but I don't have this command in C:. Should I rename "MuSetCacheMode" to "SetCacheMode", then?

A: No. The command is not in C:, and need not to be there in any case. It is a library build-in command. Besides, external commands for the setup file are not looked for in the "C:" drawer, but in "LIBS:mmu".

---

Q: The machine hangs or gurus 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. The setup scripts usually generate a remark about which line should be edited.

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, the RAM disk or "ramlib".

A: This hit is real, the FFS stack is truly 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 or the Fusion 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'm using the Fusion Mac emulator on top of the MuLib, but it seems to crash and hang more often than before. The Fusion "virtual memory" support is enabled, i.e. the "MMU" switch is on.

A: Unfortunately, the current Fusion releases hack on the MMU themselves. This is not tolerated by the MuLib and must not happen. In almost all cases, leave the virtual memory switch OFF. In case you really, really must use this option, edit or create the ENVARC:MMU-Configuration file with a standard text editor and add the following line wherever you like:

DescriptorCacheInhibit ON

---

Even though this doesn't fix Fusion, it might be used as a temporarily work-around which might help to avoid some, but not all problems.

---

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 or the z3scsi.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. More on this is in the [P5 related problems](#) related section. Moreover, the 68060 library in this archive will be able to support virtual memory correctly, something which is usually not considered by other products.

Please do not install the 68040.library on an 68060 based board, but either keep the 68040 dummy library which was provided by the board manufacturer, or install the 680x0.library and apply the [SetPatch upgrade](#) .

More is here: [Compatibility Guidelines](#)

---

Q: Should I install the 68030 or 68020 library provided in this archive?

A: These two libraries are only loaded if you own either a 68020 or 68030 based board, and even then they're currently not really required, unless you want to run the "FPU" program to disable FPU exceptions. If this is the case, you need to install the 680x0.library as well. By using the MMU, the 68030.library will also work around a firmware bug of the 68030 which could cause hangs especially if used together with bridge-board software. If you run the 68030.library, you need not to disable caches to run this software anymore.

---

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 is therefore 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](#) and in the [P5 related problems](#) section.

---

---

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.

---

## 1.74 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

## 1.75 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.

## 1.76 History: What happened before?

Release 43.6:

mmu.library: Added another safety check for the DMA property list. MuMapRom: The reset-and-stay resident mechanism of MuMapRom makes now use of the ColdCapture exec vector and a nice little extra hack. MuMapRom: Adds now a 16MB "safety zone" around memory areas to keep some wierd memory tests working. memory.library: Changed the memory administration functions a bit by adding a "ranger pointer".

Release 43.5.1:

680x0.library: Handled the low-memory area of MuMove4K improperly and therefore broke MuFastZero. Fixed. MuMapRom: Did not work at all if the "ROMINFAST" option was not present at the command line. Should be much better now. memory.library: The library does no longer allow the attachment of an address space to the global MMU context. memory.library: The library does no longer support attachments of address spaces to supervisor contexts. This wouldn't have worked anyhow. MuGuardianAngel: AllocAbs() was still broken and returned the wrong register.

Release 43.5:

CPU libraries: All CPU libraries reset the VBR now before restarting the ROM. mmu.library: Fixed a possible race condition of the 68060 exception handler. The 68060 can report a misaligned access even though the fault address and the access fault size do not indicate that a page boundary is crossed. Included a new test tool: "SwapTest" will check whether the 68060 or 68040 library support some race conditions on swapping correctly. Note that this test will fail for most third-party libraries. - Included MuForce 40.30 (Aminet release) that is required for the latest MuGA. It won't work with former releases.

Release 43.4.2(Internal release only):

memory.library: fixed possible memory leak of the swap hooks. They should have closed files/devices on VMPACK\_EXIT, not VMPACK\_CLOSE. Updated the documentation of the mmu.library for the new functions. Included a first version for the memory.library documentation.

Release 43.4.1(Internal release only):

mmu.library:Forgot to include the 43.4 of the mmu.library in the last distribution. memory.library: fixed broken handling of private swap hooks for virtual memory pools. Fixed broken handling of "Retry" of error requesters. Added more sophisticated error handling for out of memory and swap alerts. The code will no longer try to repeat for obvious errors. mmap.c: fixed missing result code on error.

Release 43.4(Internal release only):

mmu.library: The 68040 race condition fix of the 43.3 wasn't as good as I thought. Reworked this mess again. It will now be able to handle the wierd condition where a write-back is busy and detected from a word-sized movem, even though it comes from a different instruction. Yuck! mmu.library: Added workarounds for the V37 ObtainSemaphoreShared() bug. mmu.library: Made all context locks shared as far as possible. mmu.library: Fixed a possible register trash for the shared context locks. memory.library: Worked again a bit on the memory allocation routines Examples: vmem.c and mmap.c are now ready for release. The examples look now like they are supposed to. Updated the mmu.library autodocs a bit. memory.library: added another cache at the swap hook side of the library. This should hopefully help to improve the performance a bit as it tries to bundle I/O accesses. The library limits now the virtual memory range of the address space to the user defined limit before asking the hook for the maximal size. This avoids unnecessary disktrashing for the file hook. The library should behave much better now for low memory situations and errors on the swap hook. The hook remains responsive in these situations. Added a (localizable) error requester for failures of the three built-in swap hooks. All file I/O goes now over packets rather the dos.library. This would avoid trouble in case the dos.library gets patched over.

Release 43.3(Internal release only):

mmu.library: Fixed a possible race condition of the Motorola "Diva", the 68040. Unlike what the documentation suggests, the CM bit is not directly related to access errors of movem's. -): memory.library: Fixed a bug in the final page disposal routine that could have caused MuGA hits. Fixed. Fixed a bug in the swap daemon that could have tried to deliver a motor tick to the swap hook even though the hook has been released already. Reworked the internal memory handling. The memory pools come now with scratch lists to speed up the allocation of tiny chunks, and to avoid unnecessary virtual memory accesses. Further, the library uses now its own set of pooled allocation/deallocation routines. First of all, this avoids clashes with whatever patch might sit there and doesn't know how to handle virtual memory correctly, especially the rather harsh Forbid()/Disable() rules. Second, the new pooled allocation tries a combination of a "best fit" plus "buddy chunk" allocation that is less naive than the native exec

allocation. (but still naive enough to allow improvements...) Fixed bugs in the computation of the swap pool size that happened mainly on machines with Z2 memory only. Added documentation for the PoolVSize() function that was forgotten for the 0.0 release last time. Added a tag to restrict the size of the virtual memory pool created. MuGA: Fixed Deallocate()/Allocate() patches that forgot to align memory correctly. MuRedox: Apparently, the new version never made it to Aminet, even though it was uploaded. It provides one new option, SHOWPATCHEDINSTRS, which shows the list of instructions it was able to replace by its own set of stub-routines.

The memory.library got tested now on the 030,040 and 060.

Release 43.2(Internal release only):

mmu.library: Added support for the mmu.resource. This is a system resource that defines the interface to the true hardware MMU. The library will make use of this resource whenever it is present, and will fall back to its build-in routines otherwise. The purpose of the resource is to allow emulation of the MC68K MMU on non-native CPUs (i.e. x86) without the need to re-write the entire library from scratch. mmu.library: "shared" pages are finally officially supported. Note that most of this stuff worked already in V42. mmu.library: Fixed a bug in PhysicalLocation() that did not return the true physical location in case the memory was marked as MAPP\_SHARED. mmu.library: Added GetPageUsedModified() to parse the Used/Modified flags more easily than with Get/SetPageProperties(). This is still to be documented and mainly for the purpose of the memory.library. MuGA: Fixed a register trash that broke AllocAbs() and related calls. NEWS FLASH! Finally, the first release of the memory.library is available. The purpose of this release is to supply virtual memory to the AmigaOs in a flexible and compatible way. There is not yet much documentation, but there are includes and autodoes.

Release 42.11 :

MuForce, MuGuardianAngel: The documentation stated that you must not exit MuForce as long as MuGuardianAngel is running. Since some folks do not want to read documentation, this behaviour is now enforced. (pun intended)

mmu.library: The MMU memory type tag was ignored for the spare pages for MAPP\_BLANK, fixed.

MuFastZero: Returns now a separate error code in case FASTEXEC has been specified but exec is already in fast memory.

MuMove4K: PREPAREEMUL did not work correctly on boards with true autoconfig memory due to a missing MEMF\_CHIP. Fixed.

MuGuardianAngel: Added the new critical PROTECTPOST option to enhance out-of-bounds error detection.

mmu.library: Added some private tags to GetContextData(), not to be documented.

MuFastRom: Got modified to work with even already remapped ROM images.

fpsp.resource (within the 68040 and 68060.library): Fixed the generation of the "store" flag that should have been set in exception conditions.

FastIEEE: Development passed over to Gunther Nikl; included a new 40.2 that fixes some bugs of the 40.1 and works around some old bugs of the V37 math libraries. Thanks, Gunther!

mathieeexxxx.library: The V45 math libraries support now the fpsp.resource and hence faster mathematics. They are part of Os 3.9 and not included in this distribution.

MuMapRom: This is a new MuTool that allows booting from a different kickstart release if the board is well-behaved. Be warned, this is a hack.

MuRedox: Another new MuTool that patches on-the-fly unsupported 040 and 060 instructions similar to the Oxypatcher or the CyberPatcher. Be warned, this is a hack.

Included FPSPSnoop for debugging purposes of MuRedox.

Included LoadModule as "base tool" for MuProtectModules.

MuForce: Marked the ROM area as WRITEPROTECTED rather than ROM on exit, causing GURUs on erroneous writes into that area.

Release 42.9 :

Due to a bug in the MMU-Configuration scan, Imprecise and NonSerial were mutually exclusive within the MMU-Configuration file. Result could have been less-optimal MMU setups on 68040 boards that could not have caused any instabilities, though. (Hence, this is mainly a cosmetic fix...)



MuProtectModules was updated to support to the LoadProtectModules release 40.5 and beyond.

Release 42.8.1 :

MuGuardianAngel: Fixed a bug in the page access handler that could have invalidated pages under some alignment race conditions, thanks Stephen.

68060.library and 68040.library: Added explicit masking within CacheClearE() which might work around a potential firmware bug of the 68060 or of DMA driver firmware.

68060.library: Fixed a bug in the "unsupported FPU data type" handler due to a register trash. Urgh! Thanks for the bug report, Luca.

Release 42.8 :

mmu.library: Fixed a fatal bug in the RebuildTree(s) calls that trashed the DMA relevant MMU table mapping in case the MMU context remained untouched and RebuildTree(s) could have exited immediately.

MuFastChip: Added a check whether the program is really required, and prints a warning in case it is not.

MuFastZero: Tries now to detect the lower limit of the chip memory automatically and will remap all memory below this boundary. Should help when using the ShapeShifter and "MuMove4K".

Documentation: exd\_Internal was at the wrong place in the C header files. (Urgh!) It was always correct in the .i files, though.

Release 42.6 :

Included a new tool, MuProtectModules. It write protects reset-resident programs.

mmu.library: Fixed some very rare problems on access errors of pre-decrement movems for both the 68040 and the 68060.

MuMove4k: The PREPAREEMUL (without the A1200 switch) option was broken and might have caused crashes on a reboot. MuMove4K installs now itself into MEMF\_KICK and no longer MEMF\_CHIP. If this causes problems on your machine, try the INCHIPMEM option.

MuFastZero: If ExecBase was remapped by means of the FastExec option, installation of reset-proof programs behind MuFastZero failed since these programs installed only into the non-resident copy of ExecBase. MuFastZero includes now a patch to SumKickData() which will avoid this problems.

MuForce: MuForce installed an interrupt handler directly into the autovector base which is not very system friendly. Fixed by using the official interrupt handler. If MuForce crashes on exit, disable your favourite "SaferPatches" clone and either avoid these tools, or get "TRSaferPatches.lha" from Aminet.

MuGuardianAngel: Supports now some not yet available memory pool enhancement patches.

P5Init: Fixed a potentially dangerous unaligned CopyMemQuick().

A new MuLib based, tested and VMM aware 68060.library is now available.

Included a new tool, MuProtectModules. This will write-protect resident modules loaded by "LoadModule". (See Aminet).

MuEVD: Forgot an ATC table flush, result might have been partial display glitches caused by erroneously dropped refresh cycles.

FPU: Forgot to include this FPU control program. Fixed.

680x0.library: Did not identify a 68060 FPU correctly if used to bootstrap the 68060.library by the patched SetPatch.

Release 42.4.1 :

CPU libraries: The FPU test was super-critical and failed if the FPU could not complete a float->integer conversion before starting a concurrent frestore.

CPU libraries: The GetMsg() replacement is now more conservative and will return NULL in case the message list of the port is not properly initialized or the linkage field of a node is (incorrectly) NULL. Thanks Ian!

Release 42.4 :

Fixed several flaws of the 68060.library: Several cache flushes have been missing, FPU setup has been fixed as well.

Fixed the installer script: FixCybAccess must be "run" in background.

MuGuardianAngel did not record the return PC correctly for the pooled memory allocations.

The mmu.library crashed on a plain 68000. Fixed.

Release 42.3 :

Included a MMU driven ShapeShifter video driver, MuEVD. (new)

MuGuardianAngel: Does no longer call RawIOInit() and hence will no longer conflict with programs using the serial port even in cases MuGuardianAngel never required the port in first place.

MuForce: Does no longer call RawIOInit() but fully relies on the Os and/or the user to initialize the serial port correctly, in the same way the old Enforcer did. This means that MuForce will now safely cooperate with software that requires the serial port in cases MuForce output goes to the parallel port or other destinations.

Note that there is now a new release of "BlizKick" which fixes the problems mentioned below.

Updated the documentation, fixed a minor mistake in the MMU developer's manual.

Included an upgrade patch from SetPatch 44.13 to 44.14 which, once again, opens the 680x0.library instead of the 68040.library. The RamLibFix is then no longer required since it is included now in SetPatch anyhow.

MuGuardianAngel: The DUMPLIST routine had a bug and might have allocated too less memory. Luckily, this had the only side effect that not all of the mung list would have been dumped correctly, it did not trash any memory.

Installation scripts: Due to a bug in the BuildMMUConfig.rexx script, the on-board hardware of certain P5 boards has been disabled completely. Urgl.

mmu.library: Added a new command for the ENVARC:MMU-Configuration, "FOR". It executes the command given as argument if the manufacturer product code fits, possibly several times if more than one board is in the machine.

mmu.library: Added another new command "ClearMMU" which can be used to re-run the MMU table setup procedure for some computer resources, mainly to ignore a previously loaded MMU setup at least partially if it is sub-optimal. This seems to be required for some Apollo boards.

Note that there is now a new release of "BlizKick" which fixes the problems mentioned below.

Release 42.2 :

The 42.1 release automatically marked all hardware pages as cacheinhibited serialized. Looks like even that was too much for some hardware boards, I don't know why. This should be the proper default anyhow. I disabled this again, even though this means, as for 42.0 and before, that a MMU-Configuration is *\*mandatory\** if you use the library as "stand-alone" instead on top of a thirdparty 68040/68060 library.

Release 42.1 :

MuFastChip: Forgot to include the latest mmu.library in the latest upload. Sorry.

Installation: P5Init, PPCIdentify, P5Identify reworked again, it enables now explicitly the bus error generation of the A4000 motherboard resources.

Installation REXX scripts: Added a "NoP5" keyword to disable explicitly the P5 identification steps which seem to be problematic for some boards for reasons that are beyond me.

MuManual: Fixed some typos, corrected some mistakes, clarified some formulations. Thanks to Etienne Voigt for proofreading!

Organization: The "MMULib" archive is now the user archive, all developer information went into the "MuManual" archive, including the autodocs, the includes, the bmaps and some example sources. This will help to keep the archive short.

mmulib: The CurrentContext() function forgot to Forbid() properly. Note that you still need a Forbid() bracketing or the result code might be pretty useless. The propability that this broke code is very low, though.

mmulib: GetMappingProperties() was simply broken in V42.0 and below. Sorry, this got fixed. This function hasn't been used yet, so this bug was left unnoticed.

mmulib: Even if "ClearTTX" is missing in the MMU-Configuration, the library cache-inhibits now the access to expansion boards. This is just a safety bonus.

Included a debug version of the library in the MuManual archive.

BlizKick: In order to avoid a yellow alert, either BlizKick must be modified or must be run behind SetPatch. The reason for the alert is that BlizKick opens the mmu.library before the 68060/68040 lib is open, which is and never has been legal. I just added

an explicit check for this condition in V42 because too many people ignored it. As I said, "no discussion". This is a side effect of how the library works and has to work.

Release 42.0 :

mmu.library: Added more error checking for the startup code, esp. the MMU-Configuration file. Added a check for proper configuration, i.e. whether the library was (incorrectly) loaded in front of SetPatch. Added a new function: RunOldConfig().

68040.library: Added an explicit check for correct configuration, it will generate a requester in case no 68040 is available.

MuGuardianAngel: Fixed a bug in the mung-wall check which could have reported one additional mung-wall damage in case the front wall was found defective. Added a workaround for a possible 68060 firmware bug.

In case you see MuGuardianAngel hits of the z3scsi.device, run the FixCybAccess program. It will work around the z3scsi.device hits as well.

MuMove4K PREPAREEMUL moves now the low chip memory end to the 16K line, not to the 8K line. This might fix some Fusion problems. Thanks, Pavel.

Improved the error messages of MuFastZero a little bit.

Reworked P5Identify and PPCIdentify to make these two more stable.

Added another external MMU setup command, P5Init. It should keep care about all P5 specific cache settings and should setup the PPC and the BOOT-MMU-Port automatically. All manual P5 specific entries in the MMU-Configuration except graphic board cachings are obsolete now and should be replaced by P5Init.

Rewrote both setup scripts to reflect the changes in the P5 setup logic, i.e. ScanMMUPort has been replaced by P5Init and all P5 specific cache settings have been removed.

Added stack increasement patches for the mfm.device (CrossDos) and IPrefs 40.7 in case you do not yet use Os 3.5. Thanks Gene.

Added a fix for two bugs in ramlib. First, its stack is too low. Second, it uses SIGF\_SINGLE as message bit for its process port which could cause some race conditions with semaphores in library setup code.

Release 41.4 :

mmu.library: The CachePre/PostDMA() functions are now a bit more error tolerant and handle cases where a DMA device attempts DMA transfer to a non-existing memory region more gracefully.

680x0.library: Sets now the memory attributes of the supposed-to be remapped low-memory header to MEMF\_FAST instead to 0 to avoid a zero return value of TypeOfMem().

MuFastZero: Sets now the memory attributes of the supposed-to be remapped low-memory header to MEMF\_FAST instead to 0 to avoid a zero return value of TypeOfMem(). Lowered the priority of the rendezvous port.

FastIEEE: Optimized the IEEEEDPFloor and IEEEESPFLoor in case a 060 processor has been found. Note that this makes currently no difference at all since there is currently no 68060.library that provides the fpsp.resource.

Release 41.3 :

disassembler.library: fmovem.x <dynamiclist>.memory was disassembled incorrectly. Fixed.

mmu.library: SetPropertyMapping() accepts now base=0 and length=0 as a special case to transfer the complete list.

68040.library: Removed some unnecessary strings and references to the math libraries.

mmu.library: Added workarounds for possible DMA disable counter underruns that might have happened in case the library gets active while DMA is active.

MuMove4K: Added more options: IGNOREVERIFY/S disables the verify check for the reboot code. REVERSE/S allocates the memory for the resident tags in reverse direction. LOWPRI/S lowers the priority of the MuMove4K resident tag as a possible workaround for some wierd other hacks. Added more error messages in case something goes wrong.

MuForce: Forgot to restore the Z-page mode and physical location on removal. Oops. Thanks, Heinz!

MuFastZero: Added a "FastVBR" option because it was easy to do and canonical to have. NOTE THAT THIS OPTION IS NOT REQUIRED IF YOU REMAP THE VECTOR BASE ANYHOW.

MuGuardianAngel: Prints now the stack boundaries in case of stack problems, let it be overflow, underflow or nearly out of stack. Thanks to Heinz, again. Provides an option to disable these warning messages, but beware! This doesn't mean bugs will go away, it just silences MuGuardianAngel. Remember, this program is provided to scream upon problems it detects. Setup scripts: Greatly enhanced! The new setup script will also re-set the memory caching modes for boards whose boot-roms already run the MMU. Fixed a minor glinch that produces a (bogus) error report in case LIBS:MMU already existed. Remember that you can't run the ppc.library on the MuLib setup, sorry.

Added a new setup script "ScanToConfig" that takes a "MuScan" output from a system running under a third-party 68040/68060.library and generates an MMU-Configuration file from it. To use it, re-install a non-MMU-aware processor library, boot the system, then enter the "Install" directory and run the script. MuScan must be available in the current directory as it is in the archive. Please let me know how this new script goes, maybe it's smarter. Added a new program "MemModes" to the installation tools. It will auto-generate the memory caching setup as the mmu.library does if the MMU is found non-active. Possibly an improvement for those boards which - strange enough - turn on the MMU before booting. Added a patch for the new SetPatch in the BoingBag, check the "Fixes" directory. Again: When will people finally learn to read the FAQ? \*Sigh\* In case MuFastZero complains "The zero page is already remapped", READ THE GUIDE.

Release 41.2.1 :

Updated the 68040.library to 40.6. It builds now the fpsp.resource which offers opcode emulation to external hosts.

Included FastIEEE which re-directs the mathieee-library routines to the fpsp.resource for optimal performance.

FastIEEE fixes, too, some bugs in the mathieee libraries. IEEEEDPCmp() is broken and orders some numbers "the wrong way". IEEEEDPPow() and IEEEESPPow() are broken and return non-sense for special arguments.

Release 41.2 :

The MMU Library build-in AddMem failed in case the base or length were not aligned to 64K boundaries. It now rounds to the next 64K boarders such that the supplied area is at least mounted partially.

The idea to disable the TTx registers in the mmu.library was unfortunately not a very smart idea as it broke the code on some machines. Re-established the old rule with the only exception that the TTx registers are initialized for the EC040.

The ColdReboot() patch of the 68040.library used the MMU registers even on a system without MMU. Fixed.

MuMove4K checks now all libraries, devices, ports, resources and resident modules whether they violate the 8K boundary.

68040.library: Forgot to disable the caches in ColdReboot().

68040.library: The motorola OpErr handler did not consider tracing correctly. Fixed.

Finally wrote and included the 68020.library and the 68030.library, hence "FPU" will work on these machines.

Updated MuGuardianAngel: Added a new keyword "DUMPWALL" that prints the contents of a broken mung-wall and of broken memory cookies.

Release 41.1 :

ScanMMUPort was broken and scanned for the wrong port. \*Sigh\* This release should work better on Blizzard boards.

AddMemList() uses now RebuildTreesA() to rebuild user and supervisor tables at once and is therefore a bit more error tolerant.

The mmu.library will now support CachePreDMA/CachePostDMA for the EC040 and EC060 processors as well.

The 68040.library will now disable the TTx registers manually such that the "generic" board does no longer require an ENVARC:MMU-Configuration file.

The 68040.library will now enable copyback caches for the EC040.

Some of the MuTools did not unload properly if loaded from the workbench, fixed.

The MuTools are now a bit more error tolerant due to a new function in the V41 mmu.library which gets used.

Release 40.60 :

Fixed the shutdown code of MGA, thanks to Stephan!

Disabled a kludge in the mmu.library which write protected a kickstart image at 0x00200000 and up by checking the name of execbase. This kludge might have conflicted with some softkickers.

Because people tend not to read guides, I added the arguments "WRITEPROTECTED" and "INVALID" to the library build-in "SetCacheMode". Note that they work different to what "MuSetCacheMode" does. Here, "WRITEPROTECTED" and "INVALID" are just aliases to "ROM" and "BLANK", hence enable the most defensive protection strategy.

Added a new LVO "RebuildTreesA" which is "officially" not yet existing and will be documented in V41. More LVOs might be added up to V41, but please **\*DO NOT\*** yet call them as they did not exist in V40.50.

Fixed the installation script, due to a typo the P5 MMU boot hack was not detected.

Fixed the P5Detect program which just looked to the wrong identification resources. Oops. Installation on "non-standard" boards should be smoother now.

Fixed a bug in the disassembler.library which disassembled the lea (offset.L,pc) wrong. The offset was wrong by two bytes.

The release number in the MuForce guide was wrong.

When will people finally learn to read the FAQ? *\*Sigh\** In case MuFastZero complains "The zero page is already remapped", **READ THE GUIDE.**

Release 40.51.1 :

Disabled the layers.library kludge for MuGuardianAngel if V40 is found active. It is luckely no longer required.

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.

## 1.77 Index

[68000 and 68010 based systems](#) [68020 based systems](#) [68030 based systems](#) [68040 based systems](#) [The V40 68020.library](#) [The V40 68030.library](#) [The V40 68040.library](#) [68060 based systems](#) [The V40 68060.library](#)

A...

[AddMem Non-Auto-configuring Memory](#) [What is AutoConfig, please?](#)

B...

[Boards with a PPC processor](#) [Installation of the BVisionPPC fixes](#)

C...

[ClearMMU](#) [ClearTTX](#) [Compatibility Guidelines](#) [Contents of the Archive](#) [Credits: People I'd like to thank.](#) [CybSCSI/z3scsi problems](#)

D...

[DescriptorCacheInhibit](#) [The disassembler.library](#) [DMA based interfaces](#) [What is DMA, please?](#)

F...

[FastIEEE](#) [FixCybAccess](#) [FixP5Scsi](#) [Installation of the BVisionPPC fixes](#) [For \(configuration command\)](#) [Frequently asked questions, did you check these?](#) [Future plans about the mmu.library](#)

G...

[Glossary](#)

H...

[History: What happened before?](#)

I...

---

---

[Installation of the 68020 Library](#) [Installation of the 68030 Library](#) [Installation of the 68040 Library](#) [Installation of the 68060 Library](#) [Installation of the BVisionPPC fixes](#) [Installation of the debugging tools](#) [Installation of developer material](#) [Installation of the mmu.library](#) [Installation of the MMU-Configuration](#) [Installation of MuOmniSCSIPatch](#) [Installation of non-autoconfiguring memory](#) [Installation of the SetPatch upgrade](#) [Installation of Software Fixes](#) [Installation of Zorro-II memory fixes](#) [Introduction: What's the MMU.library?](#)

L...

[Logical vs. Physical Addresses and the DMA problem](#)

M...

[The MMU-Configuration File](#) [MMU Guide](#) [What's the MMU.library?](#) [MuEVD](#) [MuFastChip](#) [MuFastROM](#) [MuFastZero](#) [MuForce](#) [MuGuardianAngel](#) [MuLink](#) [MuLockLib](#) [MuMove4K](#) [MuOmniScsiPatch](#) [MuOVLYMGR](#) [MuProtectModules](#) [MuScan](#) [MuSetCacheMode](#)

N...

[Non-Auto-configuring Memory](#)

P...

[P5 System compatibility problems](#) [P5Init Boards with a PPC processor](#) [Programmed I/O interfaces](#)

R...

[Software replacement list](#)

S...

[ScanMMUPort](#) [SetCacheMode](#) [How to speed up the computer](#) [Software replacement list](#)

T...

[The THOR-Software Licence](#)

V...

[What is virtual memory, please?](#)

Z...

[CybSCSI/z3scsi problems](#) [Zorro-II 16-bit Memory Problems](#) [What is Zorro-II memory, please?](#)

---