The x86 Memory Management Unit

Some years back, while I was reading a quite recent book in operating system development, I found some information about memory management in operating system design and how it's done out there with some specific operating systems, being it hobby or professionnal. I read something like "...the kernel is loaded in the upper 4GiB of address space". I also saw weird memory addresses which were 32 bits. At that time, I knew that most computers could have a maximum of 512 MiB of memory onboard. I thus asked myself: How is it possible to access memory above the gigabyte when you only have say 128MiB onboard? Weird thing with yet, a simple answer: the MMU is the guy behind this "Mafia".

The MMU stands for Memory Management Unit,also sometimes called Paged Memory Management Unit. It is computer hardware component responsible for memory access handling by the CPU. Its function depends on the type is system and type of CPU:

1. Virtual to physical address translation (virtual memory management). This will usually be found on more modern CPUs such as x86 based processors and others.
3. Memory Protection.
4. Cache Control / Bus Arbitration. This will be found on CPUs having a cache for cache control; and bus arbitration will be found in multi-core systems.
6. Bank switching. This is usually found in small 8 bit microprocessor systems such as the 6502 processor.

The MMU can reside in the same IC package as the CPU (that is the case in major 32 bit CPUs) or could be external to the CPU IC package.

As the major and most important (in our context) use for the MMU is for virtual memory management, we will not disgress further in it's other uses and concentrate on this topic.

To achieve its goal of address translation, the MMU divides the address (virtual) in little fields that will permit the location of data in memory. The entire virtual address space is divided in little chuncks of memory called pages. For those who are lost, a virtual address is the address the cpu 'sees and sends' to the MMU. The MMU will then translate this to a physical memory address. This is depicted in figure 1.

As said above, the linear address space is divided into little fields. The upper bits are used to point to a page an offset in a page (figure 2).

Each page directory is an array of page directory entries. Each directory entry contains the base physical address of a page table, i.e the physical start address of the page directory. It also contains attributes of the page table, the

most common being a flag to indicate if the page is present in memory, a read only, system or user page table (figure 3).

The page table is an array of page table entries.Each page table entry contains the base address of a table, that is the address (physical) of the first byte of a table. Also, it contains attributes that are quite similar to the ones in the page directory entries (figure 4).

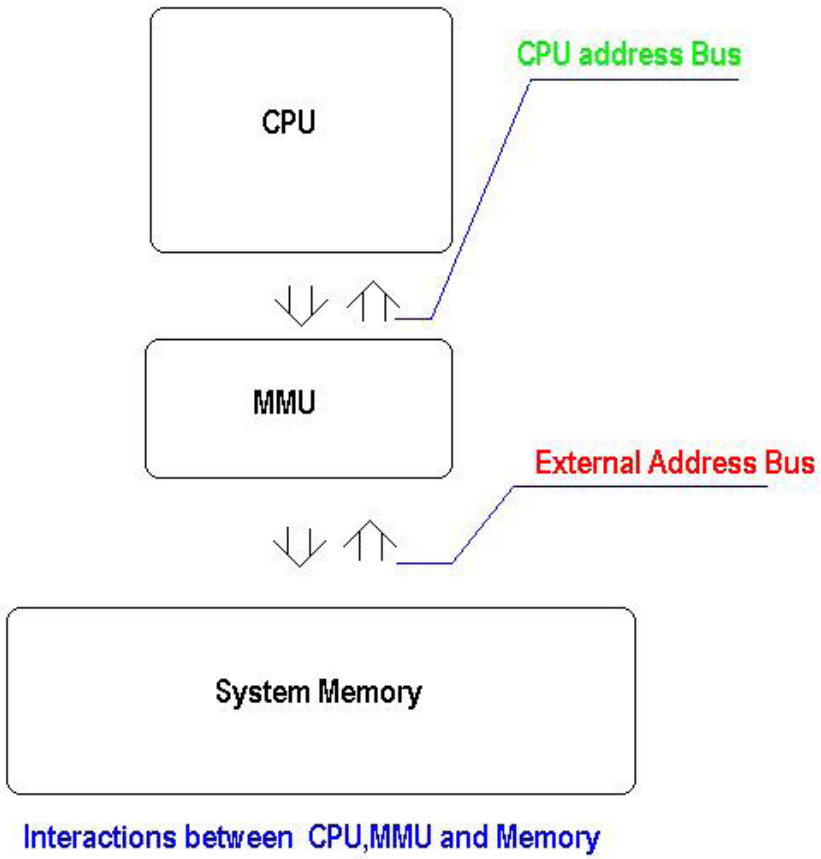


Figure 1

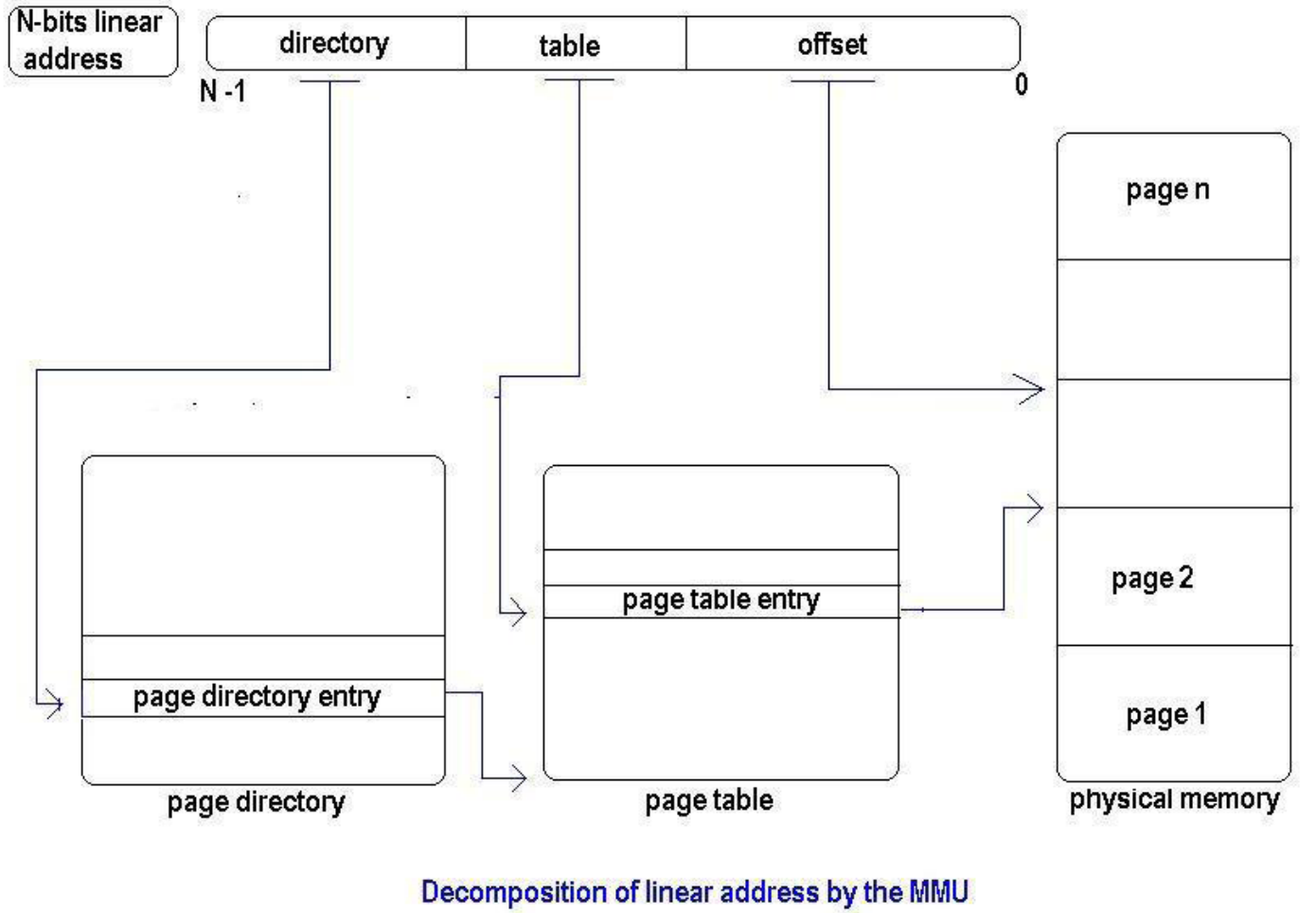
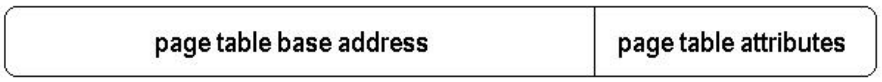


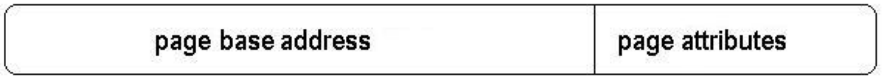
Figure 2

So to resume, the page directory contains contains entries that tell us where the page tables of a program are located in main memory, the page table tells us where the pages used by this program are located in memory. All of this excluding their properties such as the page being present in main memory or not and others.



structure of a page directory entry

Figure 3



structure of a page table entry

Figure 4

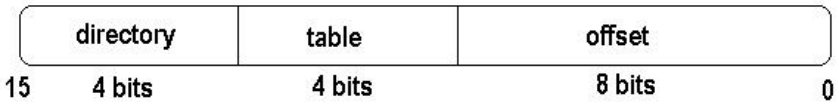


Figure 5

The main purpose for keeping track of which pages are not in main memory is to simulate the existence of much more physical memory than what is present on the system. When the CPU tries to access a memory location in a page that is not in main memory, the MMU will generate a page fault.

A page fault is an exception indicating that a page is not present in main memory. So a system with 128 MiB of ram could use this mechanism to increase it's memory capacity to say 1GiB if we suppose that there is a secondary storage device such as a hard disk which can provide us with the rest of the 1GiB we need.

When a page fault occurs, the CPU invokes a programmer defined routine usually called the *page fault handler* that does the necessary routines to bring the missing page in main memory, by say sending an unused page into secondary memory.

The operation of moving an unused page to secondary memory is called swapping, and the file where the unused pages are stored is usually called the pagefile or swapfile.

I will take here a concrete example of how all of this works. Some people with some experience will sort out that my system is feasible, but non economic. My hypothetical system will consist of a CPU with an MMU.
This imaginary CPU will have a 16 bit address bus. This address will be divided as follows:

1. The directory field will be 4 bit wide so will have a max of 16 entriees.
2. The Table field will also be 4 bits, so 16 entries max also.
3. The offset field is 8 bits wide so each page will have 256 bytes.

This addressing scheme is shown in figure 5.

We will now suppose that the cpu has output address 0xFAAA on its address bus. All addresses are in hexadecimal. Since the upper 4 bits of the address are used to point to the page directory entry,it will point in this case to entry 0xF, i.e the topmost entry in the page directory. Lets further suppose that each of the entries in the page directory is in the format shown in figure 6.

Figure 6

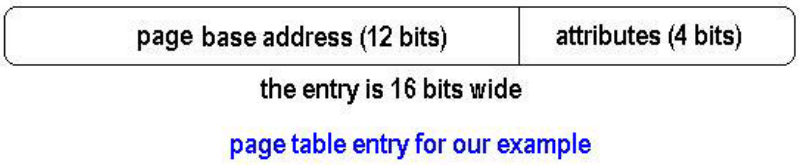


Figure 7

Lets again suppose that in our case, entry 0xF contains 0xA002. 0xA00 i.e the three first bytes gives us the physical base address of the page table being used at the moment. Let's not care about the attribute bits for now.

Taking another look at our first address from the CPU, we see that the directory field (which is 4 bits in our case) is 0xA , i.e 10 in decimal. This means that entry 10 in our page table, the page table being located at address 0xA000 as we saw above, contains the address of the first bytes of the page where our information is located.

Now lets suppose (again!!) that our page table entries have the format like in figure 7.

In our case we will suppose (for the final time) that our entry number 10 contains 0xCB1A. This means the physical base address of our page being accessed is 0xCB1 or 0x0CB1.

Our offset into the page in this case (looking again at our CPU address 0xFAAA) is 0xAA, so our physical address the MMU sends to the memory chip is 0xCB1 + 0x00AA = 0x0D5B. So the virtual address 0xFAAA "becomes" the physical address 0x0D5B.

All of this is supposing the page is present in memory (that is the present flag is set). If it were not, then a page fault exception would have been generated and software would have been invoked to take appropriate actions to move the page from the pagefile to main memory. Many algorithms out there have been designed for that such as the LRU (Least Recently Used) algorithm.

My hopes are that this tutorial has not been long and boring and that many people gained something when reading this. Thanks to all. If you have any queries, email:

dzouato@yahoo.fr