An Introduction to Intel Memory Management

Jeremy Pierre
jeremy@0x90.org

1 Introduction

This document exists to give a basic overview of how memory is managed by Intel processors running in 32 bit protected mode. It is not intended to explain how one would bootstrap an operating system or anything else of the sort. Readers are encouraged to read the texts given in the bibliography as these can greatly further your understanding of how processors operate on the lowest levels. From here on, assume that we are discussing memory specifically from a protected mode point of view. Any addresses given will be in hexadecimal, preceded by the conventional 0x prefix.

Memory on an Intel compatible processor is managed on three levels. From bottom to top they are:

- Physical
- Linear (paging)
- Logical (segmentation)

A program never actually operates directly on the linear or physical levels, although you will see that it is possible to give this illusion. The linear address space is typically handled by the paging mechanism. The paging facilities on the processor allow us to remap physical memory to suit the needs of the program running. Logical addressing, or segmentation, allows us to separate different aspects of the program from each other, such as the program code itself and the stack.

The majority of this document will focus on how segmentation and paging mechanisms relate to each other and create an address space [1] for a program to operate within. Protection mechanisms themselves will not be discussed in much detail. Readers interested in learning more about the protection facilities on Intel processors should refer to the materials listed in the bibliography.

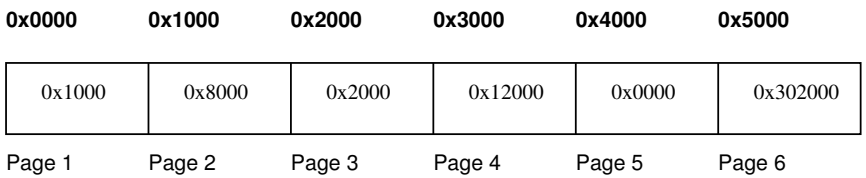
2 Physical Memory

Physical memory is the most basic address space on Intel processors and is actually never seen directly by programs. Essentially, byte 0x0 is the first byte in RAM. Naturally then, the byte addressed by the index value 0x10FB359 is located at location 0x10FB359 in RAM. There is no form of translation or mapping at the physical memory level.

[1] Address Space refers to a specific range of memory that a program(or process) has access to read from and/or write to.

3 Linear Memory

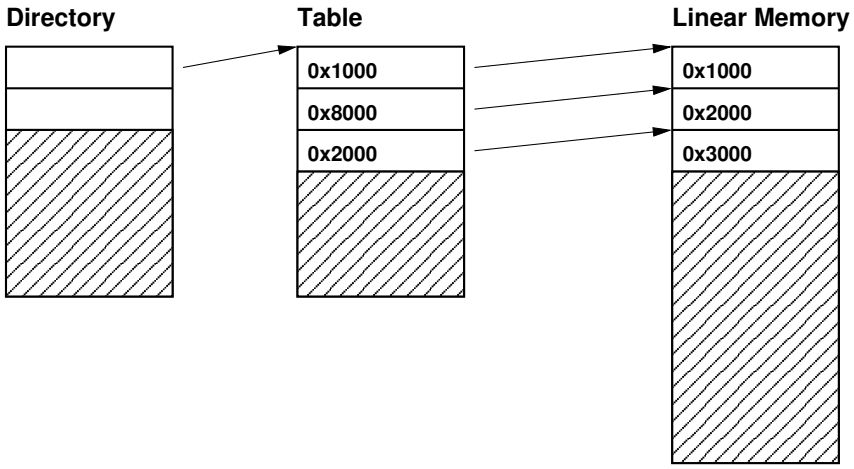
Linear memory, also known as virtual memory, looks very much like physical memory, except that we can remap sections differently by means of the paging mechanism. Paging, for example, will allow us to make location 0x0 actually map to location 0x1000. As you can imagine, if we are not careful, memory will start to look like a bad jigsaw puzzle. The following figure illustrates a small example of a paged area of memory. Each box in the figure represents a single 4k page of memory(default page size). The addresses listed above the boxes are the linear addresses(what the program sees) while the addresses inside the boxes represent the physical address of each page, or the actual address in RAM of the page.



As you can see, paging allows us to represent portions of physical memory any we want to. For example, even though Page 5 actually is stored at location 0x0 in physical memory, as far as our program is concerned physical address 0x1000 comes first because it is mapped to address 0x0.

Paging on the Intel processors is handled by two control structures, the Page Directory, and Page Tables. Both structures occupy exactly 4k of space in memory. The Page Directory is basically an array of pointers to each Page Table. Each entry in the directory contains the base physical address of the table it points to. The directory entries also allow us to control attributes such as read and write access to entire Page Tables and the pages the tables point to. The Page Tables are almost identical in appearance to the directories, except that each entry points to the base address of a single page. If we wanted to, instead of controlling the attributes of memory at the directory level, we can accomplish it on a page by page basis. The following figure is a basic representation of a Page Directory

pointing to a Page Table which points to several individual pages.



The addresses in the table entries are the physical base addresses of each page being pointed to, where the addresses listed in linear memory are what software will see. The maximum amount of memory we can address with a full Page Directory and full Page Tables under it totals 4 gigabytes, assuming we are using the default page size of 4k [2].

Now for the location of Page Directories. The CR3 control register points to the physical base of a Page Directory. This register is also known as the Page Directory Base Register(PDBR). Since the CR3 register is changed upon a hardware context switch [3] , we can actually create completely different linear memory maps for different programs running simultaneously.

4 Logical Memory

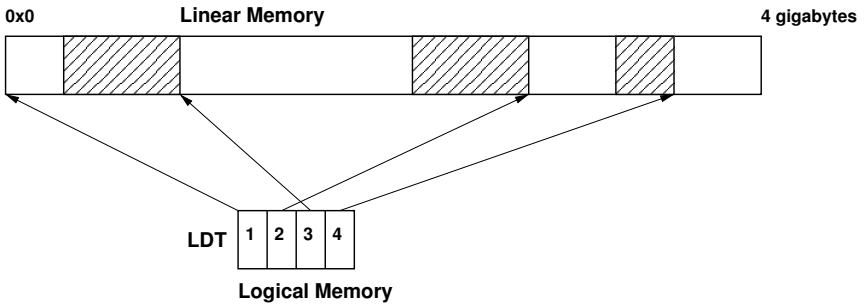
Logical addressing using segmentation has been common on Intel processors almost from the beginning. To date, there is no way to operate an Intel compatible processor without segmentation, we cannot turn it off as is possible with the paging mechanism. Originally, a segment was always 64k in size, and to address memory one needed to supply a two part address composed of two words(16 bit values) separated by a colon such as 0xDEAD:BEEF. The first word is called the segment selector. The second word is called the offset. So 0x0:1000 refers to the first segment in memory, at an offset of 0x1000. The problem comes when you try to access memory location 0x0:10000, as this is an offset bigger than 64k.

Since the 80386 processor, we have had a much more flexible implementation of segmentation. A segment can be any size needed, up to the addressable limit of 4 gigabytes. Much like there are tables to keep track of our linear memory pages, there are table structures to keep track of our logical segments of memory. Two types of tables exist:

- Global Descriptor Table(GDT)
- Local Descriptor Table(LDT)

Each processor must have one and only one Global Descriptor Table(GDT) and it cannot be larger than 64k. The upside is that we can have one Local Descriptor Table per process so that each process on our computer can have its very own logical memory address space, just as each process can have its own linear address space.

Our GDT looks almost exactly like any LDT with one exception: the first entry in the GDT must always be filled with zeros. This is for several purposes such as debugging, access checks and so on. In any given LDT, the entry 0 can be anything we want. The GDT and LDT's contain segment descriptors that allow us to arbitrarily define the base address, size and attributes of a segment for use by a process. The following figure illustrates a few segment descriptors pointing to arbitrary linear memory locations.



[2] There are extensions to the paging hardware on a number of processors since the Pentium that allow 2 and 4 megabyte pages as well as an address space up to 64 gigabytes. See the bibliography for further documentation on which processors support this as well as specifics of operation.

[3] A context switch, or task switch is when control of the processor changes to a different piece of software(multitasking/multiprocessing).

The empty boxes of linear memory represent arbitrarily sized segments. The LDT boxes are individual segment descriptors. The numbers inside the segment descriptors are called segment selectors. Specifying a logical address is the same as it was 20 years ago, except the offset could be just about anything. For example, if segment 1 had a size(or "limit") of 2 megabytes, the logical address 0x0001:00040558 is perfectly legitimate.

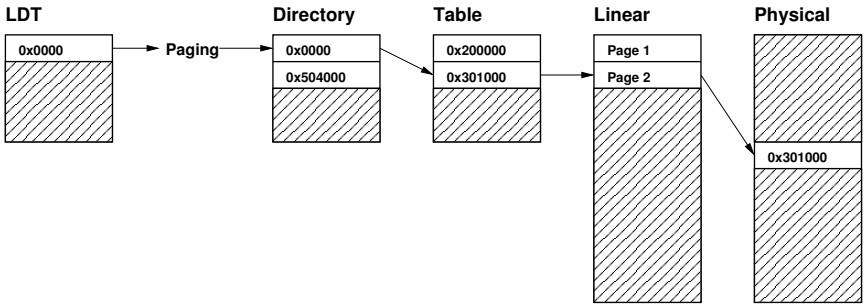
Our previous example is just fine, except that the supplied segment selectors are completely incorrect. Segment selectors are still 16 bits long just as they were 20 years ago. The difference is that now we use the first three bits of a selector for some extra information like priviledge levels and indicating which table to use. Priviledge levels are beyond the scope of this text, but the table inicator bit is important. The following figure illustrates two segment selectors at the binary level.

The grayed out areas in the selectors are a Requested Priviledge Level field occupying 2 bits of space in the selector. The Index section is simply an index into the descriptor table containing the segment descriptor for the selector. The main difference between these two selectors is bit 2, the Table Indicator(TI) bit. When set, the selector is for a segment contained in the current Local Descriptor Table. When cleared, the selector indexes a segment in the GDT.

5 Combining Segmentation and Paging

Now that we have covered logical and linear addressing, it is time to put them back together. Quite simply, a logical address consisting of a segment selector and an offset is translated into linear address for the paging mechanism. The Paging mechanism then translates the linear address into a physical address and your memory access completes itself.

For an example, let us consider a segment with a base address of 0x0 and a limit of 1 megabyte, or 0x100000. The following figure illustrates the translation from logical to linear to physical address assuming we are looking for address 0x0004:1000. Note that our segment selector has bit 2 set with an index value of 0. This indicates that we are looking for the first segment in a Local Descriptor Table.



Our LDT descriptor 0x0000 points to a base address of 0x0. The paging mechanism translates this and passes us along to the Page Directory. Since we are operating within the first 4 megabytes of address space it forwards us to the first page table. The offset we supplied is the first byte in the second page, so the translation passes us along the chain through linear memory and on to our physical address of 0x301000. A few details were missed along the way, but that is basically what happens.

5.1 Simulating a Flat Address Space

We will end this introduction to memory management with an example of a simulated flat address space, such as the one that Linux implements. A flat address space is basically a uniform memory region used for both code and data with no logical seperation defined between the two areas. As a code segment cannot be written to, we cannot simply discard the data segment altogether. Since a data segment disallows code execution, we cannot be rid of the code segment.

The simple solution adopted by the Linux operating system is to have one code segment descriptor in the GDT that spans a full three gigabyte memory region. Also housed in the Global Descriptor Table is a data segment descriptor that overlaps the exact same three gigabytes of memory. Since each process can have its own Page Directory Base Register value, each

process can use exactly the same logical address space for every operation, but maintain separate linear memory regions to avoid tampering.

References

[1] Intel Architecture Software Developer's Manual, Intel Corporation

[2] Hans-Peter Messmer: The Indispensable PC Hardware Book, Addison-Wesley, 2002