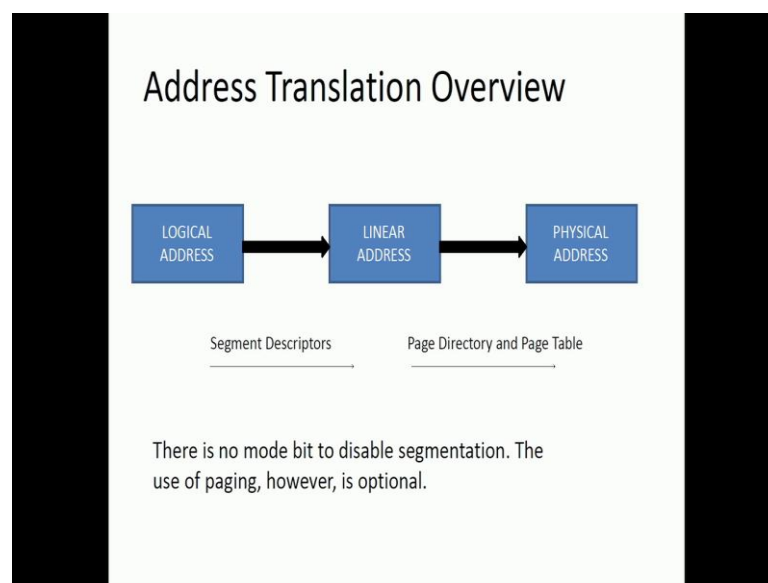


Information Security - II
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 27
Memory Management

We had seen Paging yesterday. We will just do a very quick exercise on Paging in the next 20 minutes. I will just give you a very quick overview. We have covered this in detail yesterday.

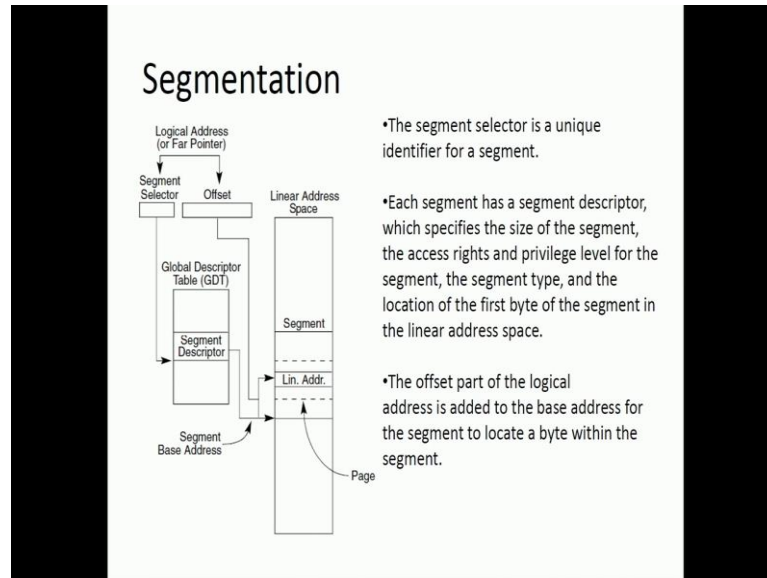
(Refer Slide Time: 00:22)



So, there is segmentation, before segmentation there is something called a Logical Address, which is generated by the compiler. When the compiler compiles a code, it assumes that the code segment starts at 0, it assumes the data segment starts at 0, it assumes that the stack segment starts at 0. When you load these segments, you update the segment register with the base address. Every time a data is accessed or a code is accessed or a stack is pushed or popped, it is with respect to that base address, so this ensures mobility. The address generated by the compiler is called Logical Address and that when you add to this segment base, it becomes a Linear Address. That linear address

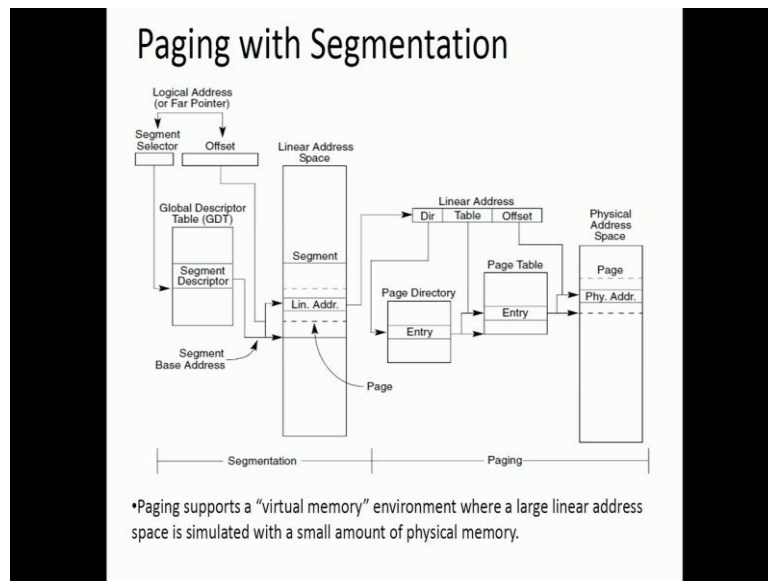
directly maps on to the address in the memory, if no paging is enabled. If paging is enabled, it does go through a translation before you get into the Physical Address.

(Refer Slide Time: 01:25)



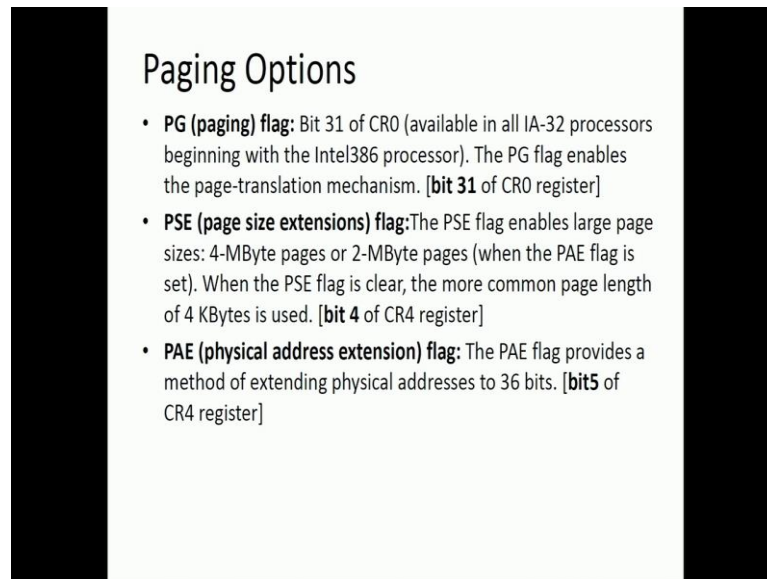
Now, this clearly explains what is segmentation? When I add a logical address that is generated by the compiler to the address given by the segment to the base of the segment, I basically generate an address in the logical address space, in the linear address space. Linear address space is that is not 0 follows 1, 1 follows 0, 2 follows 1 etcetera. So, I now get an effective address this is called an Effective Address. Effective address or linear address is nothing but your logical address generated by the compiler plus the segment base. Now you get a 32-bit address and what you do, you split it as 10, 10, 10, 12 each.

(Refer Slide Time: 02:13)



Your entire memory is divided into, physical memory is divided into page frames of size 4096 bytes, which essentially needs 12 bits for addressing and your entire logical address space is split into pages of size, again 4096 bytes which each requires 12 bits of addressing. So, you split the entire 32-bit address into 20, 20, and 12. So with 10, 10 and 12 and the first 10 bits, you index into the page directory whose base is given by CR3. Then that will give you an entry for the page table, the base for a page table and if that entry is valid that means a page table actually exist. Now, you go to that page table you use another 10 bits to basically index into that page table that will give you the starting address of a page. To that starting address of the page, you put this 12 bit offset to get an actual value of the page. So this is how we are doing it and we have already seen how this will work.

(Refer Slide Time: 03:30)



Paging Options

- **PG (paging) flag:** Bit 31 of CR0 (available in all IA-32 processors beginning with the Intel386 processor). The PG flag enables the page-translation mechanism. [bit 31 of CR0 register]
- **PSE (page size extensions) flag:** The PSE flag enables large page sizes: 4-MByte pages or 2-MByte pages (when the PAE flag is set). When the PSE flag is clear, the more common page length of 4 KBytes is used. [bit 4 of CR4 register]
- **PAE (physical address extension) flag:** The PAE flag provides a method of extending physical addresses to 36 bits. [bit5 of CR4 register]

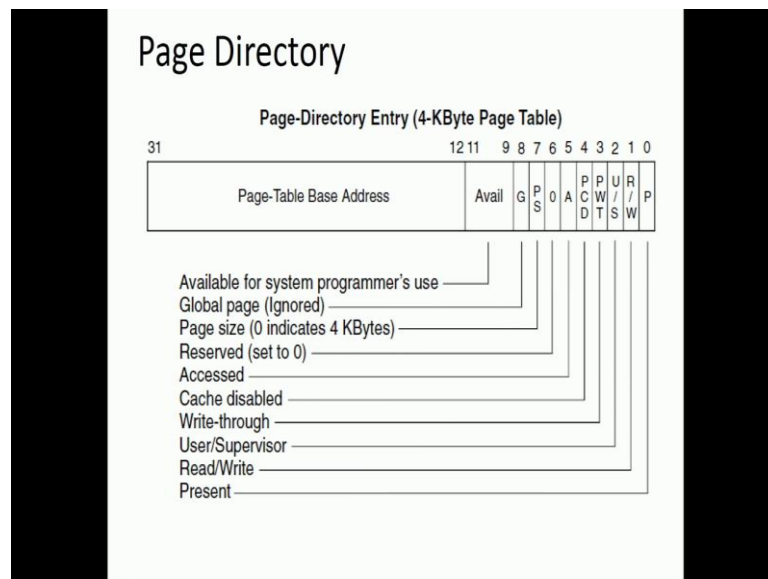
Now, please note that by default in the protected mode, you have segmentation. You cannot get away from segmentation but paging is optional. So, once you come into the segmentation, once you start the protected mode, which essentially has segmentation, you can go and set the 31st bit of CR0, the control is the 0 and then immediately paging will be enabled. But before you enable paging, you have to go and set up the page tables everything properly. If you do not enable the page table because the moment paging is enabled, every address including your access to code, the next instruction that you are going to access that address of the next instruction essentially will gets translated. So, before you enable paging, you should go and set up everything properly that coat-uncoat properly is what we are going to see.

There are some very good important subtle points that we need to follow, when you want that properly. So, you set all those things properly and then you start, you enable this paging, what do you mean by enabling paging? It is not rocket science, just go to 1 bit in CR0 and make it 1. But note that you can enable paging, only the privilege 0 code can enable paging, why should privilege 0 code, enable paging? Because, the privileged instruction moving into or out of the control register is a privileged instruction. So, only PL0 code can enable paging. Then there are 2 things that we can do here there is something called page size extension. So in the bit 4 of the CR register, if we go and set

1 flag as 1 then if it is flag is 0 then the page size is interpreted as 1 byte that is if I interpreted as 4096 bytes or 4 kilo byte. When I make it as 1 then this is interpreted as 4 mega byte pages. Now, 2 megabyte or 4 megabyte pages and whether it is 2 mega byte or 4 megabyte actually depends upon the PAE bit which is actually the 5th bit of your CR4 register. So, if your 4th bit is 0 of the CR4 then it is 4 KB byte. If your 4th bit is 1, let us call VSC bit page size extension bit is 1 then you go to the 5th bit, if that 5th bit is 0 then it is each page is 2 megabyte in size. If your 5th bit is 1, then each bit becomes 4 megabyte in size.

You could have small pages, large pages and medium pages, but this 4 KB to 2 MB is a big deal and then 2 MB to 4 MB is again doubly. So, you do not actually get good granularity when we start looking at paging. But note that, at paging from a security point of view, we have already seen the page table entry as you see here.

(Refer Slide Time: 06:50)



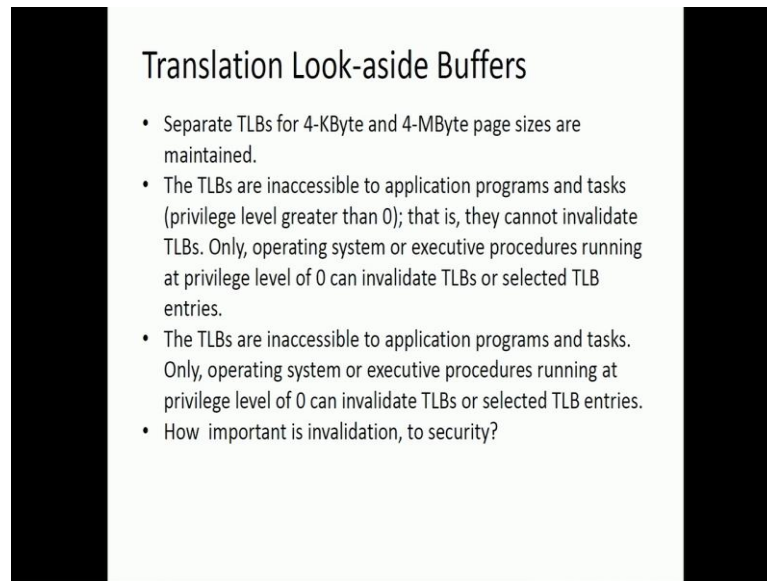
There is U bar S bit, which is the bit number 2 and there, if you put U it has 2. Then basically you can set that bit or reset that bit. Basically, you can allow PL0 code alone to go or any code to go. It is a user slash supervisor bit that we can we can handle there. Similarly, this is true with your page directory that is the first level paging which use the top 10 bits of your 32 address to index into and this is also true for all the second level

page tables where use the second 10 bits to index into it. And the last but one architectural concept before we go into the assignment is that, we have something called translation look a side buffer because when I want to access something in the memory for paging, I have to go to page directory that is a memory read. From there I have to go to page table that is another memory read. From there I have to go to the actual page in the memory and then read. So every memory read now becomes 3 memory reads.

In the case of segmentation, it was 2 memory reads and we solved by having a hidden path there. Now, in case of paging it now becomes 3 memory reads. So 2 of the memory reads are done for translation purpose that is your converting your effective address to a physical address. The address got after segmentation to a physical address 2 memory reads were dedicated or used for doing this translation and the next memory read is basically to get the actual value from memory. To stop this 2 additional or minimize these 2 additional accesses to the memory, we use something called the translation look a side buffer, which is nothing but fully associative cash that is staying inside your processor. So, if your page is already translated, you need not go and translate it again. All the recently translated page addresses are stored in your TLB.

So, when you generate a page address, what is the page, the first 20 bits gives you the page address the remaining 12 gives you the offset within that page. First you go and ask the TLB is this entry translated and available with you, if the TLB says yes use it, if the TLB says no then you do the translation. And then if you look at you know principle of locality of reference and special and temporal localities, this TLB will essentially lead to lot of performance benefit, which is actually in practical scenario we see that improvement in performance, significantly large improvement in performance because the page translations are not going through every time but once in a while. This is again a gist of paging; I have covered paging in the previous sessions much in detail.

(Refer Slide Time: 09:56)



Translation Look-aside Buffers

- Separate TLBs for 4-KByte and 4-MByte page sizes are maintained.
- The TLBs are inaccessible to application programs and tasks (privilege level greater than 0); that is, they cannot invalidate TLBs. Only, operating system or executive procedures running at privilege level of 0 can invalidate TLBs or selected TLB entries.
- The TLBs are inaccessible to application programs and tasks. Only, operating system or executive procedures running at privilege level of 0 can invalidate TLBs or selected TLB entries.
- How important is invalidation, to security?

But before we go through the assignment, we are just getting into some details so that we recap those concepts that we had covered in the various sessions.

Now we will move into assignment, but what primarily we are going to do in the assignment is to the actual program which will demonstrate paging for you on the x86 machine. We are going to do two things there, one thing is what it means to set up before we switched paging, what do mean by proper setting up and then we will demonstrate paging by saying that we will jump to a very far location but at will be my next location, so the translation will take care that it becomes the next location. That is one step that we will do and the next thing that we will do is a very, very simple thing like we will go and make a particular page not available and then we will make it 0 and then we will try to go and we will demonstrate a page fault, that such a page is not available how a page fault we will show that how it is going to the page fault handler. This will give you a broad spectrum of this. So, with this as the background I will just open it for a 2 or 3 minutes of doubts and then we immediately go into the code.