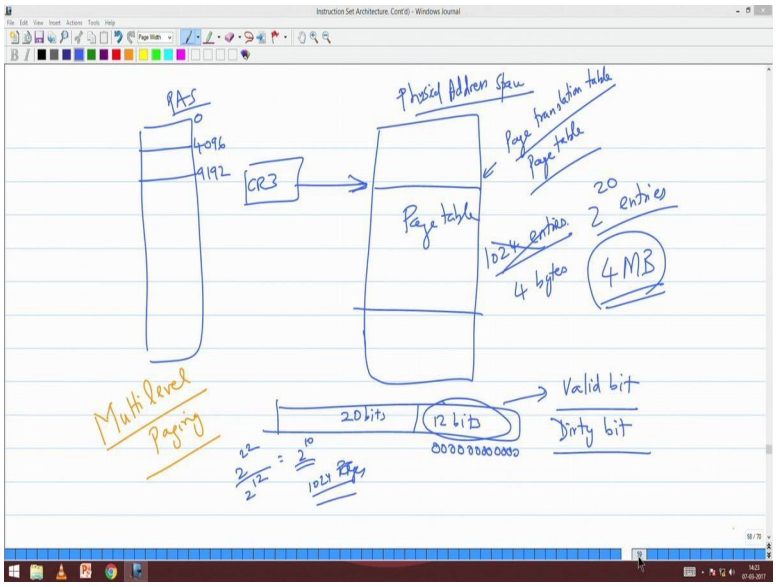
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 28
Multilevel Paging, Translational Lookaside Buffer

Good afternoon.

(Refer Slide Time: 00:27)



So, we continue with morning's discussion. So, we had in the entire virtual address space, basically we had the page page translation table this essentially we will call it as page table. The word that is commonly used in the literature is page table and we basically the start of the page table was given by CR3 right I am sorry.

So, this was a physical address space there was a page table here inside this and then the

start address of this was given by is stored in CR3. Now what basically happens is that in the morning discussion we said that we need something like 1024 entries and each of each of 4 bytes right 1024 entries no, 2 power 20 entries sorry each of 4 bytes. So, this is something like 2 power 20 to which is 4 megabytes of memory will be used for setting up this page table right because 2 power 20 is 1 megabyte each 4 bytes. So, 2 power 40.

So, before going further, so that is what this is big disadvantage like 4 MB of the memory is just used for you know the storing this page translation mechanism this is one important thing. But before going further on that please see that the page frames in your physical address space which I call it as pas actually starts at. So, since each page is 4096 it starts at 4 KB boundaries and so on right.

So, the page the starting address of a page will be in a 4 KB boundary; that means, the last 12 bits if we take this as 32 bits the last 12 bits will always be 0s for the start of a page because your page frames are 4 KB it starts with 0. Next will be at 4 KB next will be 8 KB 12 KB and so on so; that means, it is at a 4 KB boundary; that means, the start address of every page will have the last 12 bits as 0. These 12 bits I can basically use for other constructive purpose right and I will tell you 2 interesting things that we use this page bits for one thing is there is something called a valid bit.

The valid bit says that whatever I have entered see if I look at any 33 bits it will be a random 0 and 1. So, any combination of 0s and 1s in that 32 bits will makes some sense to me right. So, so in this 12 bits one of the bit will be valid bit which I will set it to 1, if a real the corresponding page is loaded I will set it to 0 when the when the no page is when the corresponding page is not loaded. So, I have two 20 pages 2 power 20 pages and for each page I have an entry in that entry the last 12 bits I will be using for some constructive purpose one of the purpose would be one of those 12 bits what is that bit Prasanna will tell you at 3 o clock that bit essentially if it is 0; that means, page is loaded in memory.

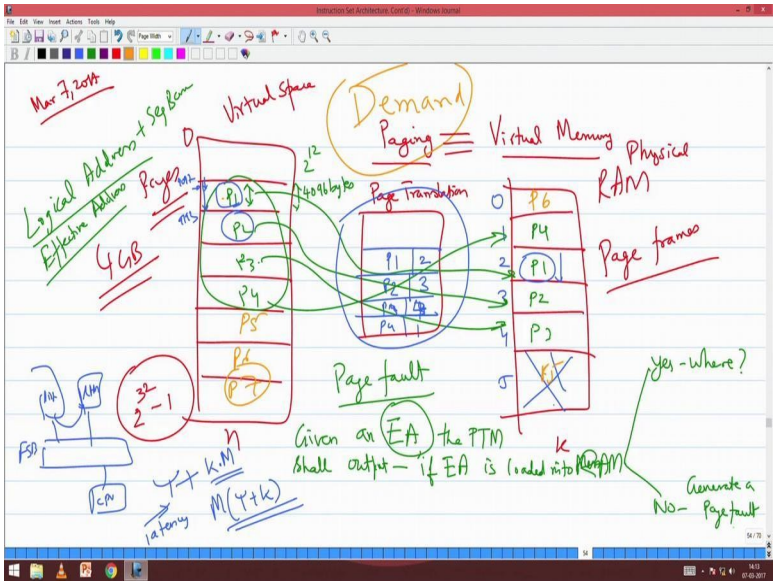
If that is one means that page is loaded in memory. So, when I load a page I will go and make that bit as one correct that valid bit as 1, to start with all the bits will be 0 I will make that bit as 1; that means, that page is loaded. So, when I get when I have this 32 bit affective address right I take those 20 bits and index into this table. After indexing into this table I will go and check that valid bit if that valid bit is one then we go ahead the valid bit is 0 me means who the architecture the hardware right it will go and if the valid bit is 0 then it will raise a page fault correct because that page is not actually loaded. Then the page fault handler will start working on it are you able to get it yes or no. So, this is very very important. So, that is that is something that we need to keep in mind. So,

so one bit is the valid bit another bit I call it clean bit, but western world call it dirty bit, but what do you mean by dirty bit. So, so we will go with dirty bit because ultimately somebody else will call it as for dirty bit.

So, see why we need this dirty bit can anybody tell me why we need this dirty bit dirty bit is after I load that page from the disk to the memory have I gone and changed something in that if I go and change something there immediately that dirty bit will become one. If I do not change anything there then the dirty bit will be 0. Why do we need this dirty bit.

Can you guess when I want to replace a page. So, morning somebody asked now what happens when all the pages are full I go and replace a page when I replace a page I mean I am going to over write on an existing page correct. I am going to over write on an existing page when I am over writing should I go and copy the original page back and then over write or should I not copy the page and just over write right.

(Refer Slide Time: 07:08)



So, let us go back to that 54 yeah. So, I want to replace say P 1 by P 6 or because, P 7 I want to replace P 1 by P 7. Now if P 1 is may be P 1 was the least recently used I have only

6 pages now I have to replace someone. So, the interrupt service routine the page

fault handler decides to replace P 1. If after loading into the memory if P 1 was changed right then I have to copy the entire P 1 back here and then load this P 7 after loading into the memory P 1 was not changed it was red, but not written into then I can just overwrite P 7 now taking this 4096 bytes and putting back it to the disk is also a very time consuming affair.

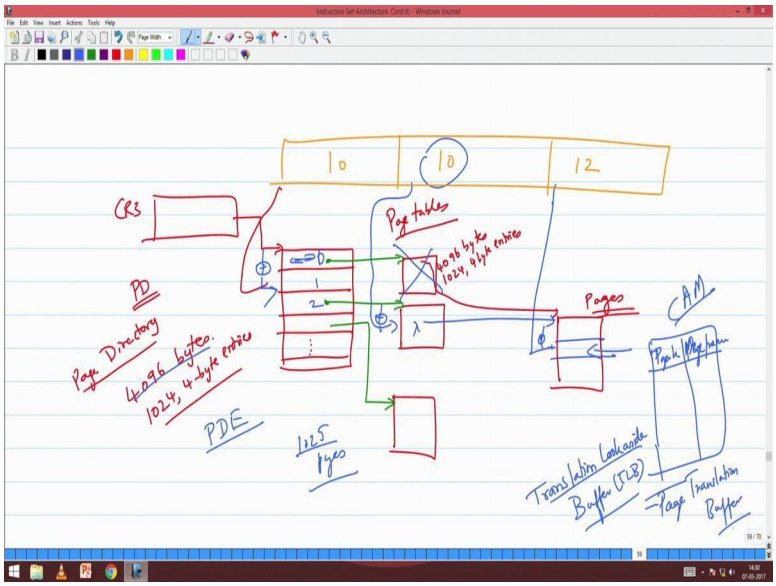
And there is no need for me to do this right there is need for me to do this if this fellow is not changing at all if P 1 did not change at all why should I copy the same thing back again if the copy on the virtual address space on the disk and the copy in the main memory are same I need not go and rewrite it correct. If the copy in the virtual space in the disk and the virtual memory and the ram the physical ram are different then I have to go and write back P 1 and then write P 7. So, the moment I go and change something in the page the hardware hardware goes and makes that dirty bit as one. So, that in case you are going to replace this page what you do you go and copy this entire page back to the virtual address space and then and then you know copy this and then overwrite on top of it ok.

If this dirty bit is 0 then you do not do anything just overwrite correct. So, that is one another interesting bit of course, there are some privilege levels also that the some more bits that we can use and then there will be still lot more bits available and these bits can be used to implement page replacement policies the operating system will judicially use these bits can use this bits to basically implement certain policies the replacement policy like least recently used which page to replace given this are you able to follow any doubts. So, now, do you understand how I am exploiting the availability of some redundant bits the 12 bits for you know basically doing this valid check and the dirty check and we get lot of performance enhancements based on this, is it fine, any doubts.

Now the big challenge that we have is this 4 MB should I have 4 MB right now very few programs would use all 4 GB address space 4 GB is a huge address space your gmail is only 12 GB 15 GB right and it takes a life time for you to fill it up. Now your 4 GB is a large address space for many many practical user unless you are going to do some high end compute supercomputing problem or very high end VLSI problem assumption basis for normal usage even for normal sort of you know even online web transactions or

database even banking system or your railway reservation you will not actually load that entire 4 GB address space with your program right it. So, one of the thing is that this 4 MB that I want to basically store for page table can we do something better. So, that is question. So, now that gives us. So, not only in Intel there are many other architectures which follow this today what we call as multi level paging.

(Refer Slide Time: 11:27)



Now what is multi level paging we will just look into it very shortly it is very very simple actually in multi level paging what we do is the entire 32 bit right I make it as 12 bits 10 bits and 10 bits. Now CR3 point to the start of one page start of a page which I call it as page directory P d I call it as instead of page table I call page directory.

So, this page directory will be one page; that means, I will have 4096 bytes; that means, one 1024 4 byte entries. Each of these entries here, each of these entries here will point to another page will point to start of other pages and these pages also have these are also 4096 bytes and they will have one 1024 4 byte entries. Now each of these entries will point to the actual pages. So, this is called page directory this is called page these are all called page tables and these are all and they will point to the actual pages right. So, instead of having one level of translation now I have multiple levels of translation. So, what I do I take

these 10 plus 10 plus 12 the first 10 bits I will index into I will index into

this fellow because there are 1024 entries, I have 10 bits 2 power 10 is 1024 I will index here.

This will point to another page table, to this page let just say this pointing, to this to this I will take the next 10 bits I am marking in blue here to index into this because this will also have 1024 entries right. So, I will use these 10 bits to mark on to this 1024 entries. This will point to some page some start of a page that page will have 4096 bytes I will use this 12 bits to index into that and that will give me the actual physical memory that I need to access.

You are getting this. So, now, instead of having only one translation of 4 MB size what I am doing I have one page directory which will have 1024 entries 4 byte entries I use that 10 most significant bit into index into the table into the directory that directory we call each of these entries as PDE page directory entry, each of those entries will point to a page table. So, I go to corresponding page table namely here and use the next 10 bits, next 10 most significant bit index into this page table correct and that will give me the start address of a page to that page I give that page has 4096 bytes. So, I use the 12 bits index and this is this will be my actual location I want to access. So, instead of one direct translation I go to 2 levels of translation to basically get the addressing of it is it are you able to follow.

Note that each of the page directory will fit in 4096 bytes and each of these page tables will also fit in 4096 page. So, somebody ask me why 4096 bytes in the morning this is one explanation for 4096 bytes. So, this will also fit. So, each of the page table the page directory everything will fit in one page itself right. So, what happens now instead of one look up now I go to 2 look up first time I just go and look up one table and then just add the address and take now I look up one table that gives me another table look up that table add the address and go and access.

But since everything fits into a page the last 12 bits of everything for example, for the last 12 bits of this page table what does this say what does this entry in the page directory say its points to the start of the page table and again since everything is page aligned that last 12 bits will be 0. So, those 12 bits can be used to find out if there is a page table

associated with it is it present or not I can tell that. So, I can do one check here of whether the corresponding page table if nobody in this address is basically access then that page table will not be even touched up right.

So, then what happens then, even in the page table again the pages are again 4096 bytes aligned. So, ultimately the 12 bit will be 0 and that could be again used for all the other purpose. So, this multi level paging what I have created I have 1024 plus 1025 pages are needed to finish this 1025 pages are needed to you know get this things up. Right, in the previous case we had 1024 entries right or what did we have 2 power 20 entries divided by sorry we had 2 power 20 2 divided by 2 power 12 we have something like 2 power 10 ten 1024 page tables we had last time.

1024 pages it will occupy now this will occupy actually in the worst case this will occupy in the worst case this will occupy what 1024 plus 1025 pages one more page extra. But then please understand that when I am see always the 4 GB will not be full for all the all the holes or wherever it is not being used that corresponding page tables can be thrown out it is not mandatory for me to have all the 1024 pages here, I certainly need the page directory which will occupy one pages one page sorry right I will I need to have the page directory that is always there.

Now, the respective page tables inside that page directory need not be populated immediately as and when I need I want to I will populate and I do not need I can throw it off if there are no programs. So, all the programs that are executing falls within these 2 addresses these 3 addresses then I will have only 3 page tables for this the remaining things I will just leave it off as invalid. So, and suppose the program corresponding to this has finished I can throw that page off and I can use that page for some other purpose. So, this means that the size of the page table is directly proportional to the amount of memory you are actually utilising correct, if you are not utilising some amount of memory then the corresponding page tables corresponding to that address can be thrown are you getting this right.

So, the amount of paging I use the amount of page you know number of page tables I use the size of the memory that is actually reserved for page table usually depends upon size

of the actual memory being used. So, this essentially this multi level paging basically gives us a mechanism of a dynamically growing and shrinking page translation mechanism right. So, when there are lot of pages being used corresponding page tables and page directories that corresponding page tables will be loaded when some pages are finished I can throw them when I want again I can bring. So, I have an on demand page table right like what we talked in the morning as on demand paging now I have an on demand page table is it all fine. So, this is the advantage of multilevel paging in contrast to the single level paging, any doubts, are you able to follow right.

So, the next thing is that now what has happened. Now, I want I do some amount of segmentation I get an affective address, after getting the affective address I first go to the page directory entry from there I get some entry using that I go and index that and I go to the page table entry there I get some value that I take and add the remaining 12 bits and access the page. So, there is a lot of memory access one memory access now assuming segmentation we are not accessing memory even before I find where that particular data is there and gone right now that 2 now have become 3 and 4 right. So, it is just increased are you able to follow this. So, what we do is that we also have an associative memory or content address content addressable memory where in we just go and store what we call as page translation buffer; in this I go and say this page k is loaded in this memory location this page frame. So, I have the mapping between the actual pages that are loaded in the memory and the corresponding starting address of the page file. So, I have a mapping between the affective address and the physical address.

So, what I do is I go to this cam I go into this P t b page translation and ask hey can you do the translation for me this fellow says no I cannot do because I do not have that entry then I go and do all the by the by the principle of locality of reference and locality of spacial locality and the temporary locality I know that this page translation buffer as concept succeed it correct the P t b as a concept will succeed because. So, this is also called in literature as translation lookaside buffer. So, I could have what do you mean by translation lookaside buffer do not go and do the translation just look aside it then try it off come to me I will tell you what the translation is. So, for many instances this t l b will help because we are having this spacial locality and temporal locality. So, what will happen is at many instances this translation lookaside buffer will give enormous

performance boots to the virtual memory system.

So, now what we will do in the lab starting now the last statement will be given, but then we will also. So, it will be extension of your third assignment, but the point first assignment extension of your first assignment just check if you have remembering that. There are some points that you will do when you do the code you will understand and tomorrow morning I will debate on that in great detail. So, what is it to enable paging till now first 2 3 you are not enabled to paging what it means to enable paging what are the cast that need to be taken when I want to enable paging. So, that is one thing that we need to keep in mind.

The second thing that you know we need to also talk this what are this page replacement algorithms how they are going to come. So, several of those background I will try and cover the difference between a single core and a multi core is there are 2 major differences how these cores are interconnected there is an interconnection work that is happening. The second thing that we need to be careful about you know moving from a single core to the multiple core is that how the memory is being accessed, how they are interconnected is one story, how they are how the memory is going to be accessed is another story. So, tomorrow we will do a deep dive on this how do how do we set up and what are the things why 4 (Refer Time: 25:28) all these why and what we will try and answer in tomorrow morning class any other doubts yes.

Student: (Refer Time: 25:36).

See, so when I give a address of a page right instead of going if I do this 32 bit instead of going here then here and then here after I do one page translation that data right this page is stored at this page frame. So, if the second page is stored at this frame then I cannot I need not do the other processing correct. So, it is something like I give an address to that P t b or your translation P and b and ask hey is this address already loaded is this page loaded.

It will not only say yes, but it can also say that where it is stored right it will tell you where it is loaded. So, that will be given in this translation because this is very close to

your b t b what branch target buffer says whether the branch is taken or not and if it is taken then it will also give the target address buffer. Similarly here this t l b will tell you where it is loaded and if the pages is at all loaded and where it is loaded which page for (Refer Time: 26:48), directly I can go to memory t l b you cannot see, t l b is invisible to the assembly program, but you can basically see the I think there are some performance counters which can tell you how many t l b s are there that we can probably program, but not in your we are not looking at performance in this course at all, but we may look I will use some pointers about performance come to later.

But otherwise we get basically implement this page. So, what will you be doing in the assignment is you set up this entire translation mechanism. And what you do is that this page directory there will be one thing separate. And you will make CR3 point to the start on the page directory it will be one page there will be 1024 entries then what you do that and the pages here right you, there will.

You take some 2 pages and you will be mapping. I think Karthik will be explaining you that right and then some of the things that you need to take care is that page table the page containing the page directory the page containing the page tables these need to be identically mapped.

Identically mapped means if I am storing it at say 4097 right I need that same 4097 if it is not identically mapped then you go into sub task. So, the page that I mapping on to is identically mapped you are getting this. And if you have any doubts you can go to my information security 2 course again we have done the same thing there. So, you could have a feel of how things work there. So, all the things that I am teaching are also part of information security 2 it is not there in the web, but I think you have a link in the module right just go and understand that that will give you lot more.