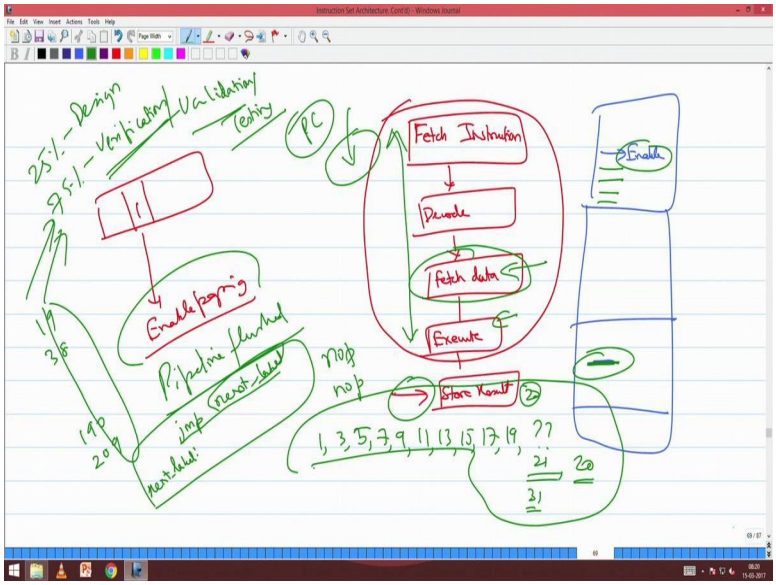
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 32
Cache

So, we continue with whatever we left out yesterday. Yes, I open a thought; we need to take care of lot of things while paging is enabled should I take care of something more, which is architecture specific. While enabling, paging and with that we will end page. So, I will give you a clue pipe lining, with respect to pipe lining what will happen.

Student: (Refer Time: 00:45) we fetch operating system (Refer Time: 00:59).

(Refer Slide Time: 00:58)



Exactly, beautiful see now, that is, that is, So I have, I have fetch instruction stage I have decode, fetch data, execute, store result, either I just take this common thing. Where will the pipelining get too enabled? When will be the paging gets enabled? Which stages will the paging gets enabling?

Student: (Refer Time: 01:27).

See there is a bit in a register, and that bit when it is set to one it will go and enable paging. So, when will paging get enable.

Student: (Refer Time: 01:42).

Decode.

Student: Store.

Store the result, because this pattern has to go and sit in the register. After that only that bit gets enabled and then. So, by the time you do several things can happen in this pipeline. And just for the fun of it. If I am going to say that I will plan in such a way that the instruction that enables paging, the next instruction is exactly in some other page somewhere so the, but I will map it properly, are you get are you able to get this? Suppose I could map it pro properly. So, just to make fun of the architecture, what I can do is, say this is the instruction that enables paging. And there is some other page somewhere. So, the next instruction I wanted to execute would be say here, somewhere here right somewhere here.

So, the moment I enable paging the next instruction to be executed will be fetched from here, should be fetch from here right. Suppose I write a program like this I can write a program like this. Then if a processor is not working correctly according to this, will your processor actually work correctly no because by the time you store this cell, this instruction this 4 of this instructions could even eventually be populating this pipeline right. By that time you realize that this instruction is enabling paging there will be 4 other instructions that follow could have come and populate this pipeline. And they need not necessarily be the instructions that should be executed after it, number 1. Number 2 suppose they are fetching some data, assume it is a low store instruction there may be actually there is instruction assuming even that I am identically mapping the page which enables paging, let these 4 be the instructions they have come here. But it need not necessarily mean that the data fetched by this instruction and this instruction they exactly the same. They can now because of paging it can fetch from some other location, are you able to get this, right?

And in the act of (Refer Time: 04:02) getting fetched fetching this it might also lead to a page fault right. So, all that I have done in terms of without realizing that the forward instruction is enabling paging, all that I have done in the past after we completely under. So, the pipe line has to be flushed while I enable paging right. Pipeline has to be flushed it is not just flushing the pipeline, please understand. I should also go and reload my PC

my program counter, So then after this is over the next we have virtual address is given, and then it is no I have to subtract it by the number of instructions that I have fetched right. And interestingly there could be in jump instruction after this, conditional jump and that would have predicted correctly saying that can and that would have gone and adjusted this PC some other instruction could have also come.

So, So many complications can happen by the time this fellow realize that paging is, are you able to appreciate understand what I am trying to say? So, that is why so, the architecture. So, the way of handling this is the moment I say enable paging or moment I say enable segmentation for example. Both can have the same issues. I can basically now go and say that flush all the pipeline. So, I will make the architecture go and flush the pipeline. One minute, otherwise what we should do? The complier should do something to flush the pipeline, that is what is you know when we enable paging, then just after that there will be one jump next label and next label code, just this statement. So, this is a unconditional jump and the moment I put a jump there are certain architectures which will flush the pipeline right. If the no prediction the moment I see a jump I flush it flush the pipe line. So, while the jump can certainly execute this is PC relative jumping. So, this jump will execute irrespective of whether you have enabled paging or not.

And even if it executes it will just go to the next label, it will not do anything. And after that obviously, you can paging gets anyway in it. So, so this is this is something very interesting. Now the way the complier could have handle this that after enabling paging it could put several nobs. But if I am going to have such a compilation then as my architecture changes compiler also has to become different right. So, so yes as we go on improving the you know performance of the processor, there are lot of things that come as a baggage to keep consistency with the execution of the program.

These are all very, very simple examples which basically tells you that as your processor becomes complex, as you go on increasing your pipeline all these things, what is with that we have to take care, where will we fail right. So, if you if you look at the entire processor design process the 75 percent of the time is spent on what we call as verification, whether after doing a change if all my original programs will work properly or not right. So, the entire design designing at processor only 25 percent is actual design, the remaining 75 percent is actually verification, and this is very, very crucial.

Verification or validation you can call lot of things, or this thing are these are all not really synonymous, but we can use it at least one ok.

So, I am just giving you case studies relating to what the OS or the compiler needs and what we basically land up complicating the architecture, and how careful we need to. And you will agree with me that is a very, very simple obvious fact correct? You didn't strike us right. So, that is why you know you should start your imagination should be imagination level I do not know whether you call it analytic ability or this thing I think imagination ability should be much stronger if you want to be a computer architector. So, the word architect architecture has complier, architecture needs more imagination. So, that is very, very important. So, you should start imagine then only you start when the world thinks in one way you should start thinking in some other way right. This is one very interesting thing 1 3 5 7 9 11 13 15 17 19, what is next?

Student: (Refer Time: 09:31).

Can you tell me, could be anything I can.

Student: 20, 21.

So, can you give me a reasonable argument?

Student: For 21 like.

21, 21 anybody will say, like I will say.

Student: If I take multiples of 19.

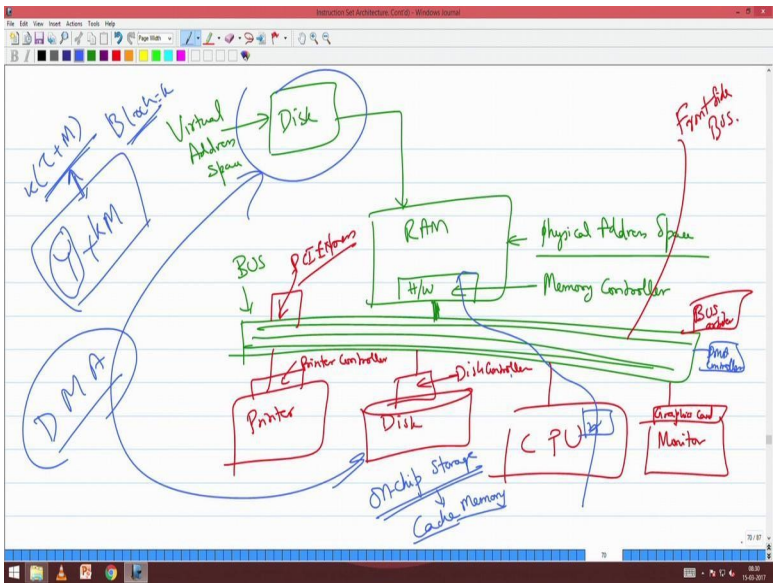
You have statue otherwise?

Student: It may be (Refer Time: 09:55).

So, you know that somebody has thought you. Can you tell me 31 why? Do not know can you tell me 31? Why 31? Because if I look at you know I am constructing all these number it only odd digits and the next odd digit after 19 is 31, correct? I can say 20 that fellow said 20 just take 19 table right. 19 ones a 19, 19 twos a 38, like that you come 19 tens a are 190, 19 elevens a 11 is 201 right. Just do mod 10, you will get 1 3 5 7 9 11 19 and then you will get twenty. So, after this I could have twenty. But you can actually fit

functions to say that I could have anything after this, I could have even minus 5 minus one anything right. This is very interesting. So, this is what we term as something out of the box.

(Refer Slide Time: 11:07)



So, let us go to the next part. So, we had disc where we had virtual address space, and then we had the RAM, where we had the physical address space. Now this is a bus to which this RAM is connected. The RAM is connected using something called a memory controller; this memory controller is basically an hardware. And then here fix the CPU. Here set your printer there will be a printer controller, here fix your disc and you have a disc controller disc controller and this is your hard disc. Here is your graphics card then here fix your monitor right. Then here there are peripheral interface like PCI express. You can put anything your sound card whatever you want to put. This is a basic system right.

Now, when the CPU wants to access RAM, it actually goes through this. This is a common bus, sometimes this is also called as a first beam front side bus right. So, when the CPU wants to access RAM it goes through this bus right. It access RAM and so, whenever it has to go through this bus again it will wait for so, essentially this disc is here. Essentially it again wait is for getting that bus so there is a latency tau, plus then there is a memory access time. Memory access is read or write. So, I will call it as

memory access times, and it there is and it reach so, suppose this tau plus m is the total

time required for reading a reading some 1 byte from the memory. So, tau is the latency. The same thing we have talked when RAM wants to talk to disc, it have to actually when I want to move from memory RAM to disc, I have to wait for the bus and then.

Now, also understand that the very few peripherals talk directly to the CPU. The way by which peripheral actually talks to the memory is through talks to the CPU is through memory. Very few peripheral talks directly with the CPU, in the sense, can you tell me one peripheral which will directly has to talk to the CPU? And in what? Which way does it talk? Keyboard for example, right. Has to talk to the peripheral right. Sorry keyboard has to talk to the CPU why control c, you press it has to stop na, when a program goes on infinite loop you go and press control c it has to stop. So, keyboard talks to the CPU through what we call as the interrupt mechanism.

So, that is why we have several other interrupts right. And sometimes it is a non mask able interrupt I cannot stop that interrupt. So, if I press control c respective of what is happening when a you know great program is running VVIP program is running go and stop right. But many peripherals will not transfer data directly from disc to their from the peripheral to the CPU, it will write into the memory and then it will take into from the right. And one minute when just a minute when there is a data transfer between the peripheral and the RAM the CPU need not be involved right. Why should the CPU, you just keep watching what is going on it will say I do that and it can be basically start doing some processing right.

So, this is called DMA, direct memory access. So, there will be something called a DMA controller in this bus. And when a peripheral when the CPU want some data, it will tell the disc it will tell the disc controller transfer this much amount of data and then it will start doing it is own activity some other activity. And then the disc controller will transfer the data to some location in the RAM. And it will to bypass the CPU right. CPU will just say I need this data.

Disc controller and RAM will talk to each other and they data transfer will be as really I want to write back to the disc I will tell the RAM, I will tell the again the DMA saying this amount of data from RAM has to go to this. And then this as a CPU I do not get involved further and that transfer happens, that is why we call it as disc direct memory access. We will understand more about it when we go into the IO part, it has a last part of

this course right. But as of now this is we need to understand that if the CPU wants to access RAM, like how RAM when it wants to access the disc right. It has to wait it has to request the bus it has wait for some latency, tau time and then it gets control over the bus, and then it goes and access. And so, the total time involved is not just access time, but is some tau plus m. And so, this I will come back to your question, sit down.

The same argument. So, suppose I want to read k bytes from the memory, for example, exercise 6 is a byte addressable memory. So, suppose I want to read k bytes then it becomes k into tau plus m. I access one byte give of the bus again request for the bus then access next byte. Instead of doing that whenever I access one byte I do not just bring that byte alone, I bring a block of data the word block is used here block was used even in the line.

One block was for me one page in the case of paging; here also I bring one block of data from the memory. So, so then what happens is let that block be k bytes then the total time involved now will become tau plus k into m rather than k into k into tau plus m. So, this is now becomes tau plus k into m. So, my I reduced I actually improve the performance by several false because my tau which is reasonably significantly large amount is not getting multiplied by k. And this type of the a transfer is very good for me because, it will be beneficial for me because of what?

Student: (Refer Time: 18:40).

Locality of reference; so I have both spatial and temporal locality, because of locality of reference, this particular block transfer is going to be very, very beneficial. Now So, so what I do is instead when I am accessing RAM, I do not I just bring a block of data I need one byte, but I do not bring just that byte I bring a block of byte and that I need to store somewhere. And that is that is some storage which is on the chips. So, this is So, it is called on cheap storage and that where. So, I am bringing k bytes no where I will store it here. So, I store it in something called the cache memory.

So, the cache memory is a intermediate memory which is very close to the CPU, very totally, very tightly coupled with the CPU in the heart of the CPU right. So, and I can access instructions very, very fast. And what is the role of the cache it will store whatever is fetched from the RAM it will store it in the cache and then So, the subsequent accesses I need not go to the ram, but I could keep accessing it from the cache, are you able to get

this? So, this is one very this is how cache memory has come into place. We will not talk about cache in today's class and probably next week we will talk more about caches doubt.

Student: Sir, you are said last semester that keyboard is also a memory mapped IO?

Yeah, keyboard has been a keyboard is a also a memory mapped IO in the sense that the data that comes from the keyboard, essentially goes to a buffer in the memory and then you keep reading it from that buffer.

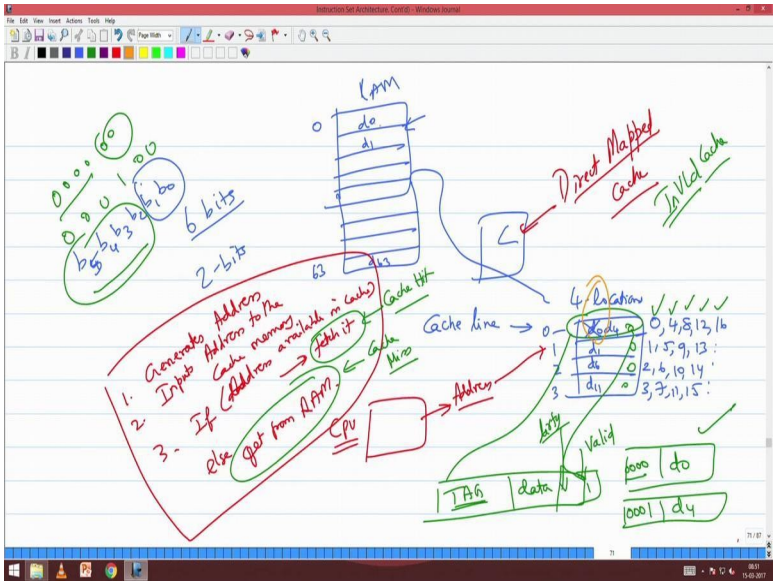
Student: Then like you told that.

But here is only the control signal goes directly into the CPU. So, I am not transferring data, but I am transferring some control. Control c is not a data right. It is not you are typing that is a difference between you are typing logging name was it is control right. So, that is it right. And you did implement keyboard in your last semester using memory mapped IO right. So, you got this now we will go into how cache memory is organized.

So, the difference the difference between virtual memory and cache memory is that, in virtual memory I had an elaborate with the, purpose of the virtual memory was different. And I add an elaborate page translation mechanism. Now I cannot have another cache translation mechanism here. The purpose of the virtual memory, the purpose of paging was to provide you that 4 GB virtual memory that was the main intent. While we look at caching, the purpose of caching is what? To improve speed of processor right, this is one purpose, as we proceed when we go on to you know task switching etcetera, the purpose of cache I can give use different varieties colors of purpose of cache. So, somebody ask you what is purpose of cache you can say I am improving performance that is that is a good answer, but I will give you excellent answers, but as we proceed right.

Without cache your operating system itself will go for die it, will become useless you cannot even get performance in the operating system just not just that I could access some memory fast. But there are many more RAM executions of that we will we will look at in great detail. But a very simple answer which any tom dick and harry can give for this is that, by having a cache I will improve performance. And that answer is sure. Now how is a cache organized? I cannot have a major another page table sitting inside my chip. So, it has to be very straight forward.

(Refer Slide Time: 23:00)



So, let us say I have, let me say that there are 0 to 62 locations. So, how any bits I need? 6 bits I need ok.

Suppose I say there are 64 locations in my main memory. And I ask say 4 locations in my cache right. So, how will I, so what I can do? First I assume I will not look at blocks now, let me just bring in 1 byte. Now block size is just 1 byte, just for our initial understanding. Later we are increase in several bytes. So, I need to have a mapping mechanism. So, what I do is this is 6 bits and this is 2 bits. Let me call this 6 bits as b 4, b 5, b 4, b 3, b 2, b 1, b 0. Now if I get, if I bring say a 6 page, I will store it in location specified by b 1, b 0. So, if I bring 0 from the RAM to the cache, I will store it in 0. If I bring 1 I will store it here, 2 I will store it here, 3 I will store it here. Then 4 I will store in the same location where 0 is 4, 5, 6, 7, 8, 9, 10 11. I will do for 12, 13, 14, 15, 16 dot, dot, dot. So, 0, 4, 8, 12, 16, 20, 24 and all will be stored in the 4th location and the 0th location of the cache. 1, 5, 9, 13 if at all they block they will be stored in this. Then 2, 6, 10, 4, 14 will be stored in the second location and the third location. So, what is the cache doing? It is actually partitioning your entire address space into 4. One partitions has 0, 4, 8, 12, 16 stores these addresses. The other partition stores 1, 5, 9, 13 etcetera. Other partitions stores 2, 6, 10, 14 etcetera, and 3 7 9 are. So, the entire address space of 0 to 63 is now partitioned into 4. So, each one of this I call it as a cache line. So, I have 4 lines here, in each line these are the addresses that could be stored. So, what I do? I bring 0 let me say this is d 0 d 1 to d 63. I so, d 0 is essentially gets store here ok.

Now I bring d 1 then say, I bring d 6 d 6 gets store here let me say I am accessing data in this way, I can d 11 like this way. So now, I bring d 4, then what happens? D 0 is array is 10 d 4 is right. You are getting this? So, so the mapping now is very straightforward, what I should do? I take the 6 bits, I throw I take the last 2 bits and in that location I go and store this 6 bits this the data. So, my mapping from RAM to this is just take the last 2 bits and in that location go and store the data, correct? Are you able to follow? So, this is this is a very direct mapping and instantly it is called, this is called a direct mapped cache right.

Now, what happens here? When initially I was trying to bring, so, what will happen? The CPU when it wants some address it will generate that address, CPU generates the address right. Now with this address it will go to this cache, and see if that address the data corresponding to their address or instruction corresponding to their address is already stored here. If it is stored here then it will just access from the cache, access means read or write. If it is not stored then it will bring from the RAM and then access it right. So, what will the CPU used to generates address, generates the address then inputs address to the cache memory. If data found if, if address stored address available in cache that is a better word, then save fetch it, else get from RAM.

So, this is what we need to implement. If address available in cache fetches it else get from that. Address available if this if this is true then we call it as cache hit, if this is false then we call it as cache miss, we call this as cache hit another we call it as cache miss. It is not there. Now let us quickly get around with 3 or 4 implementation issues in this. What is the number one implementation issue? Now you have done paging what is number one implementation issue. In this generating an address is clean; I will input the address to the cache that is also clean. Because now in this case I will have to go and find out if address is available in the cache. If this addresses the data corresponding data or instruction corresponding to this address is that stored in the cache. If it is there I will fetch it from the cache otherwise I will go to memory. Now which the location in the cache will I look at? Very easy I take the last 2 bits and go there, and what is the problem in looking at that?

Student: (Refer Time: 30:23).

Exactly now very simple thing right. I do not know whether it is corresponding to 0 or 4 or 8 or 12 or 16. So, what I store here is in this I am zooming up this location here, what I store there is the actual data plus what we call as tag bits. What is the tag bit? The first 4 bits, the most significant 4 bits namely b 2, b 3, b 4, b 5 I will store it here correct. So, when I store say when I store d 0, I would have 0, 0, 0 0. While I am storing d 4 I will store 0, 0, 0 1. Because b 4 is 0, 0, 0, 1, 0, 0 so, when I am generating address 0 and suppose this is stored. I will come address 0 is what 0, 0, 0, 0, 0, 0 right. So, I take these 2 0s I come here and if the tag bit matches this, if the first 4 bits what I generated match the tag bit. Then I know that this is 0 that is being stored here right. Are you able to follow? Very, very simple right.

Suppose I great 4. 0, 0, 0, 1, 0, 0, 0, 0 I go to 0, 0 this location. I find the tag bit as 0, 0, 0, 0, but the actual bit I need is 0, 0, 0 1 then there is a miss match. So, the first thing is that I will use So, if my address is k bits, I will use some amount of bits to index into the location and the remaining bits I have to store it as a tag. In this case I had 6 bit address, I use 2 bits to index into the location the remaining 4 bits I store it as tag.

Next implementation, can you tell me what is the next most important issue here?

Student: (Refer Time: 32:33) will need a dirty bit.

Dirty bits next one, before that, dirty bits 7, roof.

Student: (Refer Time: 32:40).

Valid bit present bit right. So, you take some clue from what you have learn right. This is what you should establish. Any location no it will be 0s and 1s. So, any data can be valid right. So, I will have one valid bit here, if it is one; that means, whatever I have stored here is since. Because when I start nothing is there nothing is there means what there will be some 0s and 1s there right. And everything will make sense right. I do not have a insensible binary combination every value I store every 0 and one pattern I store will make some sense. So, I have to so, first I should say that is it a correct? Valid value which is which is written by the CPU or fetch from the memory or not, so I will always have a valid bit.

So, everywhere when you start making this type of storage mechanism, the storage will have some random values to start with. And so there is something called a valid bit. And So, if you look at some of our architectures including Intel there will be specific invalided, invalided, invalidate cache. So, you will g invalidate the entire cache, we will tell you what you make all the valid bit 0 for all these 4 locations. The next thing which I will finish of another 2 minutes that we have stop, we will have again this dirty clean bit, why dirty bits important? Because see I access d 0 to start it.

Let us say I am marking it in some orange color. I started with d 0, now 4 is accessed should I write back d 0 to RAM and get d 4? Or I can just over write d 4. If I write back d 0 and get d 4 overwritten then it becomes 2 memory accesses which is very, very costly for you. If d 0 has not changed that is the value stored in d 0 is here value of d 0 stored here and here are the same. After I fetch am not touching that value. Then I need not go and write it back I can just replace d 4. So, I have a dirty bit which is set to one the moment I go and write something into that location. So, who is maintaining this valid bit dirty bit all these mapping? Everything is done automatically by the hardware.

If you look at what you are done in paging, please finish the paging assignment then you can enjoy cache much more nice right. Please take some time and finish off that assignment is just a jujube simple assignment just do it of right. The point here is that I have in paging the entire translation mechanism everything was done with great support from the operating system. Here operating system will not even know about what is happening inside the cache, everything is completely handle by the hardware. And it has to be handled by the hardware because, I have an instruction bases I need the service right. I cannot have an operating system to service.

So, I could have I cannot have multiple instructions service and instruction. And that instruction which is servicing has to get serviced. So, that is a (Refer Time: 36:09) problem here. So, this has to be handled completely by hardware. So, now, what we have stopped off today, I think by Monday we should keep all these things in mind. The structure of a location in a cache will have tag bits you know, why we need tag bits? We have the data which again you know why we need it, then we have a dirty bit and then we have a valid bit. With this we will stop, Monday we will take forward on caches. But by Monday I request all of you please spend some time and finish of your paging assignment. Then it becomes easy for you to understand. Paging is very complex, cache I

sure is much, much less, less complex. And if you understand paging then this is will be a worth (Refer Time: 36:59).