Computer Organization and Architecture

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

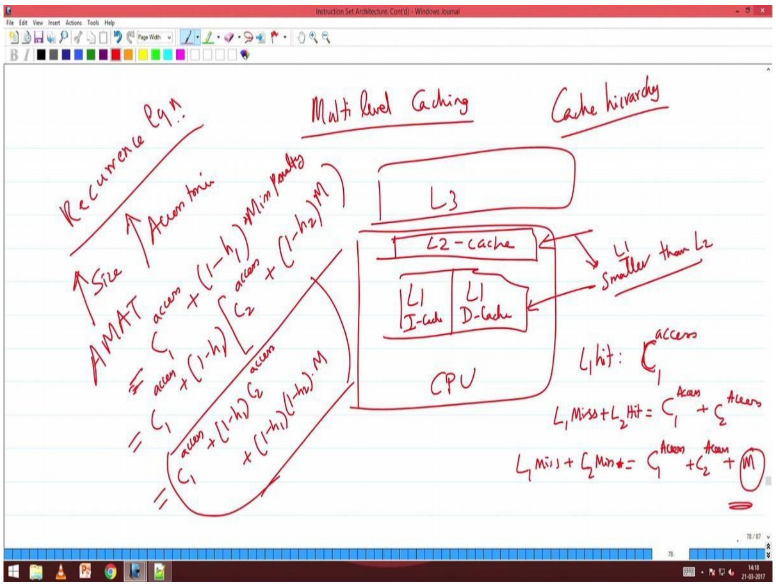
Lecture – 35
Multilevel Caching, Multitasking

Ok

Student: (Refer Time: 00:18)

So, the next step of our cache organization that we will be taking up we will look at multi level caching.

(Refer Slide Time: 00:27)


Instruction Set Architecture, Cont'd) - Windows Journal
Multi level Caching
Cache hierarchy
Recurrence Eqn.
↑Size ↑Access time
L3
L2-cache
L1 I-cache
L1 D-Cache
CPU
L1 smaller than L2
L_1 hit: C_1^{access}
L_1 Miss + L_2 Hit = $C_1^{Access} + C_2^{Access}$
L_1 Miss + L_2 Miss = $C_1^{Access} + C_2^{Access} + M$

$$AMAT = C_1^{access} + (1-h_1) * \text{Miss penalty}$$
$$= C_1^{access} + (1-h_1)\left[C_2^{access} + (1-h_2)M\right]$$
$$= C_1^{access} + (1-h_1)C_2^{access} + (1-h_1)(1-h_2) \cdot M$$

So, here is a CPU, here is a CPU, there will be L 1 split cache. So, L 1 I cache L 1 D-cache sometimes even on chip we will have integrated cache which we call as a L 2 cache right the L 1 cache will be much smaller than the L 2 cache L 1 smaller than L 2. By being small my access time would be fast, so as the size of your cache increases size, so as your size increases your access time also increases. So, we keep the L 1 cache very small fit a fair enough to actually do some of by fair enough to basically take the locality of reference within it. So, I will have that 10 percent instruction that amount of things should be there in the L 1 cache, but there in L 2 cache should be much larger than this L 1 cache, it can store large more than it. So, this is multilevel caching.

Now, what will happen to AMAT, AMAT will be again I will take one instruction. So, take any instruction L 1 hit means it is going to get level one C 1 access time. If L 1 miss means L 1 miss plus L 2 hit essentially it means C 1 access plus C 2 access L 1 miss plus L 2 miss right that is nothing more. So, this will be C 1 access plus C 2 access plus memory. So, we already saw AMAT is equal to C 1 access plus 1 minus hit 1 let h 1 be the hit ratio of level one cache into miss penalty you have seen already this equation.

So, this is nothing but C 1 access plus 1 minus h 1 into what is the miss penalty here, miss penalty is the average access time for C 2 so that will be C 2 access plus 1 minus h 2 hit at 2. And what is that miss penalty, memory access that miss penalty of level two cache will be the memory access. Are you able to follow this? A average memory access time is C 1 access plus 1 minus h 1 times the miss penalty that we have seen earlier. What is the miss penalty that is the time required to access the L 2 cache, and that will be again the same thing we can derive C 2 access plus 1 minus h 2, h 2 is the hit ratio for the L 2 cache into miss penalty. What is that miss penalty this is the memory access time what you see here, you are getting this.

So, when you just expand that it will be C 1 access plus C 2 access minus h 1 into C 2 access sorry actually we can write it a little more neatly, C 1 access plus 1 minus h 1 into C 2 access plus 1 minus h 1 into 1 minus h 2. So, there you just see this is the final equation. So, like this I can keep increasing my I could have a L 3 cache which is much more fatter than this. So, I could have a cache hierarchy, multilevel cache essentially implies a cache hierarchy. And you can use this what sort of equation is this, AMAT. What sort of equation is it? You have studied discreet right first or second semester. So, what sort of equation, this is repeatedly using it right, what it means?

Student: (Refer Time: 06:10).

(Refer Time: 06:11).

Student: Sir, sir.

Yes.

Student: (Refer Time: 06:19).

You see h 1 is the hit ratio for cache L 1 cache.

Student: sir h 1 (Refer Time: 06:25).

C 1 access.

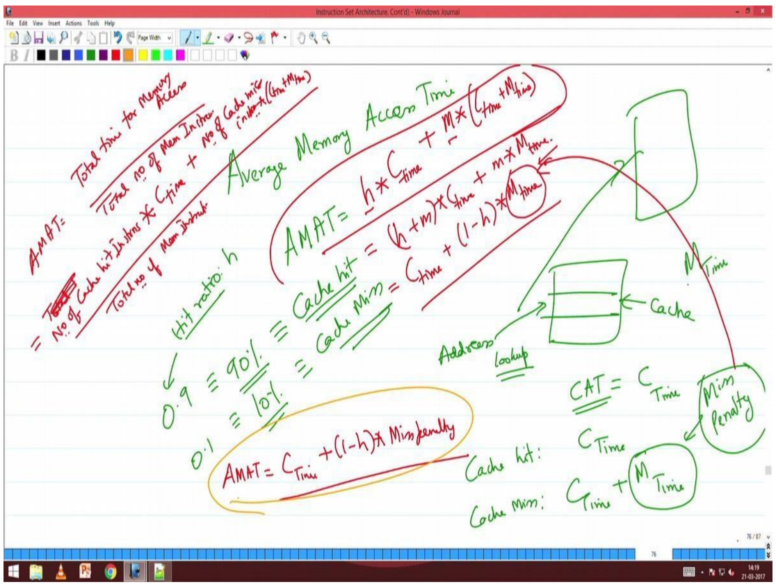
Student: 1 into (Refer Time: 06:31).

Just C 1 access.

Student: (Refer Time: 06:32).

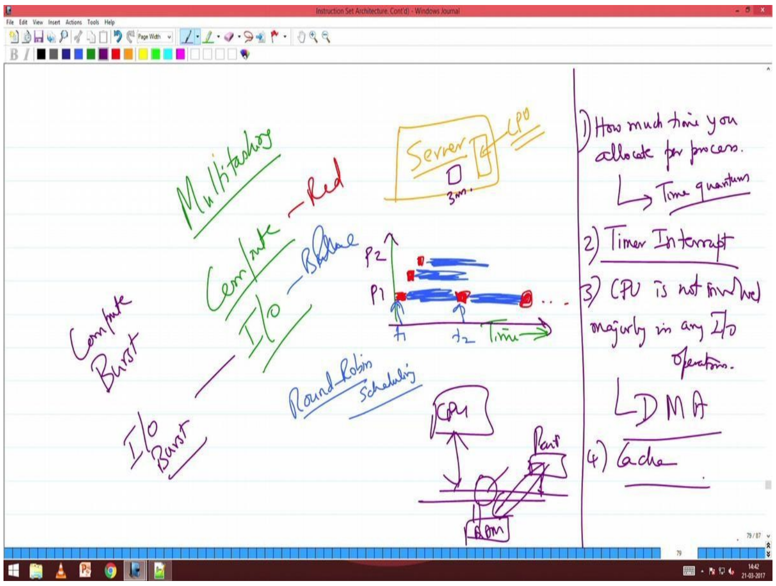
That is what we derived know.

(Refer Slide Time: 06:40)


Instruction Set Architecture, Cont'd) - Windows Journal
AMAT = Total time for Memory Access / Total no of Mem Instrn
= No of Cache hit Instrns * C_time + No of Cache miss instrn * (C_time + M_time) / Total no of Mem Instruct
Average Memory Access Time
AMAT = h * C_time + m * (C_time + M_time)
= (h + m) * C_time + m * M_time
= C_time + (1 - h) * M_time
Hit ratio: h
0.9 ≡ 90% ≡ Cache hit
0.1 ≡ 10% ≡ Cache Miss
AMAT = C_Time + (1 - h) * Miss penalty
Address lookup
Cache
M_Time
CAT = C_Time
Miss Penalty
Cache hit: C_Time
Cache Miss: C_Time + M_Time

Just look at this orange screen so followed. So, like this I can keep deriving it for n level caches. And then today when I look at say 128 cores, I will have on a chip right this is a multi core a heavily populated, then I will have multi levels of this hierarchy. So, we are not gone into multi core, but as of now even a single processor can have a multi level of hierarchy. As my levels go high your access time also increases. With this background let us now go and slowly I will start introducing the fifth assignment of your lab right how will you go. So, let us see.

(Refer Slide Time: 07:27)



So, today I already talked about. You know a server what is the difference between a server and non-server. Server is expected to do what we call as multi tasking even desktops today do, they serve you in some sense. So, what is multi tasking more than one process will execute at the same point of time. Now, let us assume that there will only one CPU what it means that more than one process will execute at the same point of time, there is only one CPU in which can execute only one process at a time. What do you mean by more than on process executing at the same point of time.

Student: (Refer Time: 08:15).

Ah?

Student: (Refer Time: 08:16).

No, no, it is not a illusion. So, now we have to go an technically understand how this going to happen. An execution of a process comprises two things, one is the compute, another.

Student: another is a (Refer Time: 08:33).

Another yeah, yes, right. This is ok. One is the compute, another is I O. If you take any program now there will be some amount of CPU they will use and they will they will start using lot of I O and the IO it sense it takes lot of time compared to the communicate

computation time. So, let us take a typical process, this is the time access and I say let me say red means which is compute and this is red means compute and some blue means IO, sorry right. So, I will start with red a bit of red, then lot of blues, then again a bit of red, then again lot of blue, then a bit of red. So, now this is say let me say that this is some p 1.

Now, we will do p 2. So, this red is called a compute burst, this blue is the IO burst. I do lot of computations and then only I do lot of IO then some curve computation. And by the time you press that key, so which is the most highly respected peripheral in your system the keyboard right because there only you can put control c and stop this term. If you do not have the keyboard the system is completely out of your control. So, keyboard gets a non mask able interrupt. So, It is the slowest of the peripheral is the most important. Now, by the time I press a key, I could execute some 10 million instructions. And sometimes you start searching where a, b, c d, you keep on using mobile and suddenly start using this keyboard you do not know where is a, b c, d right. The time you search and then some 10, 20 million instructions can happen. So, IO is much, much, much, much slower.

So, as a process p 1, I do my compute burst; and while I am doing my IO, process p 2 can do its sorry process p 2 can do its compute burst by that and then it will go to its IO burst. So, it will starting its UO burst. Then process p 3 can do its control burst then it will start doing its IO burst. So, by the time process one finishes its IO activities, several other process is can keep doing that compute. So, what happens in one time quantum when p 1 has done something and it is doing IO, so prospectively I am seeing some IO is happening. So, p 1 is also executing, at that time p 2 will also executing, p 3 will also executing.

So, since I have an inter leaving or whatever alternative sequence of CPU and IO burst and the IO burst is much, much time consuming that the CPU burst. By the time process finishes it is CPU burst the next time it needs the CPU burst by that this in that time it will be you doing IO. By the time needs the again before that time interval from where is it used the CPU that is between this red [FL] and this red [FL], you started at t 1 it releases released the CPU at t 1. By that let me say it want to sit in t 2 between the t 2 to minus t 1 interval several other process is can basically work on the CPU on to it. So, are you getting what I am trying to say yes or no?

So, what happens in practice p 1 is continuously executing, p 2 is also continuously executing. P 1 did something then it will waiting for IO. The moment from a CPU perspective you if there is its not making p 1 wait, it did something for p 1 then it started working on its IO burst. By the time it comeback it was ready. And in that mean while it was doing the IO, this fellow was basically doing some sensible computation with p 2, p 3 etcetera.

So, what are we doing here we are merging the compute and the communication, we are hiding the communication below the compute or hiding the compute below the communication which ever you want to call. And we exploit the property that communication and peripherals involved in communication are much, much, much, much, much slower than the CPU we make this as a (Refer Time: 14:02). Are you getting this? So, with this as a background, let us now start moving to you know next step on how this is going to be feasible. So, can you, please.

Student: Can two (Refer Time: 14:23) operate at the same time.

It can in some (Refer Time: 14:27). If I could have if it is going to be just with memory then it has wait for the burst.

Student: (Refer Time: 14:36).

It has to wait for the burst, but then what happens is we will talk about this. So, this question you deserve. So, now let us. So, now let us list, how are we going to implement this I assume there is only one CPU how I am going to implement this whole thing. It is something, so let me just tell you the first word is round robin scheduling. I think I talked about round robin scheduling sometime in that previous I forgot what I thought you that time, how deep I want let me just revise it.

So, what are we doing here on the round robin scheduling, we are take, so sometime quantum I give you for p 1 then for p 2 then for p 3, p 4, p 5. Again p 1, p 2, p 3, p 4, p 5, so I am going on a merry go around one after and so that is why this is called a round robin. So, I equally distribute the CPU time across multiple process this is robin hood that is how we got robin that robin is took take money from wealthy fellow and give it to each one of them equally, so that is why he call robin and I do it in a round. So, I am called round robin. Round robin scheduling you will learn in much greater in your operating system course, but this is what round robin scheduling means. And round robin basically exploits the property that there is a IO burst following every CPU burst and the IO burst is will take much larger time than the CPU burst.

So, in the same CPU when we want when p 1 is just executed the CPU will have enough time to execute p 2 then p 3 and p 4 and p 5 and again come back. Are you getting this regarding this? So, this is the notion of a round robin scheduling. Now, what are the questions that we need to answer, how do we go and implement this. What are the questions that are coming in your mind we will list all these questions.

Student: (Refer Time: 16:44) order.

That is ok that inky pinky ponky, then after that after that it goes in the same one the (Refer Time: 16:54).

Student: [Laughter].

Does not matter.

Student: How much time we allocate?

How much time we allocated for process, very good question. How much time you allocate per process? The answer will be given to you in the operating system, but I should tell you this is basically called time quantum. And this time quantum is basically decided based on the average CPU burst across all programs, based on that. If I keep on interrupting no then if I keep on I will give you something I stop, stop, stop, then stop stop, stop then nobody will do constructive work I will be sending more time in swapping you and out of it. So, that is what time quantum fixing is very, very important. So, there is lot of discussion that need to be done when you educate you on that time the operating system course will teach you how to fix this time quantum. But I will give you an idea time quantum is actually fixed based on the average CPU burst of that clash of program that you see. And this time quantum is programmable, configurable I can change the quantum. It is not that when I make the chip I may. So, the operating system is responsible for fixing this time quantum.

So, as and when you see move you know different profiles of peoples trying to access, and you see this one profile of processes, I can go and keep very this time quantum question number one. There is much more fundamental question. Let us say we nine of all the first row nine of us or working it. Now, let us say this guy is on the CPU, there is only one CPU, there is only one CPU. Did I teach it in the previous course this one this topic, known, say no anyway. So, there is only one CPU right. Now, after the time quantum let us say 2 milli seconds I give for each, after that time quantum what should happen?

Student: (Refer Time: 19:22).

He should be pulled out and then this fellow has to be scheduled. So, who will pull him out, operating system will pulled out. How can the operating system pull out? Operating system is a software for it to do some action it has to execute on the CPU, there is only one CPU and he is working on that CPU, how will the operating system pull him out. So, there will be a interrupt. So, the most important thing for implementing this type of attain is what I have an timer interrupt that is one thing which hardware has to support.

So, in a computer organization course, we talk about time interrupt in the context of operating system correct. Now, what I do to the timer, I will fix that time quantum say 3 milliseconds huge time run. So, millisecond you can do some big yaga. So, three millisecond I am big I said three. So, when I put that fellow in I means I am the operating system, I am now executing. When I trying to put that fellow in, I set this 3 millisecond and I say start, I push this fellow. So, he starts working on the CPU. Now, the hardware timer will be working, it will be countdown counting and when it reaches zero that means, 3 millisecond is that is been given to it immediately raise an interrupt, external interrupt or it is a internal in the sense it does not come out pin. But it raise an interrupts like how a divide by zero raise an interrupt, how a segmentation overflow raises and interrupt, this fellow also raise an interrupt.

The moment that interrupt comes, what will happen automatically your interrupt service routine will start executing that means, automatically his program will be moved out, and the instruction that will execute now will be the interrupt service routine who is that interrupt service routine operating system. So, that is how in a single CPU when process is executing I raise an interrupt through an external timing hardware timer hardware, and I get control over it. Are you able to follow correct? So, without this timer interrupt, this round robin is not possible, you got this.

Now, what will happen now after I push him after I pulled him out and I am pushing this fellow in, he has to start executing what will that fellow do, his IO will be happen. So, for his IO to happen independent of this fellows execution that means what the CPU should not be involved in IO operation. The third important thing is CPU is not involved let me say majorly in any IO operation if CPU is involved it will only do that IO operation. So, what sort of IO is this we call it as DMA - direct memory access.

What is direct memory access yesterday explained you we bypass the CPU. So, you keep, so when he has moved out he needs to do some IO work his peripheral there will be some peripheral which will be talking to his memory and read and write from the memory. So, all the from the peripheral if I want reach the CPU except for some interrupts the way I can reach it is write into the memory and ask the CPU to read from the memory. And when he is trying to write into the memory he does not involve this CPU that is why it is called direct memory access, because I am not involving the CPU here, I keep directly reading and writing from them. And if DMA is not possible, he cannot do his IO please understand. So, timer interrupt is very important, DMA is very important correct.

If there is no DMA, so then, so now, what happens he was executing timer interrupt came it made me execute operating system I throw that fellow out, and I ask this fellow to start executing, I ask him to start executing. Now, when he is executing that fellow is basically doing his IO; and that IO is happening with the peripheral and memory; that means, what the front side bus this is the CPU this is the peripheral which our fellow wants to access, and this is RAM, CPU is also connected. Now, this peripheral and RAM are talking to each other. So, this bus is not free correct, bus is not free. So, if he starts executing, if his instructions and data are not there in the CPU then it has to wait correct, yes or no, yes to wait.

So, how will I get rid of it, because of caches. In the cache, when I fetched his previous instruction because of principle of locality of reference I would have fetched to more instructions when I fetched his previous data again by principle of locality of preference, I would have fetched more data a closer to it and now based on spatial locality etcetera he will be using that. So, when he is pushed out and he is actually occupying the front side bus, this fellow will have reasonable amount of instructions to be executed in the cache. And reasonable amount of data that that instruction is those things will already be in the cache not by design, but by actually the terminology that we are used that we can afford to have this locality of preference.

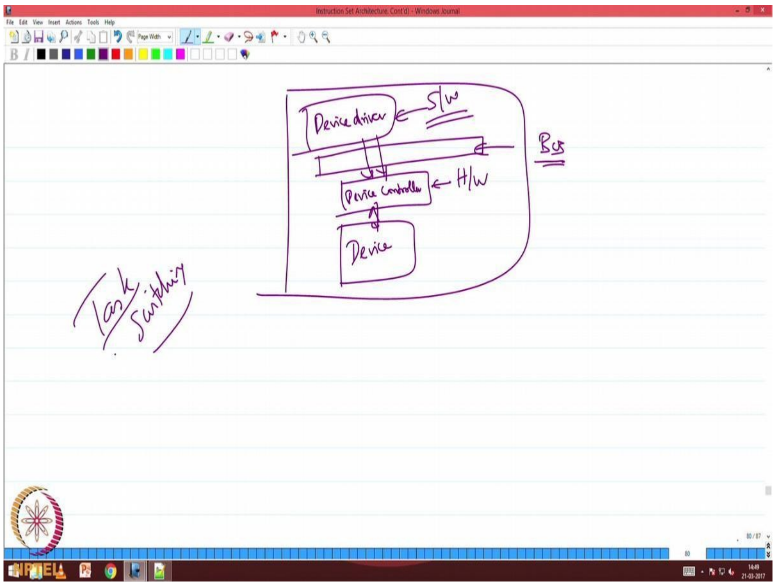
So, if cache is not there then this whole mission is a failure. When the cache is not there then the whole mission is a failure because cache gives you so I put him there the next fellow, and if he does not find any instruction or data then he has to wait for the bus the bus now occupied by a some other process. So, so p 2 cannot compute unless the data and instruction that are necessary for p 2 is already in the cache and that our spatial locality or block caches help, so block caches help. If I do not have block caches then this whole concept of round robin scheduling would be an impossibility.

So, the three questions that you have answered; one thing is there is a need there is a time quantum fixed then one minute, then there is a interrupt that comes in then from the timer. So, for the process is pulled out, another process is put in, the operating system essentially gets the interrupt service routine is the scheduler. When the interrupt comes from the timer it goes to interrupt service routine that interrupt service routine is the scheduler so that fellow takes care of pushing another process. And the CPU is not involve majorly in any IO operation. So, all done through DMA. And the last thing is that I need a block cache if I do not have a block cache the entire would be a failure. With these four points we can we can round robin. So, it will install round robin in a round robin scheduling in an environment involves these which type of discussion.

Student: (Refer Time: 28:04)

Yes, now we talk about IO burst, yes he has already made that question. Now, IO burst is nothing but some IO operation. And when two IO operations happens at the same time then this whole burst gets clocked. So, the way things have been designed.

(Refer Slide Time: 28:34)



Let us take any IO peripheral what is this one. So, let us take any IO, the IO peripheral is split into three parts one is the IO device itself then there is called a device controller. It is a hardware. And then there is something called a device driver which is a software. So, handling a device is in three phase there is a software which is a device driver that is a hardware device controller the device driver programs this device controller to keep accessing the device and give that is back. Are you able to follow? So, this is how thing. Now, what has happened over a period of time the device driver has to execute in the CPU, while then this is the bus. So, device driver what will be the execution of the device driver, device driver will pass the necessary information to the device controller like this hardware, this that that everything and then the device controller can start working.

Now, this device controllers have become so powerful right that they can do lot of activities inside them. For example, you give the so they can do say for example, if I take an graphics card, they can do the complete hidden surface elimination just like this. So, these are all becoming sort of commodity products. So, what happens is that the device controller essentially becomes much, much more powerful, so that when I want to do an IO activity, I do not pass lot of commands and data there. I give you some simple command and then ask the device controller to basically execute it on all the systems that they are coming. So, this is how your IO burst is handle.

So, if I say I want to do lot of IO activity the IO activity will not be perceived on the front side bus rather what you do is you give some instructions to the device controller saying this is one sample data and now extrapolate and make the other data something like that. Then the amount of traffic that will go on this bus will be tremendously reduced. So, by making device controllers as yet another CPU that is what happen right in media came out GPU they actually wanted to make a CPU to start with correct. So, as and when my device controller starts maturing essentially your overhead actually starts decrease. So, this is basically I try to a explain you the round robin scheduling and the notion of you know different things that come under, for example, the cache, the DMA and the hardware timer. If all these things do not exist then doing a round robin is impossible, far enough.

So, one of the things that we do is I pull him out, I push him in right, how it can this happen this is called task switching. I move from one task to another task. Your fifth assignment essentially will be based on this task switching. Any doubts? No, ok.