

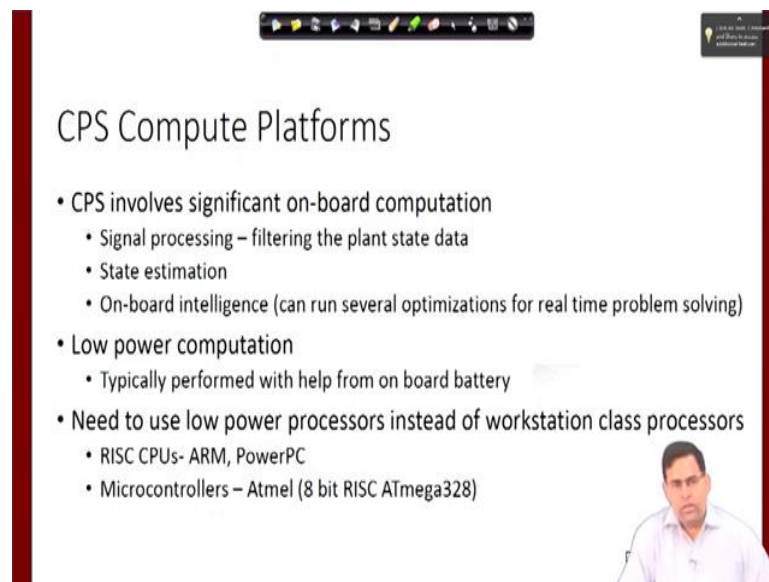
Foundations of Cyber Physical Systems
Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture – 05

CPS : Motivational Examples and Compute Platforms (Continued)

Hi welcome back to this lecture series on Foundations of Cyber Physical Systems, so, we have been giving some introductions to different cyber physical system aspects. right So, ah what we will start with is basically what are the compute platforms that are typically used in cyber physical systems and what are their attributes due to which they find their applications in such systems.

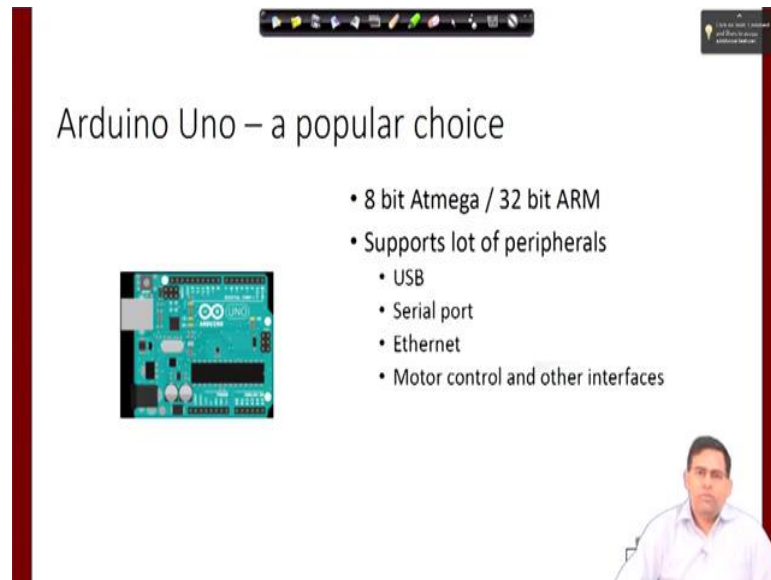
(Refer Slide Time: 00:51)



So, from this point, ah is basically having going to have the properties of real time and embed embedded systems only. Because CPS compute platforms have typical workloads like on-board computation, depending on I mean of tasks like control law ah signal processing tasks, estimation task, etcetera, etcetera. And we need to perform them inside some power budget as well as a time budget.

So, the typical choices here are RISC CPUs which by design are very simple and low power. And also power optimized microcontrollers like Atmel and several other vendors. Ah

(Refer Slide Time: 01:33)



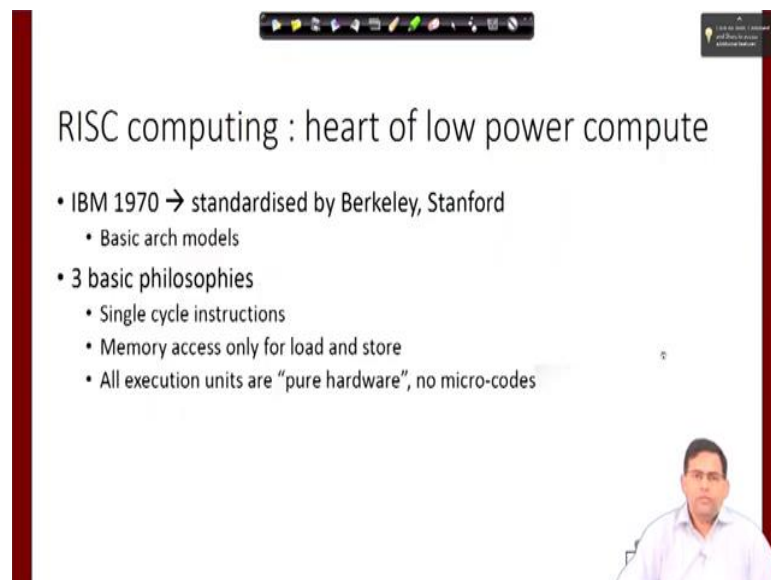
Arduino Uno – a popular choice

- 8 bit Atmega / 32 bit ARM
- Supports lot of peripherals
 - USB
 - Serial port
 - Ethernet
 - Motor control and other interfaces

So, we have just a picture of the Arduino Uno board which is kind of extremely popular for ah DIY traffic experiments here. So, this is a choice if you are doing some, I mean home security home project of CPS, I mean at your home or your at a lab scale. These are very low cost boards available with lot of interfaces to like USB serial port ethernet and you can have a general purpose.

GPI was through which you can connect and create some projects like motor control or other kinds of CPS projects like door, locking autonomous vehicle, etcetera, etcetera.

(Refer Slide Time: 02:12)



RISC computing : heart of low power compute

- IBM 1970 → standardised by Berkeley, Stanford
 - Basic arch models
- 3 basic philosophies
 - Single cycle instructions
 - Memory access only for load and store
 - All execution units are "pure hardware", no micro-codes

So, most of these low power chips that came around ah came with this RISC computing philosophy, ah RISC is reduced instruction set computing. ah That is a term which was standardized in Berkeley between Berkeley between by Berkeley and Stanford. ah Around

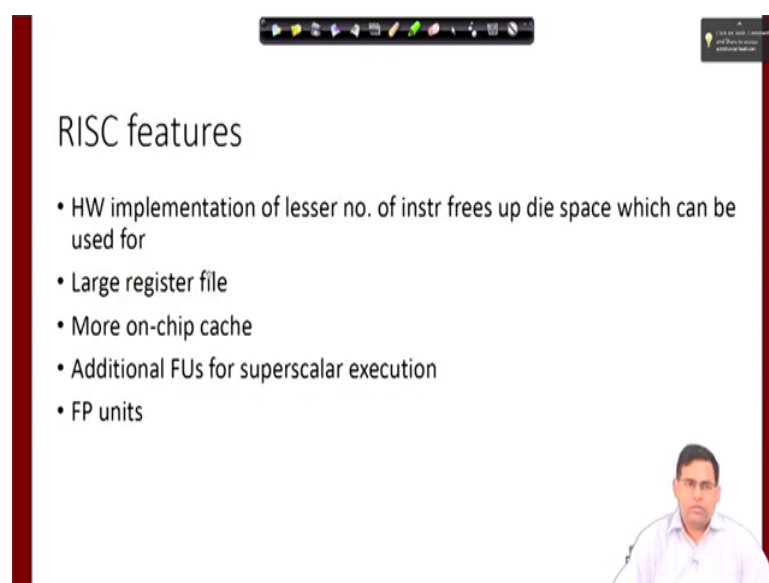
1970s and also it involved inputs from IBM right and the basic architectural model had some philosophies which was that well previous to this.

Whatever computer architectures have evolved, they followed what they called as the complex instruction set computing. That means the instructions could be of any size, they would be quite complex in their encoding and they can be also of any length. So, what these people said, ah the researchers at this universities. ah What they said ah was that ah well ah you should build a CPU where the instructions should always execute in a single clock cycle.

And the instruction encoding will be simple and all the instructions should have a same length to make the instruction set very regular. ok Not every instruction will have the power to access the memory. Every instruction will operate on operands that are in the register file apart from load and stored instructions which will have the power to bring data from memory or stored back to memory.

So, the execution units are like pure hardware. There is no lower level micro code software that is going to execute and this RISC computing standard was adopted by IBM in their 1970 I mean this this series of CPUs. ok So, on this basic philosophies RISC CPUs have been built and the simple nature of the CPUs also made them low power and embedded processor become the first line of things where this computing philosophy was adopted. And nowadays it has slowly moved in the RISC philosophy has slowly moved into the desktop world also in several ways.

(Refer Slide Time: 04:12)



RISC features

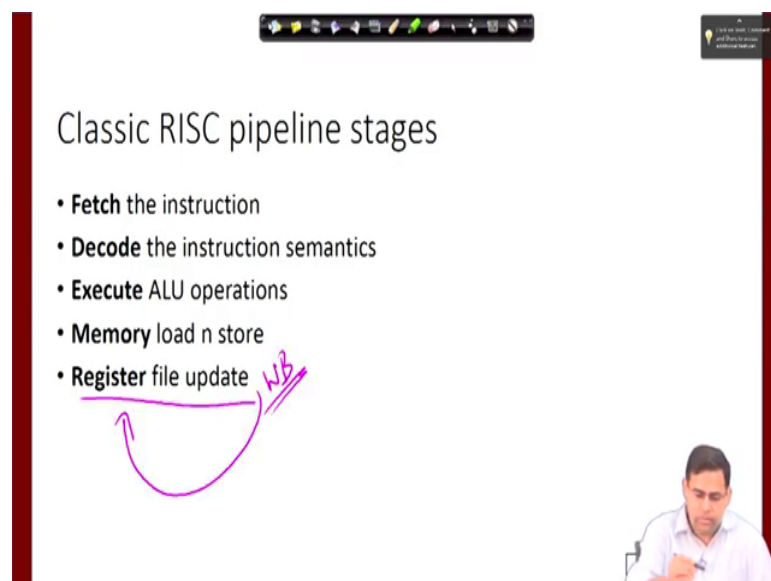
- HW implementation of lesser no. of instr frees up die space which can be used for
- Large register file
- More on-chip cache
- Additional FUs for superscalar execution
- FP units

So, the idea is that since the processor is simple, the hardware implementation is very simple. ah The instruction set is simple the number of instructions is less and execution of each of them will happen in a single clock cycle. So that makes the chip small and low power but the chip will have a large register file because all instructions, apart from load and store are now, going to work on data which is there in the register file.

And the on-chip cache size will also be high again. I require this because most of the instructions ah will work on register file. That means for any data you will, you will need to access the the memory through loads and stores and you will like to have a large cache so that ah you you you do not need to go to the memory that frequently. And also slowly reef processors started becoming super scalar.

That means they can execute my multiple instructions at a time and that would need support for multiple functional units like multiple radars, multiple floating point multipliers and other kind of functional units.

(Refer Slide Time: 05:22)

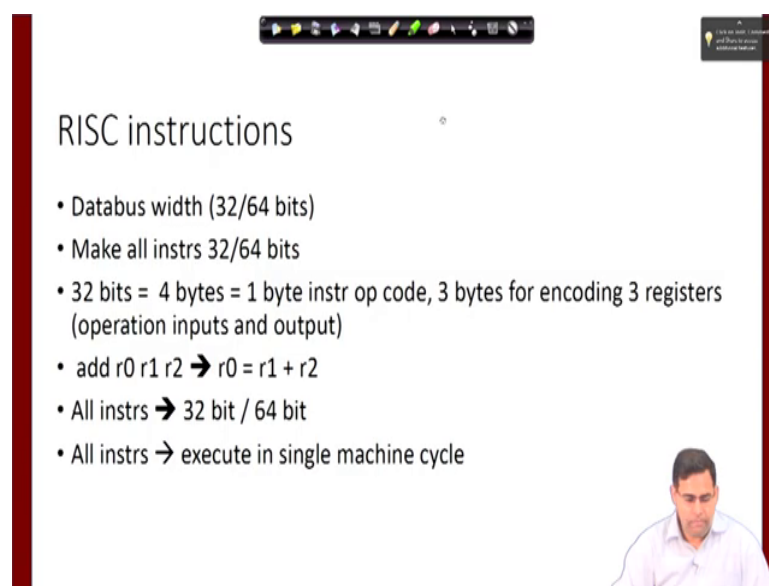


A classic RISC pipeline could have this following stages in it is computer architecture, so, it would have a fetch stage through which the instructions are fetched. Then the instructions go for decoding the decoding phase. So, the fetch stage we will see that whatever the program counter is pointing to which instruction and that would be getting loaded. And then that would be passed to the decode stage of the pipeline, where the meaning of the instruction will be decoded.

And accordingly, the hardware data path of the CPU would be activated. Suppose, ah you want to execute a multiplication then the ALU will take as operand to registers Ah I mean operands from the register file with suitable select lines being turned on and it will do the multiplication. So that would be in the execute stage where the ALU operations will happen. Then, in the memory stage Ah suppose you have a load and or a store instruction.

The load address or the store address will be computed in the execute stage and the actual load and store. That means the access to the memory, ah to load data to the register file or to store data to the memory. That would happen in the memory stage and finally, in the register, file register stage or the writeback stage as more popularly called the register file will up get get updated. So, just a correction this is more popularly known as the writeback stage where this functionality actually happens. That is the register file gets updated.

(Refer Slide Time: 06:59)



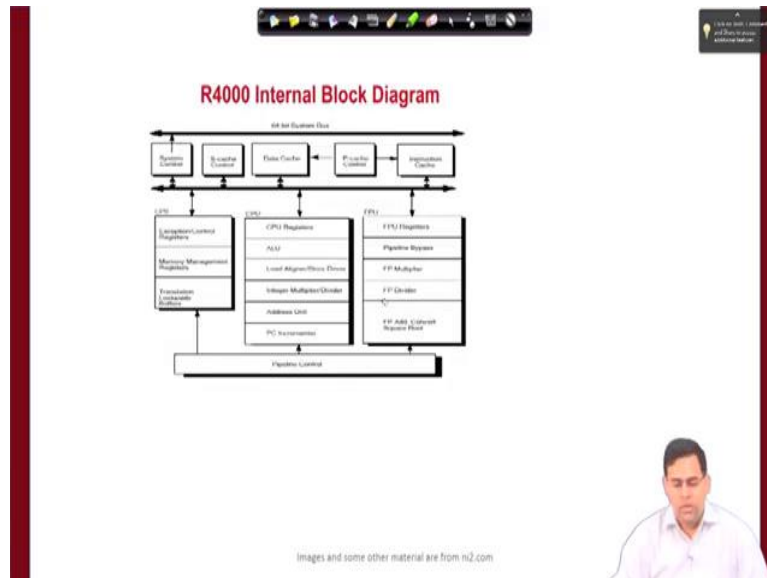
RISC instructions

- Databus width (32/64 bits)
- Make all instrs 32/64 bits
- 32 bits = 4 bytes = 1 byte instr op code, 3 bytes for encoding 3 registers (operation inputs and output)
- add r0 r1 r2 → $r0 = r1 + r2$
- All instrs → 32 bit / 64 bit
- All instrs → execute in single machine cycle

So, this is the classic RISC pipeline of five stages and typical data databus width for this CPUs would be 32 bit or 64 bit. That would mean that all instructions, since this is the 32 bit or 64 bit CPU um that is the the the the instructions should also be of this size. ok So and all the instructions would execute in single machine cycle that that is one of the RISC features. Like we said that the only instruction just to summarize this is the important thing and then all instructions should have same length.

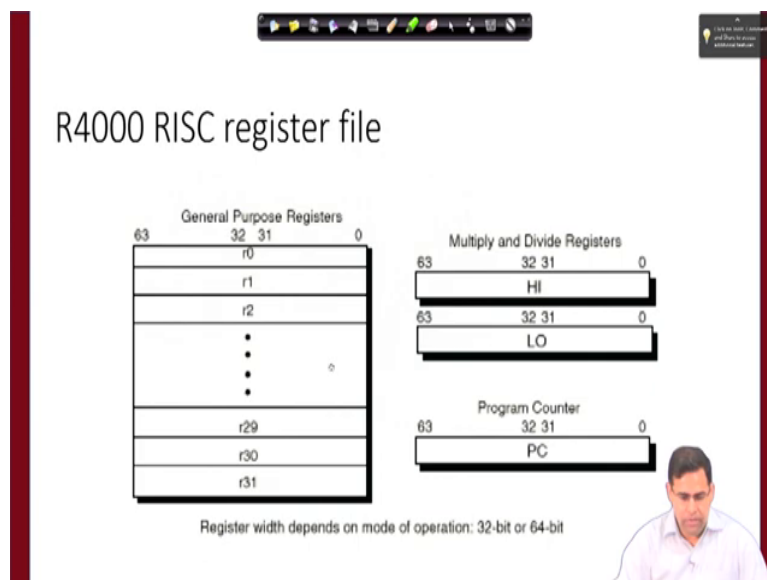
And the only instructions accessing memory is load and store. Typically, you will have a large register file to support register, more number of register based operands and the instruction set should be sufficiently simple and small.

(Refer Slide Time: 07:44)



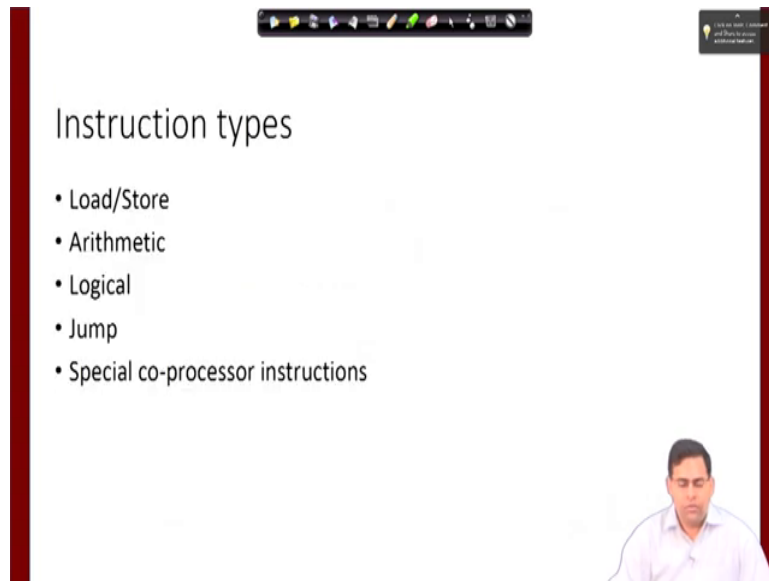
So, this is the block diagram taken from a website mentioned here of one of the earlier RISC processors R400. So, you can see several blocks like CPU registers, floating point registers, data cache and other stuff and control signals for the pipeline.

(Refer Slide Time: 07:54)



And similarly here we are just trying to tell you that well it has a large register file of 32 registers for general purpose usage by the processors.

(Refer Slide Time: 08:13)



Instruction types

- Load/Store
- Arithmetic
- Logical
- Jump
- Special co-processor instructions

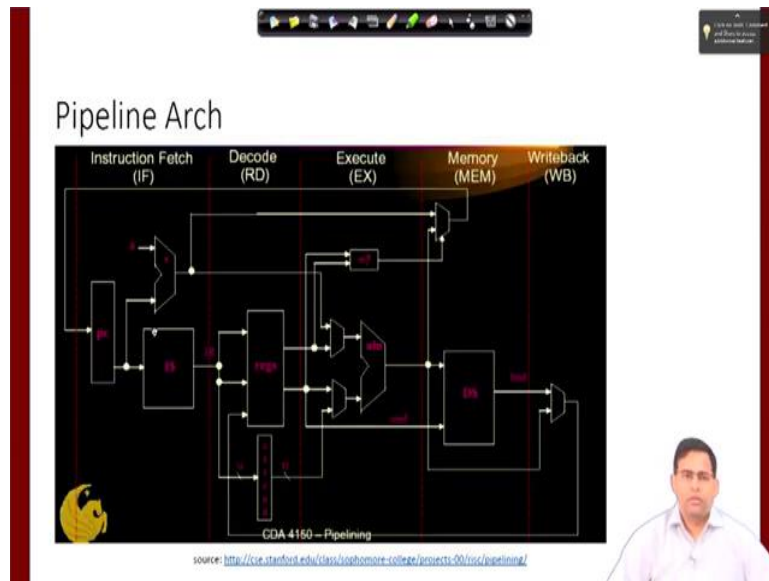
So, the point I am trying to make here is we are not here to teach you computer architecture. We are trying to tell you what are the properties of RISC CPU's which help them ah to compute for an embedded processor in low power. I mean that would because to compute in low power would also mean that the processor should be sufficiently simple and it would have certain kind of specific features.

So, we are just trying to give you an overview of a basic computer architecture at a very high level. Assuming that this course the interdisciplinary one people will be coming from various backgrounds. They may not understand that how I mean? I mean How an embedded processor works in a low power mode and what are it is typical characteristics? So, we are just trying to give an overview here.

So, the different kinds of instructions that the RISC CPU will execute will will roughly be of this type. It is a load, store instruction data storage for the memory or bringing data from the memory to the register file. Working on register file data to do some arithmetic operation or some logical operation or some jump of operation based on conditional leaps or some copyrights or instruction.

Let us say I have a separate FFT chip, where I will upload data from the CPU and ask you to do some FFT computation and give me the result things like that.

(Refer Slide Time: 09:30)



This is a simple example again taken from this resource, where we are trying to show a very simple five stage RISC pipeline, where you can see that the data path has been built with the ALU and register file, ah etcetera. The program counter different MUXs are in place. Right
(Refer Slide Time: 09:51)

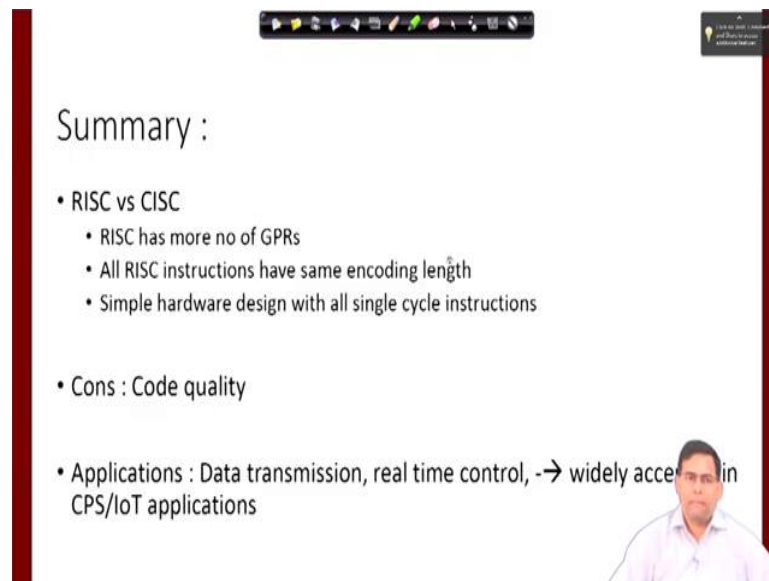
MMU

- Translates virtual addresses to Physical ones
- Some addresses are cached in "Translation lookaside buffer" (TLB)
- TLB is **fully associative** memory

Now, just to give an idea that what are the other things that may be present in the in the in an embedded processor. For example, nowadays, embedded processors are having a large address space. So that would mean ah they require a large address space but since the physical memory is limited the idea of virtual addressing which is prevalent in the desktop domain as also now prevalent in the ah embedded domain.

So that would require some special hardware to do virtual to physical address translation which is the memory management unit. And that has now found its way into embedded processors also. So, in many embedded processors you will see, for example, modern ARM processors, all from your M. They have MMUs, they also have the translation lookaside buffer or the TLB which is like a cache which stores the last frequently done virtual to physical address calculation. So that the MMU's translation hardware can be bypassed.

(Refer Slide Time: 10:50)

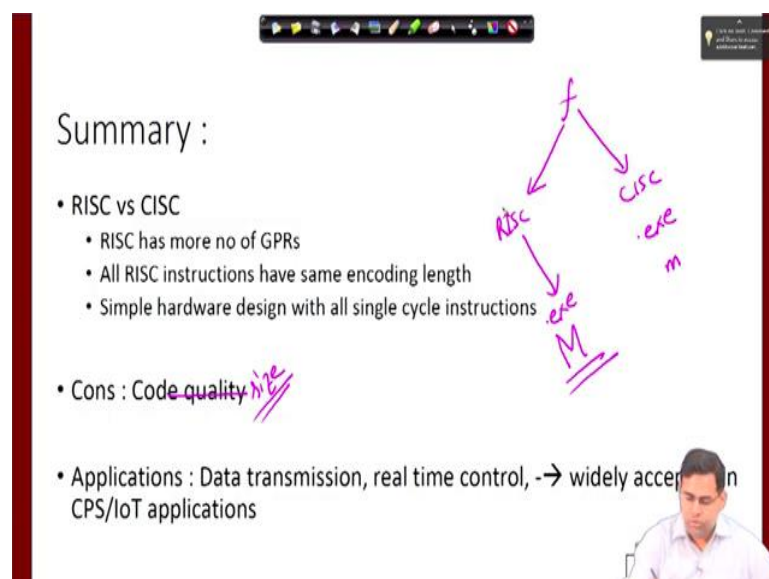


Summary :

- RISC vs CISC
 - RISC has more no of GPRs
 - All RISC instructions have same encoding length
 - Simple hardware design with all single cycle instructions
- Cons : Code quality
- Applications : Data transmission, real time control, -> widely accepted in CPS/IoT applications

So, just in a summary, if I am trying to say the differences between embedded kind of computing and desktop style of computing. The CPUs are simple here, we have more number of general purpose registers instructions encodings are of the same length. And the hardware design is simpler with single cycle instructions and the issue is ah something like this.

(Refer Slide Time: 11:25)



Summary :

- RISC vs CISC
 - RISC has more no of GPRs
 - All RISC instructions have same encoding length
 - Simple hardware design with all single cycle instructions
- Cons : Code quality *✓*
- Applications : Data transmission, real time control, -> widely accepted in CPS/IoT applications

That I would rather not say it is code quality or when because I would rather say the code size. The issue is when I have very simple instructions and the number of instructions is less. Suppose I am implementing a C function in RISC processor and in a CISC processor. The application binary that will be generated here will typically have a very large size. Whereas the one here can have a small size.

The reason is, you have very simple instructions. So, for doing a complex operation, you will need multiple simple instructions. Here you have you may do that complex operation using one such complex instruction which is anyway here right. So that would mean that the code size of the assembly file that you have will be will be larger. right But again, this makes the simple instruction approach makes this CPUs low, powered and simple and they have wide applications in the CPS IOT domain.

(Refer Slide Time: 12:32)

Summary :

- RISC vs CISC
 - RISC has more no of GPRs
 - All RISC instructions have same encoding length
 - Simple hardware design with all single cycle instructions
- Cons : Code quality
- Applications : Data transmission, real time control, -> widely accepted in CPS/IoT applications

Handwritten notes in purple ink:

$$[M_1] \leftarrow [M_1] + [M_2]$$

LD R1, [M1]
LD R2, [M2]
ADD R3, R1, R2
ST R3, [M3]

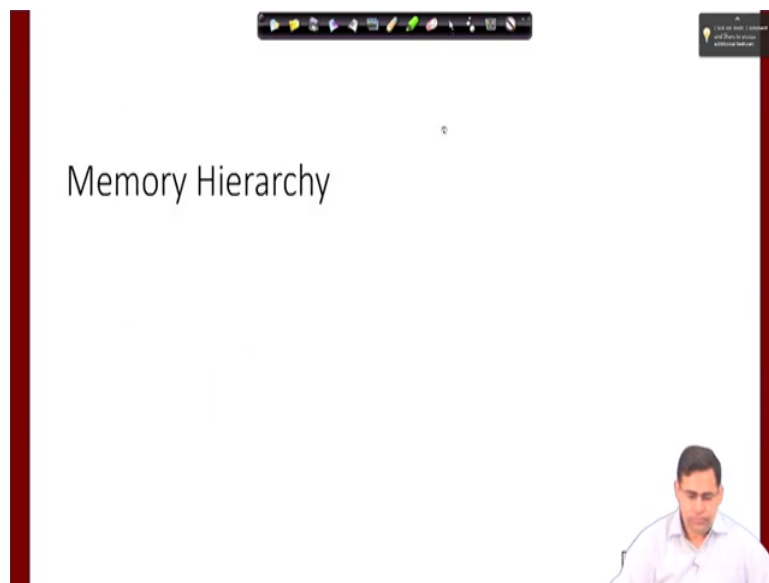
So, when I say simple and complex instructions, let take an example suppose I want something to happen like this that at some memory location M 1, there is some data. And at some memory location, M 2 there is some data. I want to add them and I want that data to go to some memory, location M 3 right. There may be a CISC instruction which does this. So, there is one instruction.

Let us say and see if you, if you execute that it will internally, do all these things through suitable hardware lines active activation. But if I am trying to do this in RISC, I have to first have an load instruction which will bring data from memory M 1 to some register. Another

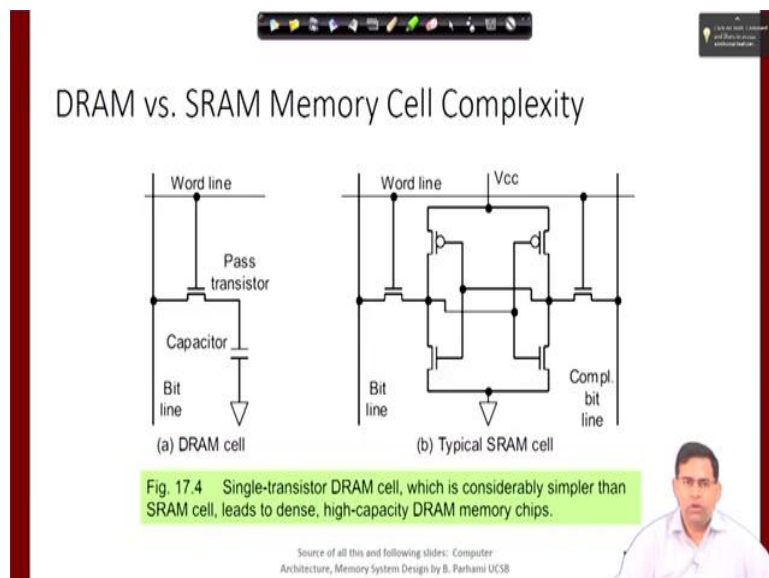
load instruction to bring the data from M 2. One add instruction to load those two data points that have been brought to the register file and store the result in the register file. Ok

So, if I just write it. So, you see, these are the number of RISC instructions. I will need to execute here. So that is an idea that why this code size should actually increase in this case.

(Refer Slide Time: 13:56)



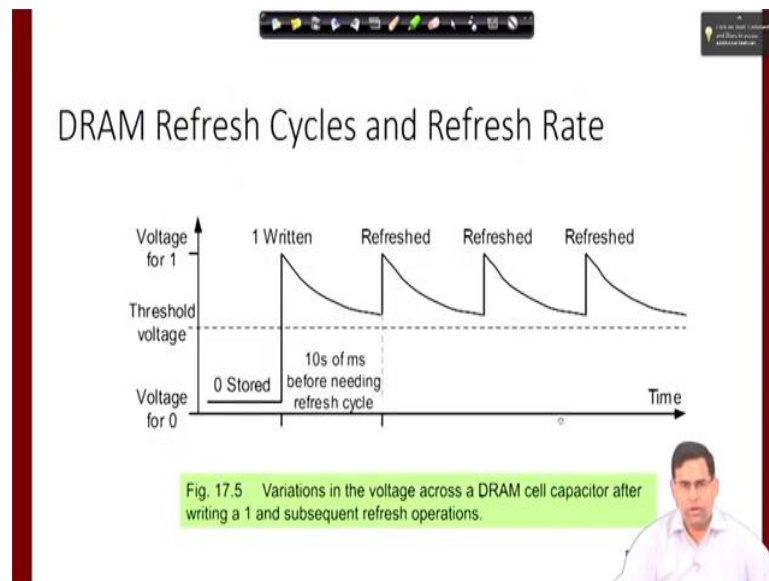
(Refer Slide Time: 13:59)



Now, so, the processor is one unit along with that you need to have the memory. Now, going into the very basics of memory let us understand how a bit is stored. So, in a simple DRAM cell ah the bit is stored by ah pass transistor logic like this. So, whenever a word is written, so, this is high. right So, the this switch will close and this capacitor will store the charge and so that is like one bit being stored.

So, these are DRAM. That means, whenever it is powered off, the capacitor will lose the charge whereas in a static RAM, Ah the SRAM Ah you will, you will have a more complex design like this. So, again we are. We are not getting into the details. There is a there is a VLSI subject. We are just trying to give you an idea what it is?

(Refer Slide Time: 14:44)



Now, when I talk about DRAM, what is important from this to understand is with time this capacitor may lose charge even if, if it is powered on. Suppose the word has been written, the DRAM is on but now no word is getting written. So, the capacitor is by principles of physics is going to lose some charge right I mean nothing is lossless here. So, the DRAMs have to be refreshed in in with some period.

So, suppose Ah I have written a one value here and then I do. I am not changing that value. What will happen is this bit is going, its charge is going down. So, after a period of let us say, 10 seconds ah of milliseconds, I I mean ah some number of 10 of milliseconds or 20 millisecond I will have to do a refreshing here. So that the capacitor charge which was actually, trying to represent a one, should again go to this refreshed higher value. right Now, the the point I am trying to impress upon here is the DRAM.

(Refer Slide Time: 15:51)

Loss of Bandwidth to Refresh Cycles

A 256 Mb DRAM chip is organized as a $32M \times 8$ memory externally and as a $16K \times 16K$ array internally. Rows must be refreshed at least once every 50 ms to forestall data loss; refreshing a row takes 100 ns. What fraction of the total memory bandwidth is lost to refresh cycles?

Figure 2.10

Solution

Refreshing all 16K rows takes $16 \times 1024 \times 100 \text{ ns} = 1.64 \text{ ms}$. Loss of 1.64 ms every 50 ms amounts to $1.64/50 = 3.3\%$ of the total bandwidth.

Nov 2014

Computer Architecture, Memory System Design

When you have it in your system ah it needs to be refreshed and this refreshing ah can lead to some loose, loss of bandwidth. Let us understand so, second example here again from this book. So, what is happening here is, let us say we have a 256 MB DRAM chip. It is organized in this way externally. That means the external interface is like 32 cross 8. ah But internal it is an array 16 K cross 16 K.

That means in each row there are 16K locations. right Now, these rows must be refreshed at least once every 50 milliseconds, so that the data is not lost. If you have some capacitors charged and if you have not replaced them inside this 50 millisecond time, the capacitors are going to discharge out. So, you need to refresh them. Now, let us say refreshing each row requires hundred nanoseconds.

So, the question is well, this DRAM bandwidth how much of this I am going to lose ah through such refresh cycles? So, if I have to refresh all these 16 K rows, what is the time required? The time required would be 16 times 1024 that is 16 K times 100 nanosecond. So that is 1.64 milliseconds. right So, this is the total time that is required ah to ah to do the refreshing of the entire DRAM and this thing I am going to do again and again. Right

So, ah I am, I am losing this time out of every 50 millisecond because I am going to do the refreshing again after after this 50 millisecond cycle after 150 millisecond cycle. So, inside this 50 millisecond cycle for doing the refreshing business, I am losing 1.64 millisecond. So that means whatever is the memory bandwidth. The read write capacity, read write speed of the DRAM out of that this 3.3 percent time goes to the refresh.

Because during the refresh no read or write will happen, any data that is coming for read and write will be kind of stalled in the buffers and they will happen after the refresh. right So, the reason we have this calculation here is. We are trying to say that when a processor is computing over some data there are several ways time can time has to be accounted for. Time will be accounted for when the processor has to bring data from memory.

Time has to be accounted for when the processor has to bring data from the cache and time has to be accounted for when the processor has to execute instructions. So, we have a RISC CPU with simple instructions all of them are single cycles. So, executing those instructions will take their own time depending on the input it will depend at which sequence of instructions are actually, executed.

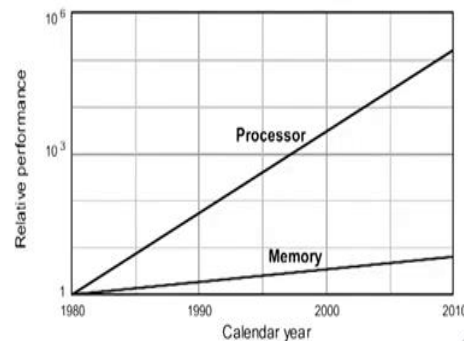
So, at different ah instances the same code have differ, can have different execution times in the CPU. And there will be time for spend for accessing the main memory and there will be time spent for accessing the cache memory. So, when I am trying to find out what is the actual time consumed by this by this by this code or I am trying to figure out in the worst case, what is the time that is going to be consumed in any run of a control program on a CPU.

I have to understand what is the components. What is the computer architecture components in that the CPU? And what are the latencies that can happen due to them? So that that is how all these things are connected? If I am trying to do some timing analysis of a code which is very much required when I am trying to figure out well due to execution of a code inside a control loop what is the delay that is created?

So, there is an linkage. right So, we need to understand how all these basic computer architecture events take up time. Ok

(Refer Slide Time: 19:40)

Hitting the memory wall



Now, the other thing I just mentioned is cache memory and accessing that cache memory. What, how is time taken? So, just to understand that what is the impact of memory? So, this is like the curve which tells you that how processor frequency has increased but how memory performance has not increased that much. Right. So, there is this processor and memory bandwidth gap.

So, no point really in making the processor fast unless we can increase the speed at which we can do read and write from memory. Right

(Refer Slide Time: 20:10)

Bridging the CPU-Memory Speed Gap

Idea: Retrieve more data from memory with each access

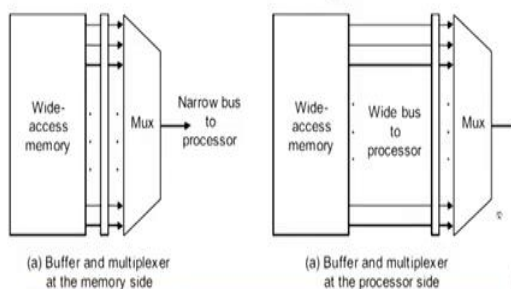


Fig. 17.9 Two ways of using a wide-access memory to bridge the speed gap between the processor and memory.

Nov 2014

Computer Architecture, Memory System Design

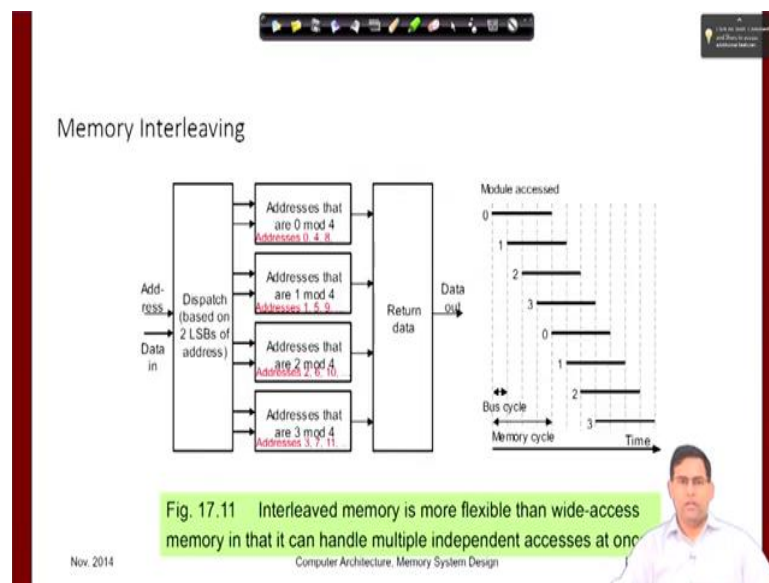
So, this speed gap between the memory and the CPU has to be hidden in some way and there are ways to hide it. Memory can be slow but memory can be accessed through a wide bus. That is the advantage of having a memory. So, although we can have a 32 bit CPU that

means it is going to crunch 32 bit data. ok ah The databus will be 32 bit but the memory the instructions will be 32 bit but the memory can be read in chunks of data which is much, much wider than 32.

Maybe let us say 32 cross 4 something like that. right So, there are two possible techniques here. The memory is wide access and this wide access of data there is there is connected to a buffer and that buffer from that so, multiple different locations of the memory are accessed. They are in the buffer and from this buffer which data is to be forwarded can be selected through a bus. And then there is a so, a MUX and then the output of the MUX is connected through a narrow bus to the processor.

The other option is have a wide access memory have a wide access bus have this have this buffer and the MUX on the processor side.

(Refer Slide Time: 21:24)



So, depends on your choice to hide this gap between the processor, speed and memory speed. The other option is the memory is divided into banks. So, when you are writing consecutive data addresses you write in different banks. So, if you are writing in different physical banks of the memory ah those writes or reads can occur in parallel. right So that is another advantage.

(Refer Slide Time: 21:45)

The Need for a Memory Hierarchy

The widening speed gap between CPU and main memory

Processor operations take of the order of 1 ns
Memory access requires 10s or even 100s of ns

Memory bandwidth limits the instruction execution rate

Each instruction executed involves at least one memory access
Hence, a few to 100s of MIPS is the best that can be achieved
A fast buffer memory can help bridge the CPU-memory gap
The fastest memories are expensive and thus not very large
A second (third?) intermediate cache level is thus often used

Nov. 2014

Computer Architecture, Memory System Design

(Refer Slide Time: 21:47)

Typical Levels in a Hierarchical Memory

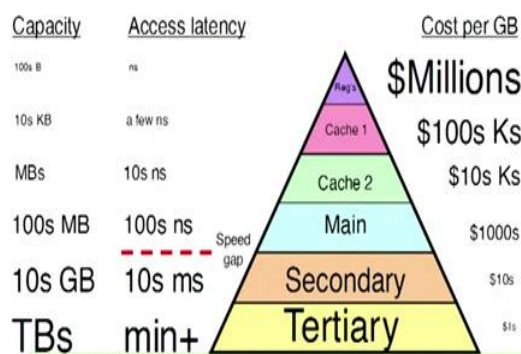


Fig. 17.14 Names and key characteristics of levels in a memory hierarchy

Nov. 2014

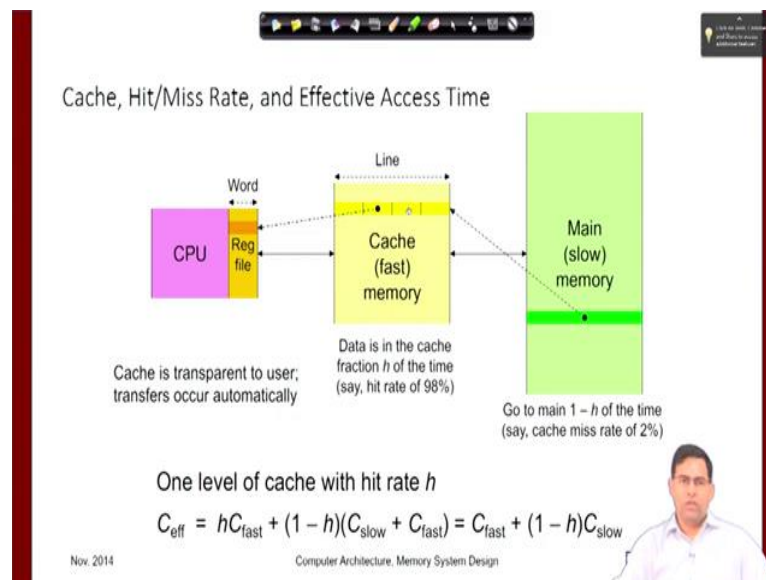
Computer Architecture, Memory System Design

Now, the question of memory hierarchy, so, we will just take very small time here. So, this is the rough idea of how the memory hierarchy is. So, you have main memory. You have the hard disk, the secondary memory you can obtain drives as the tertiary memory. As you can see and the and from the main memory you can take data to the to level 2 cache. From the level 2 cache you can go to level 1 cache. And from there you can take the data to the CPU register. Right

So, these are the things which are inside the CPU chip and they connect with the main memory secondary memory, tertiary memory like that. right Now, the question is if you, if you see, ah if you go higher in the pyramid, the cost per unit of memory drastically increases but what decreases is the access access latency. right These are first memory elements due to two reasons, the technology used and they are nearness to the CPU pipeline. Right

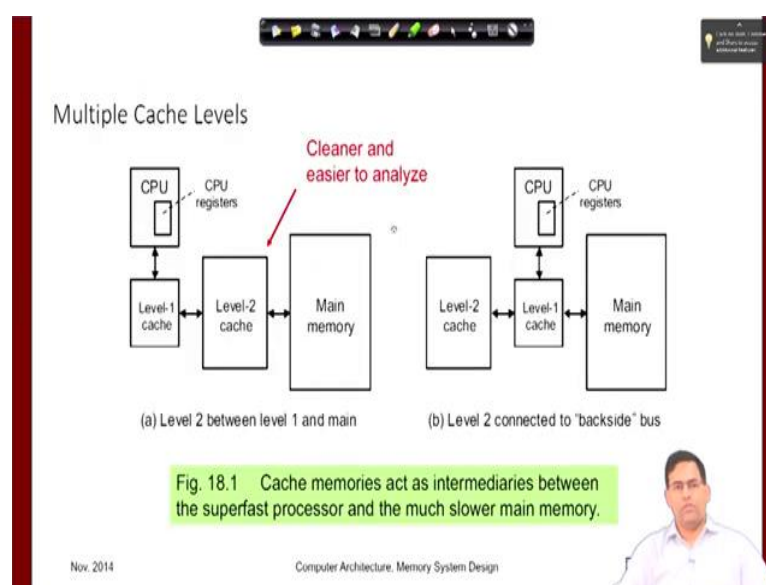
And also points like whether they are on chip or off chip stuff like that. right So, the overall point is, I cannot have a very costly memory segment for doing something in large amount. right It may be a perform, it may be a cost issue, it may be also a feasibility issue.

(Refer Slide Time: 23:00)



So, this this motivates why I need some cache memory to get data ah from the main memory. So, the cache memory helps me to hide the memory access latency because whichever data I will use. I mean once I bring it to the cache which is the faster memory whereas the memory is a slower memory. If if the CPU requires that again I can just quickly get it from the cache.

(Refer Slide Time: 23:25)



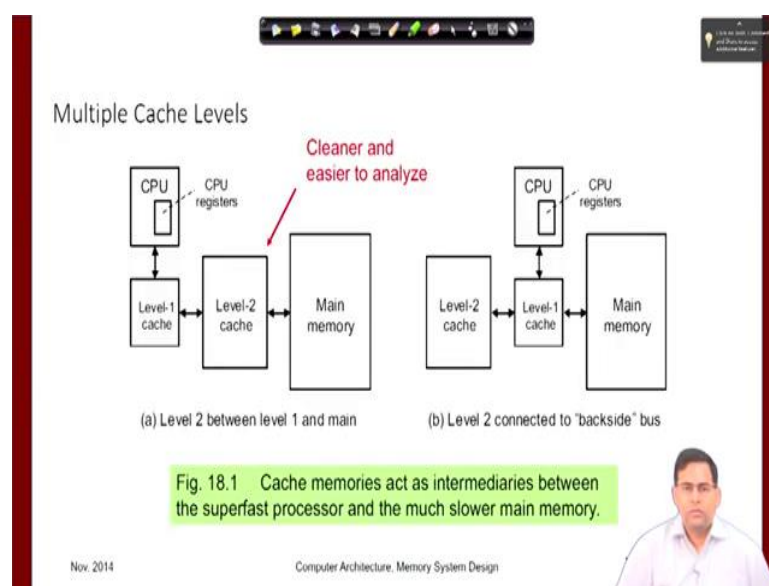
And the idea of caching also follows from basic principles like ah the principle of locality right whenever I am doing and the locality can be both both special and temporal. So, I can when I, when we write programs. Programs are more likely to fetch the same data again and again over time. And also the program is very much likely to fetch nearby locations to whatever has already been accessed in future. Right

So, this principles motivate the usage of cache memory, so, this is the faster memory than the mesh main memory. So, whenever I am executing instructions, I I I can bring data from the main memory store them into the data cache and then I can access them. So that in future I will need the same memory same locations data. I can get it just from the cache instead of going to the main memory.

So, some small math here. So, let us say I have a single level cache. There are no multiple levels of cache and the hit rate of the cache is h . That means if I request data from the cache 100 times out of that each number of times ah the cache will have the data. ah So that would be a cache hit otherwise, the cache should be cache, it would be a cache miss. And I have to console the main memory.

I have to get the data and I have to store it in the cache and then I will also use it in the CPU. ok So, the effective cache latency in this case can be calculated like this, where C_{fast} and C_{slow} represent the latencies in case of a cache miss and the cache hit. Ok

(Refer Slide Time: 24:59)



(Refer Slide Time: 25:03)

Performance of a Two-Level Cache System

A system with L1 and L2 caches has a CPI of 1.2 with no cache miss. There are 1.1 memory accesses on average per instruction. What is the effective CPI with cache misses factored in? What are the effective hit rate and miss penalty overall if L1 and L2 caches are modeled as a single cache?

Level	Local hit rate	Miss penalty
L1	95 %	8 cycles
L2	80 %	60 cycles

Solution

$$C_{\text{eff}} = C_{\text{fast}} + (1 - h_1)[C_{\text{medium}} + (1 - h_2)C_{\text{slow}}]$$

Because C_{fast} is included in the CPI of 1.2, we must account for the rest

$$\text{CPI} = 1.2 + 1.1(1 - 0.95)[8 + (1 - 0.8)60] = 1.2 + 1.1 \times 0.05 \times 20 = 2.3$$

Overall: hit rate 99% (95% + 80% of 5%), miss penalty 60 cycles



Nov 2014

Computer Architecture, Memory System Design

You can also have multiple cache levels, suppose you have a two-level cache and at each level you have different miss penalties. So, you have a level 1 cache with a hit rate. Let us say like this ah level 2 cache with the hit rate like this. And the miss penalty in level 1 is small because then you only go to the level 2. And the miss penalty in level 2 is large because then you have to go to the main memory.

So, you can see those ah access latencies are here as rate cycle and 60 cycles. So then, the effective latency can be found using some formula like this. Right So, here C_{fast} C_{medium} and C_{slow} tell you that what is the ah I mean in case of L1 hit what is the time consumed? In case of L1 miss but L2 hit what is the time consumed? And in case of L1 hit and L2 hit also, what is the time consumed? So, this is for the L2 L2 miss, sorry in this case of L2 miss what is the time consumed?

So, using them I can calculate what is the effective ah latency here? right So, ah typically the you can see that in in if you use a multi level cache system ah it can be also shown with some simple examples that they are. Ah they actually, help you out instead of having a single cache. If you have more, I mean multiple levels of cache ah that means that well you have to access the main memory even less number of times. OK So, you will have and so, from this you can just calculate an overall hit rate.

(Refer Slide Time: 24:59)

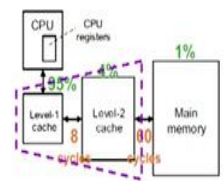
Performance of a Two-Level Cache System

A system with L1 and L2 caches has a CPI of 1.2 with no cache miss. There are 1.1 memory accesses on average per instruction.
 What is the effective CPI with cache misses factored in?
 What are the effective hit rate and miss penalty overall if L1 and L2 caches are modeled as a single cache?

Level	Local hit rate	Miss penalty
L1	95 %	8 cycles
L2	80 %	60 cycles

Solution

Consider 100 instructions, so there are 110 mem access.
 Cycles w/o misses = 120.
 $L1 \text{ miss} = 110 \times 0.05 = 5.5$, cycles = $5.5 \times 8 = 44$
 $L2 \text{ miss} = 5.5 \times 0.2 = 1.1$, cycles = $1.1 \times 60 = 66$
 Total cycles = $120 + 44 + 66 = 120 + 110 = 230$
 Hence, CPI = 2.3
 Overall: hit rate 99% (95% + 80% of 5%), miss penalty 60 cycles



Nov 2014 Computer Architecture, Memory System Design

So, using such cache based systems, you can also hide the memory latency and all these techniques of hiding the memory latency are useful ah for meeting some real time deadline. So that is why it may help to have multi-level cache systems, even in embedded processors. And nowadays they also have such systems in basically CPUs which are typically used in such applications.

So, this is one small example, ah which you can just study that we are trying to figure out What is suppose a system has a CPI of 1.2, ah considering there is no cache miss. Now, if I just consider a real system where there is really cache miss and whenever the cache miss opens. These are the miss penalties. This is bound to increase the cycles consumed by Ah suppose I am executing 100 instructions.

And then the number of cycles consumed in case of no cache means will be something right ah but which is can which is an ideal thing. right But when there are cache misses then the miss penalties will add on and the number of cycles will increase. right So, let us say there are no cache misses the CPI cycles per instruction is 1.2. Now, suppose ah there are 100 instructions and let us say the the number of memory accesses happening on the average per instruction is 1.1.

That means for 100 instructions there are 110 memory accesses. Now, ah the cycles where you do not have miss ah is this 120 cycles. right That means because these are the cycles where there are no cache misses. right Now these are the so, these are the CPUs instruction execution cycles. ok But now you have this 110 memory accesses. right So, for this memory accesses so,

if you look into this problem, what it is saying is here in this CPI calculation, the cache misses are not accounted for. Right

So, they need to be accounted for by considering this memory access space. ok So, you consider this memory access space, ah which are numbered 120. So, from this you can calculate that how many L1 misses you are going to have and how many L2 misses you are going to have. From this number of L1 misses, you can figure out how many cycles will be spent in accessing the L1 cache and how many cycles will be spent in accessing the L2 cache.

So, those will be added to this 120 cycle ideal CPI, where no cache miss was actually, accounted for. right So, these are the extra cycles which are spent by doing the memory system access. right So, what you have is 120 plus ah 110. right So that means your effective CPI becomes 2.3 here, instead of something like 1.2 where the cache misses were not considered. So, overall I can. I can calculate what is my hit rate.

So that would be 95 percent of L1 plus this 80 percent of 5 percent. So, this is my overall hit rate and my miss penalty also I can calculate it would be 60 cycles. You can just follow these effective, latency equations and this miss penalty we have already seen here. right So, with this, we will end our discussion on basic ah CPU architectures and memory systems which are used in these systems. We will end this lecture here. Thank you.