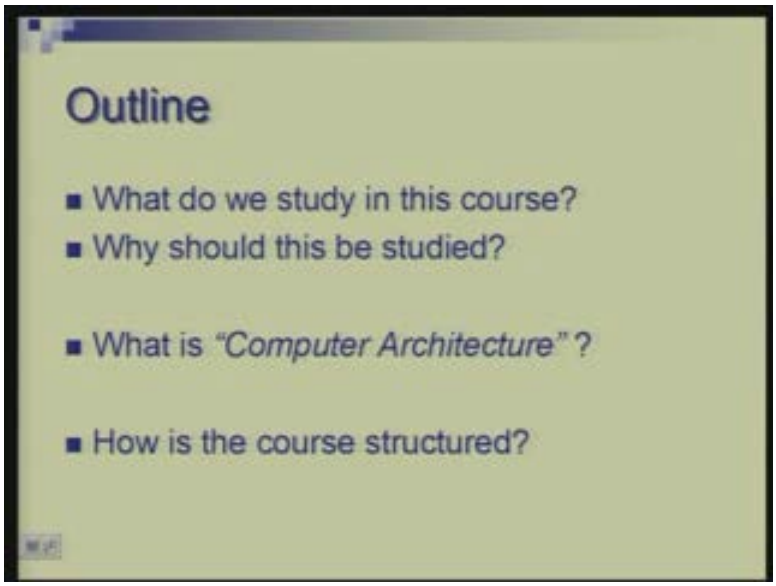
Computer Architecture
Prof. Anshul Kumar
Department of Computer Science and Engineering,
Indian Institute of Technology, Delhi
Lecture - 1
Introduction

Welcome to this course on computer architecture. Today we are passing through a very exciting area; we see computers in all walks of life; they have become integral part of most of our daily activities, you find them in various shapes and sizes and if you look at a typical computer of today it packs in more power, more storage capacity, much more IO capability than a typical computer of 1950s when they began and they occupied a much bigger hall than this where we are sitting today. But although this is the most common form of computer you see today.

There are many applications for which this is insufficient such as weather prediction, nuclear simulation, for astronomical calculation and bio informatics you need much more computing power but on the other hand, they are very tiny and small computers which you do not actually see as computers they are hidden in your home appliances, cameras, mobile phones even in the projector which we are using today and even the remote to control this projector.

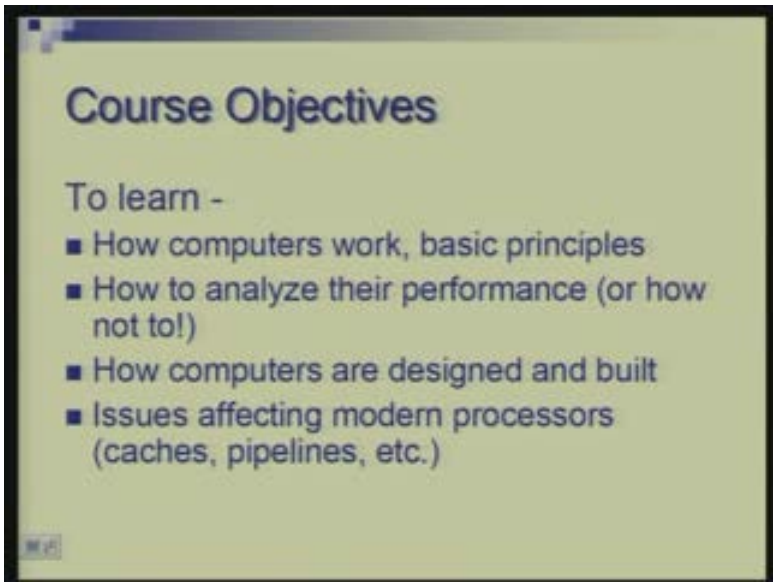
So our question is that going through this entire variety of computers entire diversity in space and time what is common and what is different. So this would be one of the things we are going to learn in this course that, what is the common principle which binds all these different forms of computers which you see today and also which you do not see today.

(Refer Slide Time: 02:39)



So, to outline what we are going to do today firstly I will introduce you to what is the objective of this course; what is it I expect you to learn, why should we study this course, what is the motivation for studying this I will define the term computer architecture and also finally talk about how this course is structured on the whole.

(Refer Slide Time: 03:06)

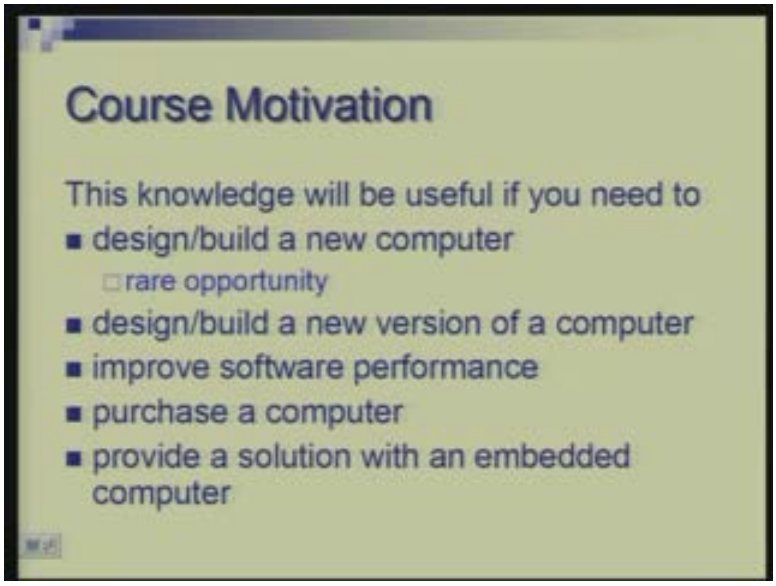


To begin with the course objectives: what we propose to learn is how computers work; what is the basic principle and we will find that this basic principle is common across all computers small and large old or new. It is not sufficient just to learn how computers work we also have to worry about how efficiently they work so performance is an important issue. We will try to understand what we mean by performance and also look at some ways where we should not look at performance people have developed wrong notion which are misleading. So, apart from performance there are other factors which are important such as power consumption and the chip area or the cost so our focus will not be so much in to those we will look at computer architecture more from point of your performance.

After having looked at the basic principles as to how a computer works we will try to see how these could be designed or built. So that is we go deeper into the design and construction aspects of computers and we will look at various sophisticated techniques which are important in today's architectures they are responsible for giving them the kind of performance we see, for example, how memory structures into caches and virtual memory and so on and how a operation could be pipelined and how instructions could be put in parallel so some of these modern concepts would be introduced. Of course the scope of this course is not to go very deep into these because there are more advanced courses probably you would do at a later stage where these things will be covered in more detail but you will get a glimpse of what are the basic features of most modern computers.

Why are we trying to learn all these?
Question is; are you going to build a computer? Are you going to design a new processor or a new computer? The answer is unlikely. The opportunity for designing a new processor or new computer is very very rare and you may or may not get it, it is very less likely that you will get that opportunity.

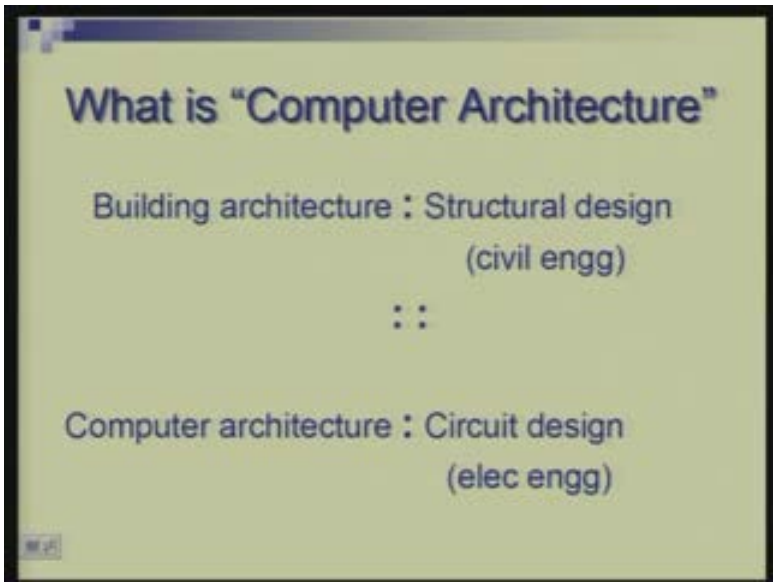
(Refer Slide Time: 05:30)



What might happen more likely is that you will probably design an improved version or a new version of an existing system because the number of new architectures or new processors getting designed is very very small today. On the other hand, many of you would go into software stream where you will be developing software and the question there would be how to get the best performance of the software so for that you need to have good understanding of the architecture and the hardware. So even though you may go in a software stream, understanding of architecture, how it influences the performance, what is the relationship between hardware and software that is going to be very very important.

Also, quite likely is that you might be a manager or an administrator where your responsibility may be to buy a set of computers. So once again understanding of the system is important and this knowledge of basic principles and the relationship with basic performance would be important. There is another exciting area of embedded computers where computer is part of overall system it is embedded somewhere deep in the system and that throws open many more design opportunities. So it is quite possible you might end up in designing a new application where computer is an embedded component.

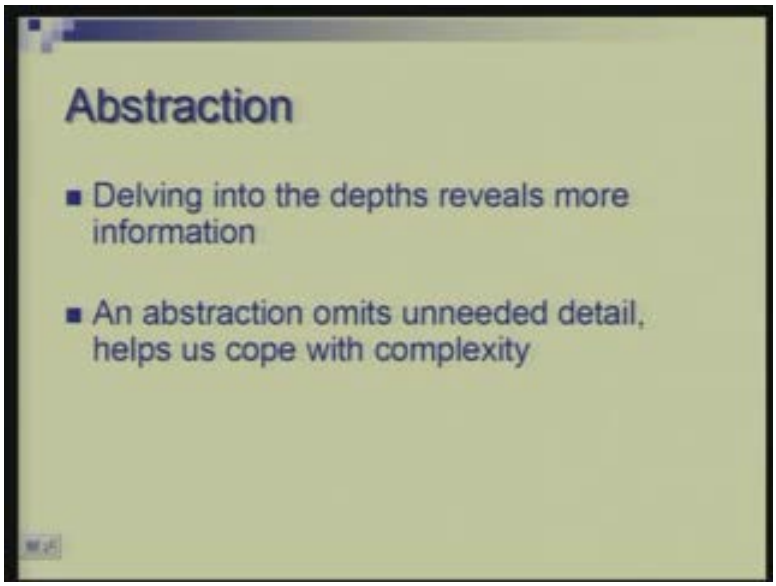
(Refer Slide Time: 07:27)



Let me now define the term computer architecture.
A simplest way of giving an idea what computer architecture is to relate it to architectural buildings. Building architecture is basically a plan of what different parts of building are to be used for it is an overall layout where you decide functionality of different components of building. For example, if it is a residential building you will say that this is the living area, this is the dining area, this is kitchen and this is balcony and so on. So you make a plan of what is the function of different parts of the building. On the other hand, a civil engineer would look at this plan, look at this architecture of the building and do structural design which means he or she has to worry about how this building is going to stand it should not fall, it should be durable he has to take care of stresses and strains, durability, finish and so on. So the relationship you see between what an architect does for the building what a civil engineer does for building is somewhat similar to what a computer architect does for a computer and what a logic designer or a circuit designer does.

Hence, architecture is once again a plan of overall functionality of the computer and what are the basic operation it can perform, how they can be sequenced and so on so that is what architecture is about whereas to realize this functionality, realize this plan you need a circuit designer who has to put basic components together transistors, registers, capacitors and so on to make all this happen. Thus, our focus would be on architecture but we will move little bit towards the design aspect; not go all the way to circuit but will have some idea about how the computers are designed and built.

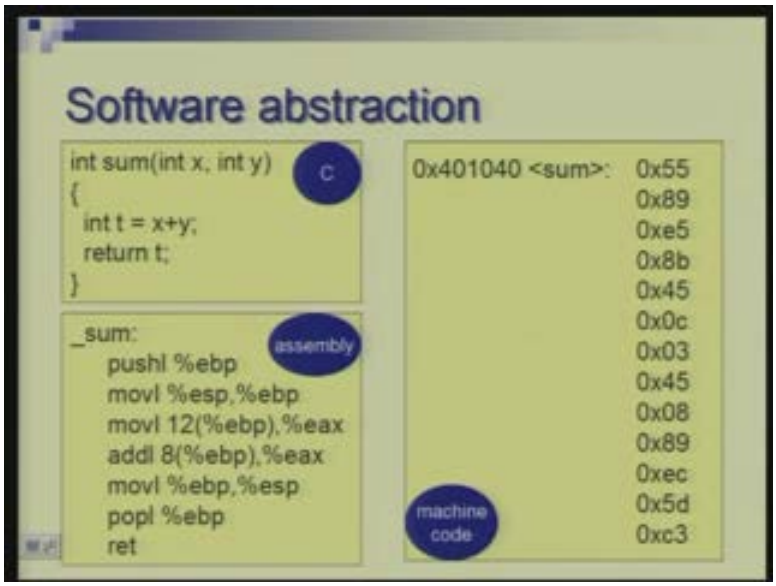
(Refer Slide Time: 10:03)



Now immediately it brings up the issue of various abstractions. You find abstractions in all disciplines of engineering and science but in computers particularly when you talk of architecture you find that there are layers and layers of abstraction. What abstraction means is that you leave out details which are less relevant and focus on matters and issues which are important at that particular time of some particular consideration.

As you go into depth you get more and more information as you have more and more abstraction by information is reduced. So when you look at the relationship between architecture and circuit essentially they are separated by several layers of abstraction. Similarly the architecture and the software are separated by several layers of abstractions. The abstractions help us in coping with the complexity because if you try to look at computer at the level of basic components from circuit point of view there are millions and millions of basic components and you if you start defining your computer in terms of these components it is extremely complex and mind boggling.

(Refer Slide Time: 11:35)



Let us look at software abstraction which you are quite familiar already. You all have done programming and typically you use C, C plus plus, java. Given a problem you can write a program which is essentially a solution for that problem. For example, this is a trivial function (Refer Slide Time: 11:49) which simply adds two numbers and returns the result. So as a high level language programmer in C you will just write these few lines and that is your solution but a computer does not understand that directly, this solution has to be translated into a language which computer can understand more easily. So a compiler will take this program in C and produce an equivalent program in assembly language so this is a process which is automated by a compiler and as you would notice that this simple computation is broken down into number of steps each step is an instruction.

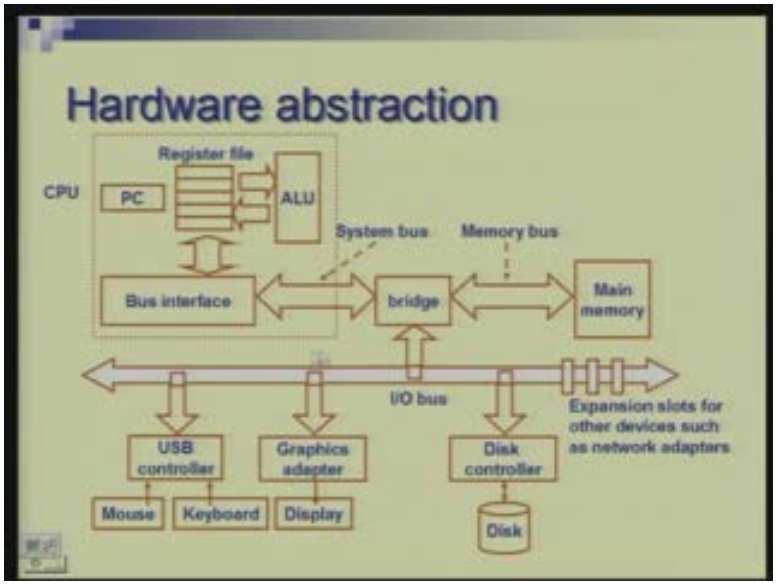
I am not going to go into details of what the instructions are. Basically what you see as addition here is this add instruction. There are some move instructions which you see above this add and below which are essentially preparing the data before addition can be done, they are putting the data in the right place from where the circuitry inside processor can perform addition and the first and the last statement here (Refer Slide Time: 13:13) are essentially required to actually link this function to a main program where this could be used.

Once again this is not the form which is understood by the computer. this has to be further translated into 1s and 0s so that is the job of a program called assembler; assembler will take these, the task here is much simpler; each step here or each instruction here can be translated into a number or a sequence of numbers which is a very straightforward process and what you get is called a machine language program or machine code. So each step here is essentially a number, the number notation I have used here is hexadecimal this 0 x indicates that the number is a hexadecimal number which is nothing but number to the base 16. So, two hexadecimal digits would mean that we are

talking of an 8-bit number or bytes. So essentially it is a sequence of bytes which represent the same program and in turn it represents this program which is originally written by a programmer.

So although I have written not in terms of 1s and 0s but hexadecimal notation has a very that corresponds with binary notation inside a computer these would be binary numbers and interestingly whether it is a data or program, data or instructions the numbers do not distinguish from each other.

(Refer Slide Time: 15:13)

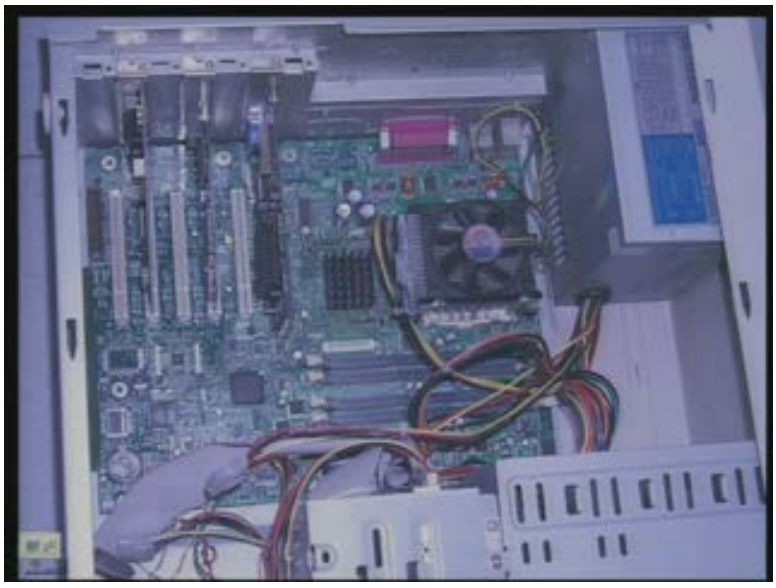


This was one side of the picture what you see has a high level language program; you normally deal with that; how it is related to something which a hardware understands. The hardware itself again has several layers of abstraction. One major module there is CPU or the processor; CPU is the central processing unit, you have memory, you have input output devices and their controllers. For example, here you are using mouse, keyboard, display, disk drive they are connected to the system through a bus and some controllers (Refer Slide Time: 16:04). The processor is connected to memory again through some buses. This is a system bus, this a memory bus, there is a bridge connecting these.

Now, if you go further to this box which I have labeled CPU you will find that there are it is further sub divided. So you have the program counter, register file, ALU and bus interface. Program counter keeps track of current instruction being executed, register file stores operand which are currently in use, ALU is the one which performs various operations such as arithmetical, logical, relational, comparison operation and bus interface is the one which connects these modules to the outside world namely the memory and IO.

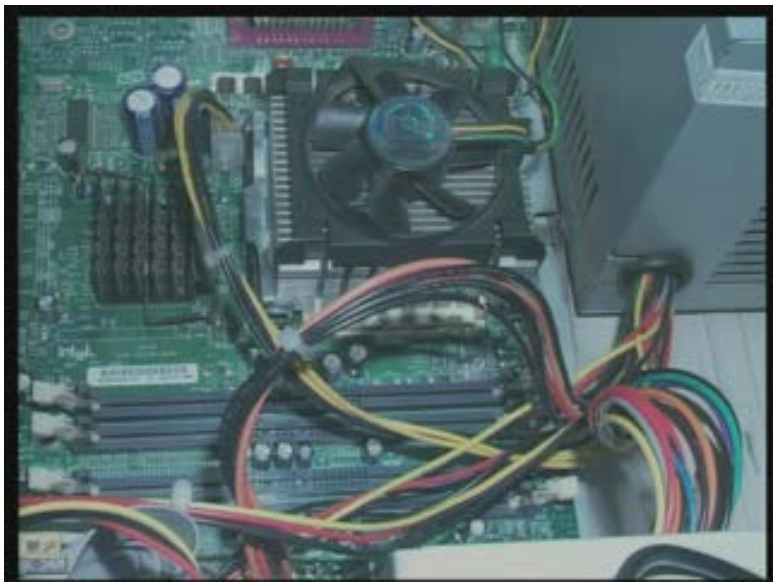
One could go further into this. For example, if you take ALU it is composed of several gates of different types; it may have AND gates, OR gates, exclusive OR gates, NOT gate and so on large number of those; these gates again would be consisting of basic devices, transistors, registers and so on. So, as you see that hardware level also you have subsystems within that you have components you go down further eventually you reach a level of transistors or basic devices which can be fabricated.

(Refer Slide Time: 17:52)



I probably would have shown these pictures to you earlier. Physically this is how you would see if you peep into a PC. What you are seeing is a motherboard; it is a complex electronic assembly where you have number of components mounted on this. On this side you have power supply; on this side on the bottom side you have some peripherals namely a floppy disk drive, hard disk drive, CDROM drive and so on. What you see here (Refer Slide Time: 18:39) is a fan actually what is under this is you see parallel bars these are fins of heat sink and under that there is a CPU or the heart of the whole system; the memory is here on the side and here you have some input output cards.

(Refer Slide Time: 19:05)

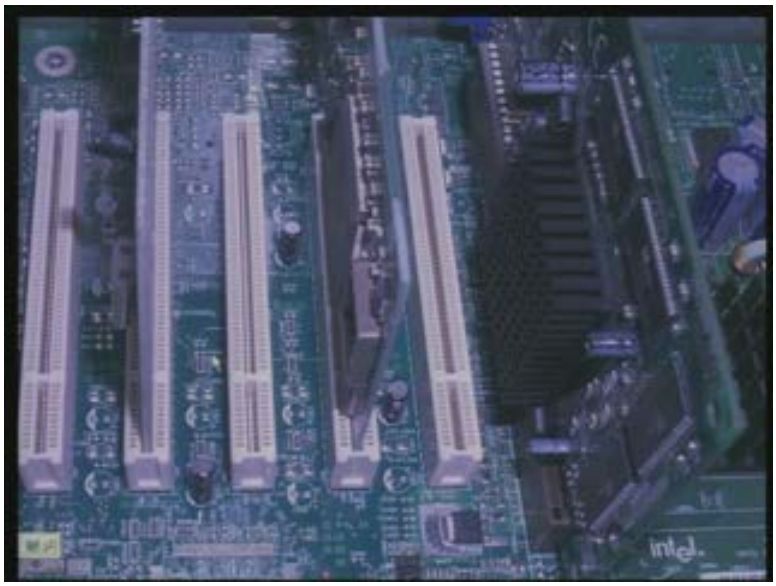


We will go further into this. You can see the processor area more closely zoomed in further.

(Refer Slide Time: 19:11)

What you are seeing here at the bottom…… some part is visible; this is a part of the processor.

(Refer Slide Time: 19:27)



These are slots where you can plug in input output controllers, these are PCI slots, PCI is the name of the bus through which various cards are connected, various controllers are connected to the computer.

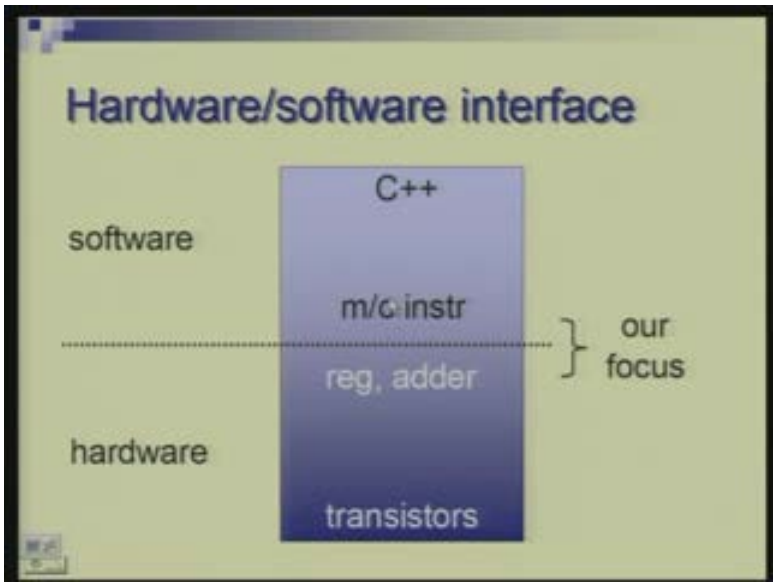
(Refer Slide Time: 19:45)

This is a memory module, this is in a package form, this is open another one, these are individual memory chips and all these put together form a memory module and several of these form the complete memory system.

(Refer Slide Time: 20:08)

This is a Seagate hard disk drive shown from two sides. This part is the disk controller this is just a casing; you are not actually seeing the medium on which the recording is done that is inside. This is the controller circuit you can see closely (Refer Slide Time: 20:25).

(Refer Slide Time: 20:34)

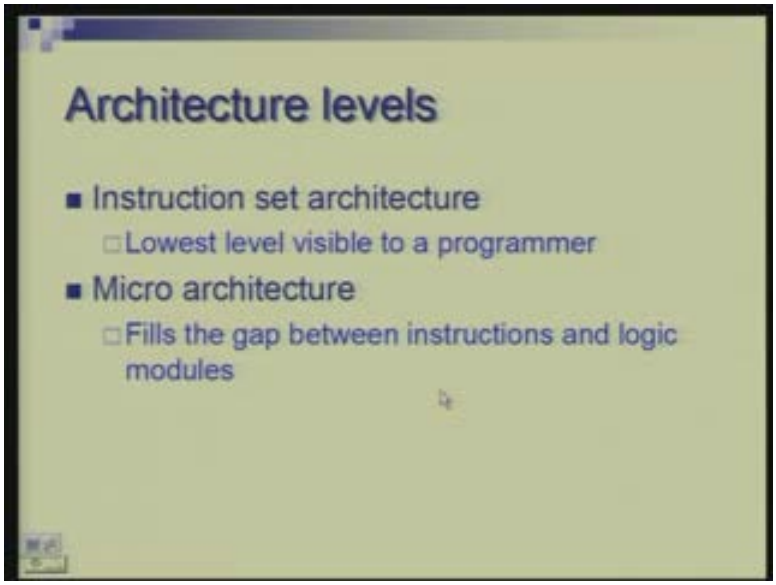


Well, now you have some idea of hardware and software what are layers of abstraction and this picture is trying to put these together. So you have in the software at the highest level you see your high level language programs C plus plus example and at the lowest

level you see machine instructions, in between there was assembly language which I showed you and there are ways in which you can move from here to there.

On the hardware side we saw major building blocks, registers, adders and so on and at the bottom you have individual components and transistors. So our focus would be somewhere here in the middle where you see hardware software boundary. So, what exactly is hardware software boundary? It is where you have a set of instructions which define the basic capability of a processor and major hardware components which are able to understand those instructions. So, if you are a programmer you will see the machine defined by a set of instructions whereas if you are a hardware designer you will see software in terms of those machine instructions which you need to interpret. So there are levels of hierarchy here within hardware and level of hierarchy within software.

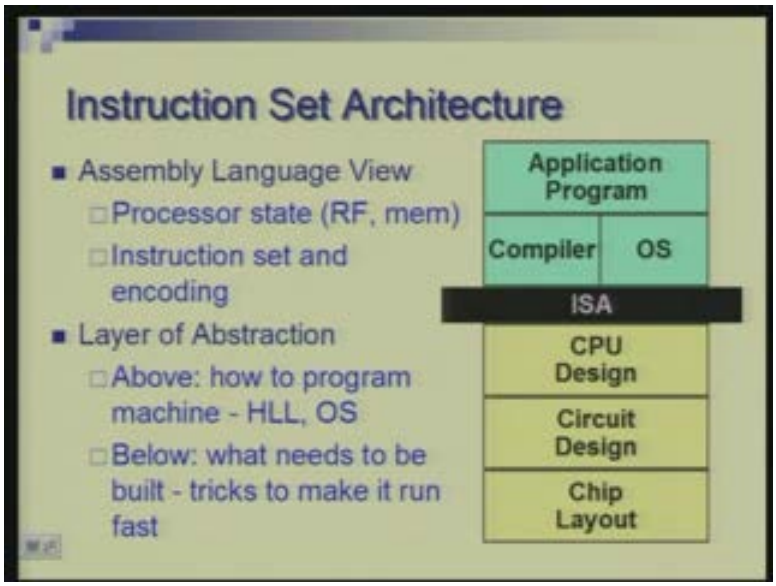
(Refer Slide Time: 22:16)



Architecture itself is around interface between hardware and software. We are going to specifically look at two layers or sometimes it is called architectures at instruction set level or at micro architecture level. The instruction set architecture refers to the lowest level visible to a programmer. The programmer is not concerned about your transistors or your gates or your flip-flops or adders and so on. The basic unit of computation is an instruction whereas micro architecture is what concerns a hardware designer more and it fills up the gap between the instruction and the logic modules.

Logic modules for example, an ALU or a register file themselves do not define any instructions but the way they are put together, the way the information is made to flow through these is what defines instruction.

(Refer Slide Time: 23:44)



This is the typical view of where we place instruction set architecture in between software and hardware. At the top you have application programs which are able to run on a processor with the help of some system software. For example, compilers require translating these high level programs to machine code and OS is required to manage the resources and make it possible to compile, load and execute programs.

Below ISA level you have the design the broad CPU design then at a lower level you have circuit design and for fabricating the circuit, for physically realizing this you need to have a layout where you need to worry about where you place the transistor where you place a wire how you interconnect them and so on so it is not just a network but physical dimension, physical manifestation is assigned to these components.

As i mentioned earlier assembly language view defines what a processor state is and how the processor state changes from instruction to instruction. as you execute instruction the processor goes from one state to other state so how the state is defined; the state is defined in terms of contents of its memory, contents of various registers and flip-flops which are there in the processor whereas memory contains somewhat more permanent data and program, what registers contain is some data which is currently under use. When you are performing an addition typically the operand will come from a register file and the result will go back to the register file.

The programmer at assembly language has to also worry about how instructions are represented; what is set of instruction, how each instruction is represented in terms of 2s and 0s or hexadecimal digits for convenience. So, above this layer of abstraction we have compilers and OS and then you have high level language programs. Below this layer you have hardware components high level as register files, ALU and so on and low level as transistors and resistances which actually make this instruction set possible. But an architect has to worry about how to put these components, these blocks together to do the

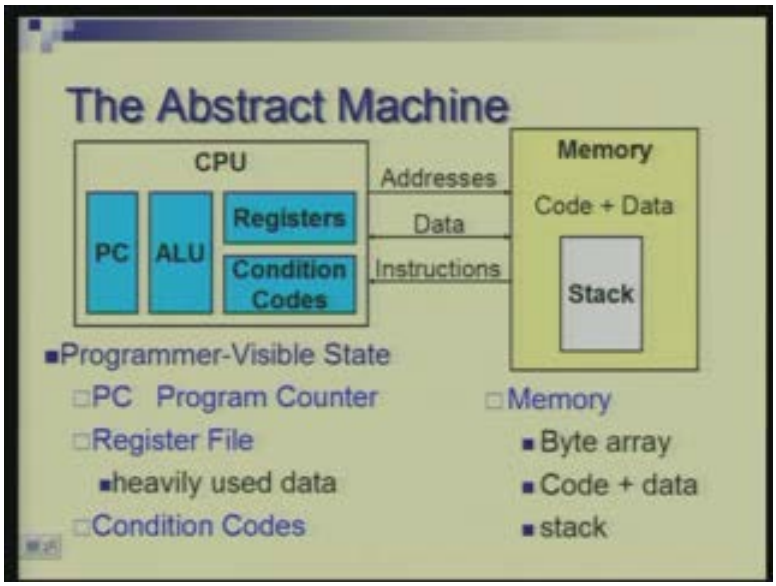
instructions efficiently. So there are lots of tricks and techniques which have to be employed to make everything happen fast.

We will be talking mostly of performance. But as I mentioned there are other issues such as power consumption particularly when you are talking of hand held devices, devices such as small computers, laptops, mobile phones so on where power consumption is important. If you are consuming power at a large rate then the battery which you are carrying would not last very long. So your task would be to carry out computation while consuming as little power as possible.

Sometimes performance is a dominating issue and sometimes power is a dominating issue and sometimes you need to have a suitable trade off you cannot let go performance and go to extreme power saving, you need to have often a combination of the two and with all these said and done ultimately the cost is very important so you may find different ways of executing same instructions but some methods could be very fast but they might be very expensive, they may require lots and lots of transistors to do the same operation.

As we progress with this course we will see these options and you will see that the judgment of an architect lies in making the right choice while designing a processor.

(Refer Slide Time: 23:44)



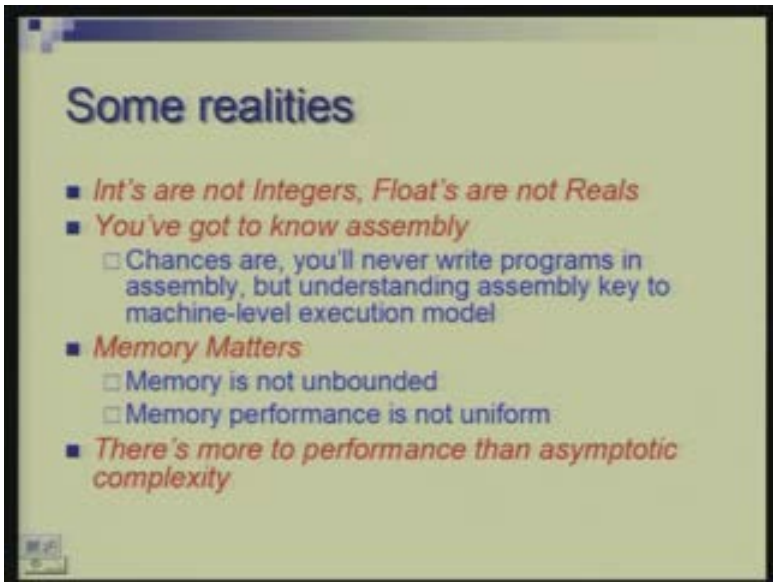
So, what is the basic principle on which a computer works is captured here that you have a memory which contains code or program in machine language and the data on which it is supposed to operate. Of course there have to be means to bring in data on which one has to operate and after computations are done take the results out. And, often memory contains a specialized structure called stack which is used to create functions or routine hierarchy of the program abstractions.

The other block or the heart of the whole system is CPU where the key components as I showed earlier is the program counter and ALU, registers and condition codes whereas ALU is the one which basically perform all arithmetic and logical operations. The operands are contained in registers so they may have to initially brought from the memory and after having operated on them they are send back to the memory. PC is the one which keeps track of the current instruction. The way it proceeds is that, PC will help in picking up an instruction, ALU will carry out the desired operation and PC will then point to the next instruction and so on. So a sequence of instructions will roughly go on in this manner. the instruction will differ from each other in terms of how they are encoded, what operations they invoke, what ALU is supposed to do so there is a controller within the CPU which has not been shown but that is the one which will guide all these components to do the right thing at the right time.

Typically after a operation is done there are some condition codes which are set these often could be a part of the register file itself; sometimes in some processor this could be separate and these could be used subsequently for decision making and that is what helps in providing branches and loops as you know in high level languages. So there may be very sophisticated processors or simple processors but this is the idea which is common to all of them. We will elaborate on this; see specific examples of this.

There are architectures which have been deviced which are which deviate from this basic idea. There are some architectures which are revolutionary, different, we will perhaps talk a little bit about those as time permits. But this is the most common theme or most principle behind everything.

(Refer Slide Time: 31:18)



Although we might make an abstraction like this simply say that a processor works in terms of simple instructions which are representation of the real problem but there are

some real situations which are often have to which often get ignored so when you do abstraction some details are getting ignored and some information may get lost.

One has to be careful when you are making an abstraction. You must know when what kind of simplification is being done and what is getting ignored. For example, in your registers or memory words you will store supposedly integers and reals. But what you are storing actually is only an abstraction, only a representation which may not be exact. So integers for example, as you understand in mathematics are unbounded but when you talk of real programs and real hardware they have to be bounded.

Integer will have to be represented by a finite number of bits. So when you perform addition of two integers in mathematics and you perform addition of two integers in finite number of bits the two may not exactly coincide so one has to understand these differences. Similarly, what you represent as far as real numbers is concerned is again an approximation because of finite number of bits you have finite precision and you have finite range. Therefore, sometimes there are nice properties which real number would satisfy when you are working in mathematical domain the same may not hold when you are talking of real instructions.

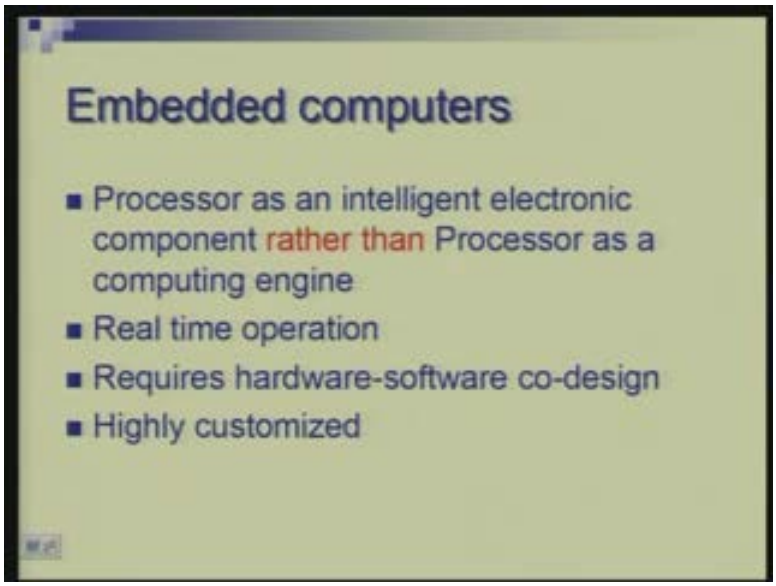
For example, when you have to add three numbers: a plus b plus c you can group them anyway. What is called associativity is a property which you simply assume but you will find that it may not necessarily hold when you are talking of numbers as abstracted in your computer system. Therefore, to understand a processor you need to understand its assembly language fairly well, although you may not in real life program an assembly.

Now, almost all or I should say all programming is done in the high level language but still understanding of assembly language is required at the back of your mind to understand how programs are going to behave when they run on real hardware. So, if you have no feeling of what assembly language is, what instructions or what their limitations are you may program in a world which is somewhat isolated from the physical reality.

Similarly, an abstract model of memory may say that it is simply an unbounded array of bytes or words but physically memory is bound. You have limited number of word or limited number of bytes and each has fixed number of bits and it may not necessarily be equally fast to read different words of memory. One access to memory may be fast next time you may another access to the memory it may be slow so what makes that happen. Therefore, memory is not just a flat array of bytes or words it is a hierarchical structure in real practice so that has to be kept in mind.

When you are talking of performance in program often the discussion focuses on the asymptotic complexity. But in real life what you have to worry about is the execution time that is a bottom line. Given a program and given an architecture how long it takes to execute that program on that particular architecture. That is the question which has to be answered. From theoretical consideration whether the program has one complexity or the other is important but ultimately what one would like to see is in terms of milliseconds, seconds and minutes if that program is taking that long.

(Refer Slide Time: 36:53)

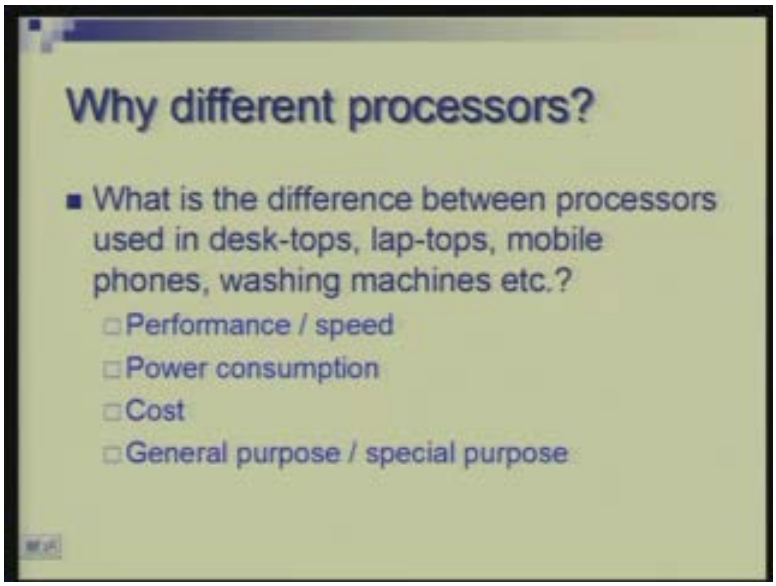


I mentioned about embedded computers domain which would probably if you are in that domain it will provide you opportunities for making new designs, it would typically require designing hardware as well as software and that is harder than designing just software or designing just hardware, it is a task where one has to worry about both the issues. So, embedded computers are treated more as components, embedded processor is part of a system it appears just as a component which performs some intelligent function rather than the conventional view of a number [……37:27] or computing engine.

A small computer which allows you to control process information or perform some communication opens up lot of application possibilities and that is what embedded computer domain is. Typically one has to worry about real time operation there. A computer in embedded domain has to work with information as it appears in real time; it has to respond to that immediately. And as I just mentioned there are lots of design opportunities because each embedded application has a customized design, you might use a standard processor but more often you may have to customize a processor so you may have to add or subtract to the architecture even if it is not completely designed.

When the basic principle behind all the computers is more or less same the simplest part the key part is same what is the difference between the variety of processors you see.

(Refer Slide Time: 38:50)



You see different processors in different applications from tiny ones to large ones, you have in desktop computers, laptop computers, mobile phones, washing machines so on so what is the difference; all work on the principle of having an instruction set and executing instructions one by one. What is the difference in these cases? Of course there may be difference in instructions, there may be tremendous difference in terms of performance or speed, there may be lot of difference in terms of power consumption and cost and in terms of instructions some may be very specialized and some may be of general purpose and general purpose means that you do not a-priori know what kind of application it is going to be used for.

A desktop computer, for example, may be used for power point presentation at one point of time and next time it may be used for emailing and yet another time it may be used for listening to music whereas a special purpose computer for example one let us say a mobile phone has a very specific task, it has to process the calls and that function is unchanged. So unlike in desktop processor where you can change the program and do different things at different times in a mobile phone the program is fixed and over and over again it is the same program which is executed.

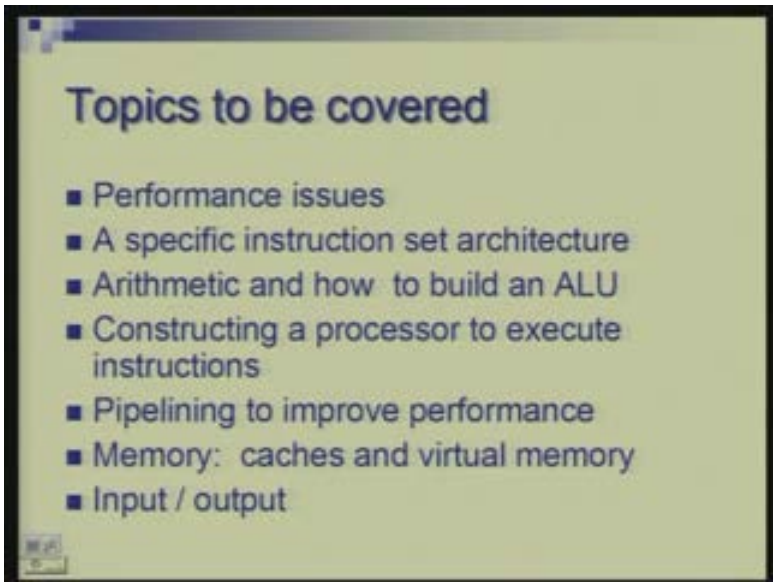
(Refer Slide Time: 40:45)

Here for example you have two different CPUs: this one on the left is a Pentium processor and one on the right is again from Intel the same manufacturer but a micro controller. It is this type of processor which is found typically in embedded application. The actual chip is this small rectangular piece that you see inside this circular window whereas in this case the actual chip is of that size this big squared chip. The size here is larger because of the packaging requirements there are something like forty pins through which you have to connect to the other circuitry so to accommodate forty pins it has to be made so big. But the same device same circuitry is also available in much smaller packages. This is roughly let us say twice this size approximately this size where pins are much tiny and they are much more closely spaced.

Therefore, these two processors are two contrasting devices. One on the left is a general purpose processor, very high performance, consumes lot of power and is much more expensive whereas one on the right is typically used for special purpose applications, it has much lower power consumption, does not care about performance whereas the one on the left would work at Giga Hertz of frequency but the one on the right works only at few Mega Hertz.

Now, this transparent window which you are seeing (Refer Slide Time: 42:44) is essentially a quartz window through which you can erase the program; it has memory which can store the program inside. Since program need not be changed frequently you can put the program inside its memory and then forget about it. Generally if you want to make correction you want to make some upgrade you can expose this to ultra violet light and the memory content will be cleared of and you can change the program but that you will not do every day. So, once you have made it a part of the circuit typically that program will remain and keep on operating over and over again and it is only at some infrequent time if you want to make a change you can do so. There are those where you cannot make a change you just write once and you do not change it often.

(Refer Slide Time: 43:45)

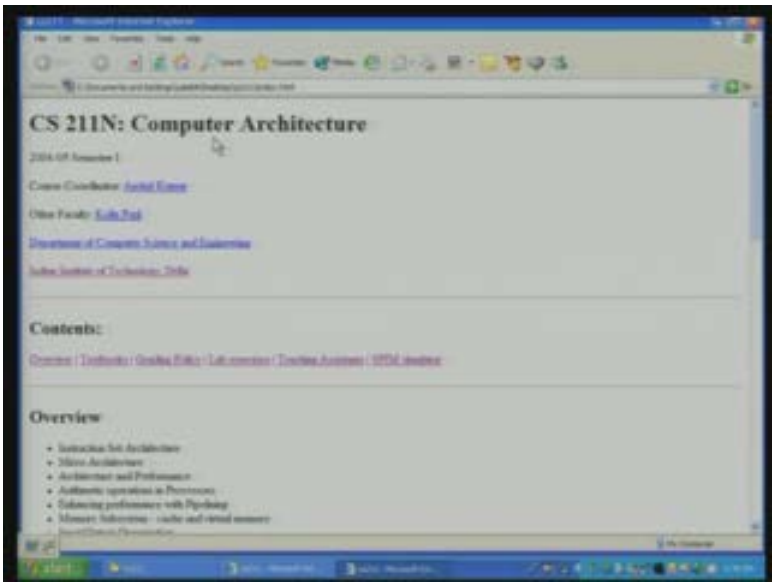


Finally let me summarize with the list of major topics which we are going to cover in this course and that forms an outline of the course. We will not necessarily go in the order in which it is written here. We will define what is meant by performance of a computer; how you define it, how you measure it and how you relate it to various architectural aspects. we will take up specific instruction set, a simple but powerful one so that you can see how instructions work, how you can express computation in terms of those instructions and thereby you will get a feel of what an architecture at instruction set level looks like. Then we will move over to the design aspects. We will see how an arithmetic unit can be designed and built; how basic operations are carried out in terms of binary numbers and how we can build circuitry to carry out those numbers those operations.

Then based around an ALU how you will build a processor to execute the instruction. ALU is just one part of the processor; you have to put register file, you have to put buses, you have to put program counter and you have to put a controller to make the data flow into various components properly. So next will be dealing with a design, the flow of data, the data path and the controller. Briefly I will introduce how a common technique of pipelining is used to improve performance.

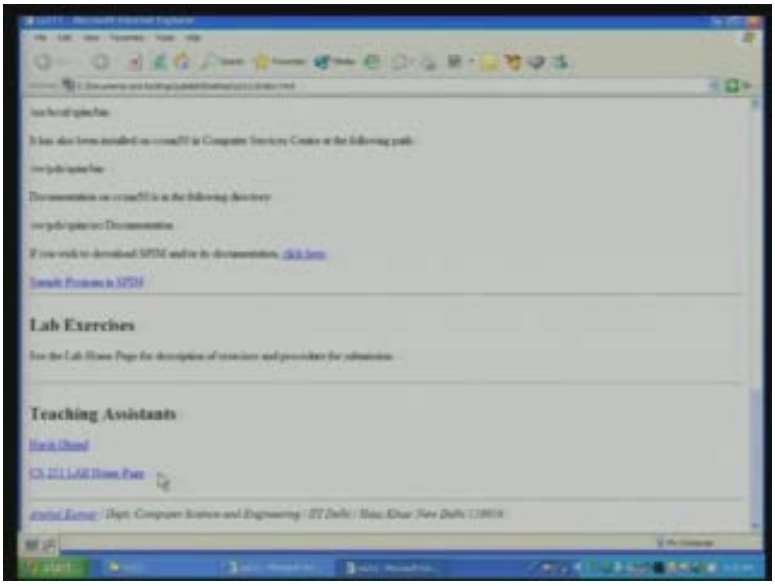
All high performance systems which you see today have pipelining of some form or the other. Then the next important component is memory which is actually structured in a hierarchy; there is cache memory, there is main memory, there is virtual memory, we will see how all these are put together to give you a good performance at a reasonable cost and finally we will talk of input output devices, input output controllers and how they connect to memory and processors.

(Refer Slide Time: 46:39)



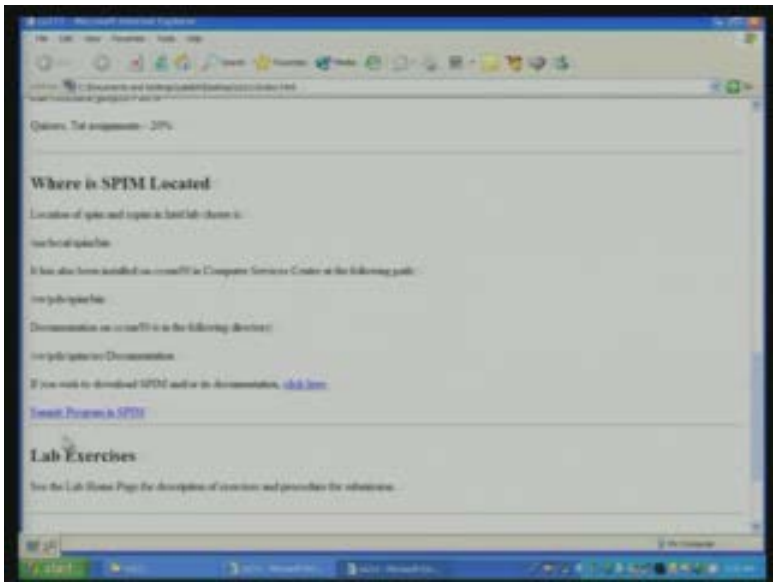
Let me give you an idea of the course homepage which I am going to use to carry out various announcements, put all the information about the course here, the description of lab exercises and so on. Here once again there is an overview, similar list of topics which I talked of. Incidentally Dr. Kolin Paul would be associated with me in conducting this course. He will be taking some of the tutorials and also he is joining me in preparing presentation material for this. There will be a teaching assistant I think more than one, at the moment I know of Harsh Dand he will maintain a lab homepage which you can reach from here where exercises will be announced and all instructions about lab exercises would be given.

(Refer Slide Time: 47:45)



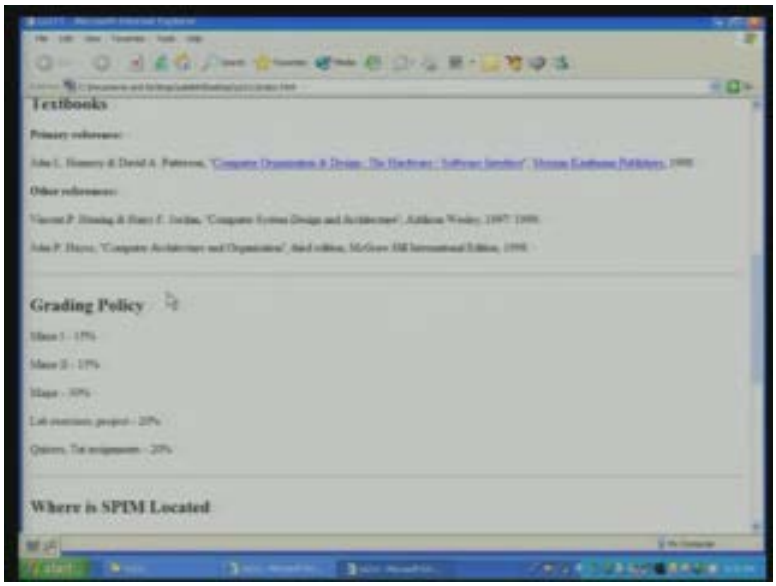
Now, even for the first lab exercise which I have announced there are some more instructions which you must see here before you make a submission. So guidelines for how to make submission of assignment will be given here so before submitting please have a look at this.

(Refer Slide Time: 48:04)



Initial few exercises would be using a simulator called spin which I mentioned to you earlier. You can download or you can use it in our department if you have an account but do not worry if you do not have an account you can use it in computer services centre.

(Refer Slide Time: 48:25)



The textbooks: this is the main book which will be used, I do not know if it is readable. This is Computer Organization and Design - The Hardware Software Interface by G.L. Henasy and D.A. Paterson. This is the main book. Other books which will be occasionally referred to is: Computer System Design and Architecture by Huring and Jordan and Computer Architecture and Organisation by J.P. Haze.

You would notice that these terms Computer Design, Computer Organisation, Computer Architecture are often used interchangeably so you can in fact in this list of books you will see that they are being used in various combinations. But if you do it little rigorously the term architecture is used in terms of describing an overall functionality whereas design refers to the hardware building aspects of it.

We will have minors, majors lab exercises, weightage of lab would be 20 percent and quizzes or any other class assignment altogether would consist of another 20 percent. So I will stop at this and if you have any questions about what I discussed today or about the course in general you are most welcome to ask.

[Conversation between student and Professor……. 50:30 PCI I think stands for….. I am not recollecting what is P for but C is for computer and I is for interface; Peripheral Computer Interface].

Any other question? Mobile phone?
Mobile phone you would consider it not as a computer system but it has a computer. In fact many mobiles may have not just one multiple processors. It has one or more processor, it has memory and it has some IO. You can call it a special purpose computer because as we understand a computer has a processor, memory and IO so it has all that. Although its main functionality from a user point of view is not computing it is communication, right? But seen from computer point of view, yes, it is a special purpose computer.

[Conversation between student and Professor……. 51:43…… like what? Palm; palmtop, yes, palmtop is a computer; in fact it is a more general purpose computer, it has various programs which can be run; it can do calculation, it can do your……it can be used as an organizer, it can be used for communication, variety of functions, it can be used for……..

[Conversation between student and Professor……. 52:22] Yes, I can make them available on the net, yes. I will in fact I will put them but there would be generally a gap of a day or two so it is only after I have delivered same day or next day typically it will be put on the net. Any other question? Okay thank you.