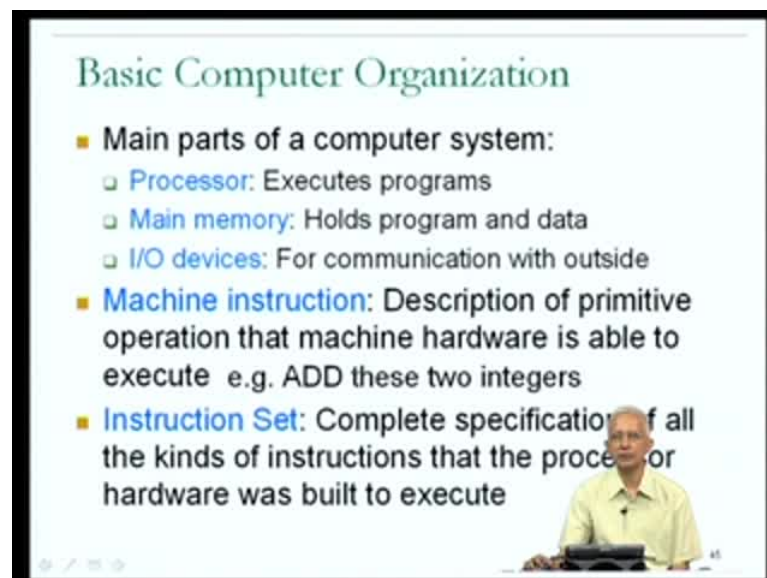


High Performance Computing
Prof. Matthew Jacob
Department of Computer Science and Automation
Indian Institute of Science, Bangalore

Module No. # 02

Lecture No. # 03

(Refer Slide Time: 00:33)



Welcome to the third lecture of our High Performance Computing course. Today, we will continue looking at basic computer organization and you will recall from the previous lecture, which I, which I will quickly review what we looked at, as far as organization is concerned, that there are 3 main parts to a computer system. They are the processor, the main memory and the I/O devices and we saw that all 3 are extremely important. We are going to spend some time talking about processors and main memory.

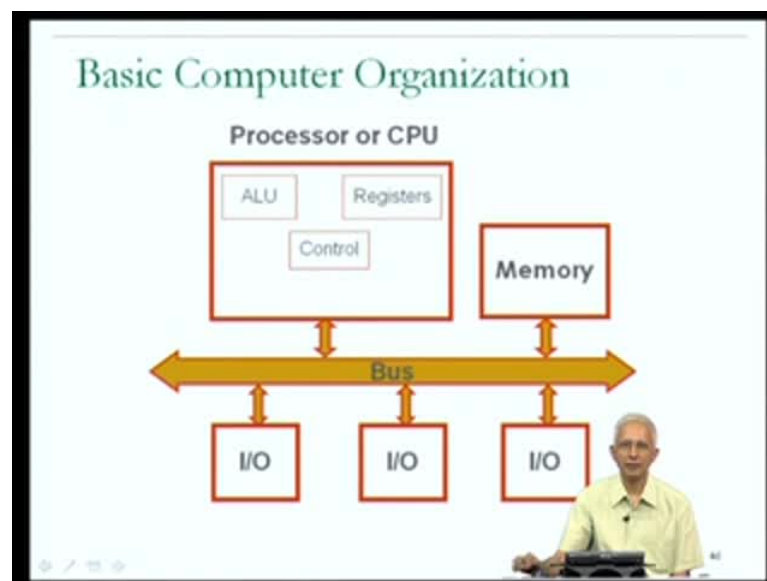
Not so much about I/O devices, I, but I will make a few comments about them. Now, one of the important concepts, as far as programs are concerned, as we saw in the first lecture, our C program gets compiled, translated by a program called a compiler into a program, an equivalent program in a language that is understandable to the hardware.

That language is called the machine language and the simplest instructions, the simplest operations in the machine language are known as machine instructions.

So, ultimately, the C program that you write is translated by a compiler into an equivalent program, in which the algorithm is described in terms of a number of machine instructions, the machine instructions understandable to the processor, which executes the program.

And, a simple example of a machine instruction is the, an instruction to add two unsigned integers, let us say. Now, as far as the given processor is concerned, I will use the term instruction set to refer to all the different kinds of instructions that, that particular processor is capable of executing.

(Refer Slide Time: 01:52)

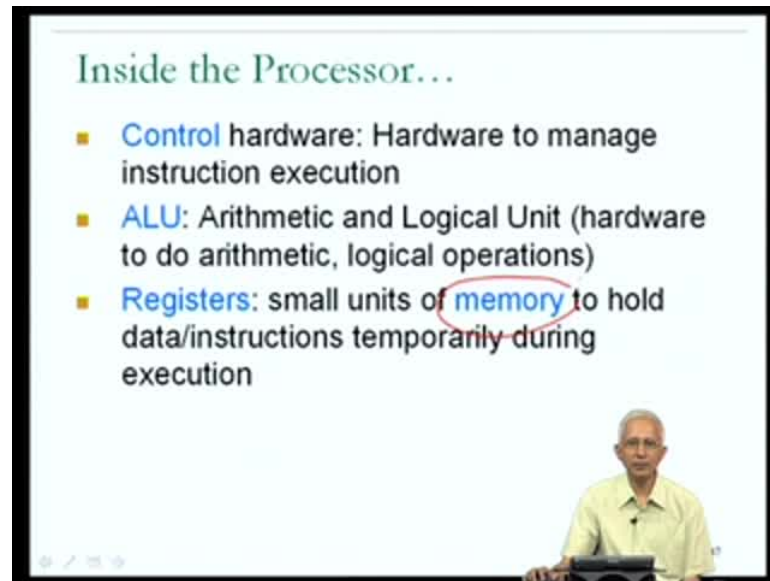


So, with these basic terms, if you look back in our diagram of the **computer organization**, organization of a computer, the building blocks of a computer, we understand that, the processor or CPU is the most important box, the most important component, but the memory is also extremely important, because that is where the instructions of your program and the data of your program are remembered. And therefore, the memory is essential and very important. Finally, the I/O devices in turn are important, because it is only through them that, interaction with the outside world happens. Finally, there must

be some interconnection between all of these components and that I show in this diagram in the form of something called the bus.

And, the communication between the, the memory and the CPU or between the I/O devices and the memory would happen, possibly through this bus.

(Refer Slide Time: 03:01)



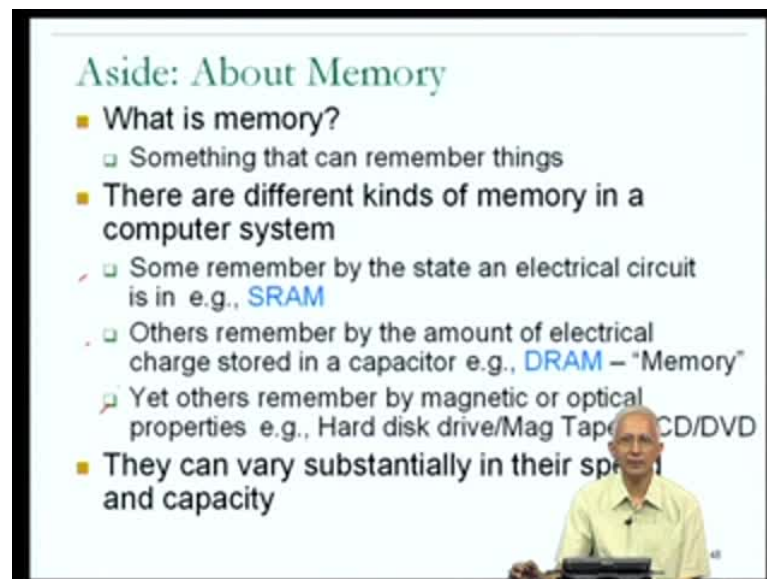
Now, we are going to spend some time looking into the components of the processor and the 3 important components of the processor, that we will look at are, the ALU, the registers and the control unit, which are shown in the, in the small boxes, inside the large box that you see in this diagram. Now, among these, the first and the easiest to understand is the control hardware. That the word hardware is used to, to indicate that, what we are talking about here, is some form of electrical circuitry, electronic circuitry. Basically, the control hardware is the electrical circuitry, electronic circuitry that manages or controls the instruction execution. And this would be of great interest. The details of, design of control hardware would be something that you would study in a course on Electrical communication engineering or a course on Electrical, Computer Science and engineering.

Now the second box that we saw inside the CPU or processor was something which was labeled the ALU and many of you would know that, this is the abbreviation for Arithmetic and Logical Unit. As the name suggests, this is again a piece of hardware,

circuitry that is capable of doing arithmetic and logical instructions. In other words, doing the calculations of the logical operations necessary, in order to achieve, what the instructions require. For example, if there is an add instruction, which is capable of adding unsigned integers, there would be a corresponding circuitry inside the ALU, which would be called the unsigned adder, which could do the operation. Finally, there are, there is a unit in the box called registers and this may be a new term to many of you.

So, let me explain. The registers are small units of memory and they are used to hold data or instructions temporarily, during the instruction execution. So, remember all 3 of these are parts of a processor. Now, there is a little bit of ambiguity here, because, as you will see, I have used the word memory over here, in a different way from, what I had used it in the previous slide. In the previous slide, where we had the diagram, I had this box labeled memory and I was indicating that, this is a component of the computer system, where instructions and data are stored. Whereas, over here, in this slide, I am using the word memory in a slightly different way.

(Refer Slide Time: 05:12)



So, what you should understand from this is that, the word memory unless disambiguated, could mean a few things. I will ((be first)) spend a little bit of time, explaining what memory is and what and I will be, a little bit more careful in my terminology, when I talk about memory, from now on. So, let us try to understand about memory.

The first question we will ask is, what is memory and in the abstract sense, you would recall that you, yourself have memory. You are capable of remembering things. Therefore, somewhere in your system, there is something that is capable of remembering. And, in general, you might say that, something that can remember things would be called memory. There is memory in the human beings, There is various forms of memory in a computer system and so on.

Now, just to be a little bit more specific, you as a human being remember things in your system, using part of your brain or wherever else the, the memory may happen, but you also use other mechanisms to remember. For example, for preparing this lecture, I made lecture notes and I made the lecture notes on pieces of paper. So, as far as I am concerned, the pieces of paper with something written on them serve as a form of memory. And there are various other forms of memory that the human being makes use of, but what about in a computer system?

In a computer system, as it happens, there are many different kinds of memory. In other words, things that can be used to remember and the box which I had labeled as memory in the block diagram of a computer system, is one very special kind of memory. We are now using the word memory in the more general sense.

So, let me just say something about the different kinds of memory in a computer system. Now, I am going to distinguish between the different kinds of memory in a computer system, based on the kind of circuitry or the mechanism that is used to do the remembering. I used the example of how a human could remember, on a piece of paper, by writing something on a piece of paper. As it happens, in computer systems, not so long ago, paper was used as a form of memory. Some of you may remember or may have heard of how people used to type programs. They did not type programs in at a keyboard, they actually used to type programs in, on two pieces of cardboard, right and this deck of cards would then be fed into the computer and therefore, the paper, the cards, computer cards were a form of memory.

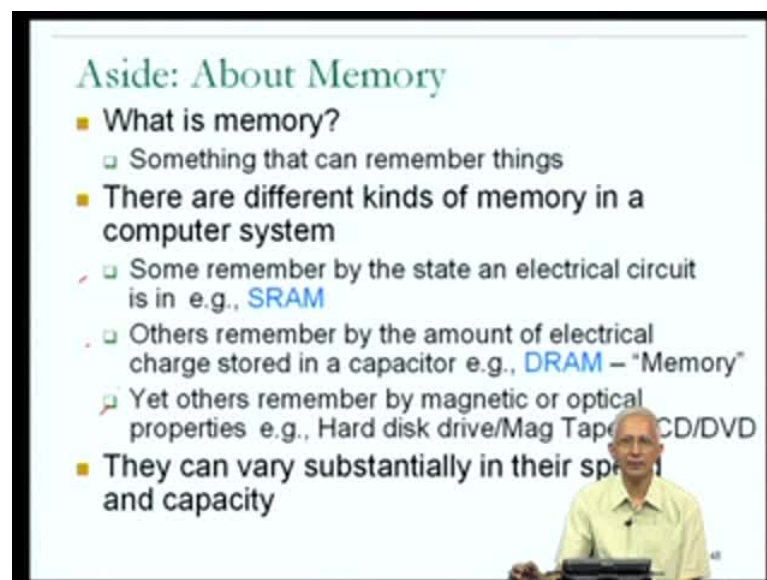
Similarly, there was a time, at which paper tape, spools of paper tape in which holes were punched at well specified locations were used as a form of memory. Paper is hardly used **for storing**- for memory and computers any more.

But it is interesting to note that, historically paper was an important form of memory in computer systems. In today's talk, I am not going to use paper as an example in the, **in the**, in the slide, because, it is actually not that common any more.

So, I am going to distinguish between the different kinds of memory in a computer system based on, how the remembering is done, how the data that is to, or instruction that is to be remembered, is remembered.

So, in that sense I could say that, in a computer system there are some forms of memory which do the remembering by the state that an electrical circuit is in.

(Refer Slide Time: 05:12)



So, they could be carefully designed circuits, which could be in, let us say, one of 2 states and if the circuit is in the state which is called 0, then it might be remembering the bit 0, whereas, if the circuit is currently in the state called 1, then I might say that, I am remembering the bit 1. I am having a huge collection of such circuits, I can remember entities which are larger than 1 bit.

So, that is a general idea of this form of memory - the idea that, I could have electrical circuits which are capable of remembering, and remembering basically happens, by the state that the electrical circuit is in. In other words, the current situation as far as the electrical circuit is concerned. Just to give you one simple example of **electric of of** a memory in a computer system which is of this form. Some of you may have heard of

something called Static RAM. I am going to talk about parts of the computer system which use Static RAM later. Up to this point in time, we have not seen the part of the computer system, which uses Static RAM.

Static RAM is an example of a device of the first kind. Now, there are other memories in computer systems, which remember not by the state an electrical circuit is in, but by the amount of charge present in a capacitor. A capacitor is a kind of a component in electrical or electronics circuits, and if memory is implemented using the amount of charge on a capacitor, situation might be that, if there is no or low charge, that would be the equivalent of remembering a 0 and if there is high or large amount of charge, then that might be the equivalent of remembering a 1.

And an example of memory which is used in computer systems, which is of this kind is, something known as Dynamic RAM and in fact, the memory block which I had used in the diagram of, the block diagram of computer organization, typically uses this kind of memory. In other words, the remembering of data and instructions is actually happening through the amount of charge when we use capacitors inside the memory device.

Now, a third kind of remembering which happens in computer systems today is that, information could be, could be remembered by a magnetic or optical property which is used and I think many of you can remember situations where this happens. For example, as far as magnetic properties are concerned, you know that there are **magnetic**- the hard disks which you have on most computer systems today, actually remember by the magnetic mechanism.

If you were to open a hard disk, you would find out that, there is something like a plate, which is coated with a magnetic material and their different sectors, different regions on this magnetic surface and those are basically the different bits of storage. So, hard disk remembers through magnetic properties and all of you have used vcds or dvds and may be aware that these operate through the optical properties of the medium.

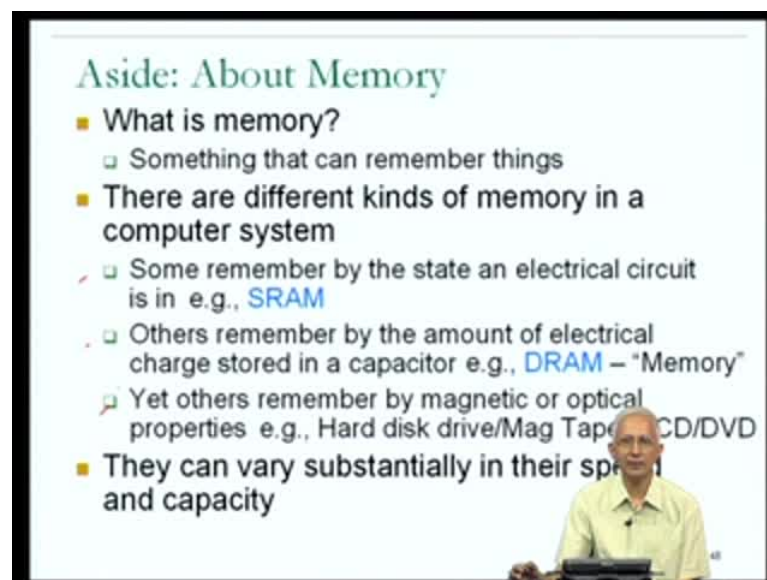
So, examples of remembering by magnetic or optical properties are things like hard disks or magnetic tapes. Some of you may have used tape cartridges in systems that you have access to and finally, vcds or dvds.

So, these are all examples of the third kind of remembering. Now, if you look at this range of possibilities, that I have outlined over here, the SRAM, the DRAM and the optical or magnetic storage, obviously they differ in some way, from each other and very clearly, they differ in terms of how they store, remembering is done.

But, there is another critical component in which they vary and that is, they vary substantially both in their speed and in their capacity. And what I mean by speed here is, how fast one can read something, a piece of data or an instruction out of that memory? How fast one can write a new instruction or data into that form of memory? That is what, I mean by speed. Speed of access and what I mean by capacity is the, the amount of memory that one can have in, let us say, a given area or for, for given amount of money.

And as you know, 1 Gigabyte of storage on vcd on a dvd is much, let me put in this way, if you were into purchasing computers or into making decisions about the purchase decisions as far as a computer is concerned, you would be aware that, it is much cheaper to add 500 Gigabytes of disk into a computer system than to add let us say, 4 Gigabytes of main memory, memory of the DRAM type, into a computer system.

(Refer Slide Time: 05:12)



Aside: About Memory

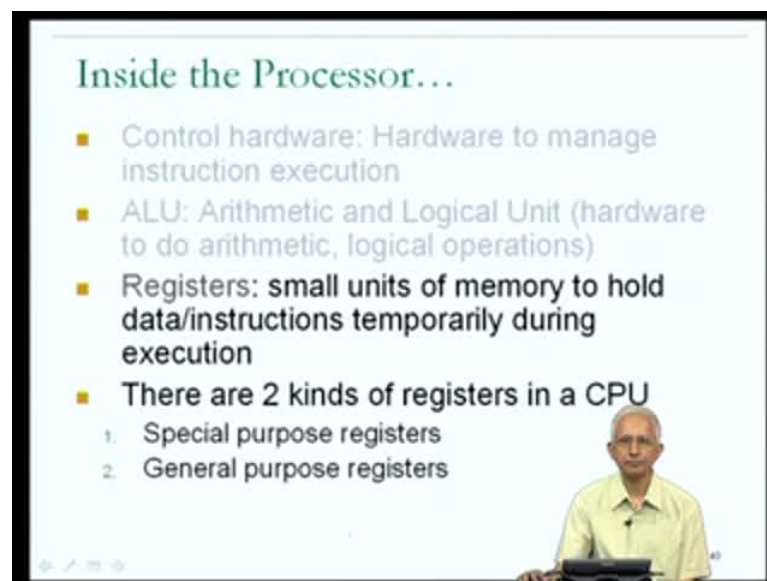
- What is memory?
 - Something that can remember things
- There are different kinds of memory in a computer system
 - Some remember by the state an electrical circuit is in e.g., **SRAM**
 - Others remember by the amount of electrical charge stored in a capacitor e.g., **DRAM** – "Memory"
 - Yet others remember by magnetic or optical properties e.g., Hard disk drive/Mag Tape, CD/DVD
- They can vary substantially in their speed and capacity

So, disk is very clearly much cheaper, than in terms of its cost for **capacitor** per, per given amount of storage, than the main memory is, the DRAM memory, and vcds, dvds are clearly much cheaper than the disk. So, that is what I meant, when I had in the slide

indication that the memories that we saw can vary substantially in their speed and in their capacity.

Therefore, we expect that in a computer system, we may see a lot of storage which is of low cost and not that much storage of high cost and that we expect that the storage which is of low cost will be therefore, of high capacity and is likely to be much slower than the storage which is of high cost. Therefore, this is clearly going to be some kind of trade off that has to be played.

(Refer Slide Time: 14:06)

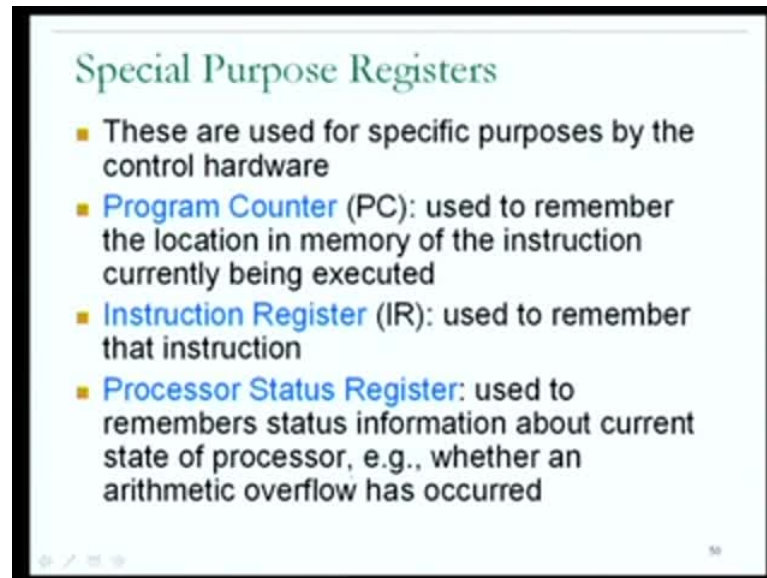


So, with this quick aside about memory, let us go back to our discussion about computer organization and move on to the next slide. Now, in the next slide, I, you look all the right slide that there are 3 important components to the processor. They are the control hardware, the ALU and the registers and I indicated that, the registers are small units of memory and they are used to hold data or instructions temporarily during execution.

So, registers are a form of memory. They are small, because they are inside the CPU, right, and if you have too many registers, then there may not be enough space in the CPU, for the other important parts of the CPU, like the control hardware and the ALU. But now, from now on, we use the term registers to refer to this use of memory inside the CPU.

Let me just say a little bit more about registers. So, registers are an important part of the processor and there are 2 main kinds of registers which are used in the CPU. They are known as the special purpose registers and the general purpose registers.

(Refer Slide Time: 14:58)



Let us look at each of these in a little bit more detail. First of all, let us look at the special purpose registers. Now, as the name suggests, the special purpose registers are small pieces of memory, which are inside the processor and they are used for specific purposes by the control hardware. That is why they are called special purpose registers. They are used for specific purposes and each of these special purposes registers would be known by a name, which is indicative of this specific specific purpose for which it is used.

I will give you 2 or 3 examples and these examples are going to be critical for our discussion of the processor, which is to follow. Now, one of the very important special purpose registers in any processor, is a special purpose register which is known as the program counter, which is typically abbreviated as PC and this is a register, a special purpose register, that is used to remember the location in memory of the instruction which is currently being executed. You will remember that, at any given point in time, the processor is executing an instruction and that instruction came from memory.

Therefore, the purpose of the program counter is just to remember, from where in memory the current instruction, which is currently being executed, came. That is all. So,

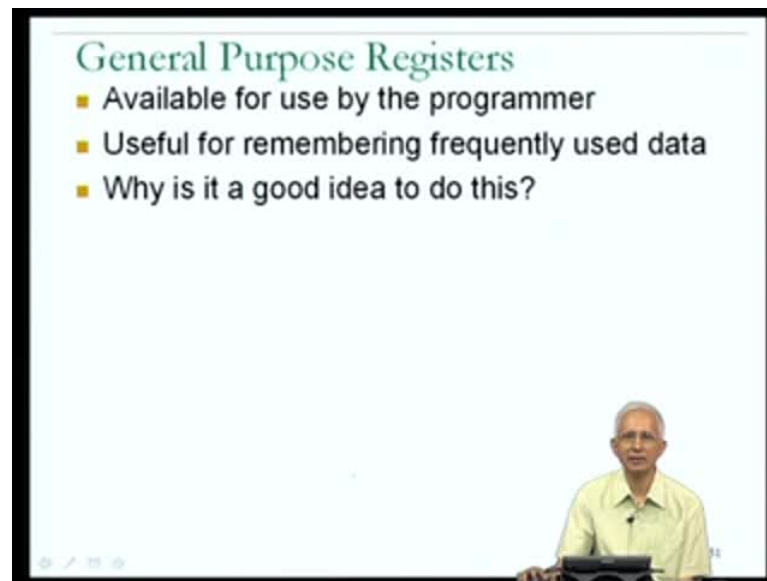
this is the special register called the PC or program counter, for this purpose. There is another special purpose register called the instruction register or IR in the CPU and this is used to remember the instruction that is currently being executed.

So, you think of the instruction register, as a small amount of storage inside the CPU, which at any given point in time contains or remembers the instruction which the CPU is currently executing. And this may be necessary because, in executing an instruction the, the control hardware may have to examine that instruction in great detail, different parts of the instruction and therefore, the instruction register is the place in the, in the CPU where instruction is located.

As one more example, most processors will have either one or a collection of registers which are used to remember, status information, about the current state of the processor. And the, this might be known as the processor status register. What I mean by the current state of the processor, could include many different pieces of information. One of the pieces of information might be, for example, just after an arithmetic instruction has executed, there is the danger that, during the execution of the arithmetic instruction, an arithmetic overflow may have occurred and if that is the case, that could be reflected as a change in the state of the processor by may be, a bit in the processor status register.

So, various pieces of information about the current state of the processor would be contained in this processor status register and used by the control hardware to manage the execution of a program. Remember, the control hardware, the ALU and the registers are the 3 parts of the processor. We are currently looking at one, some of the registers which are known as the special purpose registers.

(Refer Slide Time: 18:14)



Let us now move on to the second kind of register, which is the general purpose registers. Remember, of the registers in the CPU, some are special purpose, which are used for special purposes by the control hardware, others are general purpose registers. As the name suggests, these are registers which can be used for anything that the programmer wants.

So, they are available for use by the program. They are not used for special purposes by the, by the hardware. They are available for use by the program. Now, typically as programmers, who are, many of us are not even aware of the fact that, there are general purpose registers.

So, this may come as a little bit of a surprise. The idea that, there are registers, pieces of memory inside the CPU, that I, as a programmer can make use of. Now, the interesting thing to note here, is that, as a C programmer, even if you are not aware of the fact that, there are general purpose registers, it does not matter, because, you write a programs using variables in C and this program, later gets translated by a compiler into a language called the machine language.

So, as long as the compiler knows about the general purpose registers and uses them while translating your program, then your program will be making use of the general purpose registers. But, sophisticated programmers would, sophisticated programmers

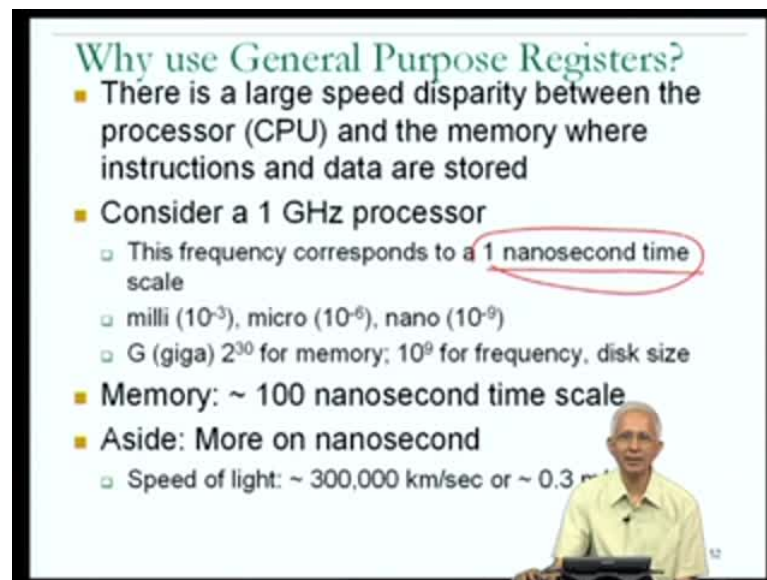
who, like to make their programs as efficient and fast as possible, may want to try to make use of the general purpose registers explicitly.

So, the question which may arise now is, so, there are registers in the processor, in other words, small pieces of memory inside the processor which are intended for whatever use the program wants to make, **you**, make of them. And typically, what these registers are used for, by the program is for remembering frequently used data.

So, if there is a piece of data, like a variable x , which your program is using, which it is accessing many times, rather than accessing that variable out of memory every time, the program may want to access it out of a register, because register is part of the processor itself, whereas, the memory is a separate block in the organization of the computer.

So, we may want to get a better idea about, why this is a good thing to do? So, why is it a good idea to put frequently used data, into a general purpose register rather than just leaving them in memory along with all the other data?

(Refer Slide Time: 20:37)



Why use General Purpose Registers?

- There is a large speed disparity between the processor (CPU) and the memory where instructions and data are stored
- Consider a 1 GHz processor
 - This frequency corresponds to a 1 nanosecond time scale
 - milli (10^{-3}), micro (10^{-6}), nano (10^{-9})
 - G (giga) 2^{30} for memory; 10^9 for frequency, disk size
- Memory: ~ 100 nanosecond time scale
- Aside: More on nanosecond
 - Speed of light: ~ 300,000 km/sec or ~ 0.3 m/ns

Now, this is an important concept. So, I will spend a whole slide, on trying to make sure that, this is clear. Then, the question that we are trying to answer is, we know the general purpose registers are available in a processor, for whatever purpose the program wants to make of them and I indicated that, they might be used for storing frequently used data.

But, why is this in the interests of the programmer? Why is this in the interest of the good execution of the program? Now the reason that, this is a good idea is that, there is a large speed disparity between the processor and the memory where the instructions and data are stored. Now, from now onwards, I am going to refer to the memory, where the instructions and data are stored, in other words, that block in the computer organization block diagram, as the main memory, because, that is the critical memory of the computer, where the instructions and data are stored.

So, apparently, there is a large speed disparity. In other words, the processor and the main memory, operate at completely different speeds. Let me give you an idea about, how fast or how slow they are relative to each other.

Now, let us look, think of the case of 1 Gigahertz processor. You may have heard of this kind of a quantification of the capabilities of a processor in, in computer ads or manuals and things like that. Now, what we have over here is the quantification of I, I pronounce that as 1 GHz, as 1 Gigahertz. You have come across G before in one of the earlier lectures I mentioned that G is an abbreviation for something. We have not come across Hz before, but some of you may have come across Hz in a course in Physics. Hertz is a measure of frequency.

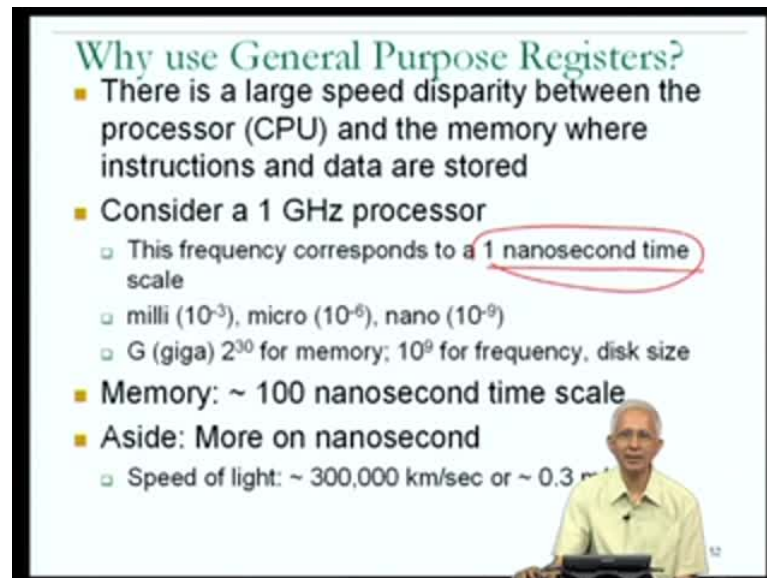
So, our suspicion is that, this is a measure of the frequency of a clock used in the design of the processor or in some sense, a measure of the speed of the processor. So, if 1 Gigahertz quantifies the speed of the processor, it also quantifies in some sense, the time scale of the processor, the amount of time in which something important happens inside the processor.

So, 1 Gigahertz is actually one billion cycles per second. So, I can easily convert from frequency to time scale by just taking the inverse and if I do that, what I come up with, is the idea that a 1 Gigahertz processor actually has, a time scale of one nanosecond and before going any further, let me just clarify what, what is happening here.

Now, first I, I will come back to the Gigahertz shortly, but first of all, what is this nanosecond? I have not used the term nano before. Some of you would have encountered it before. We talked about kilo, mega, giga and so on.

There is also milli, micro and nano and as you all know milli is 10 to the power minus 3. For example, a millimeter is 1 thousandth of a meter, a micrometer is 1 millionth of a meter and nanometer is 1 billionth of a meter or 10 to the power of minus 9 meters. So, a nanosecond is one billionth of a second or 10 to the power of minus 9 seconds.

(Refer Slide Time: 20:37)



Why use General Purpose Registers?

- There is a large speed disparity between the processor (CPU) and the memory where instructions and data are stored
- Consider a 1 GHz processor
 - This frequency corresponds to a 1 nanosecond time scale
 - milli (10^{-3}), micro (10^{-6}), nano (10^{-9})
 - G (giga) 2^{30} for memory; 10^9 for frequency, disk size
- Memory: ~ 100 nanosecond time scale
- Aside: More on nanosecond
 - Speed of light: ~ 300,000 km/sec or ~ 0.3 m/ns

Now, in the previous lectures, I had talked about G as standing for 2 power 30. Now, that is not always what the abbreviation G stands for. In computer systems, if I am talking about memory, the size of memory, then G stands for 2 power 30.

So, if I am talking about a Gigabyte of memory, that is 2 to the power of 30 thirty bytes of memory. But, if I am using the G, the, the prefix G to refer to frequency or the size of data on a disk, then G actually stands for 10 to the power of 9. In other words, it is a metric, the metric quantity. Therefore, the 1 Gigahertz that I refer to, in terms of the power of the processor, is a billion hertz, a billions cycles per second, whereas, a Giga byte is 2 to the power of 30 bytes of memory.

Similarly, as far as disk **concerned** is concerned, a 500 Gigabyte disk contains 500 billion bytes of information, it can contain 500 billion bytes of information in it.

So, this is how I came up with this quantification of time scale of the processor, that one Gigahertz processor has a time scale of 1 nanosecond. What does it mean to have time

scale of 1 nanosecond? Once again, that basically means that, in a processor an event of interest happens in about one nanosecond.

Now, the next question which arises is how fast are the main memories that we have on computer systems today? Because, my claim was that there is a large speed disparity between processors and main memory. As it happens, the main memories of today have a time scale of about 100 nanoseconds. It is an order of magnitude approximation, the time scales could be 70 or 90 or 110 nanoseconds.

But, it is approximately a 100 nanoseconds. What does this mean? This means that, the main memory is about 100 times slower than the processor and so, every time the processor needs to fetch an instruction from the main memory, it has to wait for about 100 time units in terms of its local time scale, and this is a fairly large speed disparity. The processor is about a 100 times, a 1 Gigahertz processor is about a 100 times faster than the kind of memory that we are talking about over here.

And thus, you well know the processors that are available today, are some times faster than 1 Gigahertz. More power, higher frequencies than 1 Gigahertz. For example, some of you may even have 3 Gigahertz processors in your desktop computer in the lab or at home and that would mean that, the time scale is one third of a nanosecond, right. So, the situation that I have outlined over here is a not extreme case. 1 Gigahertz processor is very common- I mean processors are typically faster than 1 Gigahertz and memories are not going to be much faster than this 100 nanosecond time scale.

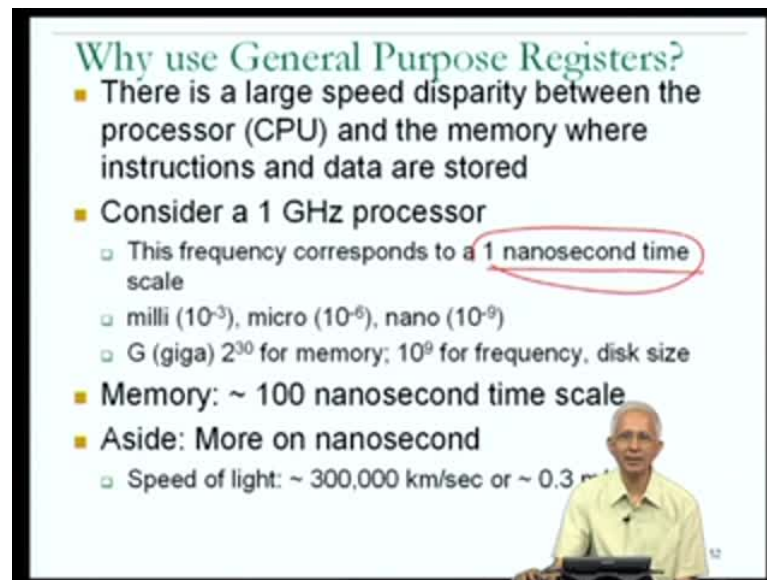
So, this is the reason that using general purpose registers makes a lot of sense to the program and to the programmer. Because, the processor is a 100 times faster than the memory and therefore, if you have data which is frequently accessed and every time it, it has to be accessed, it is accessed out of main memory, that is going to take 100 times more, than if the data had been present in a register, a general purpose register inside the processor itself, in which case, it could have been accessed in an amount of time comparable to the time scale of the processor, a nanosecond or thereabouts.

So, this is the reason that, using general purpose registers for, for storing frequently accessed entities is critical to the efficient or execution of a program.

Now, before going back to our discussion of computer organization, I thought it might be useful, to just give you some kind of a feel of what this one nanosecond time scale of the processor means. Very few of us can actually visualize what a nanosecond is.

Because, the human response time is so much slower than a nanosecond, right. We are just like there is a large speed disparity between the processor and the CP and the memory, there is an even larger speed disparity between the time scale of a human being and the, the time scale of the processor. Some, this is an interesting thing to think about. What is the time scale of a human being? What is the amount of time that it takes to do the fastest operation that a human being is capable of?

(Refer Slide Time: 20:37)



Why use General Purpose Registers?

- There is a large speed disparity between the processor (CPU) and the memory where instructions and data are stored
- Consider a 1 GHz processor
 - This frequency corresponds to a 1 nanosecond time scale
 - milli (10^{-3}), micro (10^{-6}), nano (10^{-9})
 - G (giga) 2^{30} for memory; 10^9 for frequency, disk size
- Memory: ~ 100 nanosecond time scale
- Aside: More on nanosecond
 - Speed of light: ~ 300,000 km/sec or ~ 0.3 m

And, you could try to estimate this, using examples from extreme human behavior. For example, you know that somebody who is an expert typist, can type at, let us say, a 100 words per minute and from this you could calculate how, how much time it takes for the person to press 1 key or you know that somebody is able to run a 100 meters in 9.5 seconds - from this you could calculate something about the human's time scale, but it is very clear the human time scale is much, much slower than the time scale of either the processors or the memory.

So, to understand what nanosecond is, we need some other phenomenon. So, the phenomenon that I am going to use is, in fact light, because, when we talk about 1

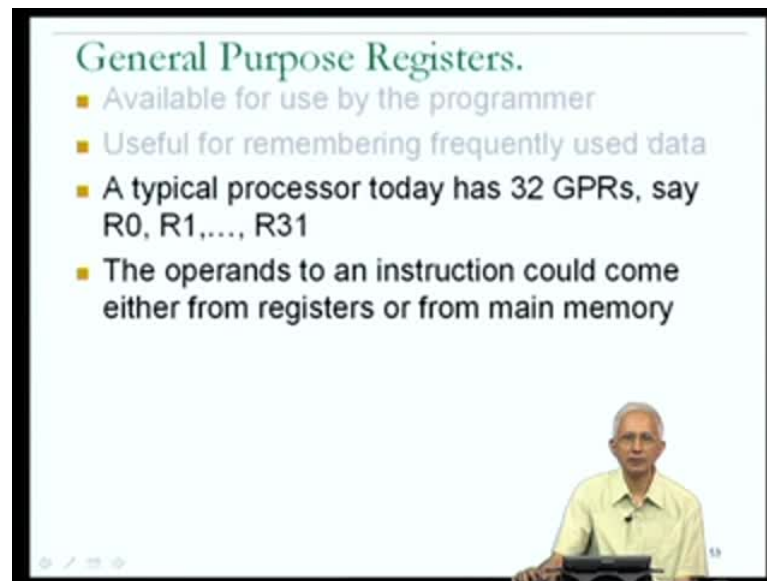
nanosecond, we are talking about something which is actually on a scale where light, the movement of light is to be taken into account. Now, you know that the speed of light is approximately 300,000 kilometers per second and if you, a quick calculation on the back of an envelope will tell you that, what this means is that, light travels at about 0.3 meters per second, I am sorry, per nanosecond.

So, a nanosecond is the amount of time that, light takes to travel about this much distance. Light travels about this much distance in a nanosecond. That is what a nanosecond is. If I have a 3 Gigahertz processor, then the, the, the time scale of the processors is about a third of a nanosecond, which means that, light could travel about 4 inches, about this much distance and we are, we are now looking at a distance which is about the size of a large computer chip or a processor and this may be a problem as far as designing computers is concerned. If the signal, if the flow of information in a computer chip is even 60 percent of the speed of light, then in a, in the time scale of one of these fast processors, it is not going to be possible for a lot of signal flow to happen inside a large chip.

So, this is the significance of the nanosecond. Processors are quite fast, memories are slower, but processors are in fact so fast that, designing high speed 1 Gigahertz, 3 Gigahertz processors is itself an art, because, one has to very carefully lay out the components inside the processor, so that, the speed of transmission of the signal does not become an issue.

So, with this we have a clear understanding that it makes a lot of sense to use, make use of general purpose registers, as far as possible for storing frequently used data.

(Refer Slide Time: 31:02)



So, let us move on. I, we are talking about general purpose registers. We understand that, they are available inside the processor, just like special purpose registers are and they are useful for, typically used by the programmer or by the compiler on behalf of the programmer, for remembering frequently used data. Now, the question which will be foremost in your mind is how many such registers are typically available in a processor? Because, you are hoping that, there are thousands and thousands of such registers, so that, you can make use of them and get the full benefit of this very fast access to the variables of your program.

But unfortunately, a typical processor today, has something like 32 general purpose registers. This is a fairly typical situation. There could be a few tens or at most a few hundreds of general purpose registers and as I indicated earlier, this is largely because the space on the processor, the amount of area of circuitry that can be built into the processor is not available only for general purpose registers. There is a lot of other critical functionality that must be included and 32 has been found to be a good compromise between what is useful to the programmer and what is possible.

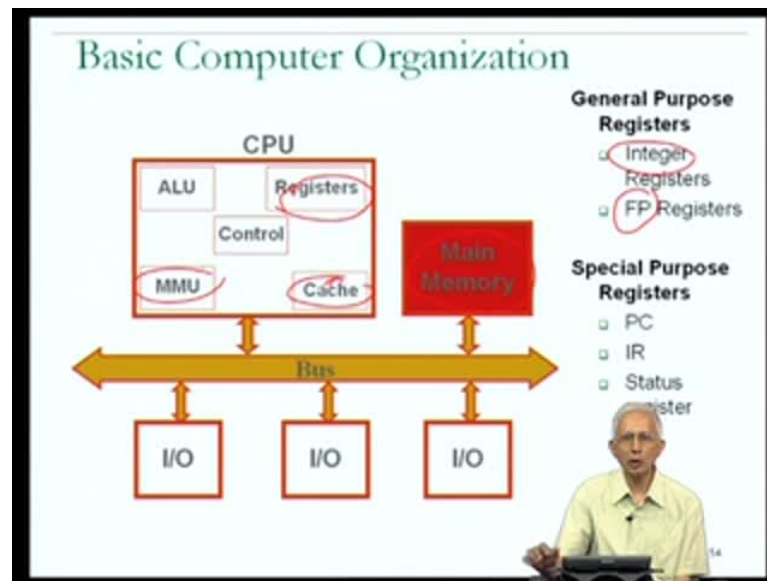
Now, if there are 32 general purpose registers in a processor, then, they have to be individually referred to, just like I talked about the program counter being referred to as the PC, instruction register being referred to as the IR. Now, the program counter as the instruction register are used by the hardware and therefore, the programmer in his

program does not typically have a need to refer to the program counter or the instruction register. On the other hand, the programmer or a compiler will have to make use of the names of the general purpose registers inside the program. Therefore, from now on if I am referring to general purpose registers, I will use names like R0, R1, R2. If there are 32 registers in a processor, I might use the names R0 through R31 to refer to the 32 general purpose registers. And, we are expected, in a program which makes use of general purpose registers, the name of the registers such as R0, R1, R2 etcetera will appear.

Now, typically, when we have general purpose registers and we have main memory, what this means is that, as far as an, a given instruction is concerned, instruction takes operands and the operands to the instruction could either come from a register or they may have to come from main memory. Remember, all the variables of your program, whether they are statically allocated, dynamically allocated, stack allocated are all going to be present in the main memory, along with the instructions of the program. The instructions get fetched from the main memory into the processor as and when they are going to be executed.

But, the variables are present in the main memory and if the program is written so that, a particular variable, let us say, the integer variable x is copied into a register, then, the value of the variable could be accessed from inside that register and therefore, an instruction which is to operate on the variable x could instead operate on the register which contains the variable x.

(Refer Slide Time: 34:22)



So, in general, the operands to an instruction could come either from registers or from main memory. So, at this point our picture of the basic computer organization looks like this. I, if you noticed that I have replaced the term memory in this block, by main memory. From now on, whenever I refer to this block, I will use the term main memory. Because, we know that the word memory is a, is a more general term, which could refer to anything from paper to a vcd to something like this main memory. Now you will notice that I have opened up the, the space inside the CPU, leaving a lot of space for some additional boxes to be drawn, indicating that very soon I am going to be mentioning some more of the components of the CPU. For the moment, we understand the ALU, arithmetic logic unit, control hardware and the registers.

Let me just quickly recap about the registers. Remember, there are general purpose registers and there are special purpose registers. Examples of the special purpose registers that we saw are, the PC, the program counter, the IR, the instruction register and the status register. The PC always contains the, I mean gives you the indication of, from where in main memory the instruction which is currently being executed inside the processor came from and the IR or the instruction register always contains the instruction which is currently being executed by the CPU.

Then, we have the general purpose registers, of which we saw, there could be, let us say, 32 inside, inside the processor. So, when I talk about both special purpose and general purpose registers, I am referring to this block, box over here.

Now, one additional thing about general purpose registers, which I would like to mention at this point is, you will remember that, as far as the kinds of data that a program might have to execute with are concerned, we saw that different kinds of data. For example, there is data which is of integer type and there is data which is of real type and so on.

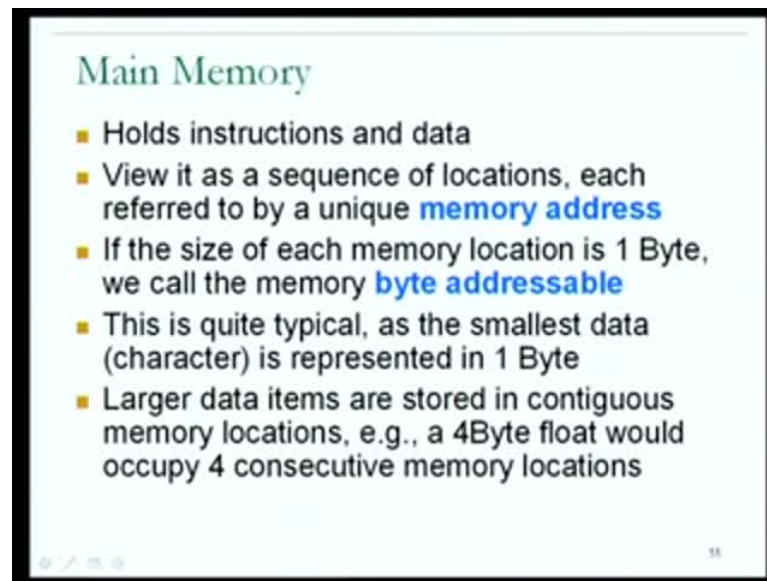
So, question which may arise is, are all the different kinds of data going to be stored in the same registers? And, in answer to this question, the designers of computers, sometimes, include separate registers to store integers and separate registers to store real values. Remember, that the real values are represented using floating point representation. So, the registers which are used to store, to hold real values might be called floating point registers.

So, in some processors, you might find out that, there are 2 collections of general purpose registers, one collection of general purpose registers in which integer data can be temporarily stored from memory and in another collection of general purpose registers into which floating point or real data can be temporarily stored from main memory.

So, there could be 2 sets, 2 or even more sets of general purpose registers. Now, with this, let us move on to, well before, before moving on, let me just, I may have raised your curiosity about what are the boxes could there be in a CPU? What are the components? And 2 of the components that we are going to see more about later - one is the cache and one is something called the MMU and we will be seeing about them later, but this is what I had in mind, in leaving some of the space blank, in this diagram.

Now, let us next, move on to the main memory. Let us learn more about the main memory. We are actually not going to learn too much more about the processor for the moment. Once we know a little bit about the main memory, we will come back and talk about the operation of the processor a little bit more.

(Refer Slide Time: 38:01)



So, remember, the main memory is that block diagram. It is a form of memory in which, the instructions and data of a program are stored. Now, the first thing which I would like to point out is that, we will view the main memory as being a sequence of locations, each of which is referred by a unique memory address. So, main memory is a sequence of locations. There is the first location, the second location and so on. Each of these locations has a unique memory address and we will refer to the address of the first location as 0.

Since the memory addresses are typically assigned as unsigned integers, so, we, the memory addresses will go from 0 on up. So, if I have a memory which is of size, a main memory which is of size 4 Gigabytes, then the first byte in the memory would be referred to by the address zero.

The second byte would be referred to by the address 1 and so on, until the last byte which would be referred to by an address, which is something in the neighborhood of 4 billion. We will have to calculate what 4G is. Remember that, 4G is going to be 4 multiplied by 2 to the power of 30 and not 4 billion. It is much higher than just 4 billion.

Now, so, we view memory, main memory as a sequence of locations and remember that, obviously these memory addresses are going to be important because, a given variable, like a integer variable x that my program uses, is going to have one memory location

associated with it and my program will refer to that variable by referring to that memory address or that memory location.

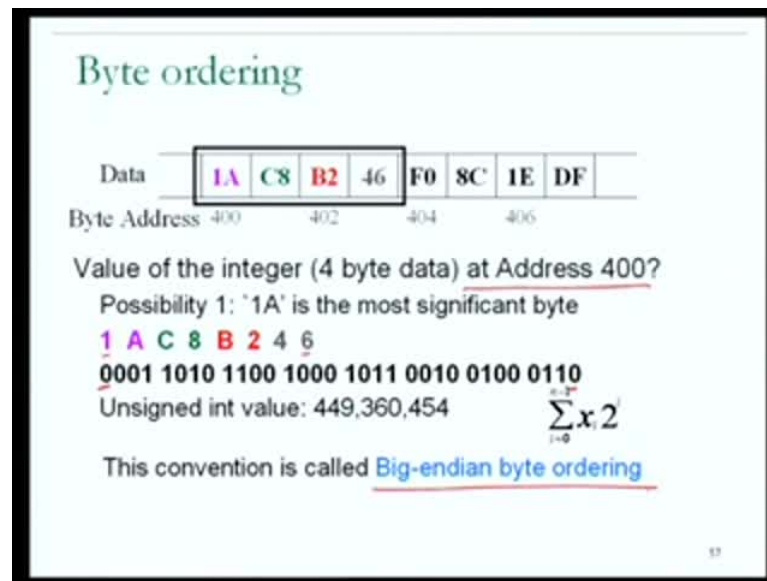
So, as a programmer I may have to know something about memory addresses. Now, one term which is often used in connection with memory systems, is the term byte addressable and the idea here is, now I said that each memory location has an address. So, if the size of each memory location is 1 byte, that would mean that, each byte of memory has a unique memory address and I would refer to that memory as byte addressable.

You should know that, it is conceivable that I could have a memory in which each collection of 4 bytes has a memory address. In other words, the size of a location might be 4 bytes and this would make sense if all of my data was integral data, integers or floating points.

But we know that, typically the smallest data that you could work with, that we have talked about, at least is character data and it is represented in 1 byte and that there are lots of situations, where programs have to manipulate character data. So, it might make sense to actually have byte addressable memory, so that, each byte of memory has a unique memory address.

Now, this raises the question of what about a 32 bit floating point value. We saw that, the real, real values are represented using the 32 bit, I triple E 754 floating point representation. So, that would mean that, a single real or floating point value would occupy 4 memory locations because, it is 32 bits or 4 bytes in size. All that this would mean is that, if I had a float x variable, a variable x which is declared as float of x, then, it occupies 4 consecutive memory locations, may be memory location 16 followed by memory, memory location 16, 17, 18 and 19, that might be the 32 bits in which, the float, float value is represented.

(Refer Slide Time: 41:46)



And so, that gives us some idea, of how data items which are larger than 1 byte could have addresses associated with them, but this has of course, raised an interesting question and that is and a question to which the answer is going to be byte, byte ordering. But, let me just frame the following question. Over here, what I have drawn, what I have drawn for you to see is a picture of a region of memory. So, I am showing you a number of memory locations and so, and as you would guess, the number which is below the value, which is in color, is the address of the memory location. Since, each of the pieces of data in the memory is of size 1 byte, I have shown each of the memory locations as being of size 1 byte and I am referring to the address of that location as its byte address.

So, this is the memory address. So, the memory addresses are shown in decimal. There is memory address 400 up to memory address 407. I am not showing you all of memory. So, memory obviously starts from memory address 0 to the extreme left, going up to memory address, a large memory address at the right side.

Now, next you will be wondering, in that 1 byte, the first byte, the byte which is, which has an address of 400, I have shown a value, but I have written it as 1A. By now, many of you would have guessed that, I am showing the values in hexadecimal. Remember the hexadecimal is the base 16 number system. In the base 16 number system, there are 16 different digits. They are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. F stands for the 50, for 15 and therefore, it takes 4 bits to represent one hexadecimal digit. Therefore, in each

of these boxes, each of these memory locations, I am showing you 2 hexadecimal digits which means that, each of them is of size 4 bits and the size of the 2 digits together is 8 bits or 1 byte.

So, I am showing you, 1 byte of information encoded in hexadecimal. So, that is what this example is showing. It is showing you a picture of a window, out of a main memory, between address 400 and address 407. Now, the question which I am going to ask is, what is the value of the unsigned integer at address 400?

So, this is actually, an interesting question, because, as it happens, it is easy for us to understand that, when I talk about the address 400, as I had mentioned earlier, I know that the unsigned integer of size 4 bytes which means that, it is going to occupy main memory from address 400 up to 403. So, that we know from what I mentioned a few minutes ago. But now, I have the question of trying to interpret the byte sequence 1 A C 8 B 2 4 6 one as an unsigned integer.

Now, as it happens, the answer to this question is that, there are a few possibilities of what the value of this unsigned integer is.

Now, let me just, mention what two of these possibilities are? Now, one of the possibilities, and I am sorry, these two possibilities depend on the significance that I have associated with these bytes. I used the term significance in a, in a previous lecture, when I talked about a least significant bit, the bit which has least weightage associated with it, as far as its contribution to the value and the most significant bit as being the bit which has the most weightage associated with it, in terms of its contribution to the overall value.

So, very clearly there are going to be few possibilities, as to how I could order these bytes, in terms of their significance, which is why we have the title of the slide byte ordering.

Now, here is the situation that we have. We have 4 bytes in memory and the question is, what is the value, the unsigned integer value of the 4 byte quantity at address 400, in other words, the address which occupies bytes 400, 401, 402, 403?

Now, one of the possible answers to this question is that, I could view byte 400, in other words, the value 1A as the most significant byte. It is the byte which contributes the most to the value, in other words, it is the most significant- it contains the most significant bits of the 32 bit unsigned int.

(Refer Slide Time: 41:46)

Byte ordering

1A	C8	B2	46	F0	8C	1E	DF
Byte Address 400	402	404	406				

Value of the integer (4 byte data) at Address 400?

Possibility 1: '1A' is the most significant byte

1
A
C
8
B
2
4
6

0001 1010 1100 1000 1011 0010 0100 0110

Unsigned int value: 449,360,454

$$\sum_{i=0}^{n-1} x_i 2^i$$

This convention is called Big-endian byte ordering

So, if this is the case, then the 1 A C 8 B 2 4 6 has 1 as the most significant byte and 6 as the least significant byte. I can expand each of these hexadecimal digits into binary, using the binary, hexadecimal to binary look up table, which I- I would not show it to you, but hopefully by now, you all remember that A, 1 will map to 0001, A will map to 10 which is 1010 and so on. So, I will end up with this 32 bit quantity.

So, this is the 32, if we, under the assumption that, 1 A is the most significant byte, the unsigned integer at address 400 will have this binary value. These are the 32 bits of the value, where this is the, this 0 is the most significant bit and this 0 is the least significant bit. I can now evaluate this unsigned int using the binary, using this summation, summation from i equal to 0 to n minus 1, where n is equal to 32, x sub i multiplied by 2 to the power i and what I come up with, is the value 449 (million) 360454.

So, under this assumption, the assumption that 1 A is the most significant byte, the answer to my question, is this value 449 360 454. This is only one possibility and this convention where I assumed that, the byte which is the, the address deciding byte, here I

was talking about the integer at address 400 and if my convention is that the byte at 400 is the most significant byte, then this is called the Big-endian assumption about byte ordering.

(Refer Slide Time: 47:42)

The slide titled "Byte ordering." illustrates a memory layout and a calculation for Little-endian byte ordering. It shows a sequence of bytes in memory: 1A, C8, B2, 46, F0, 8C, 1E, DF. The first four bytes (1A, C8, B2, 46) are grouped together, with 46 circled in red. Below this, the text asks for the value of the integer (4 byte data) at Address 400? and presents Possibility 2: If '46' is the most significant byte. The bytes are then listed in reverse order: 4 6 B 2 C 8 1 A. Below this, the binary representation is shown: 0100 0110 1011 0010 1100 1000 0001 1010. The unsigned integer value is calculated as 1,186,121,754. The slide concludes by stating that this convention is called Little-endian byte ordering.

Byte ordering.

Data	1A	C8	B2	46	F0	8C	1E	DF
Byte Address	400	401	402	403	404	405	406	407

Value of the integer (4 byte data) at Address 400?

Possibility 2: If '46' is the most significant byte

4 6 B 2 C 8 1 A

0100 0110 1011 0010 1100 1000 0001 1010

Unsigned integer value: 1,186,121,754

This convention is called Little-endian byte ordering.

Now, there is obviously at least, one other possibility and in that possibility, I view not 1A as the most significant byte, but I view 4 6 as the most significant byte. So, I am talking about the integer at address 400, but I view 4 6 as the most significant byte, which means that 46 B2 C8 and 1A are the bytes, in order of significance. 4 ends up being the most significant hexadecimal digit, A ends up being the least significant hexadecimal digit. And once again, I could expand this into binary using the hexadecimal to binary table. I come up with this value. Once again, I can evaluate this as, using the summation and I come up with an unsigned integer value which is different from the first value.

So, remember this was what happened, if I assumed that the bytes were ordered with the, **the**, the byte which defines the beginning as being the least significant byte and this convention is known as the Little-endian byte ordering.

So, once again, this is just a convention and I could build a computer which uses either of these conventions. It may be a little curious as to where these terms Little-endian and Big-endian came from? As it happens, they were, this is a somewhat recent development.

It was, I think, something that happened sometime in the 1980s, this terminology, but it is basically derived from a book, a book in which, in fact Gulliver's Travels.

So, with this, the question that you might ask is, I talked about these two possibilities, are there any others? And technically, you might say that where there are 4 bytes, the **by[tes]**- 4 bytes can be ordered in many other ways. For example, I could arbitrarily say that, this is the **mo[st]**- 402 is the most significant byte. 403 is second most significant, 400 is third most significant and 401 is the least significant byte.

(Refer Slide Time: 49:36)

Byte ordering..

Data	1A	C8	B2	46	F0	8C	1E	DF
Byte Address	400	402	404	406				

Value of the integer (4 byte data) at Address 400?

Big-endian ordering	Little-endian ordering
449,360,454	1,186,121,754

Some machines are built to use big-endian byte ordering and others are designed to use little-endian byte ordering

This can be relevant to the programmer

So, there are various other possible byte orderings, but the two most frequently used byte orderings are the Little-endian and the Big-endian byte orderings. So, you will note that, these two are different. They result in different interpretations of the value, the unsigned integer value at address 400. Under the Big-endian byte ordering, for this same, for this picture of memory, the value of 449 million plus whereas, under the Little-endian byte ordering, the value was, one, was much more. And as I had mentioned, I could build a machine which uses the Big-endian byte ordering convention and somebody else might build a machine which uses the Little-endian byte ordering convention. And the question to ask as programmers then is, does this affect us, is this something that we should be concerned about?

Now, the answer to this question is that, if you are just writing programs to run on one machine, then this is actually not a concern to you at all, because, your program will run on that machine and whether internally Big-endian byte, Big-endian ordering is used or Little-endian ordering is used, it does not matter, because all values will be interpreted using the same convention.

But there can be scenarios, in which this is relevant to the programmer and let me just quickly describe one, which we will be encountering much later in this course, when we talk about parallel programming.

Now in parallel programming, I write a program in such a way that, it runs in part on one processor and in part on another processor. In other words, I divide the computation so that, it runs on more than one processor and the two parts of the program may have to interact. Now, if one of the processor is using Big-endian byte ordering and the other one is using Little-endian byte ordering, then I may end up having a little bit of a problem. Or, let me think, describe this to you in another way. Let us suppose that, I have a program which is written to execute on a collection of computers on a network and that some of the computers on the network use Little-endian byte ordering and some of the others use Big-endian byte ordering. Then once again, if I have a program which is running across more than one processor, then this may end up being something that, I have to be worried about.

Now, in the, in one of the previous lectures, I had given this idea that, often it is possible to write a program, an experimental program to run an experiment on a computer that you have, in order to learn a little bit more about that computer. And this is clearly a situation, where one could try such an experiment.

So, the question that arises is, you have a desktop or a laptop computer with you. And now, that you have learned about this concept of Little-endian and Big-endian ordering, you may be wondering does my laptop or desktop used Little-endian ordering or Big-endian ordering? And one way to find out would be, to look up the manual or to read about the processor, read about the details of that particular processor on the internet, something of that kind, through a manual. But, a more direct mechanism would be for you to try to write a program, a small C program, which could determine whether the

machine, on which it is running, uses Big-endian byte ordering or Little-endian byte ordering.

(Refer Slide Time: 49:36)

Byte ordering..

Data	1A	C8	B2	46	F0	8C	1E	DF
Byte Address	400	402	404	406				

Value of the integer (4 byte data) at Address 400?

Big-endian ordering	Little-endian ordering
449,360,454	1,186,121,754

Some machines are built to use big-endian byte ordering and others are designed to use little-endian byte ordering

This can be relevant to the programmer

And, with a little bit of thought, you realize that, this may not be that difficult. Let me just give you a rough idea. Let us suppose that, I have a signed integer, right. So, I know that, the signed integer possibly occupies 4 bytes of memory. In other words, it is of size 4 bytes, if it is a 32 bit integer.

So, if it is a signed integer and I know that it is represented using the 2's complement representation, then I would know what to expect in terms of the most significant byte. Therefore, if I wrote a program in which, there was a declared signed integer variable x and then I put, let us say the value minus 1 or minus 3 into that variable, using an assignment operation and then in the rest of that program, I examined the 4 bytes at the address associated with that variable x, then I could actually figure out whether the bytes are being interpreted in Little-endian or Big-endian fashion.

So, a program could be written to determine whether the machine on which it is running uses Little-endian or Big-endian byte ordering. Now, another fact which I will just mention at this point in time is the following. Many of the processors which are made today are actually designed so that, the choice of whether they use Little-endian or Big-endian byte ordering can be changed, in other words, they are configurable.

So, somebody, who is knowledgeable enough, could actually reconfigure the processor, so that, the next time that it is booted up, it actually uses the opposite byte ordering, about what it was using. This, today it might use Little-endian byte ordering. Few days later it might use Big-endian **byte** byte ordering. So, this possibility is also there - processors which can be configured to use either Little-endian or Big-endian byte ordering, right.

Now, with this we have, in this lecture, I will just quickly wrap up for today. What we have seen is that the components of the computer system, well actually, the processor, the memory and the I/O devices. We spent a little bit of time looking at the processor in more detail. We understood that the control hardware is hardware that manages the execution of instructions. The ALU is circuitry that can do arithmetic and logical operations. Registers are both, either general purpose or special purpose. The special purpose registers are used for very specific activities by the control hardware. They are not available to the programmer for direct use, whereas, the general purpose registers are available to the programmer or on the programmer's behalf to the compiler for using frequently accessed variables. And, this might be important because of the large speed disparity between the main memory, where the instructions and data are stored, and the processor, where the instructions are executed.

We saw more about memory, the different kinds of memory that are present in a computer system and finally, we spent some time looking at the mechanics of main memory, where main memory we view as, being a sequence of memory locations, where each memory location has an address, which is an unsigned integer. And, we understood the notions of byte ordering in terms, it being a either Little-endian or Big-endian. Thank you.