Digital Computer Organization

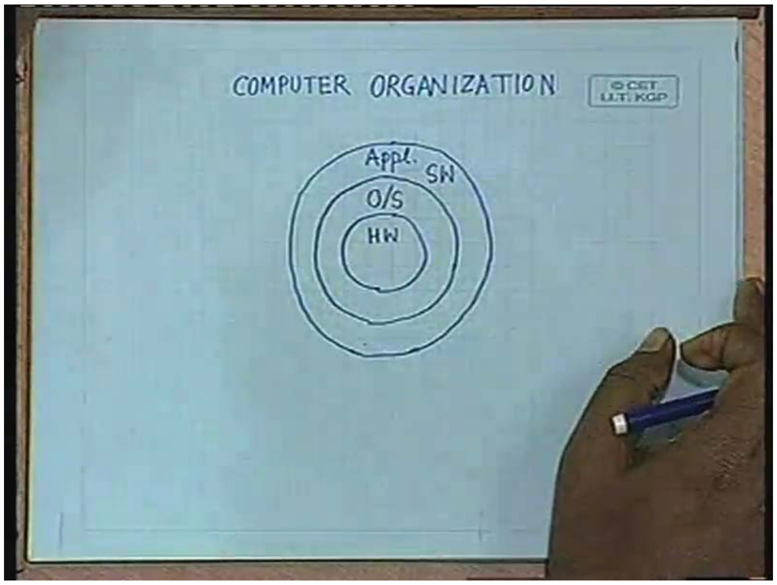
Prof. P.K. Biswas
Department of Electronic & Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture No. # 01

Introduction to Digital Computer Organization

Welcome to the course on computer organization. In this course we will try to find out that what are the resources that are available in a computer. Specifically the hardware resources and how those resources are interconnected or how the data path that go from one resource to another resource to accomplish a particular task. Now to start with let us have a high level view of a computer. As you all know that to get a task done by a computer, the computer must have some hardware resources and some software resources.

(Refer Slide Time: 00:01:42 min)

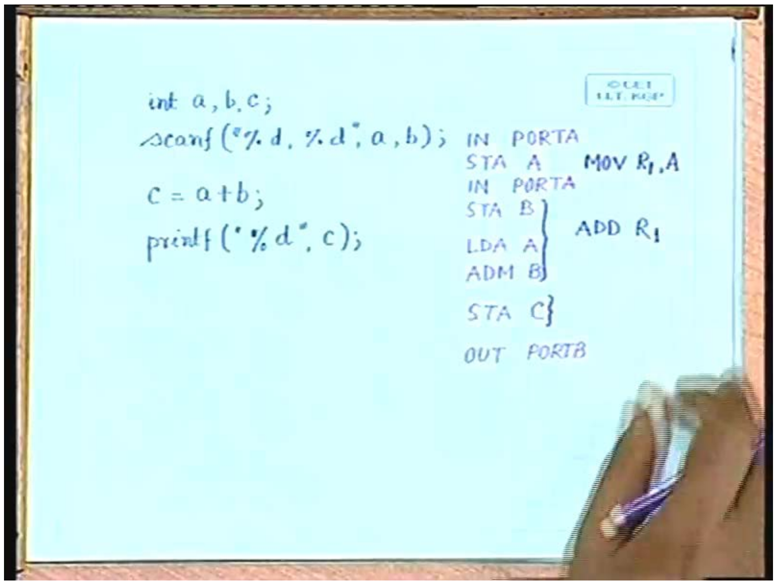


To get a high level view at the heart of the computer, we have the hardware resources of the computer. So I can put it like this. So at the center of the computer we have the hardware. On top of hardware we have a layer of software which is called operating system or OS and above this operating system we will have another layer of software which are basically the application software's. So here we have application software's. So whenever a program is to be executed, the program that we write that remains in the applications software level. Then as the program is to be executed then it is an operating system which gives an interface between the application level software and the hardware resources of the computer.

So basic job of the operating system which is in between the application level software and the hardware resources of the computer is to manage the hardware resources so that the same hardware resource can be used by one or more application level software's at the same time. Now in this particular course we will concentrate on the hardware resources. We will not talk much about the operating system or application level software.

So now to see that what are the hardware resources that we need in a computer let us take a very simple software example. So if I write a program like this, the program will read two variables, add the two variables and output the sum. So I can write a very simple software in a high level language like this.

(Refer Slide Time: 00:03:51 min)



```
int a, b, c;
scanf ("%d, %d", a, b);     IN  PORTA
                            STA  A      MOV R1,A
c = a+b;                    IN  PORTA
                            STA  B }
printf ("%d", c);           LDA  A }    ADD R1
                            ADM  B }
                            STA  C }
                            OUT  PORTB
```

So I will put scanf, so let us say have a very simple program like this. Scanf a b where the values of the variables a and b will be radiant then we perform the addition of these two variables a and b and the result is assigned to variable c and then finally we would like to print the value of c. Of course before these we have to define or we have to declare the variables a, b and c. So let us declare the variables a, b and c as integer variables. Let us try to see that how these high level program will be executed by the computer.

Now as you all might be knowing that a computer is not capable of executing this high level language program. So in between what you have to do is we have to compile this program and we have to create the executable code of the program and only the executable code can be executed by a computer. So in between actually how the computer executes this high level program, to explain that let us convert this high level program into an intermediate code. Let us call it an assembly level code. So in case of assembly level code, this high level instructions and you can see that these are the instructions of a C language. So let us try to see that how this C language code is converted to assembly language instructions. So the first operation that is done is scanf percentage d, percentage d this just tells you that what is the format of the data which will be read in and the variable names are a and b.

So if I convert this to assembly level instructions, this particular high level instructions will be converted to a form like this, IN from some PORT. Let us name the PORT as PORTA, so from PORTA some data will be read in. The data will go to some register within the CPU of the computer. What is this register that will come later on or this register is usually called the accumulator of the CPU and once the data comes to the accumulator then from the accumulator, the data is stored in a location, the location which is assigned to choose the variable a. So after reading in the data, this data will be stored in location which is assigned to location, the variable A. So once you have read in the data for A and you have stored it to the memory location, the next data that has to be read in is the value of this variable b. So similar such operation will be performed. It will performed in for inputting the data, let us assume that these data will also come from the same PORT, so PORTA and after reading in data the data will be stored in variable or the memory location which is assigned to variable B. So these four assembly level instructions perform the job of this high level instructions scanf. So we find that a single instruction in a high level language is actually converted to a number of instructions in assembly level.

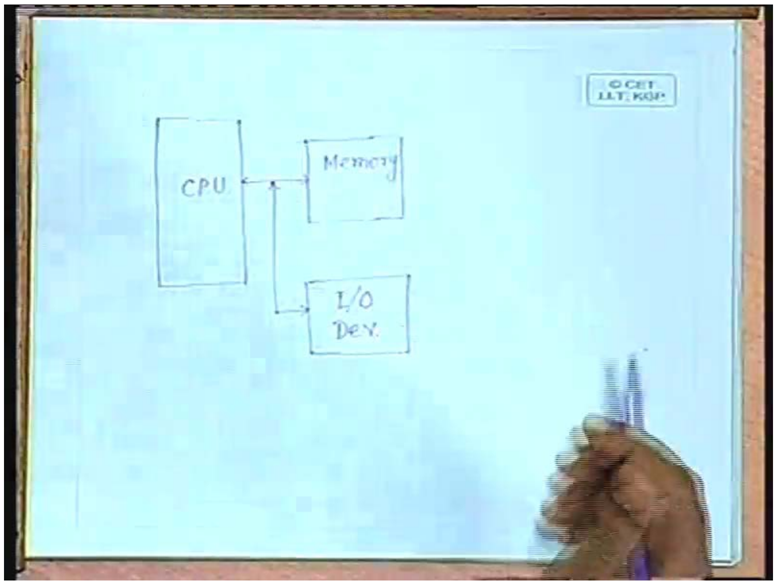
Now after performing this, what we have do is we have to perform the addition of the values of the variables a and b and after completing this addition operation, the value has to be saved in the memory location which is assigned to variable c. So for doing this once the values of a and b are saved in the memory, we have to get this values from the memory and only after reading these value from this memory, this particular operation of addition of these two variables can be performed.

So to read the values of these variables from the memory, the operation which is performed is load from the memory location into the accumulator of the computer. So we will put it as LDA then A. So from the memory location which is assigned to variable A, the data will be read in and it will be put in the accumulator of the CPU of the computer. So once I get this data in the CPU then another operation that I can perform is say add memory. So the content of the memory location B has to be added with A, so I will put it as add memory B. Here we assume that after addition of these two values of A and B, the result will be available in the accumulator. So the next operation that we have to perform is because this statement says that the result of this addition has to be saved in the variable C. So we have to perform same store accumulator operation but this time, the value will be stored in the location which is assigned to variable C and at the same time, the printf command it says that we want to get print out of these value of C which is also available in the accumulator.

So the operation that we will perform is out, some other PORT say PORTB. So we find that we have read in the values of A and B from PORTA and we have printed the value or outputted the value to PORTB. So this simple instruction or set of instructions in assembly language tells you that what may be the resources which are necessary in the CPU of a computer or in the computer system as a whole. So from this you find that for this in operation, the data is read on PORTA and it is saved in the accumulator. That means the CPU of the computer must have a register called an accumulator.

Similarly we are saving this data to a memory location which is allocated to this variable A. So which again tells that we have to have some memory associated with the computer and along with this register in this particular case it is accumulator and the memory unit. We also have some input output device because you find that the values of the variables are inputted from a PORT. Similarly values of the variable C are outputted to another port. So as a whole in a computer we must have a CPU which executes the program. We must have at least one register and we will see that one register may not be sufficient. We will have more than one registers out of that one particular register is a special purpose register called accumulator in this particular case and we also have to have some input output devices.

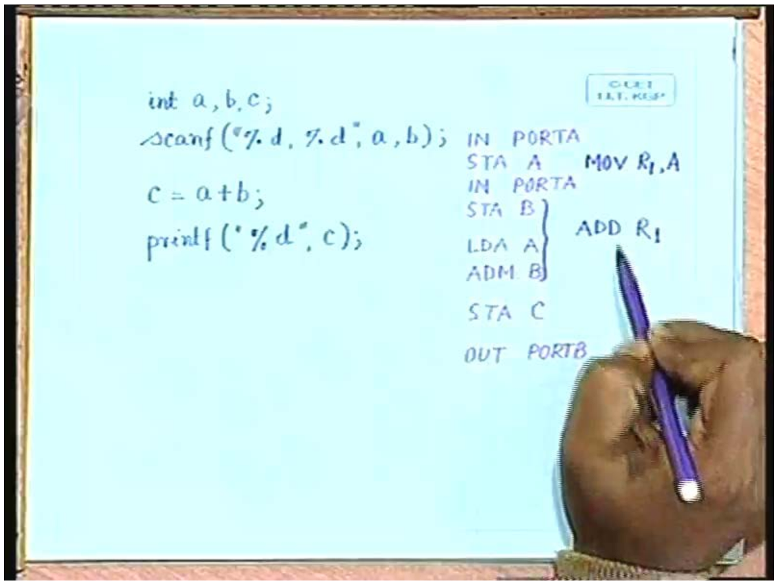
(Refer Slide Time: 00:13:05 min)



So at the high level view a computer must have the following hardware resources. We have to have a CPU or center processing unit. We must have some memory and we also must have some input output device or in short it is called I/O device. So I stands for input and O stands for output and we must have some inter connection between the memory and CPU. We must have some inter connection between the I/O device and the CPU. Now find that I have a put these links or these paths as bidirectional paths because the data may come from the memory to CPU. In case of reading a value from the memory location, the data may also move from the CPU to memory in the case of writing a data from the CPU to memory location.

Similarly if you wants to read a data from an input device then the data moves from the input device to the CPU and if you want to write a data to the output device then the data moves from the CPU to the output device. So this is the basic high level view of a computer. Now we will see in details that what are the components inside the CPU, what should be the architecture of the CPU, what should be the nature of the I/O devices in our subsequence lectures. Now today to give an overview that what are the contents in the CPU or what are the resources which are available in the CPU.

(Refer Slide Time: 00:15:14 min)



```
int a, b, c;
scanf ("%d, %d", a, b);     IN  PORTA
                            STA A       MOV R1, A
c = a+b;                    IN  PORTA
                            STA B }
printf ("%d", c);           LDA A }    ADD R1
                            ADM B }
                            STA C
                            OUT PORTB
```

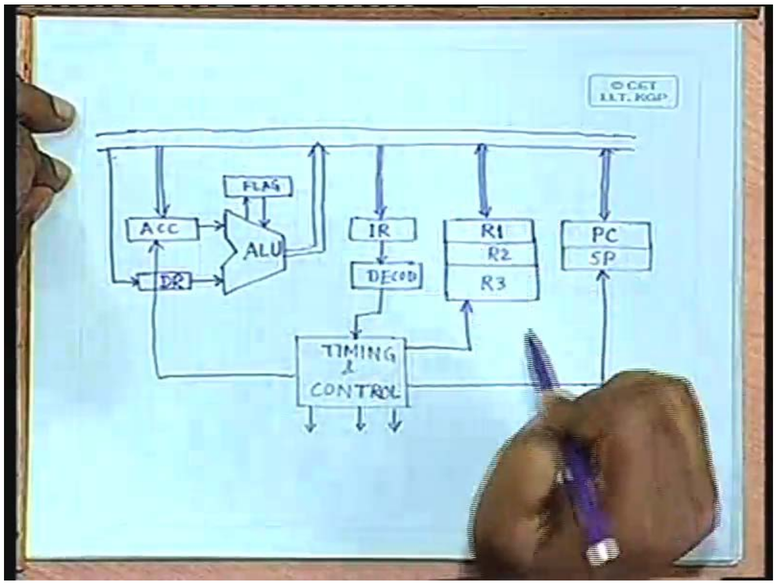
Now for that what we have done is coming back to our, this initial program, the assembly language program we have said that when the data is read from the PORTA, the data is saved in the accumulator. So accumulator is one of the registers and next operation that we have performed is from the accumulator we have returned the data to a memory location A. This is a memory location which is allocated to variable A. Similarly the next operation, from PORTA the data is read in the data goes to the accumulator then again from the accumulator the data is return to register B.

Now we have to perform this write back operation to the memory because we have assumed that we have only a single register in a CPU till now. Now let us say that what advantage we will have, if we have more than one registers. So in addition to the accumulator, if I assume that there is one more register let us name that register as register R_1. Then instead of this store accumulator to the memory location A, here I can simple have an operation like MOV R_1, A. It says that the data is moved from the accumulator to register R_1, R_1 is an additional register within the CPU.

Similarly in the next case when you are imputing the data from PORTA, the data goes to the accumulator and you find that the other data is already available in register R_1. So I don't need these operations at all. These three operations can be replaced by simple ADD R_1 because the value of the variable B it is already available in the accumulator which is done by this second input statement that is input A and the value of the variable B is already available in register R_1 because we have moved the value of variable A from the accumulator to register R_1. So by doing this the advantage is I can perform the same task using less number of instructions and not only that this operation of moving the data from accumulators to memory has been replaced by the operation of moving the data from the accumulator to another register R_1. Here because both these registers are available in the CPU, the time taken to perform this data movement can be much less than the time taken to move the data from the accumulator to memory which is actually a memory write operation.

Here again after performing this addition operation, you find that this the sum is already available in the accumulator. So this STA C which of course in the previous case also was not necessary, here also it is not necessary until and unless we want to use the value of C for some other purpose. So here we find that instead of having a simple register called accumulator, if I have more than one registers in that case I can write my program or the program can be executed in a more efficient manner. The other component so this other registers, the other component that you need is an unit which performs this addition operation. Here it is add with memory, here it is add within the registers. So this particular unit which will perform the addition operation or the logical operation is called an arithmetic logical unit or ALU.

(Refer Slide Time: 00:19:47 min)



So in a CPU I must have an ALU which is symbolically represented like this ALU or arithmetic logical unit. I have an accumulator, I have to have some more registers or general purpose registers for storage of the data. So I can have register R one as we have just seen. I can increase the number of registers as you increase the number of registers, you can write more and more efficient codes. So I can have a number of registers R_1, R_2, R_3 and so on. So if we increase the number on registers in the CPU, I can write more and more efficient code but of course there is a limit. I cannot have infinite number of the CPU. So I will have a finite number of registers not just one, more than one but having a reasonable number of registers.

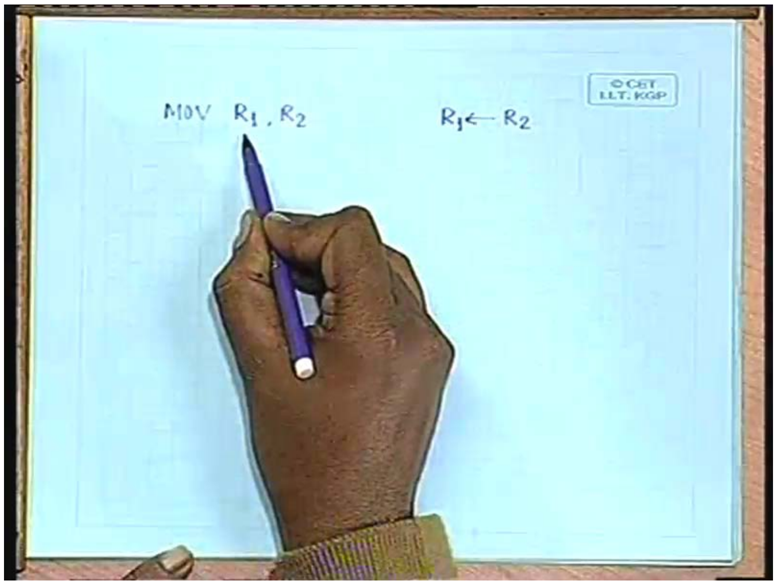
Now find that coming back to this particular instruction. After each of these instructions are executed, the CPU must know that what is the next instruction to be executed. Now how these instructions are actually stored in the memory. each of these instructions will be converted to a machine language code which is a binary number sometimes represented as hexadecimal number and those instructions will be read or machine level instructions will be read from the memory. The instructions will be decoded and corresponding action will be taken. Once a particular instruction is executed then the next instruction has to be executed. So I must have a resource which acts as a pointer to the next instruction which I must execute. So that particular resource in a CPU is called a program counter. So in addition to these general purpose registers I also have to have another register within the CPU which is called a program counter or a PC. I may also need some more registers. Say for example when you execute a program and if you write a structured program, in that case you may find that for execution of a particular task you have to call a function.

So that function can be written as a module of software and when you call any particular function, in that case all the previous intermediate results that were generated, those results you has to save somewhere. Similarly if I call a function in that case the previous program counter value, it also has to be saved somewhere because whenever I call a function, the program counter because it is pointed to the next instruction to be executed, this program counter must be loaded with the first address of the first instruction in the function which is called. So the previous value of the program counter must be saved so that when execution of that particular function is complete and I come back to my main routine where from the function was called, so I must be able to restore the value of the program counter, the previous program where from the function was called.

So to save all these information's, I must know that where in memory this information's are saved. So these information's are saved in some memory locations which are called stacks and the addresses to those stacks are given by another register which is called a stack pointer or SP. So here we find the program counter points to an instruction which is to be executed next and stack pointer points to a stalk where we saved some values of some of the registers which can be called back when we come back from the subroutine or when we come back from the function which has been called from memory.

Now another important part of the computer is that instruction you have interpretation. Before execution of the instruction, the CPU must know that what that instruction is supposed to perform. That means the CPU has to decode the instructions. So if the CPU has to decode the instruction then the up code, the code of the instruction which is the machine level code has to be read from the memory and it is put in another register in the CPU which is called an instruction register. Now from this instruction register, this instruction code goes to what is called an instruction decoder. Instruction decoder decodes the instruction and after decoding this instruction then only the CPU knows that what this instruction is supposed to do.

(Refer Slide Time: 00:26:01 min)



So before coming to further details let us assume, let us take a very simple instruction like MOV say R_1, R_2. So what this instruction is supposed to do? R_1 and R_2 are two general purpose registers in the CPU and this instruction will move the data from register R_2 to register R_1 and this is the convention which is mostly used that the first operand that we mentioned in the instruction, that is the destination and the second operand that you mentioned in the instruction that is the source. So this operation will simply move the data from register R_2 to register R_1. So for performing this particular operation because I am moving the data from register R_2 to R_1 that means this register R_1 must have a control input called load or latch and the register R_2 must have a control input which is called output enable.

So when this output enable of register R_2 is active, the data from register R_2 is ready to be transferred to some other destination. When you make the latch input or load input of register R_1 active in that case whichever data is available to the input that is loaded in to register R_1. So for performing this particular operation, I have to activate the output of register R_2 and at the same time I have to activate the load input of register R_1. So these two are called the control signals, one is controlling output of the register R_2. The other one is controlling the input of register R_1. So all these registers will have defined such control units and this control signals are to be generated in proper way so that the specified task can be executed.

So within the CPU I must have another unit which is called a timing and control unit. So let us put it like this. We have a timing and control unit. So this timing and control unit, it generates all the control signals in a proper sequence so that a specified task can be a executed. You find that the task which is to be executed that is dictated by the instruction which is under execution and that is interpreted by this instruction decoder. So this instruction decoder, after decoding that instruction gives some signal to the timing and control unit and after getting this signal, the timing and control unit generates all the control signals in a proper sequence so that the specified task can be executed.

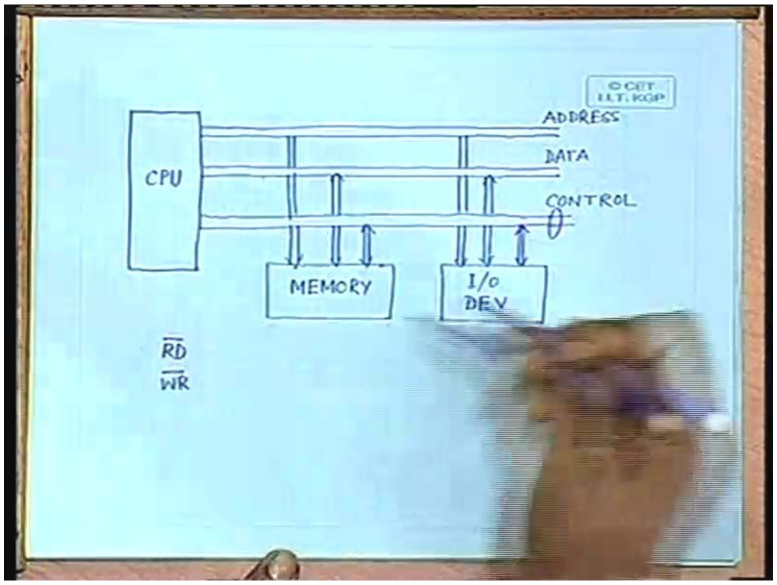
So you find that all the units in the CPU they get the control signals from the timing and control unit, all of them. So there are various such control signals which are generated by timing and control unit and these control signals are distributed to all other units in the CPU.

Now in many case you find that you have some instruction or jump instructions. The jump instruction you decide that whether to take a jump to some other instruction or whether to call a subroutine, these decision depends upon the result of computation. Say for example we gave an instruction like jump with carry. So after doing some arithmetical logical operation, if a carry is generated in that case the next instruction to be executed is taken from some function or from some other location in the simple program. It is not the immediate next instruction which comes out after jump. So I have to have some unit or some resource in the CPU which can save the status of CPU or what is the nature of the output after execution of any arithmetic logic unit. So these resources are called flags which saves the status of execution of some arithmetic or logical operation. These flags are also sometimes known as processor status lot. Now these other resources that I need in a CPU for execution of any particular task.

Now in addition to the resources, I also must have some agent which can move the data from one resources to another resource. So I have to have a data path using which the data can be moved from one resource to another resource. So all these units whether these are general purpose registers of the CPU, whether this is instruction register of the CPU, whether these are program counters or stack pointers, whether this is accumulator and of course the output of the arithmetical and logical unit, all of them are contacted to this data path. Now coming back to this ALU, the ALU for operation in most of the cases needs two inputs. So one of the input to the ALU comes from the accumulator and the other input to the ALU say for example coming to this particular operation add R_1. Here one of the operands is already in the accumulator and the other operand is available in register R_1. So from register R_1 I have to bring the data to some location so that from there it is fit to the ALU. So this particular register in some of the CPUs is called a data register or DR. So one of the inputs to the ALU is given from the accumulator, the other input to the ALU is given from this data register DR and obviously this data register also should have a connection to the data path.

So in general this is the internal architecture of any of the central processing unit. Now next part which comes is that once I have this internal architecture of the CPU and if I take this CPU as a block then what I have to see is how this CPU has to be interfaced with the external memory, how this CPU can be interfaced with the input output devises. So for that some of the interfacing signals which are taken out of the CPU.

(Refer Slide Time: 00:34:35 min)



So if I represent this CPU by a single block, so this is the central processing unit. It will have a number of pins using which we can interface the CPU with external memory or we can interface the CPU with input output devices. A state of such pins are called the address lines or address box. A state of such pins is called the data lines or data box and we have some more state of pins which actually generates or provides the control signals. So these other lines which are available for interfacing the CPU with external memory and also to interface the CPU with input output devices. So we can put a memory block, we can also put an block for input output devices. So we have this block of memory, we have a block corresponding to input output device and for interfacing this memory block and input output device to the CPU, we have to make use of this address lines. We have to make use of this data lines. Similarly for this I/O device we have to make use of the address lines. We have to make use of the data lines and we will also have to make use of this control signals.

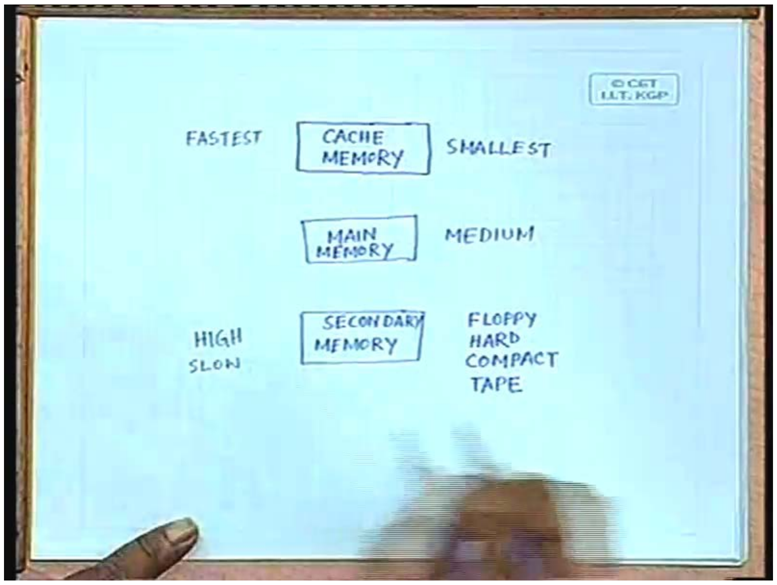
Now here we find that I have put this address lines as unidirectional or else the data lines I have put as bidirectional. The reason is whenever the CPU wants to read something from any memory location, it is the responsibility of the CPU to specify the address of the memory location from where the CPU wants to read the data. Similarly whenever the CPU wants to write some data to any of the memory location, it is the CPU which will specify that what is the address of the memory location to which the CPU will write the data. Similarly for input output devices. So addresses are generated by the CPU, address is not generated by the memory neither it is generated by input output device or else in case of data if the CPU wants to write the data to memory in that case the data will move from the CPU to memory or if the CPU wants to write the data to any of the output device, the data will move from the CPU to output device. Similarly if the CPU wants to read data from any of the memory then data will move from memory to CPU or if the CPU wants to write any data from an input device, the data will move from the input device to CPU. So this memory lines, data lines they are bidirectional whereas the address lines are unidirectional.

Now coming to this control lines, what are this control lines that we would like to have in case of CPU, control lines which are to be generated by the CPU? You find that we have two basic operations that are to be performed. there are other operations as well, we will come to all of them in our subsequent lectures but here let us see that two basic operations with the memory or with the I/O devices which are performed by the CPU is one of the operation is read operation. That is read a data from memory or read a data from one of the input device. The other basic operation that you have to perform is writing the data to one of the memory location or writing the data to one of the I/O devices. Now to identify what kind of operation is to be performed whether it is read operation or it is write operation, the CPU has to generate the control signals. For read operation the control signal which is generated is a read control signal. In most of the cases this comes in negative logic or we put it as read bar. For write operation the control signal which is generated is write control signal. Again it is given as a negative logic that is write bar.

So here if read line is low the control signal, read control signal generated by the CPU is low. In that case the CPU wants to perform a read operation. If the write bar line is low in that case the CPU performs a write operation. Obviously both read bar and write bar cannot be low simultaneously because in that case what operation has to be performed whether it is read operation or it is write operation that is not defined. So you find that whether the read bar signal has to be low or the write bar signal has to be low that is decided by the CPU and it depends upon the instruction that is going to be executed. So as we have said that in case of CPU, we have a timing and control unit. It is this timing and control unit which will generate the corresponding control signals after it gets an information from the decoder that which particular instruction it is going to execute.

So this timing and control unit or timing and control circuit in the CPU plays a very very important role. In fact if this timing and control circuit does not perform properly, the entire operation of the CPU will be totally erratic. So this just tells you that what is the basic structure of a computer. So what basically we have defined here is the organization of a basic computer. So in a basic computer we must have a CPU because without CPU, the computer is meaningless because it is the CPU which performs every task. We must have some memory unit interfaced to the CPU or the data can saved or the data can be read from the memory and we must have some input output devices to feed some data to the CPU for processing or to get the output from the CPU or the processed data from the CPU. So when we have these three interfaced together, what we have is a basic computer. Now coming to the memory you all might been aware of that we have various kinds of memory. We have cache memory, we have primary memory and we have secondary memory. So this memory is actually a hierarchy of different types of memory.

(Refer Slide Time: 00:43:18 min)



So at the topmost level, the type of memory that we have is called the cache memory. At the next level we have the main memory and we have at the next level in the hierarchy what is called a secondary memory. So all the disks say for the example floppy disk or hard disk or compact disk they are all in this category of secondary memory. We can also have other kind of secondary memory like magnetic tape. The hierarchy is like this, whenever the CPU wants to access something, it first tries to find whether that particular data is available in the cache memory or not. If it is not available in the cache memory then CPU tries to find our if it not available in the main memory.

Now this cache memory and main memory, the CPU has direct access over the cache memory and main memory but the CPU does not have direct access over the secondary memory. So if the data is neither available in the cache memory nor in the main memory then data has to be read from the secondary memory and then it has to be put in main memory or cache memory and then only the CPU can access it. Now size wise the cache memory is the smallest among all of them. Main memory size is medium and secondary memory is high. Speed wise cache memory is the fastest, main memory is medium and secondary memory is very slow. So we find that the nature of access of these different kinds of memory are very much different because the CPU has direct access over the cache or it has direct access over the main memory. So these memories, from these memories the data can be accessed as well as data can be written byte by byte. Whereas for secondary memory the data has to be accessed or written in the form of a block. The CPU, the data from the secondary memory cannot be accessed byte by byte or character by character.

So this is about the memory hierarchy. we will elaborate on each of these topics in our subsequent lecture and this particular CPU architecture or CPU organization that have told, this forms the basic of one of the simplest CPUs which is very popular which is known as 8085 manufactured by Intel. Per this registers R_1, R_2, R_3 they are represented as register b c d e h l and so on. So we will elaborate on each of this topics in our subsequent lectures.