Computer Organization
Part – I

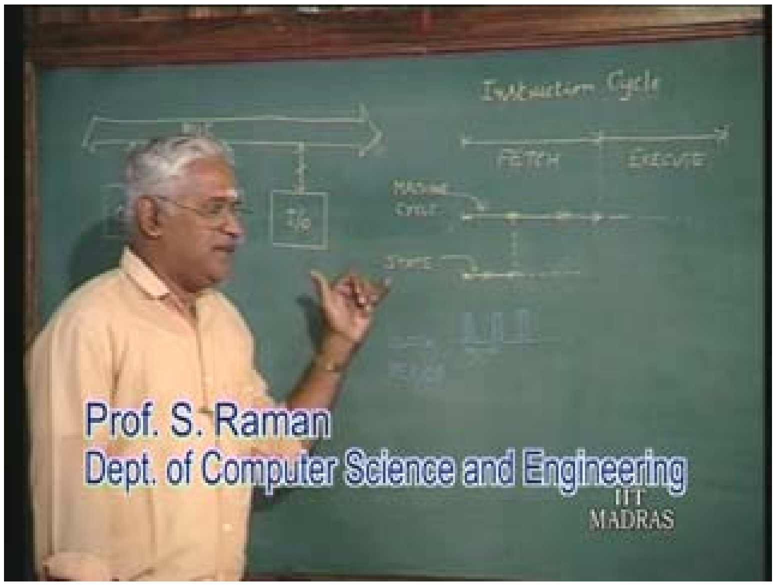
Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology

Lecture – 4

Processor Activities

In the previous lecture we were going into the details of the instruction cycle – this is an instruction cycle consisting of machine cycle, which in turn consists of state or a low level activity as I said.

(Refer Slide Time 00:01:19)



This state is nothing but the one defined by the clock period, the basic clock of the processor. You can see that in each clock period, there is some minimum activity that is going on; that in fact is the state of the processor. That point to us that the study of processor is essentially the study of a state machine in general. We will come to that a little later.

How did we start? We said the overall computer system consists of the CPU, memory and I by O; and then interconnecting these, we have a set of signal lines that form the bus. So far, we got an overview of the entire computing system and also learnt something about each of these blocks; that is, CPU, memory and then I by O. I was also giving overall generalized information about each of it in detail. Now the time has come for us to go into the details of each of these blocks; in fact we can get started with the CPU.

Though we may identify each of these as CPU, memory, etc., we cannot study them in isolation. Some kind of interaction will be there; but first, the focus will be on the CPU and later on the memory, and in between, we will also focus on their interaction.

This is the design of the CPU, what we are going to take a look at, though not in detail, so that we will be able to understand what a processor is doing in the overall system. Towards the end of the previous lecture, what we started discussing the instruction cycle. In this lecture as well as few lectures to follow, we will be going in to the details of processor: what it is and so on. At least we know that a processor is a state machine; it is, in fact, an example of a digital system which is a state machine, because it has to go through state after state as defined by the clock period and then it will be executing its overall function. What is that overall function? Essentially, it is to execute instruction, mainly because the processor executes a program and the program essentially consists of instructions and data. We have talked about this earlier too. These instructions and data are stored in the memory. In other words, what we have is the stored program that is what essentially a digital computer is. That is, the program is stored and the computer takes this particular program and keeps executing it. In this, essentially we have identified that the instructions are the ones which tell what operations are to be carried out, and on what operands the operations must be carried out – these are the two parts.

Now these instruction and data form the program, and that program is stored in the memory. That, in fact, points to what we know as von Neumann architecture, because it was a mathematician by the name John von Neumann, who had thought of storing a program in the memory and executing them. In fact, even today, for instance 95 98 95 of the computing systems are based on this architecture, that is, Von Neumann architecture, in which we have the memory in which the program is stored and that is executed. Later on, we will also talk about a few other architectures, which have been derived from this particular main thing. Though this series of lectures is not on architecture, it will be interesting for us to know some details about this too.

Now let us go back to the program consisting of instruction and data, where the program is executed because some data will have to be processed and the results must have to be delivered. So instruction after instruction is given and the program is executed. That means the processor is going through a series of instruction cycles; what it does is it is fetching the instruction and executing the instruction. I already mentioned that in the execution of the instruction there will also be data fetch. Data fetch or instruction fetch will be similar, because both are stored in the memory. So in this execute phase, you may have data fetch also; because the program is stored in the memory and the data also stored in the memory and so the CPU fetches an instruction or when it fetches the data it is behaving in the similar way. The CPU will address the instruction; the CPU may later on address a location which contains the data. So CPU addresses memory and the memory responds – this is what we are talking about. In one case, that is, during instruction fetch case, the memory will be responding with an instruction, and if there is a need for fetching the data or the operand, it will take place during the execute phase.

The memory will then place the data, address of data or the operand, and the memory will respond. The CPU will address that particular location, which contains that data. So each time, whenever the CPU accesses the bus and makes the demand, we said there is an emission cycle.

Whenever there is a bus activity – bus activity is essentially for fetching an instruction or fetching some data – both are stored in the memory. So whenever CPU accesses the bus for fetching instruction or for fetching data, there is a machine cycle. During the machine cycle, the processor goes through a series of states. As shown here, this fetch phase of the instruction cycle consists of three machine cycles, which means essentially the processor, for fetching these instructions, makes use of the bus three times; that is, it is fetching the instructions from three memory locations. As shown here, for example, this particular first machine cycle consists of two states – basically the processor is going to make use of two clock periods or two basic activities, and we will see what these are in detail later.

For doing this particular thing, we say the processor enters state one and then it enters state two. So essentially, the CPU can be looked at as a state machine; that is, it is going through a series of states. That, in fact, is the basic concept of any digital system. What is the processor CPU? It is an example of a digital system. According to the Von Neumann architecture, we are considering memory as a single block, in which both the instructions and data are present. Later on, we will see how this memory can be split into different blocks and then different architectures also will keep following. When it comes to the state machine, that is, where the basic activity is going on, it may be as simple as moving contents from one place to another within the CPU.

Memory may be organized so that it consists of some locations, each of which can be uniquely addressed. So there is some unit of data that is present there. Normally the memory stores a word; the word will consist of a sequence of binary digits, 0s and 1s. But be careful whenever you come across the term word – it is sometimes taken as some specific width. But the term word is also used in a very general sense. For instance, we can talk about what a binary digit is. It is a bit, for instance, you may talk about a 4-bit word, 8-bit word, 16-bit word, 32-bit word in that sense that the word is. Otherwise normally word could mean a 16-bit word. Whatever be the length, say n bits, each location holds an n bit word.

Similarly, in the case of the CPU, since this information, that is, the unit of information or data comes over the bus; will it not be meaningful for you to assume that if this is n bit in width? It is also meaningful for the bus also to be n bit and then CPU also has some internal storage facility, each with n bit; though not always it need be true, but it is meaningful. Then we talk about matching; that is, the CPU, which can hold an n bit word and the bus, can transfer the n bit word and the CPU can receive it internally. So the element, which stores such unit of data in CPU generally it is called a register. But it need not always be true; one need not have an n bit CPU and then n bit wide memory and n bit wide bus. It is not always necessary, but it is meaningful; we will assume it that way. Instantly, why don’t I point out something – now there are two ways through which I can diagrammatically represent parallel lines? For instance, n means n parallel lines; here I have marked two things. I will use another character; I will say m, which means basically there are m set of signal lines – this is one way of representing. Another way is just put one thing and then have a slash there and then m, which means this particular thing is m. So there are two ways in which I can write.

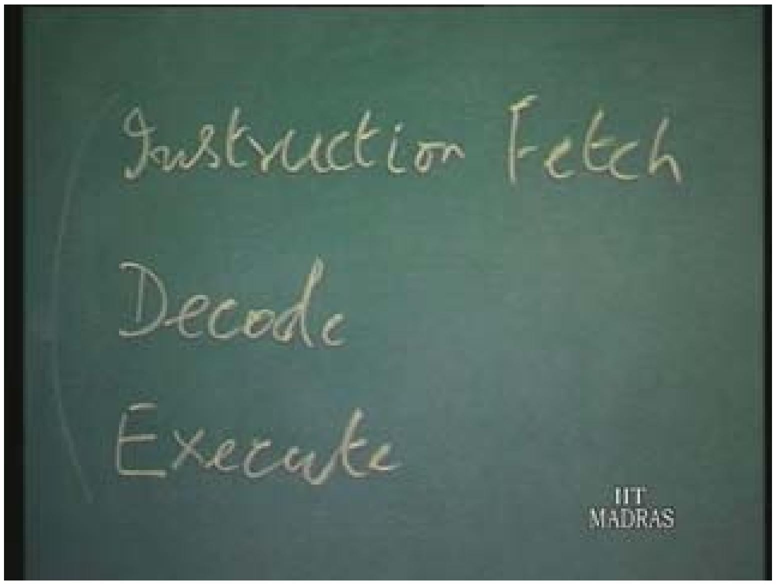
I can either have a parallel line like this and indicate m, or if both are identical, I could put a thing like this and write m – both are just the same. Let us note that down. So it will be meaningful to see that the bus width, the memory word width and the CPU word width are all matching; they are all meaningful, but remember need not really be so all the time.

Now let us come back to this register. The CPU holds some data in the register; as long as the CPU holds a data, the CPU need not address the memory and get that – it is available internally. If the CPU does not have the data in itself, only then it need access the memory. When we say that this particular thing consists of two or rather three machine cycles, basically we are saying three times a CPU is accessing the bus.

Now let us assume that as part of execute phase, we have an operand fetch – that is a data fetch. Suppose the data that need to be fetched from the memory happen to be a two word, then during data fetch there will be two machine cycles. On the other hand, if the instruction that is fetched points to a data, which is already available in the register in the CPU, in that case the CPU need not access the bus; it means the data is already available here. So CPU always accesses and gets what it does not have; since the instructions are always stored in the memory it needs to fetch. The data sometimes may be available with the processor, in which case it need not make use of the bus. That means there will not be any machine cycle.

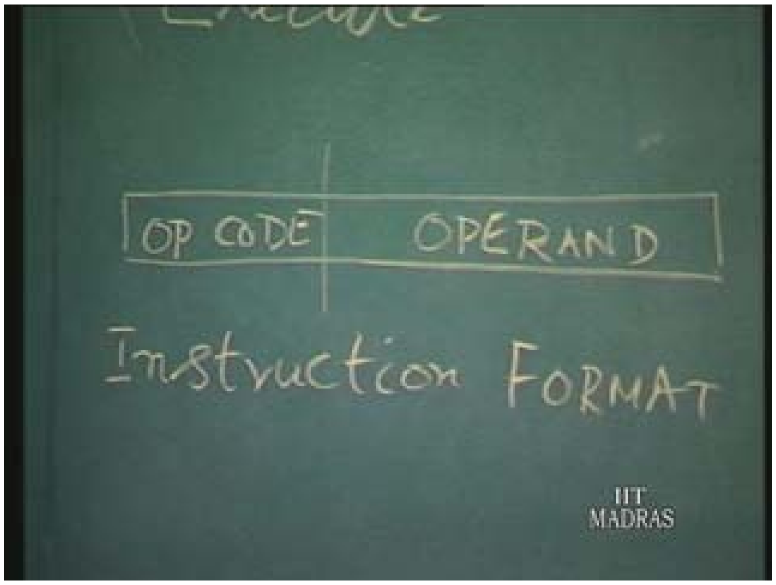
Now the CPU must have the instruction; that is, fetch the instruction and it must fetch or have all the data that is required; after that, the execute phase starts, which means basically whatever the particular instruction says, the corresponding operation will be executed. So if the instruction is an add instruction, the operation of addition will take place after it gets both the numbers. Suppose you have an instruction like add a b, once the CPU has a and b, whether a and b are fetched from the memory or they are locally available from CPU, then addition operation will be executed. In other words, what is happening is that the instruction must be fetched. This in fact we have talked about earlier too. The instruction must be fetched and then what the particular instruction is must be interpreted or decoded, and then that particular instruction must be executed. That is, instruction fetch, decode, execute – these are the things that take place during an instruction cycle. Then the particular instruction in a given program is executed; the processor keeps repeating this one after another. This is the way a program is executed.

(Refer Slide Time 00:19:07)


Instruction Fetch
Decode
Execute
IIT MADRAS

Whatever may be the application, finally there is going to be a program and that program is going to be executed by the processor in this particular manner. Fetch the instruction, decode the instruction and execute the instruction. So what happens is that first the instruction is fetched and that itself will be in a coded form. Essentially, what we are talking about here is the instruction format, meaning the way in which the instruction is organized. This instruction format will be some words' length; now one part of it will be the opcode the other part of it will be referring to the operand or the data.

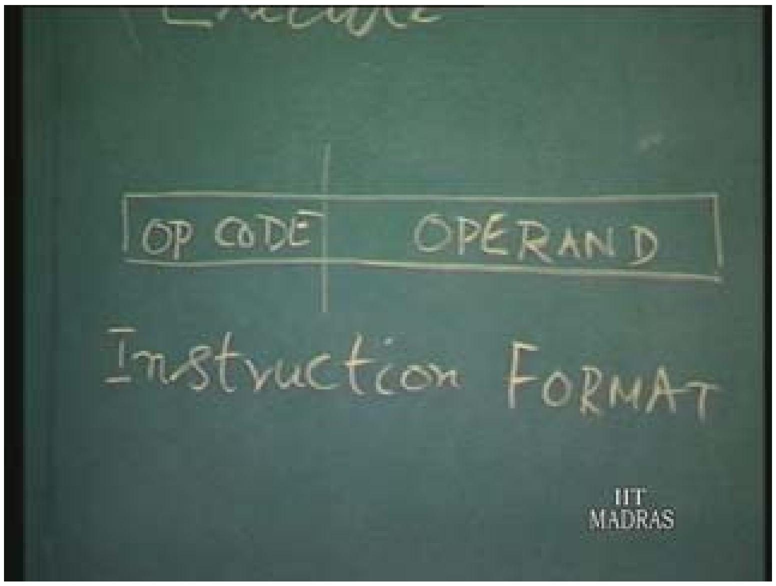
(Refer Slide Time 00:20:25)



So this opcode is the one which will be decoded, and the operands, once fetched from the memory, will be executed as per this particular operation. An instruction format essentially consists of opcode and operand, which means essentially, after an instruction is fetched, the decoding will be starting from the opcode side – that is where you have to decode, and depending on the instruction, it will also indicate how many operands are there. Sometimes, there are some instructions in which no operands are referred to at all.

We were talking about addition. Addition implies at least two operands; so we talk about a double operand instruction; that is, addition is one such double operand instruction. Then there may be an instruction in which there is only one operand, a single operand. For instance, supposing it says just change the sign of the data. It says just take one data and then changes its sign. For instance, the register contains some data and there can be an instruction which says clear that register. Clear that register is only one thing; clear one location. So if you are just referring to only one unit, and it is going to refer to only one particular operand, that operand which contains that particular operand or the data must be cleared single operand. In an image, there is no operand; so it will be zero operand; it seems no operand even will be referred to.

(Refer Slide Time 00:22:25)



There is no need for explicitly stating what the operand is. For instance, you have an instruction that says simply halt the processor; that is, there is absolutely no operand involved. So the instruction format, which generally must take care of reference to at least two operands – there can be multiple operands, but there are processors, which take care of even more than two operands, but of late generally we have this double operand, single operand, or no operand instructions. While decoding this opcode, the processor will know which type of instruction it is – whether it is double operand, single operand, or no operand. Then it will look into the other part, and then it knows that when it is no operand, there will not be any reference to operand at all. For instance, if add is the instruction, at the minimum, it is a double operand.

Then it will fetch the two operands. Once it has the two operands in, then it will carry out that particular instruction. Here what is important is that we will now go into the details of this instruction set or how an instruction is executed; there is something more we need to note down here. I said it is meaningful if the width of memory, the width of register and the width of some part of the bus – that is the data part of the bus – match, and I also said they need not match. What do I mean by that? Now I will try to introduce a few things. Suppose we talk about an 8-bit CPU or a 16-bit CPU. What is it I am actually referring to? What about the memory? Again with memory also, I have to talk about the width; I will come to it a little later.

An 8-bit CPU basically means the data that can be handled by the CPU is of 8 bits; that is, the data being processed by CPU is of width 8 bits. In the case of 16 bits, when I say 16 bits of data being processed by CPU, what we are talking about is what the processor would be doing internally. A processor may be capable of processing a 16-bit data, but the memory may supply only 8 bits, in which case the memory must do it twice; that is, memory must supply the data twice – two 8 bits, and now the width of the bus also comes. Specifically what we are talking about is the data width of the bus. The width or the data part of the bus – that's what essentially we are concerned in this particular thing.

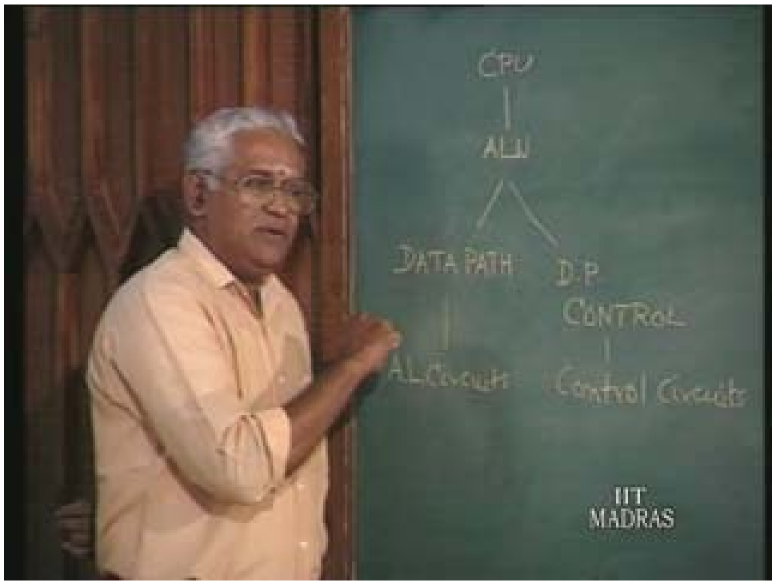
Generally the memory is organized so that the width of the memory is 8; n = 8, meaning 8, bit stands for what is known as byte. So when a memory is organized so that the width of the memory word is a byte, then we talk about a byte organized memory. Why do we organize memory in this particular way? Now, 8 is a very convenient figure; in case of more than 8, we talk about multiples of 8 bits and so on. May be in an application, the data that is referred to in one case may be an 8-bit data, may be for some other thing the same program we may be referring to double the size of that. So we may also deal with 16-bit data, with generally less than 8; these days we do not bother but it is possible to organize if one wants it. So generally what we have these days is a byte organized memory, in which case the memory is going to hold the data in units of bytes and similarly the bus will generally be capable of transferring an 8-bit or 16-bit or 32-bit data or CPU. We need not worry very much about the memory because it is byte organized.

What is the bus width and what is the CPU size? As I said, an 8-bit CPU means the data being internally processed by CPU is of width 8 bits. In other words, we are not talking about the width of the bus or the way the data is organized. What we are talking about is the internal processing capability of the CPU. In other words what did I say earlier? Processing essentially is either arithmetic or logical. Arithmetic processing will be carried out or logical processing will be carried out, and for carrying out these we have the arithmetic logical unit or ALU. Essentially when we say 8-bit CPU, what we mean is 8-bit ALU, and when we say 16-bit CPU, what we imply is 16-bit ALU. It does not tell anything about the memory or the bus or something else also – we will talk about that a little later. Essentially, the internal processing by the processor is carried out with ALU as the main thing, and we talk about that particular width. Having seen that ALU of a particular width defines the CPU of the same width, what is this ALU? I said arithmetic and logic processing and the processing is going on, on the data. So the CPU will consist of some paths or routes over which the data will be routed.

That is what we talk about – given the two data how this particular data must move over, that is, the path which the data will be taking over, and then the other part of it is this particular movement of the data over certain path – in what sequence, in what time sequence this must be going on. That is, in other words again we talk about data path. So I just put it as data path; specifically this is the one which talks about the control aspects. That is data path and the data path control – these are the two aspects of the circuit, which constitute the CPU.

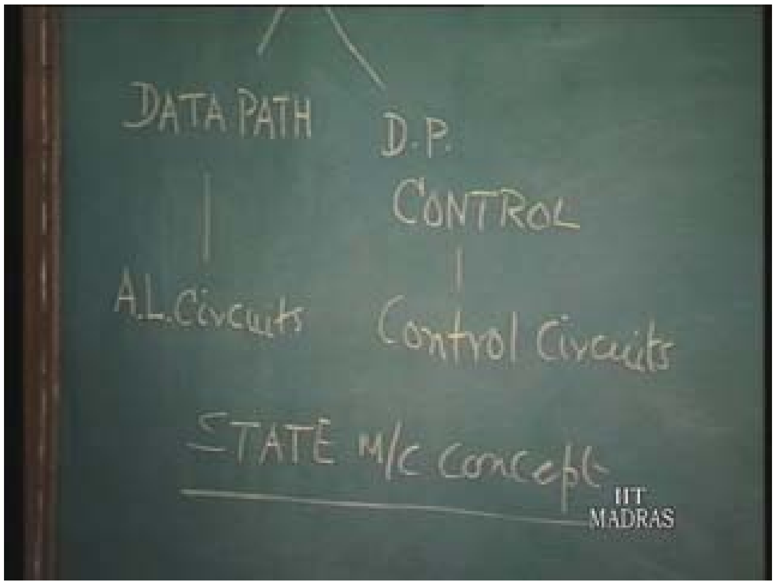
So essentially, the CPU design boils (32:00) down to design of data paths and the design of the control. The control is nothing but the controller, which is going to orchestrate the movements of the data over these paths. Then, for one particular instruction, there will be one path and one set of control signals. For another instruction, it will be a different path, may be even the same path and different control signals certainly. We may just put it as arithmetic logic circuitry, that is, arithmetic logic circuits and then we talk about the control circuits here. So this is what the design of a CPU will be.

(Refer Slide Time 00:33:00)



Now let us go back – this data is processed based on a given instruction that means we have to go back to that instruction cycle consisting of machine cycle, each machine cycle consisting of state, in other words the particular CPU design is nothing but a digital system design, an example of a digital system design, which means CPU has a state machine. So we have to develop some idea about a state machine concept; we have to have some concepts about that.

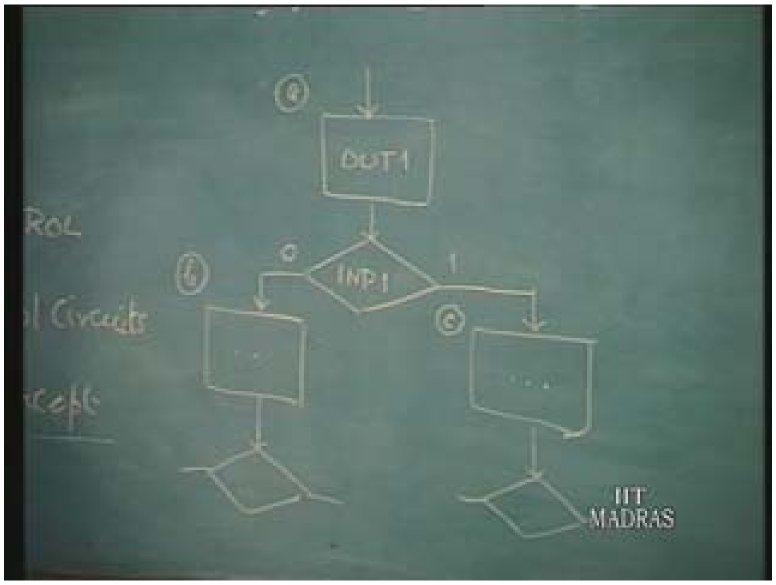
(Refer Slide Time 00:33:47)



Now what is it? In each state or the clock period, the CPU enters one state and it carries out some basic activity recall that is the instruction consisting of machine cycle, consisting of states, and in each state some basic activity was going on, on the data, and within the CPU, we have registers. So the basic activity which will be going on during a state, since essentially it goes on at the register level, can be considered register transfer activity – register to register, some transfer or clearing a register, and so on. So this will be generally called a register, transfer level activity or just simply RTL. And in fact, we have a language which explains the sequence in which the data is moved, for which the different registers in the CPU will be made use of. That, once you state, will describe the whole behavior of the CPU or the processor. So at the register transfer level the CPU is going to do and the design of a CPU will consist of design of data path circuit and the design of the control circuit and the entire behavior will be at the basic level, that is, the RTL level.

Now let us develop some concept of the state machine before we proceed to take a look at the RTL. For the present, note that at the state, that is, during one clock period, this is what is going on. A state can be represented let us say using a box. A machine enters a particular state – I will just call state A, and in that state it generates some output which I will enter. I will just name that particular signal as OUT 1 and in the same state A, it checks for some input. So I will just name that signal as IN 1, and depending on the state of that input being, let us say 0 or 1 or the logic states being false or true, the machine may enter another state like this. The other state is what we may call B and the next state is C. For the present state A, the machine generates an OUTPUT 1 and in that state, it checks for an input, and depending on that particular input being false or true, the next state, the false state is B, and the next true state is C, and again in the next state we talk about as we did here –the output and the inputs it checks, and so on so forth. This is a simple picture of a state machine. This is what precisely the processor is doing.

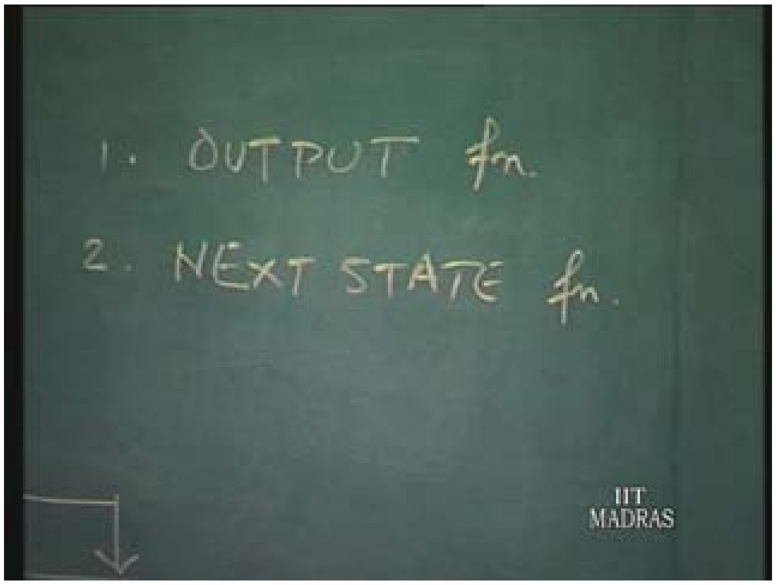
(Refer Slide Time 00:38:15)



The processor enters during the instruction cycle, and in that particular state it generates some signal. It also looks for some input signal and then it decides on the next state. Shown this way, it may appear as if in any one state, only one input can be checked or only one output can be generated. That is not necessary – there can be more than one output and here also, there can be more than one input checked. It is possible to do that, but this is one simple way of looking at it and this will help us develop some concept about the state machine.

Let us not discount the other possibility of having more than one input to be checked and having more than one output to be generated. You can now see that here computing system essentially evolves around the processor and the processor as a state machine finally just is going through state by state as per some algorithm, and that particular thing in each state is going to check for an input and it will generate output. In other words, the whole system design will boil down to two things – one is output function generation, and second is the next state function.

(Refer Slide Time 00:40:14)



1. OUTPUT fn.

2. NEXT STATE fn.

That means, the processor, to start with, enters a state – we will call it an initial state, and in that initial state, the output function will tell the output that needs to be generated. In that particular state, the next state function will tell what the input that needs to be checked, and depending on the condition of the input, what the next state is. So the design of a processor is essentially arriving at these two functions. This is the output and input; with reference to our CPU we need to know that.

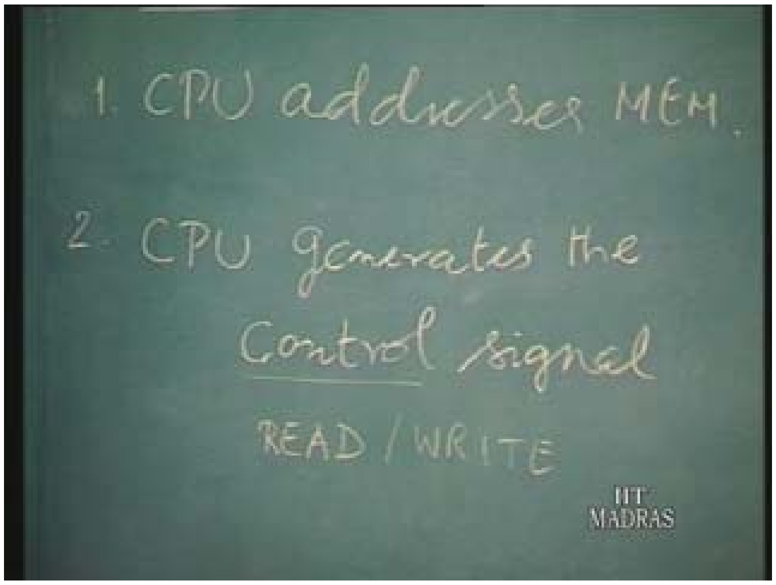
Let us go back and see what the processor was doing. The processor during its first instruction cycle will take a machine cycle. We have seen what is taking place during a state; that is, in every state it is going to involve in some RTL activity. If you have not got right picture of what is the register transfer we will come to that later; some basic activity is at the lowest level. So we will go to the next level, that is, machine cycle.

What is the CPU doing? First the CPU addresses memory, and we will assume an instruction fetch, in which case the CPU is placing the address of the instruction on the bus. Now the memory must know whether it must accept – because memory is always a slave; it does not know whether it must accept something coming into or it must generate. How should it respond? The CPU must also indicate that. So the CPU generates, I will say in general, the control signal.

What is that control signal? In the case of instruction fetch, it is essential that the memory must send the instruction. That is with reference to CPU, we talk about the instruction word, which is coming over the bus as a data. Remember I was telling you once when I say data, do not confuse program and its instruction and data. From the memory point of view, whatever is stored is a piece of data, some unit of data. In the instruction fetch, the memory must respond with the instruction word – that comes as data over the bus, which the CPU will read.

So the appropriate control signal in the case of instruction fetch will be read. That will be the control signal. That is, the CPU tells the memory, let me read the content of this address, which is an instruction. So the CPU places the address of the instruction on the bus, and CPU generates the read signal in the case of instruction fetch. In some cases, it may be right, so I use the word control.

(Refer Slide Time 00:44:35)



1. CPU addresses MEM.

2. CPU generates the <u>Control</u> signal

READ / WRITE

But if the CPU wants something to be written into the memory, what is the implication? The CPU must not only address, but it must also place the data. Only then, that data can be written into the memory. We will come to it later; I thought I will just mention that too in passing. The CPU addresses the memory; the CPU generates the read signal. As a true slave, memory must respond, and memory must place the data on the bus, so that the CPU may read this data, and in the case of instruction fetch, what is this data? In the case of instruction fetch, which is the first thing in any instruction cycle, this data is nothing but one instruction. And remember here you don’t know about the size of the instruction; the instruction may be a 1-byte instruction or a 2-byte instruction, or a 3-byte instruction.

So assuming it is a 1-byte instruction, the CPU will place the address of that byte. If it happens to be a 3-byte instruction, the CPU must place the three addresses one after another. So let us remember that all the time, because we talked about the width of the memory, width of the bus, width of the CPU and then in general, when we talk we must always correlate, because the instruction size can vary. The CPU addresses the memory, CPU generates the read signal, and memory places the data on the bus sizes and obviously the next signal will be the CPU reads in the instruction, because I said instruction fetch, but in general, this is nothing but data which the memory has responded with. This instruction is essentially nothing but data, which has been placed by memory in response to CPU’s request. We just assume that we are discussing instruction fetch – that’s why we are talking about an instruction, which the CPU reads.

Now let us get back to our basic concept, that the CPU addresses the memory so the CPU must place the address on the bus. I will try to translate it into this: CPU addressing the memory – I will rewrite this particular thing as some address is placed by the CPU on the bus. The CPU generates the control signal, I will assume read, so I will say some control signal called read is placed on the bus.

All these are put in different parts of the bus – we will come to that later – then memory places the data on the bus. I will just put it as memory – that is memory part does it – data goes on the bus. The CPU reads the instruction, which means whatever was there on the bus is actually the data, which was available on the bus is going into the CPU, meaning it is going into a register. We have assumed that it is an instruction. So the instruction is going into the instruction register. Do you see the register transfer activity? Some part of the bus is going into the register – this is the basic activity. Similarly, the CPU placing the address: this may come from an address register of the CPU.

(Refer Slide Time 00:50:07)

Some address register of the CPU goes into the bus; it is just transferred and then here this is a control signal. So this is in fact a read signal; it does not matter. The read signal is placed on some part of the bus. Here again there is a simple transfer and then here, the memory contains that data placed on the bus. That is, the contents of the memory location are transferred onto some part of the bus. So now you must have got a good concept about what we mean by register transfer and that is precisely what is taking place in each state. The processor must go into one state, do this, and then it must go into another state and do that, and so on and so forth. Where are the inputs coming from? We will have to wait to know that; may be in the next lecture we will talk about that too.