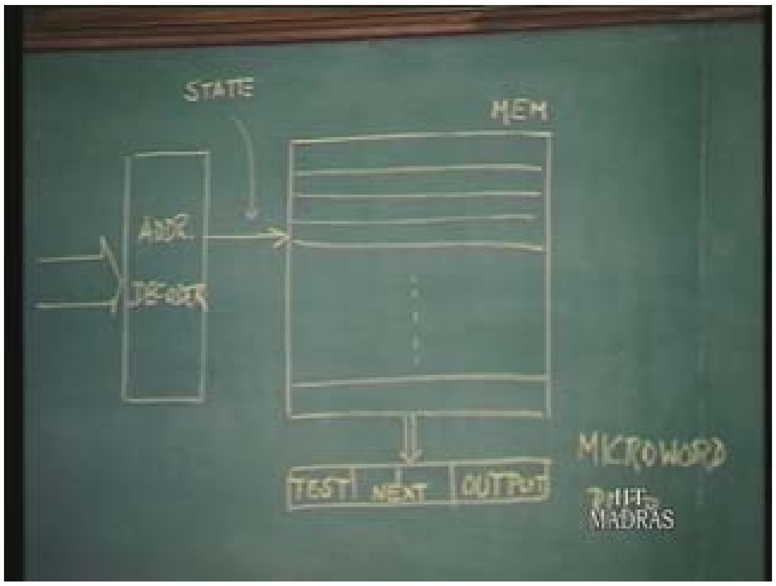
Computer Organization
Part – I
Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology
Lecture – 7
Data path Controller: Micro Programmed

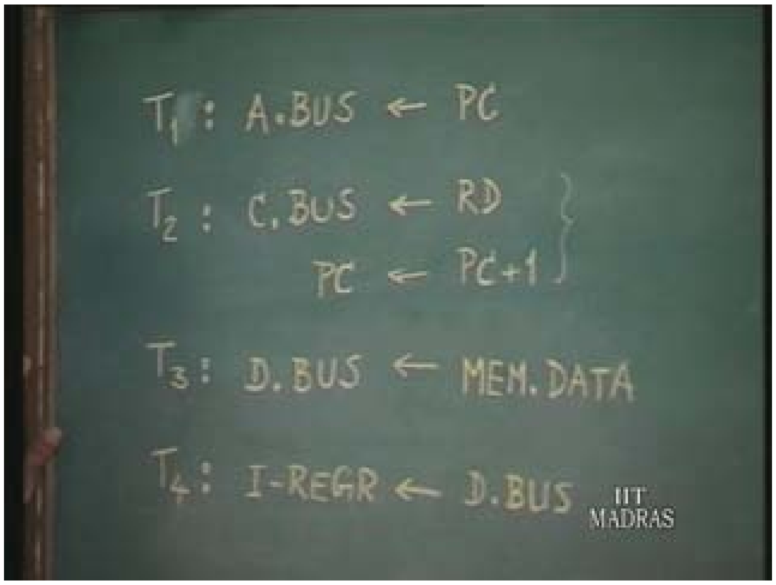
In the previous lecture we were discussing how exactly the data path control signals form the output of a micro-word, provided for each state we have an appropriate description.

(Refer Slide Time 00:01:20)



Specifically we took for this particular state and then we were working out the details of the code. This is one way of implementing. Now we will come to this point a little later. Earlier we were talking about the state; that is, specifically four states for the instruction fetch – that is only one machine cycle – and we were working out the details.

(Refer Slide Time 00:01:50)



T_1 : A.BUS ← PC

T_2 : C.BUS ← RD
PC ← PC+1

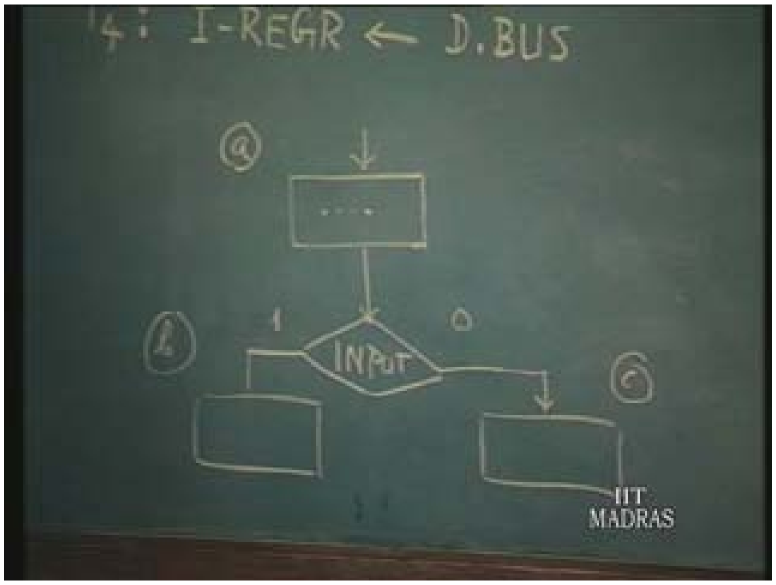
T_3 : D.BUS ← MEM.DATA

T_4 : I-REGR ← D.BUS

What exactly is the purpose of working out the details? Essentially it is to design the processor because the processor consists of data path and data path control, and these state signals actually describe the data path control, which helps set the data path; and for one specific architecture – we just took some typical architecture – we were seeing the data path in the previous lecture. I said that for one state, in case the code which we have arrived at can be put in the memory, then one word in the memory will contain information about that control data path control. And then I also showed the T1, T2, T3 and T4 states with reference to the ASM chart, showing how other fields in the particular word must also be added in addition to the control; namely, the next state, and some inputs that must be tested in each state.

So what do we have here? We have a micro-word, which is specifically for one state and you can see that once you store it in the memory, the address of a location is synonymous with the state. Here if you go address by address, then the memory as you can see has state by state information in each location. Now as I said, this is one way of implementing. To proceed further, we must know a few more things about these other fields because we worked out this output thing in detail only for this T1 state. In general, if you just take any present state, you can call it a. You have a state a, which we call the present state. Some output may be generated in that state; some input is also checked, and depending on the input being 1 or 0, we said there are two possible next states and we can call them state b and state c.

(Refer Slide Time 00:04:43)

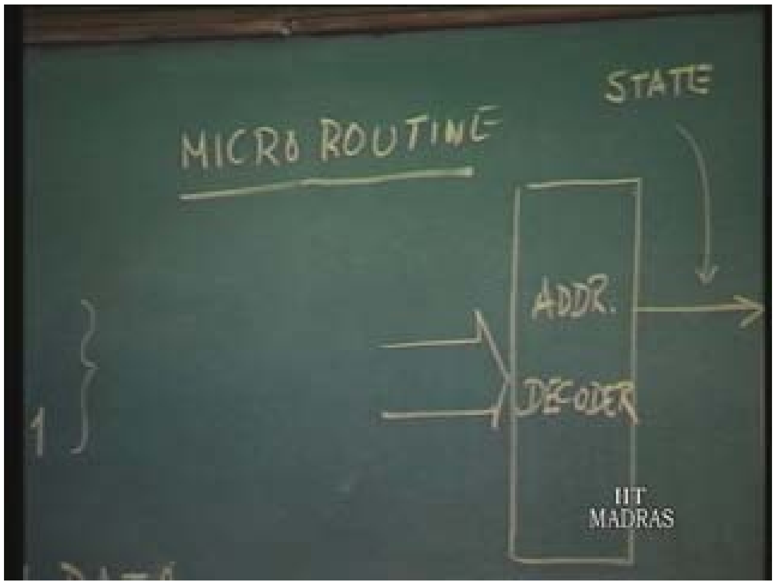


What it means is that for the present state a, if the input in that particular present state a is 1, then next state is b. Besides that, b is the next true state for the present state a. On the other hand, if the input is 0, the next state is c. We call c as the false next state of a – the true or false is really with reference to the input that is being checked in that state. Suppose this state happens to be T2 – remember in T2 we were actually checking the input condition; rather we may say we were testing the condition of the signal ready. In T2, we are trying to check ready; then, depending on ready being 0 or 1, the next states are arrived at. In fact I showed you two things – one is that one next state may be T2, another next state may be T3 or one as Tw another as T3. So now that particular information has to be accommodated here. Really speaking, the so-called next state field should indicate what happens if this test fails; meaning, if the particular input that is being tested is 0, or it is false. In case the test fails, we talk about a false next state. If the test passes, then we talk about the true next state. So really speaking the next state may be true next state or false next state. This will have to be accommodated, but there are ways in which this particular thing can be reduced to 1 also; we will talk about it a little later.

Here you have given a state – that particular micro-word tells us of the inputs that must be tested, and it tells us, depending on the condition, what the two next states true or false next states are; and that word also gives us information about these signals, which the controller must generate: that is, the controller must provide output so that the data path may be set up. We now have it for one state; when you work out for all the states – here we have just worked out some detail for one machine cycle – this will have to be done for all the instructions, which the CPU supports. If the CPU support has 250 and odd instructions, for each the details must be worked out but some of these will become common; for instance, this instruction fetch as you can see will become a very common thing for any instruction. Now I will be accommodating this particular thing in four words like this: in four micro-words, one for each state. Remember I was telling that it is a small step; and so we can even call each step a micro-instruction.

These four microinstructions together can form a micro-routine; just these four micro-instructions can form a micro-routine.

(Refer Slide Time 00:08:44)



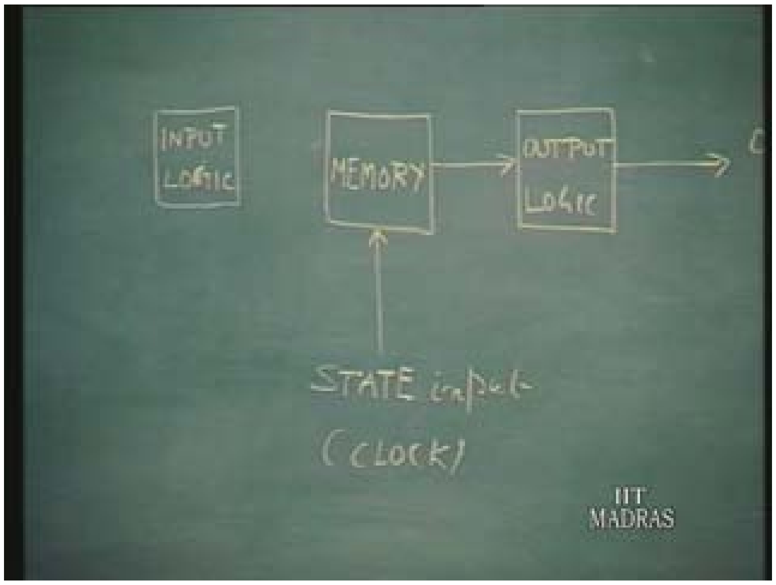
So really the controller will have to store these micro-routines for all the data paths; it must set up this thing. What about the sequencing? That sequencing information is available here; this is only for the data path setting. Here you have the sequencing; it says it is state a. In state a, it says if this is state a, this word will say test one; some input information will be available here; then it would also give these two true or next states. So given a state, what is the data path we set up? Given the state what are the inputs to be tested and what are the next states? This will have to be done for all the states, and then we have the entire control information in this memory.

Once you design the processor, these memory contents need to be only read, because it the behavior of it will be frozen once you have these codes. Then you would find that this memory is nothing but a read only memory, because subsequently only the contents of the memory will be read. Specifically it is used for control purpose; it will be also called CROM or control read only memory. Now suppose you allow the user to change the contents of this memory, then it is going to change the behavior of this machine, because the sequence will get altered. Then the behavior of the CPU itself will get altered. If that is possible we talk about a micro-programmable processor; otherwise if this is only read only memory, which means it cannot be written, it can only be read. Then we freeze the behavior of the machine; and sets of words like these form the micro-routine and sets of micro-routines will form actually the routines needed for executing one instruction, and then sets of instructions form the program. So this is the way you go up from the lowest level to the highest level, that is, the program level. Now we must have some more information or understanding about the behavior of this, that is, in general I was calling it a state machine. So I would try to give some information because this particular thing will have to be completed.

All we have is some information about behavior for one state; that's all we have arrived at. Now I would try to give you a generalized state machine model and then show you how this particular scheme can be further developed. What is this generalized state machine model? Now you have some idea about the computer going into a state; it moves state after state and then it executes some simple register transfer level activity; it carries out some activity. Here specifically what we are talking about is a CPU as a component, which is an example of a digital system or a digital machine. The processor will stay at any time in one state, which means essentially I can use a memory element to show that at a given instant the processor is in one specific state. Do not confuse this memory with the memory we are talking about in overall computer system – it is different. This memory in fact stores the information, not very much different from this. If the memory is in state T1, then the next instant it may go to T2; so correspondingly the contents of that memory will be changing. In other words, to the input of this memory specifically there is time input, which in fact gives what we may call a state input. Generally, the clock of a system will be meant for that.

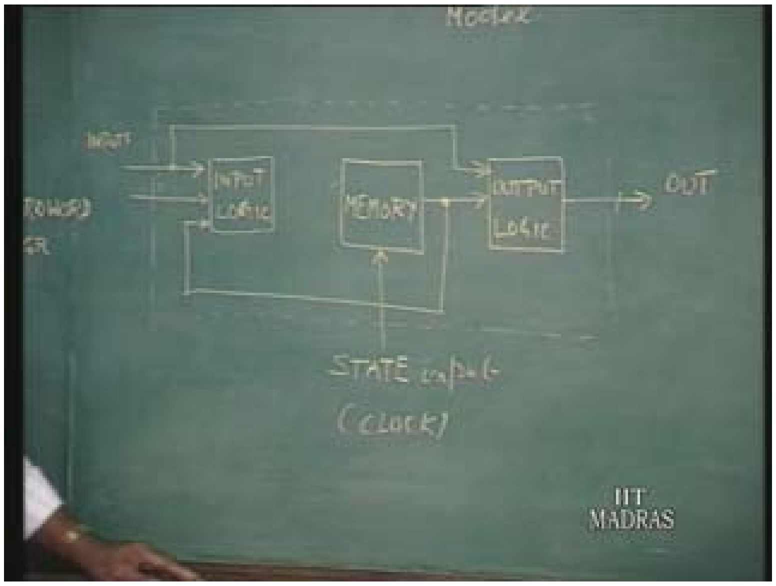
What exactly is the state memory? The memory holds information about a state. I am going to use this ASM chart; by looking at the memory I can say that it is in state a or state b or state c – that's what we mean. The clock is the input to that. Then, given the specific state, the output from that can be defined. Now I am going to complicate a little by introducing another output logic; then I will derive the actual output. So given a state, in a state it generates an output. In a state, it generates an output – there is some logic behind the whole thing. Similarly in each state, I am going to receive some inputs; in each state I am going to check for some inputs. So I will also introduce logic for the input part of it.

(Refer Slide Time 00:16:26)



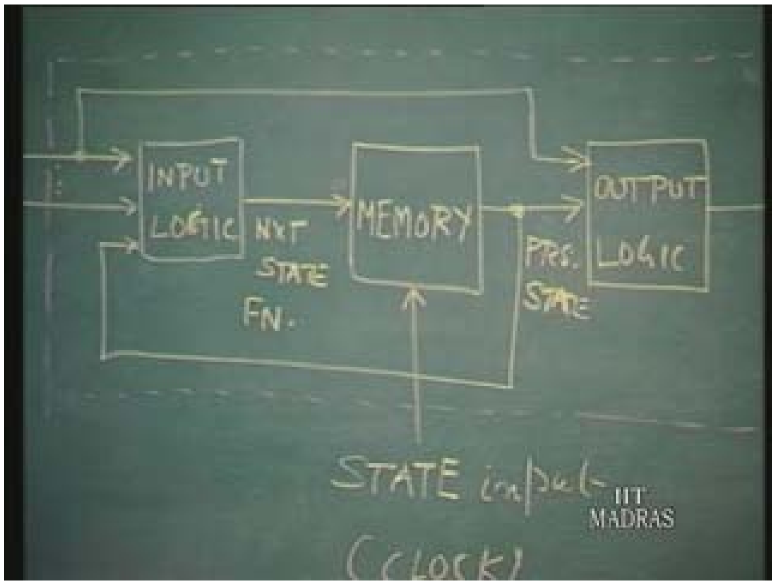
Now if I take this state information and feed to this logic, and actually feed some inputs these are the actual inputs coming from the external world – for instance I can also take some of these inputs to this output logic. I will explain the significance of this. Here what you have inside this box is the state machine. There is an input which defines the time and then you have other data inputs and data outputs similar to what we had as data inputs and control inputs. There can be more than one inputs and outputs.

(Refer Slide Time 00:17:32)



Here I have to complete this – we will just try to say this is in state a; and then some input must be tested; depending on the input, the next state of the memory may be b or c as shown here. Depending on the input, it may be b or c; and then in that state it generates an output. So that is what you have here. Given the state a, in the state, the output logic helps generate some output and in the state a, it checks for input and decides what must be the next state. So you can say that this particular thing is performing a next state function; the next state function is performed for a given state. Now given the present state, the next state is decided depending on the inputs; and given the present state the outputs also are defined.

(Refer Slide Time 00:18:54)

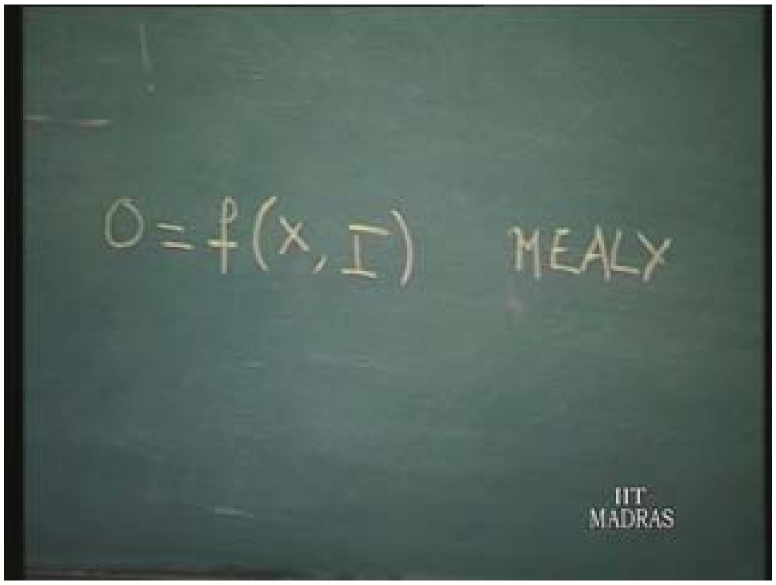


As shown, the output is not only a function of the present state, but also the input. This is one model. For instance, if you see here I have not done such thing. I am saying in the present state a, some output is generated; I will just call it OUT1 or something. As per this ASM chart, in present state a, an output named OUT1 is generated; actually it is not depending on an input. So this is another model, in which case, actually this link is not really there.

In the generalized model we have this link; sometimes it may not be there. In the sense that the output is only a function of the present state if you don't have this. So if the output is only a function of the present state, suppose I call this g; suppose I call this f, then suppose I indicate the present state as x, then I will just make a small change – I will just call this i for input and o for output – this is easy to remember.

It is like this o; and suppose this logic function performed is let us say f and the logic function performed let us say is g, then in one model what we have is the output o is a function of not only the state x, but also some of the inputs – not necessarily all the inputs – in the generalized model. So output is a function of not only this state but also the input. This particular model is called a Mealy machine model; it is called a Mealy machine.

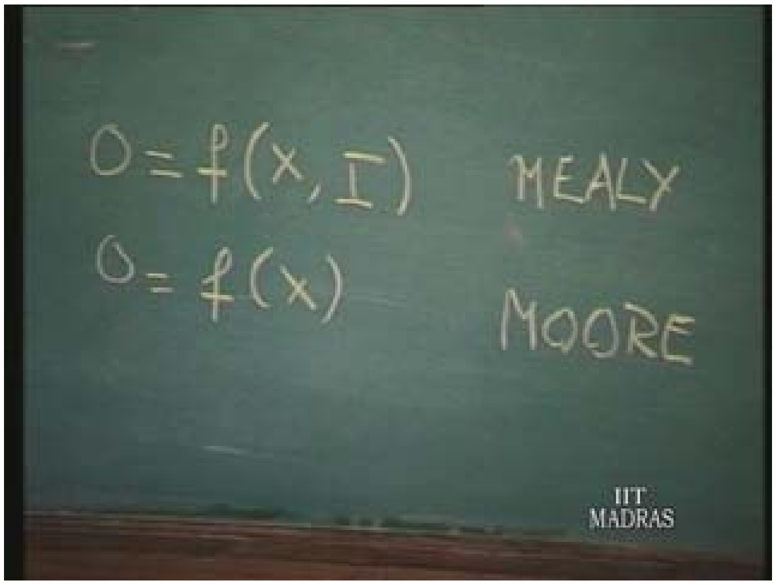
(Refer Slide Time 00:21:42)



$O = f(x, I)$ MEALY

On the other hand, if you do not have this, then the output is only the function of the state. Then this particular one is called a Moore machine.

(Refer Slide Time 00:22:03)



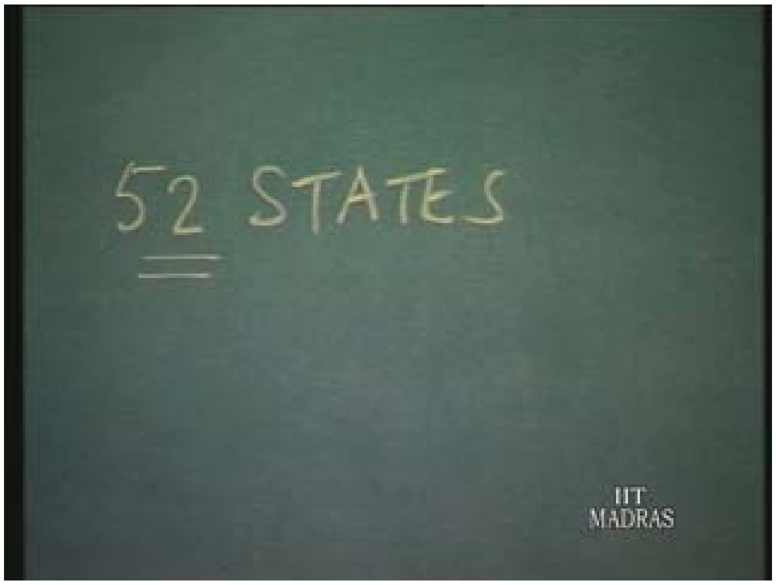
$O = f(x, I)$ MEALY

$O = f(x)$ MOORE

So we have different models like this. The next state is a function of – as you can see here the next state has the input; so I will just put it as next function that is g –x and I. I have shown here the next state is a function, that is, g of x as well as I. So it is the same thing as saying the next state is a function of I as well as the present state. In some we may have this link; in some we may not have.

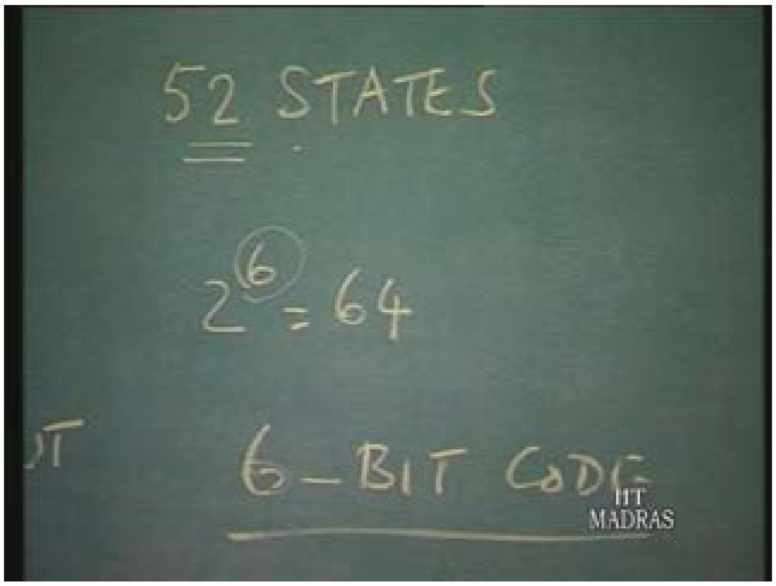
Specifically as you can see in this case I have already pointed out that OUT1 is depending only on the state it is not depending on any input. It can be the other one; not shown here in this Mealy output; what we have is only the Moore output. This is very general; now I am trying to tell you this mainly because you should have this in mind when we are going to further develop this particular one. Now what do we find here? This memory I said is the one which is going to store the state and each one corresponds to a state. So suppose we have a system with say 52 states, suppose we have worked out the design of a CPU and we have arrived at 52 states, these 52 states are going to represent only one; one after another at any time.

(Refer Slide Time 00:24:01)



So how many bits do you need? Suppose you use a binary code for this with 5 bits; you can have 2^5, which are 32. Since 2^6 are 64, basically you need a 6-bit code to represent a maximum of 64; but we are going to represent only 52.

(Refer Slide Time 00:24:45)



Basically it means one – what is the bit? Bit is a binary digit and 1 binary digit is stored in one memory cell. And you all know that a flip-flop can be used to store 1 bit.

(Refer Slide Time 00:25:12)

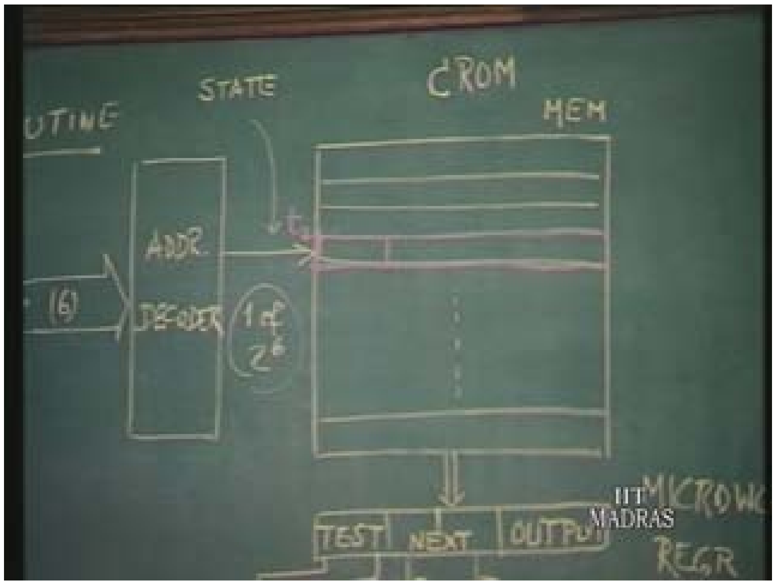
So having six flip-flops in this memory, you can see that this machine goes through all the 52 states. Coming to this specific thing, suppose we have 52 states, basically it means we need a memory of 52 minimum; minimum of 52 locations. That is what it means, because each location corresponds to a state. And at any instant, the system stays only in one state. That is fine with us; at any instant you can only read from one location. So really what we are having here is a 6-bit code – that is, I am talking about 52 states.

A 6-bit code comes here right and here you can have one of 2^6 outputs. So for this particular decoder, the output is always 1. Here it is from 2^6 to 1, which is how the particular decoding will be taking place. So we say the output will be 1 of 2^6 for this specific example. If you take an n bit code, there the c code will be 1 of 2^m in general. Now what it means here is that on this side you must have the memory elements, so that there are these six flip-flops. Let us say if you are implementing flip-flops, what you need is six flip-flops here. We will just call them FF_0 and FF_5 and then the next state, true next state or false next state – one of these will be selected.

So this next state will also be a 6-bit code. So it is a 6-bit for true and 6-bit for false; actually the whole thing will be 12 at any time. Only one of these will be used. We will have a selector here to either pass on the true next state or to pass on the false next state, which is again 6-bit code. There will be a control signal because this particular one essentially is a multiplexer, which passes on either this leg input or this leg input on to the output. What we need essentially is a 6-bit next state code for the present state. The whole information is available and it says depending on the test being false or true, one of these will be passed on. Why do not we complete that? One of these will be passed on and that particular thing will go here; that is, the 6-bit code will be going on to that. Now which of these to pass on? That depends on the selector logic. Here we can call it a simple test logic; the output of it is really used to select either false or true next state. It may be many bit wide. Suppose you allocate 4 test bits; possibly four signals are being tested. That is what it means.

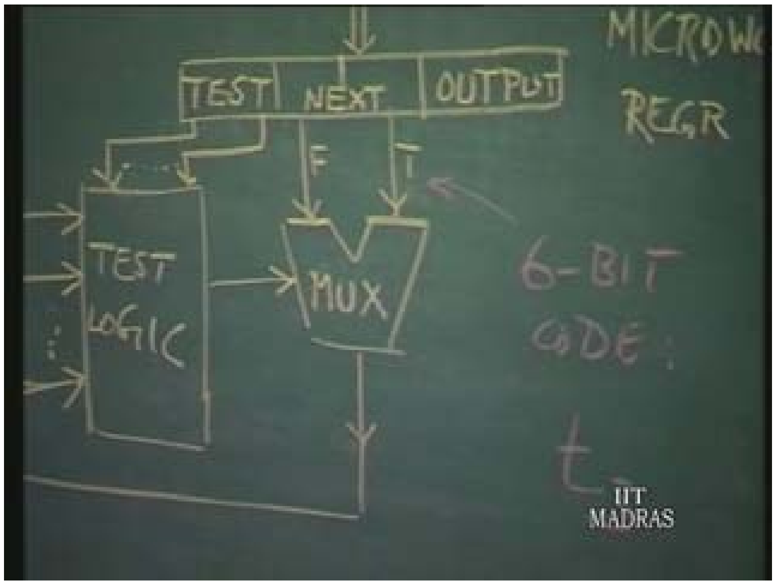
There are many other ways of looking at it – and the actual signals from the real world will be given here. In other words, the actual inputs which must be tested will come from the real world, and will be given here. For instance one of these – in our case, just one of these – may be this ready signal which is needed. In T2 state, we are checking whether the ready is 0 or 1. So that information will be coming to this. Suppose this is T2 state. There are two outputs here – these are the two outputs – and then ready is the input that is being checked. Remember in one we were saying it goes to the next state wait if it is 1, it goes to the next state T3, which corresponds to this here. So ready in fact is an input to be checked during T2. That information must be available in this. Suppose this were for T2; this particular one, this word is for T2. The test field in this must indicate that in T2, ready is the signal which must be tested.

(Refer Slide Time 00:32:37)



So it is only information that comes; this whole word comes down here. This test field will indicate ready is the input which must be tested given this state and the actual input signal will be going to this logic. Basically its only information says check; check for the state of ready and then depending on ready being 0 or 1, if ready here is 0, pass on the next address as the address corresponding to Tw or what will be one 6-bit code. This F will correspond to Tw, the code for Tw. Otherwise, if it is true, it would say this particular one, T, will correspond to the 6-bit code corresponding to T3.

(Refer Slide Time 00:33:48)



Now you have the whole picture – how the sequencing of the state is going on. So start from any state; in that state you have information of what all inputs are to be tested. What are the possible next states the machine can assume and what are the set of output signals which must be generated? In our case, these output signals are all for setting the data path. So here you have a ROM-based implementation of the controller.

Here you have a micro-programmed controller, mainly because the whole control aspect is in the form of a micro-program. Now this is one implementation as I was telling earlier, this in fact is called a firmware implementation. Why do we call it firmware? Because as long as we have the code, all the codes for all the states here are frozen – you don't change anything, the behavior of the machine is there all the time; you are not changing anything. It is firm enough. But then, suppose I remove this particular memory chip and put another memory chip with different codes, the behavior of the code machine is changed. So it offers me a flexibility; that is why it is not hard. At the same time, it is not soft. So we say it is firm. For you to gain an understanding of what would be a hard wired controller or hard wired oriented controller, I will take an example. But unfortunately I cannot take a CPU design in the lecture because it consists of many states. I will just take an example of just four states and then work out the same thing for you, and then I will show you the hardware as well as the firmware, so that you can have an idea about what would be a hardware controller. We will come to that later.

Now this particular one is the ROM-based implementation, which is essentially a firmware solution of a micro-programmed controller. Let us quickly work out. The read only memory will have information about all the states. When I say all the states, I mean all the states for all the machine cycles for all the instructions of the controller. Given any state, the details in that will be such that it will tell what the output to be generated in that state is. For instance, in T2, these are the outputs to be generated; that is three control signals must be placed on the control bus and program counter contents must be incremented. These in fact are done in parallel because they go to different parts of the processor whereas these are in sequence. So both of these are going to different parts of the processor and so you can have these things implemented in parallel. These outputs are generated in T2 and it says in state T2, check for ready input; this information will be available here.

Depending on the ready input being 0 or 1, the two next states, true next state and false next state, the codes of these will be available here; and depending on this, whether that test has passed on not, depending on 0 or 1, that is, a control depending on 0 or 1, which depends on this particular one, either this or this will be passed on to this. Now here also, with what is called a state generator, you are generating all the states. That sometimes will even be called a major state generator. To this also the clock will be given. The generalized one you see has the state input, where you are seeing the clock. So the clock is the one which goes to this memory. This memory is a state generator with the clock; that is what you have here. And there is some input logic to which inputs from the real world go. The next state function has been implemented. That part of it is also here, and the output is generated or a set of outputs is generated. That part of it, in this case, is available here, the details of which we saw in the previous lecture.

So this is how you have the control of a data path. Now you can see that data path is set up having the appropriate sequence of the control signals. Really speaking, data path and data path controller are not really different because the data path is always there; that is, all the data path combinations are physically available and only when the control signals come, the paths get activated. It is like body and breath force or soul, whatever you want to call it, that's what it is.

Now this as I said one implementation only; and as I already explained this implementation has the flexibility mainly because it is not completely wired. What we mean is the ASM really gives you the behavior of the whole machine. In the ASM chart you see the behavior of the whole machine and in terms of codes this read only memory is what it is going to have. Now instead of these codes if you pull out and put another memory with different codes, the whole behavior is going to be altered. So the read only memory, through its codes, actually captures the behavior of the whole machine. Since one can change – one doesn't change just like that – we say it is firmware.

Now in the case of hardware, later on I will show you it is going to be very difficult because the whole thing will be wired. You cannot change things here; these are all hardwires, only hardwired part of the circuit. If you want to change anything, you have to remove all the things and then de solder and what not. So in firmware implementation the essential thing will be in read only memory flexible form, but then associated with that there will always be hardware. But the behavior of the machine will be available in such a way that you can change if you want. In the case of hardware there will be a problem.

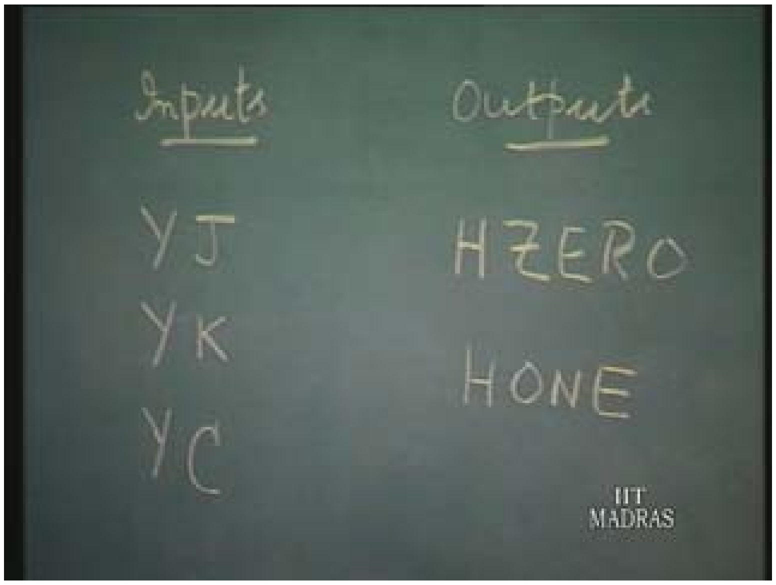
Now I hope you are able to map this generalized state machine model with this, and then understand what is going on here with this. That is what the whole thing is about. We were just looking at the micro-program controller and how to design that and then I said that it is essentially a firmware realization; and then I also mentioned that there is a hardware realization of the same circuit.

For a complex system such as the CPU design, it is going to be very much difficult discussing through lectures like these. So I will take an example of a digital component, a simple one which has a few states and few inputs to be tested and a few outputs and then work out the system design of it, both from hardware point of view as well as firmware point of view, and then try to show you the parallel between this and what we have seen for the CPU. So for taking up the design of a flip-flop which stores either a 0 or 1, specifically to accommodate a few inputs, we will make it a little complicated in the sense that we will take up the design of a master–slave flip-flop. I will explain what this means and one specific type of flip-flop is called JK flip-flop – this is the one we will consider.

J and K are the actual inputs; so let us just take a look at the set of inputs and the outputs that the flip-flop must generate. Since I said essentially a flip-flop contains either 0 or 1, the output must be just one of these. We will just name this one as H ZERO and the other output as H ONE that is the output. There is something more, which must be said for the master–slave; we will talk about it a little later.

As for the input, this particular flip-flop has two inputs called J input and K input. We will just name these inputs as follows: YJ is the J input and YK the K input. And then any system will be having a clock, especially when it is a synchronous system; there will be a clock so you just call that YC.

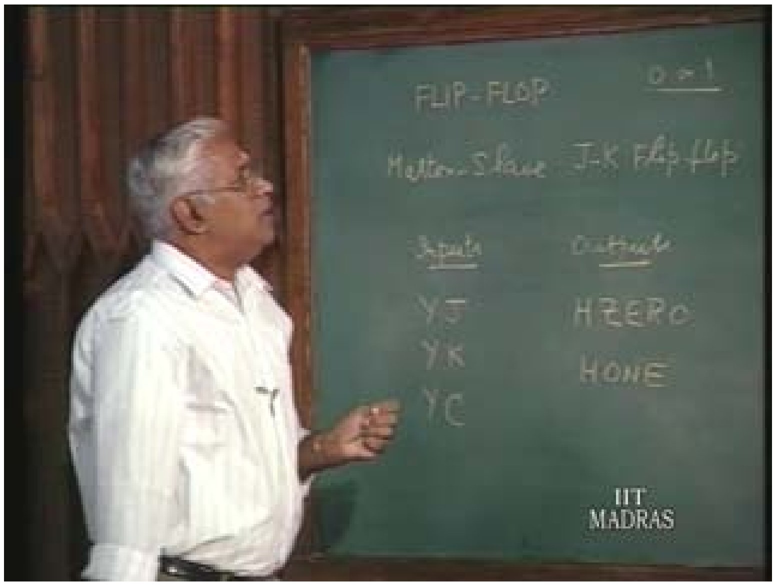
(Refer Slide Time 00:45:52)



Inputs	Outputs
YJ	HZERO
YK	HONE
YC	

Now as you can see, we are putting just a prefix Y for the inputs and the prefix H for the outputs – just a naming convention, nothing more. Then you may know about this JK flip-flop. I will just briefly tell what it is.

(Refer Slide Time 00:46:21)

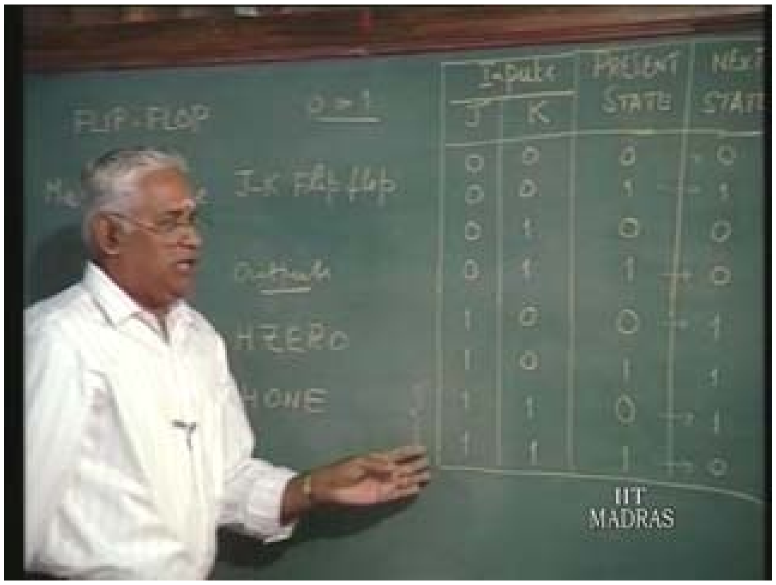


J and the K are the inputs, and the output will depend on the state of these as well as, in the case of flip-flop, we have also to take into account what was the previous state and depending on the input what is the next state. Now if you have these as the inputs and if the present state of the flip-flop is something, what happens to be next state? This is what we have got to take a look at. What we are currently going through is defining this system, namely, master–slave JK flip-flop.

The present state, if it were say 0, and if J and K both the inputs are 0, the next state is also 0. Roughly, you can take it that J is something like setting the output and K has resetting the output. We generally use the word set to set the flip-flop output to 1; we use the term reset to reset the flip-flop output to 0. Here we find that it says do nothing; on the other hand if you have J, if the present state is 1, then J and K both are 0, the next state continues to be 1. In other words what we have is there is no transition between these. If the present state is 0 and if you have K, the input is 1. I say K is something like reset; reset means 0, so the effect of that will be to see that the output is still 0. Now on the other hand with J 0 and K 1 – the present state is 1 now. It says effect of K is to reset; that is, 1 must be reset to 0. Similarly, we have to go on, which will describe the whole behavior of this JK flip-flop.

What is next? Next would be this one and then it takes J 1 K 0 effect. I said J means set. So that means the next instant, next state of this must be to set; that is, to become 1 to make the output 1. On the other hand, if it were 1 already, if J is 1, K is 0, the output will continue to be 1. Then we have two more conditions to be checked, namely, if the present state is 0 and J and K both are 1, our definition of set reset fails. Because one says set; the other one says reset. For this particular condition, two input conditions will occur, the present state is 0 and when the present state is 1: these are the two conditions. Elsewhere it is either J or K – only one of them is 1; here it is the two input conditions, for which both J and K are 1. For this we do not talk about set or reset; we say if both happen to be 1, it toggles, meaning whatever the state it is in, it just goes into the other state. That means, if the present state is 0 the next state is 1; if the present state is 1, the next state will be 0.

(Refer Slide Time 00:50:56)



Inputs		PRESENT STATE	NEXT STATE
J	K		
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

This in fact is the behavior of the JK flip-flop. I have not yet explained what the master–slave is, which we will take up in the subsequent lectures and also work out the ASM chart for this behavior and carry out the design, both hardwired as well as firmware. We will see these two approaches in the subsequent lectures.