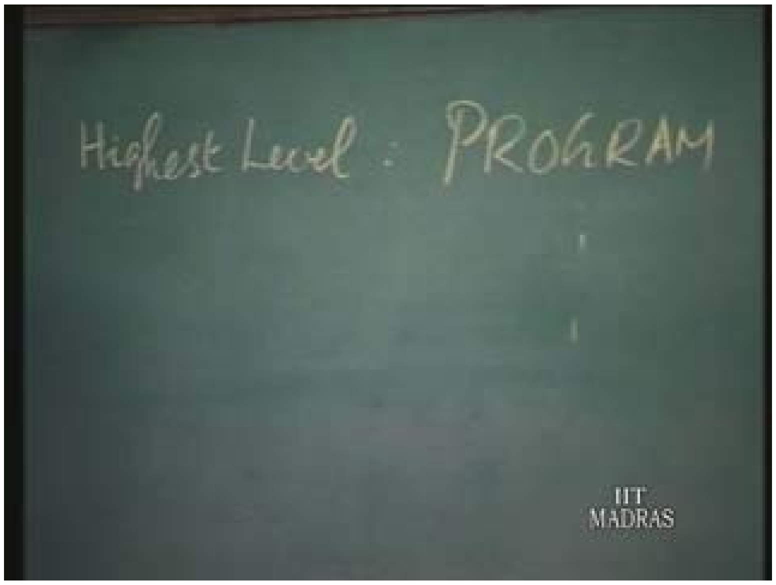
Computer Organization
Part – I
Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology

Lecture – 6

Data path Architecture

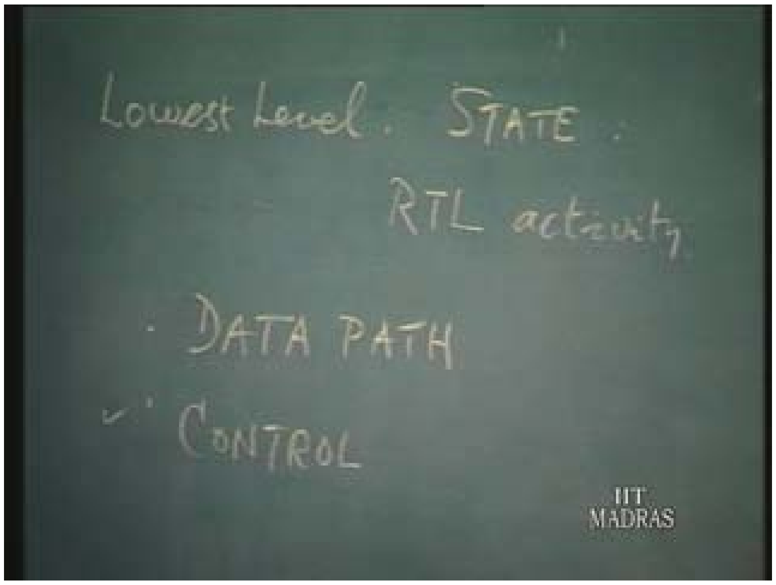
In the previous lecture you got some idea about the control, that is, specifically data path control. Now in this lecture, we will take a look at the data path architecture. Let us go back a little and then see the highest level we have, the user point of view, the program; this goes through many levels.

(Refer Slide Time 00:01:41)


Highest Level : PROGRAM
IIT
MADRAS

That is, we talk about codes and so on; and at the lowest level, we have the state information, that is, in each state there is a small activity, which we had identified as the registered transfer level activity. So while discussing this RTL activity, we saw that the sequence in which the register transfer actions must take place will be decided by the controller or the control signals, and the activity itself we saw that it consists of moving the data around some specified paths and controlling this.

(Refer Slide Time 00:02:39)



Lowest Level. STATE :

RTL activity

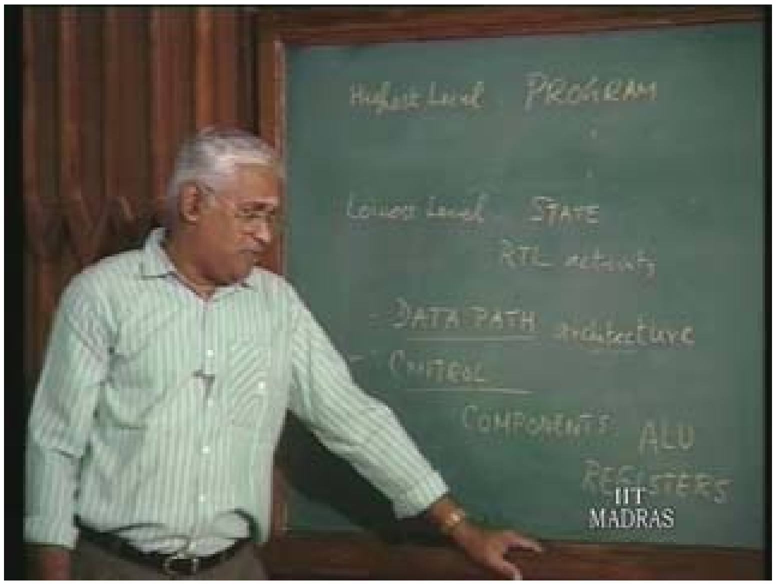
· DATA PATH

✓· CONTROL

So it gave you some idea about these control aspects in the previous lecture. Now we will take a look at the data path part of it, for which we will assume a typical architecture. I hope you would recall we said earlier that architecture is different from organization. That is not precisely what we are talking about here. Here basically it means how you can put the various components together, so that whenever the processor needs, the controller will issue the appropriate signals so that the physical connections will be enabled so that the data will move around.

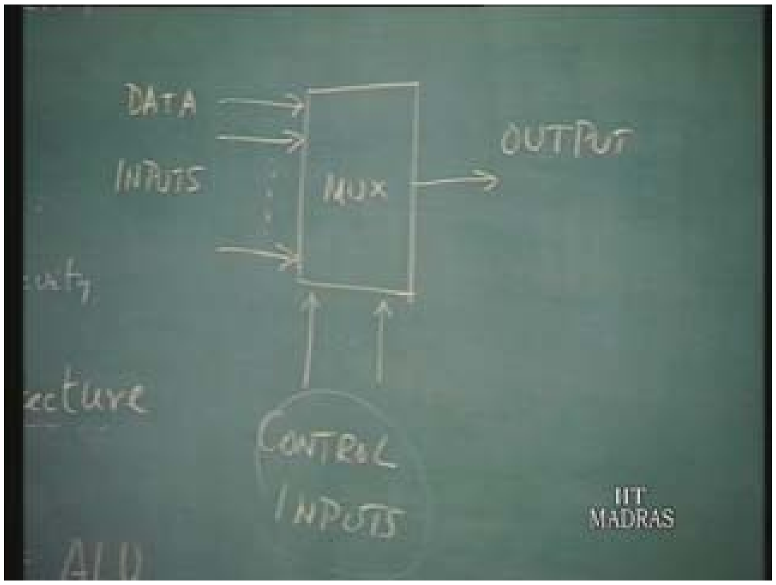
Now we will see more of it – as I said we must have some idea about the components involved in it. Some of what we knew already is what we had seen. First I said ALU or the arithmetic logic unit, which forms core of the processor. Then, while talking about the processor, we also came across this component, that is, the CPU consisting of registers; we saw some of these registers earlier. Basically it is the one which holds information on the CPU side. Now we will see some more components, after which I will discuss typical data path architecture.

(Refer Slide Time 00:04:16)



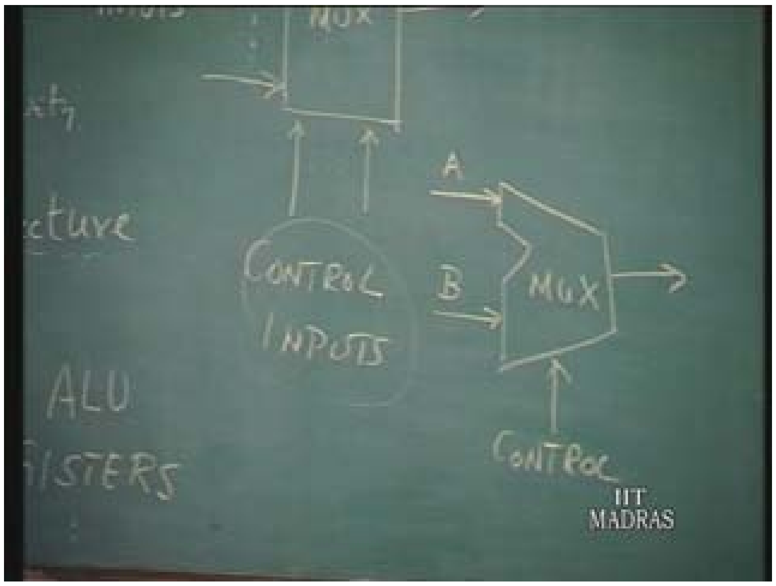
Today I will introduce one more thing; we will see more about ALU. Before that, I will talk about one component, which would be used in the architecture level for the data path; that is called multiplexer. What exactly is a multiplexer? It is sometimes simply referred to as a MUX. Essentially it has multiple inputs, which are called data inputs. They are more than one; it can be many. These are called the data inputs, and depending upon another set of inputs called control inputs, one of these will be output. So the output of the MUX is one of the data inputs – which one will be decided by the control input.

(Refer Slide Time 00:05:37)



You can see parallel to what we are talking about earlier. We are talking about the path on which the data will move, and that particular path will be decided by a set of control signals. In fact you can see the summary of it even in the single component, that is, multiplexer. So multiplexer is what we will use; for instance, if the multiplexer has two inputs – I will be using a diagrammatic representation like this – that is, a MUX has two inputs called A and B. Sometimes we refer to this as A leg of the multiplexer and B leg of the multiplexer. Now one of these will be output – which one will be decided by a control input. It is easy to visualize. You can have more than one in which appropriately we must have the set of control signals, which uniquely identify them. Similarly, let us take a look at this ALU we have been talking about earlier. ALU is the arithmetic logic unit, that is, a unit which is capable of performing a set of arithmetic functions and a set of logic functions.

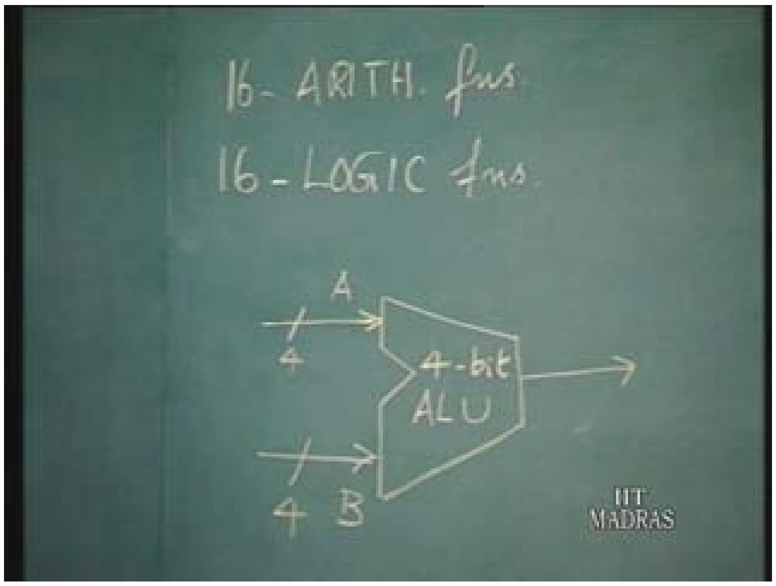
(Refer Slide Time 00:06:36)



I will assume for a purpose that it is an ALU, which is capable of performing let us say 16 arithmetic functions. We will talk about a set of data and then it will be carrying out some functions – 16 arithmetic functions and, let us say, 16 logic functions, that is, the ALU is capable of executing 32 different functions. Now we have to assume a few other things like what is the data size of the ALU? We will assume again an ALU accepts two sets of inputs – what is the size of the data? In this particular one, I was just mentioning data input; we are not talking about the size.

In the case of ALU we have to talk about that too, because one of the arithmetic functions can be addition. When we talk about addition, we have to indicate whether it is a 4-bit addition or an 8-bit addition – 4 bit or 8 bit actually refers to the size of the data. Now suppose this were a 4-bit ALU, then precisely what we mean is the data input, which comes on the A leg, is going to be of 4-bit size, or is 4-bit wide. Similarly, the one on the B will also be 4-bit wide. If it were an 8-bit ALU then we will be having an 8-bit A input or an 8-bit B input.

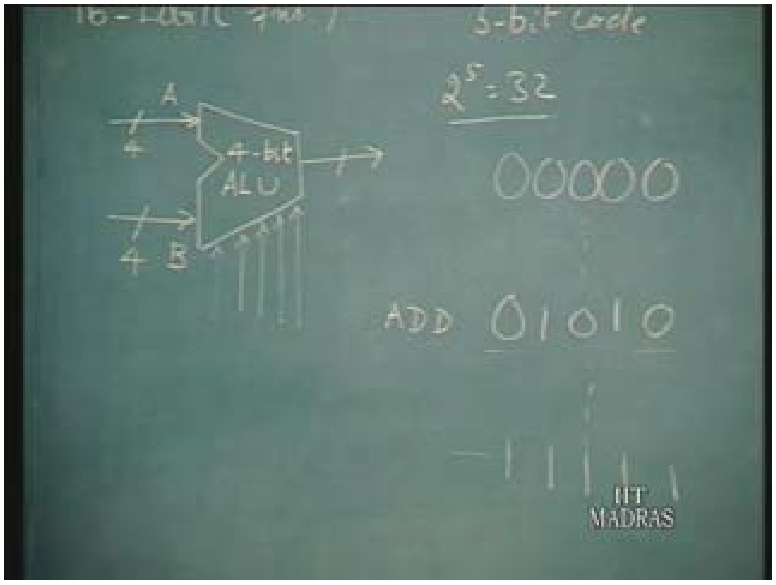
(Refer Slide Time 00:08:54)



That apart, there can be more than 4-bit output; finally there is a possibility that the addition of two 4-bit numbers will result in a 5-bit number. Generally what we say is a 4-bit sum and then an extra output. We will not bother about that because we are not only talking about only addition here.

Now how will these 32 arithmetic or logic functions be indicated? They will be indicated using the set of control inputs, which go to this. Because we have 32 functions in total, if we have a code, a binary code, let us say then we need a 5-bit code, mainly because a code which indicates the functions, mainly because 2^5 would correspond to 32. So a 5-bit code will uniquely represent constant codes; such as this will represent one function; another code at the arbiter level will represent specifically another of these functions and so on and so forth. So, with a 5-bit code you can have as many as 32 combinations and that code uniquely will identify what exactly has to be done. For instance, this may be a code for one of the arithmetic functions, let us say, add. And this possibly may be a code, which represents one of the logic functions.

(Refer Slide Time 00:10:46)



We have not talked about logic functions; I will possibly give some idea about this later on – and sometimes we may have even a code for a simple thing like passing on just the data A on to the output. I will just call this as the output of ALU, name it as OUT, in which case, given two sets of data, A and B, I am just assuming there is some function, which simply processes on to the output leg the one which is on the A.

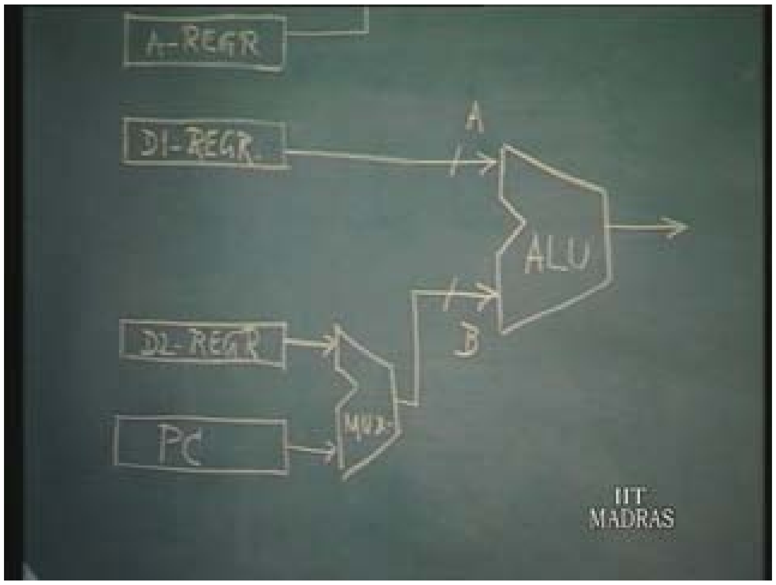
There may be another code for this; now let us just say this is the port for that and so on and so forth. Like this, you can go on defining different functions. So ALU, which is the heart of it, because it has 32 functions, will be getting a 5-bit code, which will uniquely identify what this particular one is going to do. Similarly, the MUX should have a control signal, which will pass on A or B. Now MUX only passes on the data, whereas this ALU will be carrying out some function, arithmetic or logic, as in this case. This is also one of the functions – it simply passes on. With these functions, let us try to develop the data path architecture; we have to start with the ALU as the core. Having seen ALU as the core and then the function of the multiplexers and registers, now we will try to put them together and develop what I will call typical data path architecture. It is typical in the sense that there is nothing very sacred about this; it is something with which we can carry on our discussion about data path.

What did we see earlier? We had some registers, for instance, and I will just call instruction register I register. Then we were talking about an address register. I will call the address register A register. Then we were mentioning about data registers so I will just assume that we have say two data registers; I call this D1 register or just one of the two data registers. I will assume another data register, let us say D2 register and then I was previously telling in the previous lecture, I was talking about the address being placed on the contents. I was trying to tell you that the program is actually executed instruction after instruction, there will be one register which works more as a counter and the name of that particular one is program counter.

That is, program counter usually holds the address of the instruction so you call that a PC. The address of an instruction to be fetched will normally be available in the PC. This will be passed on to the address bus and then at the bus level essentially we have the address register interfacing with the address bus part of A or just call it as A bus. Let us not worry about what the width of that particular one is. It is meaningful if this is a 16-bit bus; then it is meaningful to have a 16-bit register. We have discussed this part of it earlier. Now the ALU will subsequently be involved in carrying out some processing on the two data, but then subsequently we have to possibly also pass the program counter contents to the A register.

So the path for that also must be created; what I will do here is introduce one multiplexer; if necessary we will name it later. So either content of this or the program counter content will be passed on to this ALU. Then, in case of arithmetic or logic processing, the D2 register contents will go over to the B leg and I will assume here the other data A is available from D1 register.

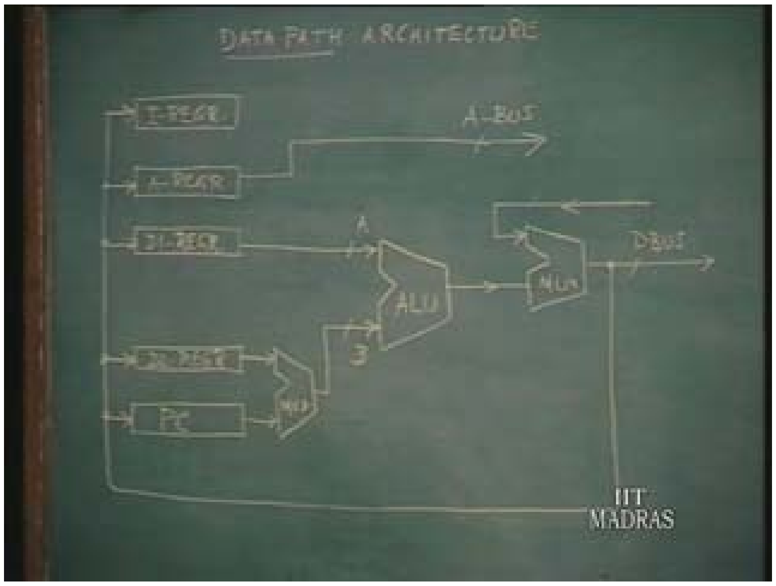
(Refer Slide Time 00:16:19)



I am trying to keep the number of components as low as possible. The ALU output goes through another multiplexer; why? We assume a multiplexer here. We will see that sometimes the processed data may be passed directly over the data bus or the D bus. Specifically, it is the output part of the D bus or the data bus and there is also the input part of the data bus, that is, data may come in. This is also in fact part of the D bus; so you can see that sometimes the CPU will place data on the bus and sometimes the data will be accepted by the CPU. So when accepted by the CPU, the data will be coming over this; when the CPU places the data, it will be going over that. For instance, in the case of memory read, the CPU reads from the memory, so the memory will place on this and the CPU will accept and route it internally. When the CPU writes into the memory, the CPU will send the data over this.

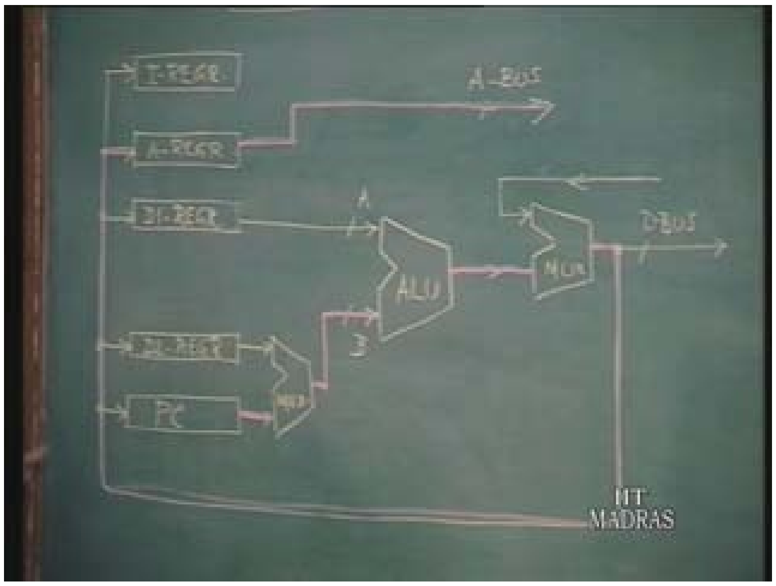
So that is the arrangement made, and for simplification I will just assume the data that comes over this gets routed to any of these registers like this. Of course, one can go on making more complex diagram than what is shown here.

(Refer Slide Time 00:18:14)



For the purpose of our discussion, I will just assume this is the data path architecture; now what is this? For instance, when the CPU places address, now shall we go back to that state after state information? In T1 we are saying the CPU places the address on the bus; so in other words the CPU places address on to the A bus, address part of the bus. We were in fact discussing instruction fetch; remember? In which case, the instruction will be pointed to by the program counter contents; essentially, in T1, it is the program counter contents which will be placed on the A bus. Now for this architecture, what exactly it means is this – a path must be set so that the program counter contents will be placed over the A bus and this in fact is D path and then it is placed over the A bus; this is D path.

(Refer Slide Time 00:19:51)



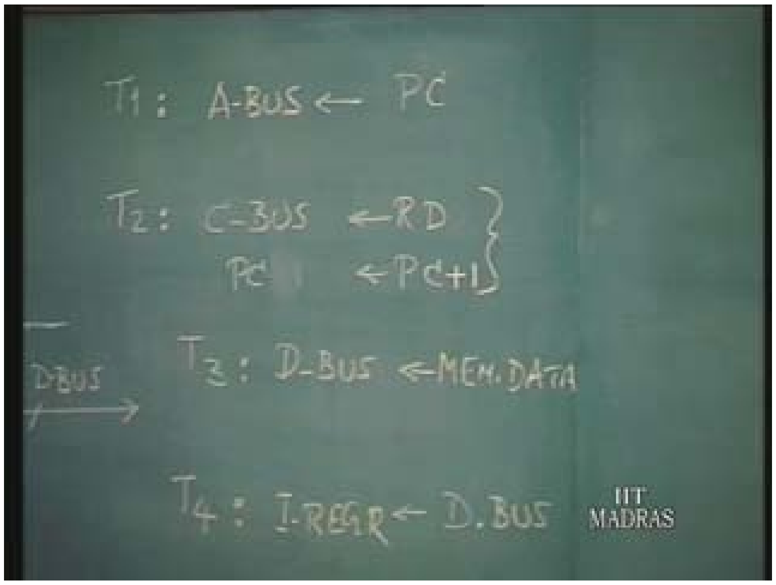
So now if this path is set up during T1, then the goal is achieved. The content of PC includes the address of the instruction to be fetched, which is available here. This must be routed through this multiplexer, passed on to the B leg of the ALU, and the output of ALU must be passed on through the MUX to the A register, which must be enabled so that whatever is there in the A register is passed on or placed over the A bus. So essentially you can see ultimately the contents of PC – it is again a register or counter – will be transferred to the A register and that is made available on the address bus. This is precisely the data path that must be set up during T1. Normally we say that since a program is executed instruction after instruction, we would expect the PC contents to pass on to the next location; so that is the next step that must be done.

We did not discuss that when we were discussing the previous thing earlier. Then I think we said during T2, the CPU puts a read control signal. Now it is not really that it can be shown here; I will just write down nevertheless. Because we have not shown where exactly the control part of the bus is here, we will not bother about the read control. So the controller must place the read signal on the control bus during T2. Then in T3, assuming the memory responds immediately, the CPU is not involved.

We said the data from the memory, which I will call as the memory data, the data from the memory is placed on the data path of the bus. Essentially the memory will respond and put it on this. Now again, CPU comes into the picture and it moves whatever is there on the data bus. Since we started saying the data that is fetched is an instruction, it must get routed to the instruction register. This is what we discussed in detail in the previous lecture. Here, after program counter contents are placed on the address bus for which specifically the data path is shown, when during T2 the CPU is placing the read control signal because it has to fetch, the CPU must read it.

At that particular instant, for instance, we can see that the program counter contents are incremented so that next time, that is, in the next machine cycle if necessary, the program counter can point to the next byte or address location. That means basically these both, these signals, can be done in parallel. The contents of PC must be incremented and placed in PC.

(Refer Slide Time 00:23:50)



$$\left.\begin{array}{l}T_2: \text{C-BUS} \leftarrow \text{RD} \\ \quad \text{PC} \leftarrow \text{PC+1}\end{array}\right\}$$

T1 : A-BUS ← PC

T2 : C-BUS ← RD
PC ← PC+1

T3 : D-BUS ← MEM.DATA

T4 : I-REGR ← D.BUS

DBUS
IIT MADRAS

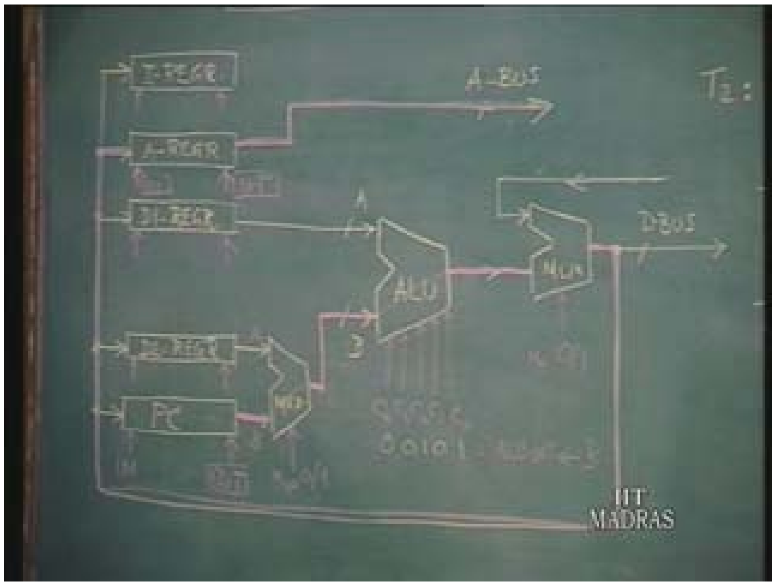
The contents of PC must be incremented and the new value must go into the PC. Now let us say this: I will just take this data path, which has been in T1. I hope it is clear in each state the data path such as this must be set up similarly, in the case of other states also. For T1, it will be shown in the case of other states also; similar T must be done. Let us see here – what is this multiplexer's role? The multiplexer essentially passes on whatever is there on this leg or on this leg on to the output.

So the control signal here must be given so that shall I again call it A or B. Either the one data register contents can come on A or this must be passed on. So the control signal here must be such that B gets passed on to the output during T1. Suppose I will just assume that on to the output either this or this is always passed on; then it is enough if I have only one control symbol. Now depending on whether it is 0 or 1, if it is 0, let us say, A is passed on; if it is 1, B is passed on to the output. So this in fact is the control signal, whereas this is the data signal; this is the control signal.

Similarly we will assume the same old 32 function ALU, in which case there will be five control inputs. So a 5-bit code is basically what you have. I will just call this particular code C_0, C_1, C_2, C_3, C_4 – that is a 5-bit code. So one particular code, which says the output of ALU will be what is placed on B that particular code must be here, in which case the output will assume the B input. That code must be generated during T1. Similarly, here too I will just assume a 0 or 1 – a simple signal. So, why not I call this particular signal M_0? This may be 0 or 1 and I will call this M_1.

This particular signal is 0 or 1. And what should this be? I said some code; you just assume if this were 00101, that particular code implies that out of ALU, on to the output of ALU, the B contents will be placed.

(Refer Slide Time 00:27:16)



So this particular code must be there. What about the registers now? We have two things: one is the input for the register and the other is the output. So we must have control signals enabling the input side as well as enabling the output side. The way I have drawn this particular figure, it may appear as if something enters and travels down the register as in the case of shift register. It is not so really; what I mean here is if a 4-bit or an 8-bit code comes, it is going to be passed in parallel. Some control signal will enable the IN or OUT of these various signals.

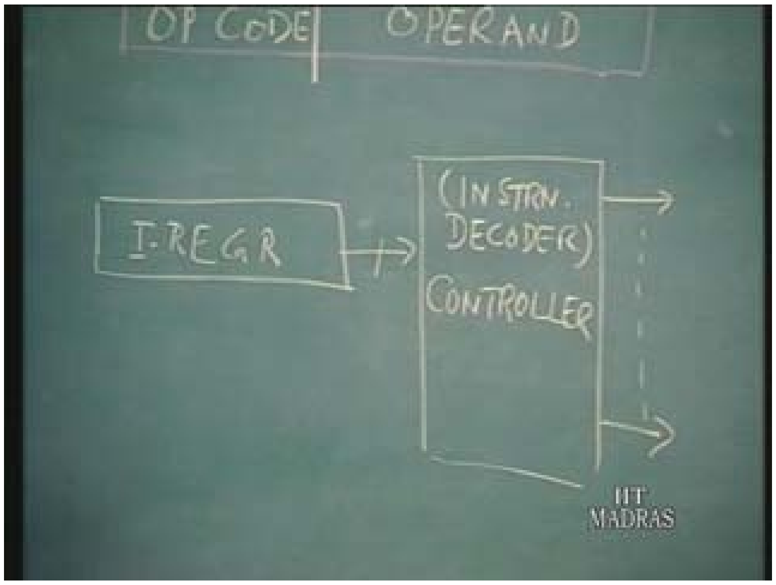
During T1, we have to see that this OUT signal is enabled for the PC register so that this OUT signal will enable the contents of PC passed on over the v leg input of the MUX. Similarly, during T1, we have to see that this input of the A register must be enabled; similarly output of the A register also must be enabled so that it will move on to this. In other words, we need to set this IN and OUT of A registers enabled so that the information can go through. Sometimes it may be done in two steps. Just the information that the program counter must be passed on to the A bus and one state T1 to the register transfer activity means generating the PC OUT signal and setting M_0 to 1, so that this information can go on to the output and generating the appropriate code for implementing this function for passing the B input on to the output and enabling this MUX so that this information and not this is passed on to this. And finally, the program counter contents go over this path and finally they must reach A register. This control signal must be enabled and then this also must be enabled so that it is placed on the A bus. So now you can see that the architecture, which we have assumed, talks about a collection of components, say registers, multiplexers, and ALU in this case, and the data path is set up essentially because of the generation of the necessary control signals.

These control signals when generated and given to the respective components at a given instant, that is, a state duration T1, will achieve place transfer of PC contents on to a bus. This is one step – like this in each state, the control signal must be set, which will set the data path. We can go on discussing for each of these. For instance, I will go to the last state T4; let us say that we have assumed that memory has placed the data on the bus and it is available on the input part of the data bus and it must get routed to the instruction register. So the data path for that would be enabling so that the data coming over this gets routed to this and finally it must reach the instruction register.

Now once the CPU gets the instruction, then depending on what that instruction is, it will start generating the necessary signals for their implementation of that specific instruction, because I was telling that the format is the set instruction format. It is going to consist of essentially two things: one part of it is the opcode, which tells what the instruction is. The other part of it is giving some information about the operand. How many operands are there and how these operands can be referred to will all depend on what exactly is the instruction it must carry out. Now once the instruction is fetched into the instruction register, the CPU proceeds and the instruction may be just say 1 byte or 2 bytes or 3 bytes.

Suppose we have a 3-byte instruction and if the data path width is only 1 byte, as I mentioned earlier that instruction must come in three steps, which means three machine cycles. Once the entire instruction comes in here, then it can proceed. It all depends, but the very first byte which comes into instruction register itself will give some information to the processor. So once the instruction is fetched into the processor, this will be given; I will write it here. So once the instruction comes into the instruction register, this information – it's a multi-bit instruction – will go out to the controller because our basic cycle is fetch the instruction; decode or interpret the instruction; and then execute. So the instruction register must go to the controller, which actually will include the appropriate decoding part of it; that is, I will call it instruction decoder. This must be part of it; and once the instruction is decoded, then the controller will generate the necessary signals.

(Refer Slide Time 00:35:27)



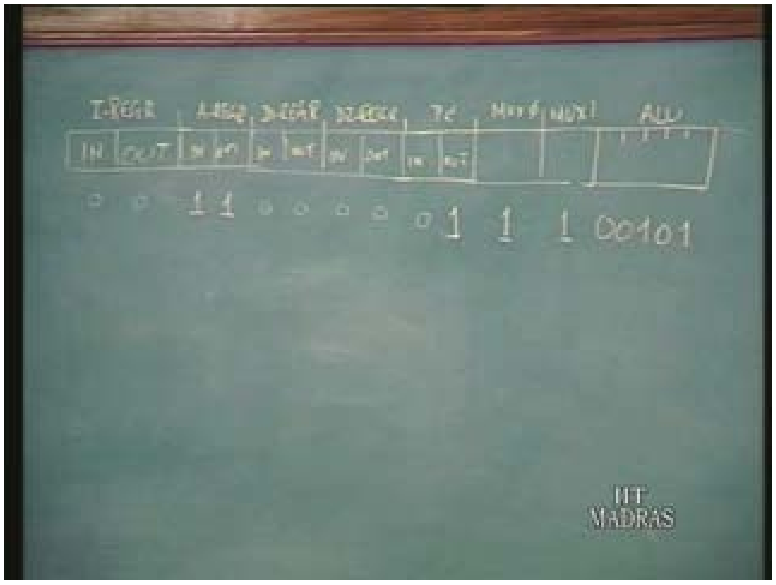
Once again, to make sure that the different data paths are set up for that particular instruction, essentially the architecture is going to help define the data path and the set of instructions. As I said in the previous lecture each instruction must be analyzed to create all these low level activities. So they would tell what the sequence is in which the control signals must be generated. So the instruction register will get the instruction; it will be decoded and essentially that will be the information to the controller, so that a subsequent data path setting up and the sequence of control signals that must be generated will all be decided subsequently.

Now let us get back to the original data path discussion, that is, just for one state in which the PC contents are placed on A bus we said this is the data path and the path is specifically set up mainly because the controller is generating these various signals. Now let us go into detail and work out; that is, in this particular MUX, M_1 must be specifically 1, so that this is passed on to that, and ALU must be set in a specific operation note, so that the code let us say C_0 to C_4 is this one. This is very arbitrarily done; that is 00101.

Assuming that is the code which helps the contents placed on the input B leg of the ALU input to be routed to the output, we get this over here and then this particular MUX. If not, again, let us say it is set to 1, so that this information goes there, and to see that the PC contents are placed on the B leg of this MUX, this output of the PC must be enabled. Going forward, the output is enabled so that it comes here and this is 1; so that this comes and not the other one and so on. When this comes, let the PC contents get routed over this and it must enter only the A register. So this input must be enabled – this is another control signal – and since this must go on to the A bus also the output of this register must be enabled. Now we have other control signals also; that is, in and out of a register like this in and out of all these registers, specifically only out of PC and in and out of A register must be enabled during T1; then we get this. Similarly we have seen these things in detail.

Now let us work it out. Let us say if we have this code, that is, I register has IN and OUT signals and so on. Then we have the A register; similarly for that also we have IN and OUT; then we have the D1 register with IN and OUT; then we have the D2 register with IN and OUT; then we have the PC with IN and OUT; then we call this MUX 0 and that MUX 1. We have MUX 0 and that is only one particular one then we have MUX 1 when we have the ALU, which is a 5-bit code. This is only 1 bit here and then a 5-bit code; now to see that this data path is set up during T1, essentially you start with this: PC OUT; that must be 1. Then MUX 0; that must be 1; for ALU we have just arbitrarily assumed the code is 00101; then MUX 1 again is 1 and then A register both IN and OUT must be 1; now we set rest of them as 0. Now we have a specific code.

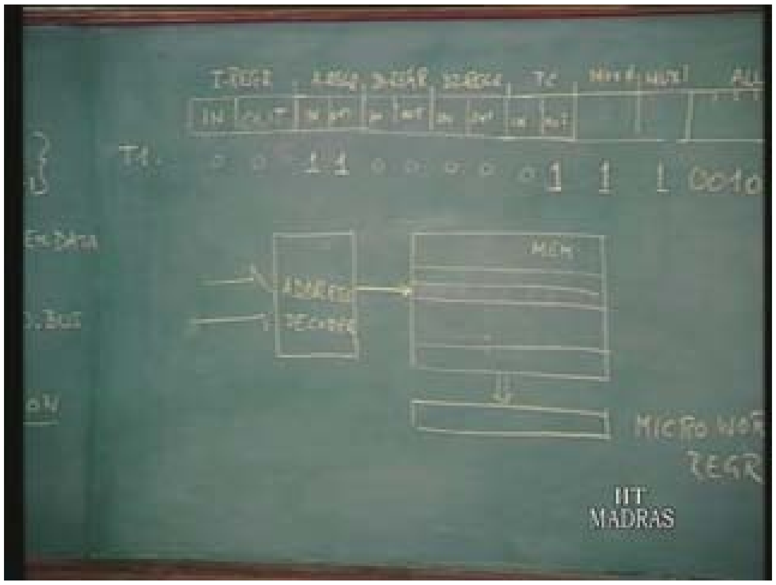
(Refer Slide Time 00:41:57)



This specific code, when generated by the controller, will set this data path so the controller must generate this signal this particular code during T1. What is the architecture of the controller? Let us see whether we can extend further. Since we know that during T1 this code must be generated, similarly for T2 we can arrive at another code; another code for T3, another code for T4 and so on for one machine cycle. Then for each of the instructions, we have many machine cycles. For each of those machine cycles, we will arrive at the same thing. So if we put them all together, we would have got the codes that are needed for executing all the instructions of the CPU. So assuming that you put all these codes in memory, we will put all these codes in the memory and so on. From the memory, if you any time read only one of these codes, that is, in our case let us say this location contains T1, this location contains T2 and so on and so forth, it will be similar for all the instructions, and anytime you just read only one of these words, then in that particular small step, which is a micro step because an instruction is executed using these micro steps and so each of these is also called a micro instruction. We are seeing specifically that microinstruction is nothing but something that sets the data path; that's what we are finding.

For each instruction the set of microinstructions must be executed, and now see also that each microinstruction specifically is going to set the data path and tell what the CPU must do in that particular state – take any state, that particular state is 1. Generally this one is called data path control signal. This particular word is essentially for the data path control, so it is essentially a control word and since it is one small step giving the microinstruction, generally we call this one micro-word. This one micro-word is in the memory; when you read one into a register like this generally this will be called a micro-word register. Recall our discussions on memory earlier – set memory usually is organized so that we talk about each location and each location has a unique address. We find that if we talk about address of this which we say contains this code, essentially address is nothing but the state number itself, that is, given a machine cycle a specific state in that machine cycle is unanimous with this particular address also. So if we have a decoder, which generates a specific unique address, so this particular decoder will take in some address whatever be the size let us not worry about that some address.

(Refer Slide Time 00:47:56)



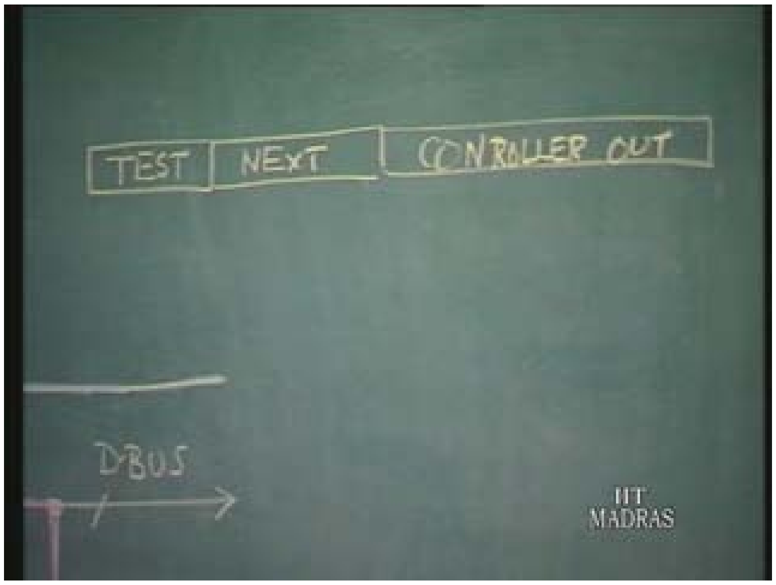
That address must uniquely generate the code, and the data contents from that memory will be read in to it. Then we would be having the data path control signal for that particular state. We said the CPU is essentially a state machine. It goes through state after state. So at any time there is only one state, which means in that particular state the information needed for stating the data path is made available. It so happens that what we are seeing here is only the control signal and this entire thing is to be used for setting the control part only. So to this memory word, we have to add other things so that we can really sequence through these various things. We will just put it as what we have is the data path control word and that is essentially the output of the controller in setting the data path. That is just what we have here. So we can allocate one location for T1; in another word we can accommodate the control signal code for T2 and so on and so forth. But then, how do we sequence through these? That is another step; for a study of that sequencing you have to recall what we discussed in the previous lecture.

That is, we were discussing T1 in terms of some state code, T1, T2, T3 or T4. So during T1, the output that is actually generated will be this particular code; that is, actually the output, which enables the PC contents to go on the PC contents, and getting routed to A register, actually, to the A bus or the address bus. We saw this in the previous lecture and we also introduced a ready input signal; that means, the sequencing is associated with a given state – what is the output that is to be generated so that the data path may be set up; and also, what is the next state it must go to; and in some state there may be occasions when some input also must be checked.

For instance in T2 we were seeing in the previous lecture that a ready input must be checked by the CPU; depending on that the next state will depend. That is, for instance, if T2 is the present state depending on the ready being 0 or 1, the next state will be in this particular aspect, in the ASM chart, the next state will be T2 itself or it will be T3. This sequencing information must also be included in the memory; that is, this particular micro-word will not only consist of whatever has been shown there, which essentially is controller output. So I would say it is the controller output signal, which will be used for setting the data path; it must also say what the next state is and what the inputs to be tested in each state are.

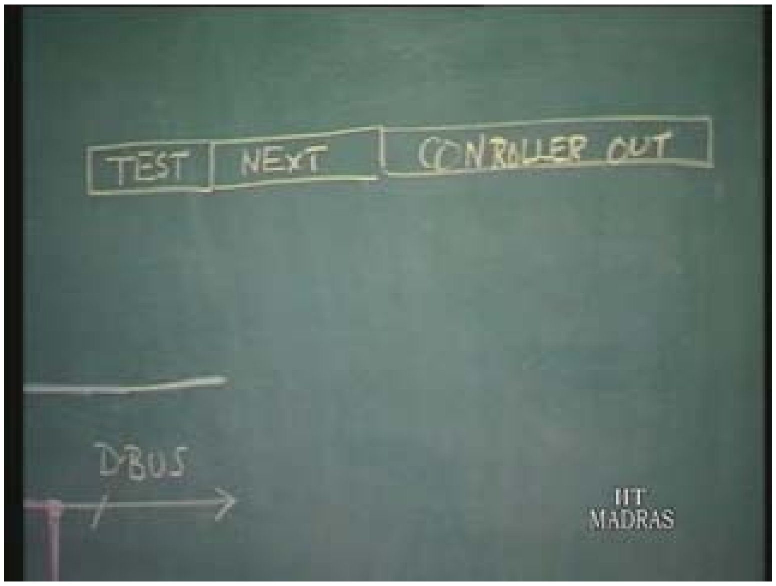
Now what exactly is this? This is for a given state – we said for T1 state, which corresponds to this state code, T1, here in this chart. Now we must also have information about what is the next state; it so happens that for T1, the next state is T2, but in the case of T2, it depends on an input. It may be T2 or T3, and then another one is in a given state. What are the inputs to be checked here? That is generally called inputs to be tested or we just call that as test.

(Refer Slide Time 00:52:00)



So a micro-word in a memory, which describes the control sequence, will not only have this, which is only the output part of it, but it must also give the sequencing information that is giving the next state for a given state. What exactly is this word? This word is for a given state; so given a state what are all the inputs to be tested? That information must be there. Then depending on the condition of this, what is the next state, it must also give this particular one. This must be worked out for each state, for all the machine cycles, which means for all the instructions. So the CPU design as I was mentioning in the previous lecture starts with getting the instruction set; that is, sometimes I use this word instruction repertoire. That is nothing but the set of instructions for a CPU, then each instruction will be worked out having the necessary machine cycles, the number of machine cycles depends upon the type of the instruction and the width of the instruction.

(Refer Slide Time 00:53:29)



As I was already mentioning, if an instruction word is 3 bytes, you need three machine cycles for fetching the instruction itself. And then you have the execution part also, and then each machine cycle consists of specific states and in one state, one low level activity, a register transfer level activity takes place. What we have here is one such activity, for which the data path will be set up. Now for a given state T1, this word will give information on what are the signals that must be generated and what are the inputs that must be tested for the given state. For instance here in T1, there is nothing to be tested; in T2, we will see that ready is an input which must be tested – that information will be here. Then it also says – depending on the condition of that – what must be the next state. For instance, in the case of this T2 state the next state is T2 or T3, depending on that particular input. So there are essentially three fields of the micro-word: one is the output field, and one is the next field, and another is the test field. The test field essentially is telling what the inputs to be tested for that given state are.

So once we have this information and put the entire thing in the memory, then the whole behavior of the state machine is there in that memory in terms of these micro-words, for which the starting point is what are the instructions to be tested? And what is it we have assumed here? We have started our discussion with an assumed architecture. There are different reasons based on which this architecture will be developed; we are not really going into that. So we have only taken this architecture as the starting point and then discussed how these various things get related to each other.