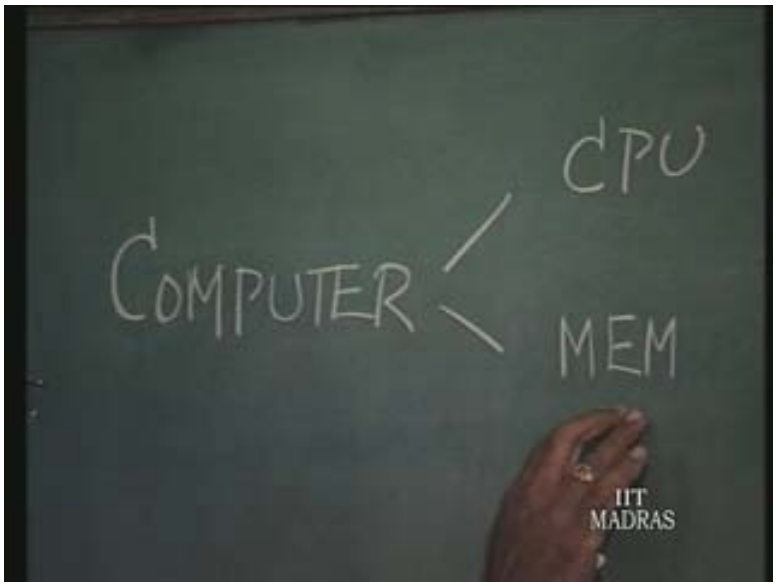
Computer Organization
Part – I
Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology
LECTURE – 11
Typical Micro Instructions

In the few previous lectures, we had worked out the details of the control unit of a computer. Now before we proceed further, let us quickly review what we have done so far.

(Refer Slide Time 00:01:24)

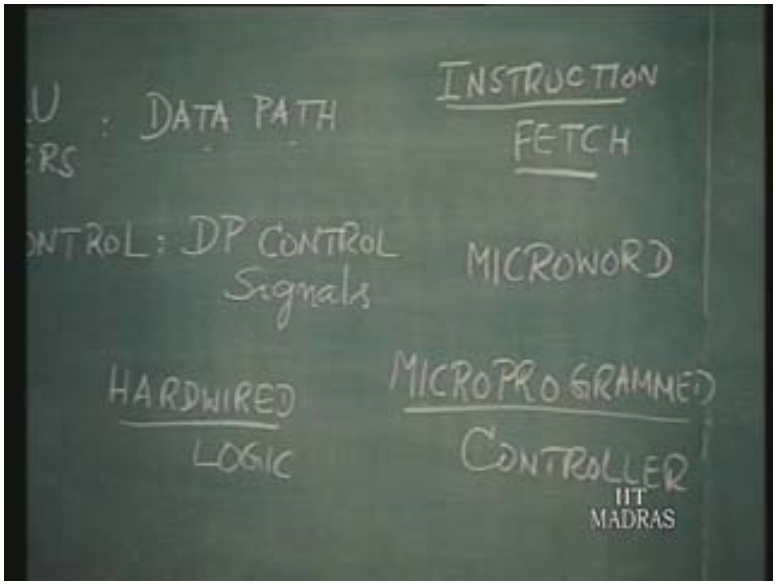


We started with saying that a computer system essentially consists of processor or central processing unit and memory, and we had also discussed quite at length I by O as an extended memory and so whatever we may discuss about CPU memory interaction, roughly it would also hold good for CPU I by O. We will talk in more detail about it later. Then we took up the study of the processor. CPU essentially consists of a processor, or we may say the arithmetic logic processor or arithmetic logic unit, plus a few registers. Together these consist of what we may call data path architecture. The processing is either arithmetic or logic as it says; and registers are part of CPU holding information much as memory will hold information. Then we worked out the data path. In fact we just considered a typical data path architecture; nothing specific but a much generalized structure and worked out the sequence of actions, and the sequence in which the data should move for which the path must be set up with the aid of the appropriate control signals. So we said CPU essentially consists of the processing unit, which are both generally arithmetic logic and the other part as the control.

In fact regarding this control we went into much discussion; we worked out. That is what we were doing in the last few lectures. The controls will essentially set up the signals, so that we will call this data path control signals. The control unit or the control will generate the necessary control signals, so that whatever that is necessary in the processor, that is, in setting up the path for the data, will be achieved by these control signals. We worked out the details specifically for instruction fetch, from which we arrived at the concept of micro-word, because essentially the instruction fetch, as we saw, consisted of four states' information, that is, information related to only four states. For each state the signals that are necessary to set up the data path; that information is available from the micro-word.

So we directly went into one specific implementation of the controller, which we may call a micro-programmed controller. Basically, to get an idea about other types of implementation, that is, implementation other than the micro-programmed type of controller, we also studied the details of a hardware controller. We may call it a hardwired logic controller.

(Refer Slide Time 00:05:45)



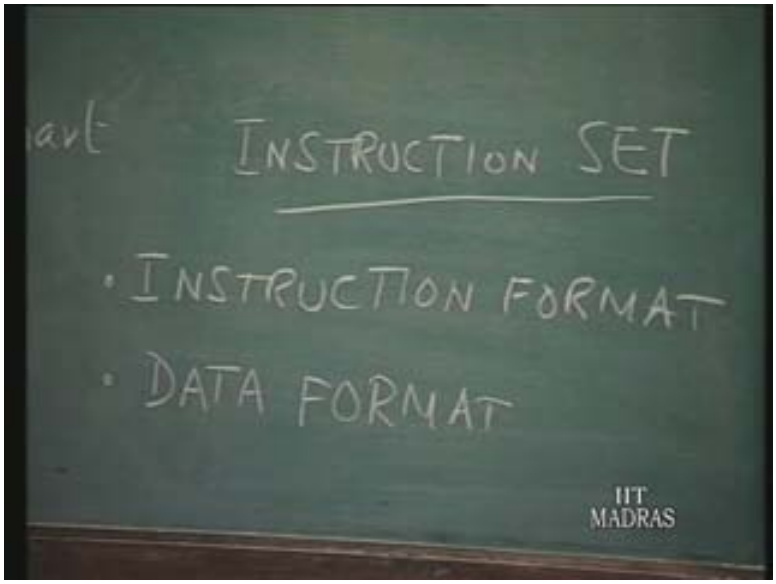
That is, in contrast to the micro-programmed controller, you have a hardwired logic controller. Both will achieve the same, but then the implementations are different. Now we saw that a processor is essentially a state machine; as a state machine, the particular thing is derived because the machine goes through state by state, and corresponding to each state, a set of signals that are required, essentially control signals that are required, will be available in the micro-word in the specific implementation of a micro-programmed controller. The processor is a very complicated state machine; we cannot really work out the full details of what will happen in each state for the entire processor. So what we did was to consider a simple example; that is, we took as an aside actually, a master–slave flip-flop, specifically JK flip-flop as one example of the state machine.

In this particular case, there are only three inputs: YJ, YK and YC to be considered; that is, J and K being the two inputs and C is the clock input, which essentially defines the state duration. And we said it will generate two outputs, one for 0 states and one for 1 state; so essentially we call them as H ONE and H ZERO. In trying to work out the details of the relations between the outputs and the inputs, we saw that we call this particular one a system or a machine or a component – this particular machine was going through four states. So with four states we are able to describe the behavior of this entire flip-flop, and with three inputs, two outputs and four states, we are able to completely work out the details of first the micro-programmed controller, and then also go into the details of the hardwired logic; this is what we were doing in the previous class. Now state by state, any system's behavior can be expressed, for which we were using the algorithmic state machine or what you may call an ASM chart.

We saw how this ASM chart gives you the information about state by state behavior, and we consider state by state what is going on. And then I asked you to extend that concept in the case of micro-programmed controller or a hardwired logic controller of a processor also. Whatever we did for four states, in the case of processor we will have to do similarly for 40 states, may be 400 states. Actually I have to tell you that the detailed study which we were carrying out in the previous lectures essentially is part of architecture; it is not really part of an organization. But an understanding of what is going on there is essential to appreciate what is going on at a higher level. So because you can see that all these things are from the designer point of view, whether it is a micro-programmed control or whether it is a hardwired logic control hardware control; the user is not really concerned with it, it is only the designer part. Essentially, it is the architecture; that is why we are also talking about the data path architecture. Now let us go to the user point of view – what are the things he will be essentially concerned with?

First, what are the various instructions that the processor supports? At this point, let me recall what I was telling you a little earlier also. For each instruction, that is, we were talking about a set of instructions or instruction repertoire – I will just use the word instruction set. A processor is capable of executing a set of instructions for which it will have some controller. For each instruction, details will have to be worked out about all the states, through which the processor goes and then finally the controller for the processor must be designed. Now talking about the instruction set, that is, the different types of instructions which a processor can execute, we have to know two things – one is, what is the format of the instruction? This would also to a large extent decide the internal architecture also. Associated with that, we also have to be concerned with the data format. What is it we are talking about here?

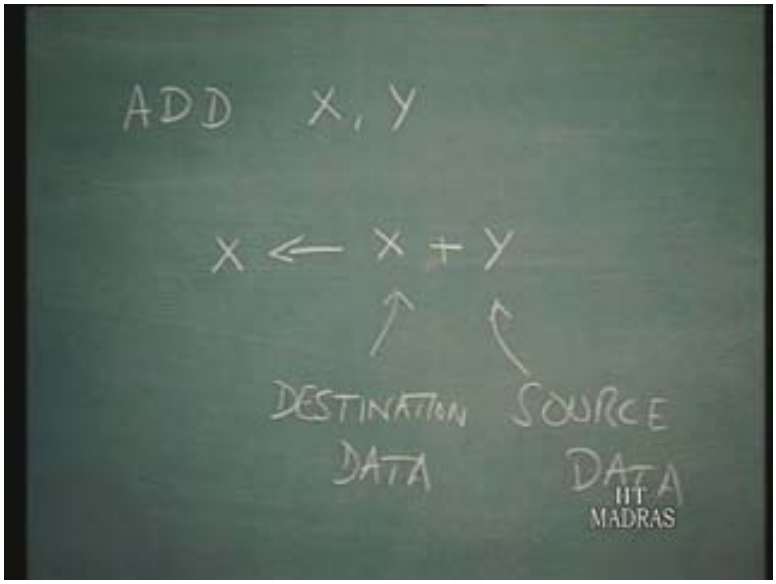
(Refer Slide Time 00:12:00)



Let us first take this: basically it is the format or the form in which the data is available or the form in which the instruction is available. So we will just take the instruction format. Now an instruction essentially would consist of indicating what the operation is, and so there is a code which indicates. It is the opcode, which says what actually the type of instruction is. For instance you can have a simple move instruction, that is, moving from, say, one register to another register, and there may be another instruction, say, for add, which will tell ALU that it has to carry out addition like this. So the code will uniquely identify what that instruction is and the second part of the instruction is the operand or operands; it all depends on how many operands. So that is what it is – there must be one field, which indicates the opcode and another field, which indicates the operands in which the operation will have to be carried out; that is, the instruction format essentially will talk about that.

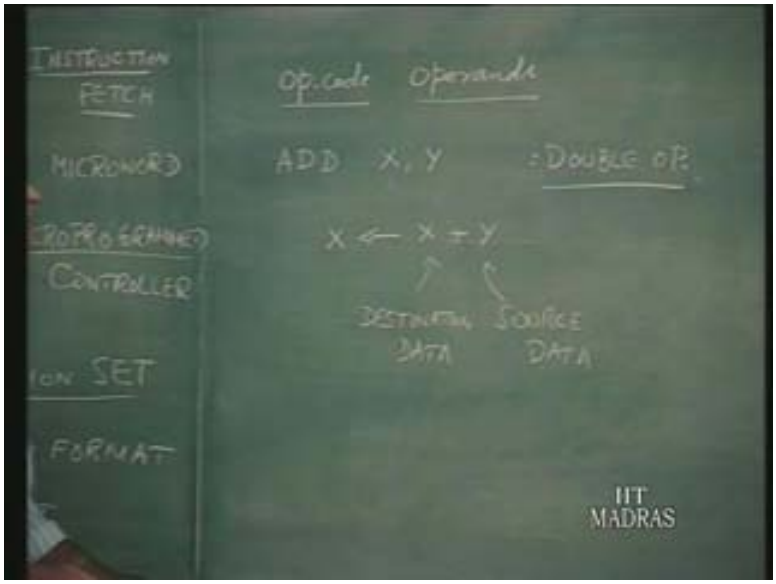
Let us assume that we have three types of operands: for instance, it is easy if you have addition. Then certainly, we would talk about add as the operation and the operands will be at least two, say A and B, or X and Y. So now, if you have this kind of an instruction – this is an instruction, an add instruction – this instruction says add X and Y. X and Y are added but then where will the sum be? You can also say, for instance, that the sum may be in some place or you can just assume that this particular one will carry out this addition of X and Y, and let us assume it will store the sum in destination source. So there are different ways in which it is done generally this will take it. It is what we call one data as source data, that is, the source operand, and the other one we may call it a destination operand. Why do we call them so?

(Refer Slide Time 00:15:05)



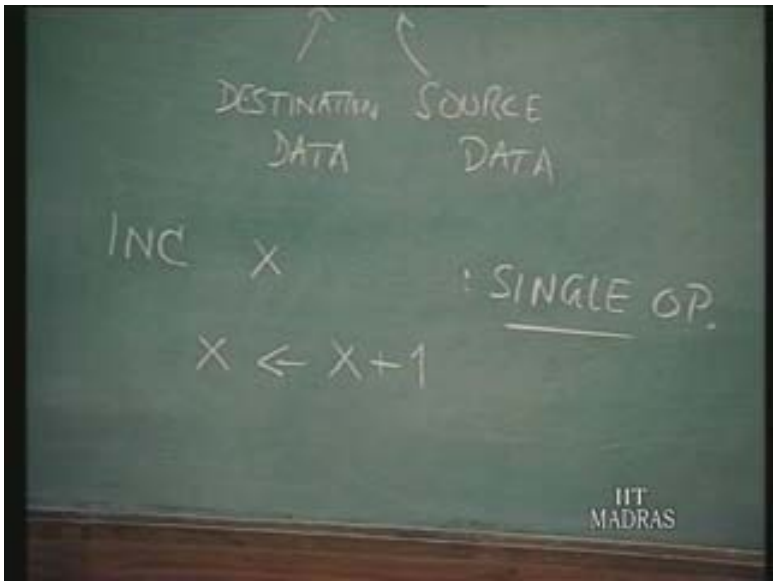
You will be able to say it right here, because the sum finally goes into this. Let us assume X and Y as two registers – there can also be memory locations – so X is one data available in one location and Y is another data available in another location. So from that location – we can call that a source location and destination location. From the source location check the data and from the destination location take the data, add these and send the sum to destination location. Because this particular location happens to be the one which receives a sum, this is called destination. That is how the name comes. Now for instance, instead of ads X Y, you can also have add X Y and another thing S; then S can be the third location. There are different forms, but here we will just assume that this particular one is in fact an instruction or operation, which refers to two operands. So this is, in fact, a specific case of a specific example of a double operand instruction.

(Refer Slide Time 00:16:28)



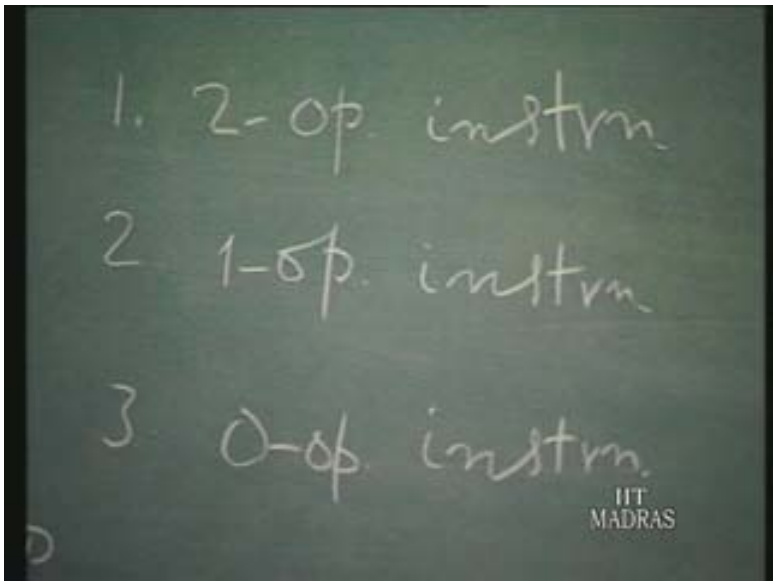
Now suppose there is another instruction, which says just increment the contents of a location, or say a register, and for instance, INC may be the code for the increment, in which case you just need to specify only one data because basically here actually it means there is a location X; from location X take the contents and add one to it and put it back. In other words the end effect of this will be to take the contents from the location X, add, and then put it back in X. Now the source and destination both happen to be the same thing. So this, in fact, is one example of a single operand instruction.

(Refer Slide Time 00:17:37)



Now there can also be a third type of category, in which no operand is specified at all; that is, there is no operand instruction; there is absolutely no operand referred to. This no operand for instance is an instruction to halt the computer; so data is needed to be specified in that case. So we have three types: a double operand instruction, then a single operand instruction, and a no operand or 0 operand instruction.

(Refer Slide Time 00:18:37)



Whatever may be the class of instructions, the format must be in such a way that the processor will be able to identify which particular one (00:18:54) an instruction is. So we will have a common instruction format, and some part of the instruction will indicate what type of instruction it is – whether a single operand or double operand or no operand. And so, you will find that the size of the instruction also keeps changing; now we will talk more about that a little later. Before we proceed to further discussion, let us also take a look at the data format. Now the different types of data will have to be accommodated because the numbers may be specified using some number system. For instance, you may use binary numbers; you may use decimal numbers; and any other number system and then, another thing is, you may specify a particular number with the decimal point in it. For instance suppose you want to specify 24.58. Then if the point always comes in a specific position in a given length, then we talk about fixed point arithmetic, that is, the arithmetic related to those types of numbers.

Sometimes we will let the point float anywhere and then indicate by another thing; for instance, 24.8 can also be specified as 2.458 into 10^1. Now we also specify it as 0.2458 into 10^2. So by having the appropriate index, we can let this particular point float. Now we talk about this as the decimal point; if it is a binary, we will talk about a binary point appropriately. So now, instead of this fixed here, we see that in this representation, by changing the indices what we are doing is we are letting the point float anywhere along.

So we talk about floating point; like this, the data format will have to give us an idea about how to accommodate different types of number systems and different types of representations of numbers. So the data format will have to be worried about that for that also. As you can see, this particular one is more concerned with the processing part; how exactly this is being done is again by designers. The user must know the types of number systems that can be used to represent the data and the types of number representation here – whether it is fixed or floating; if so, the range, because there are many things related to it. These are the things we must know – how exactly this is implemented is again the architecture, architectural designer point, not from really the user's point.

Now we will elaborate on the instruction format and then discuss a few things. May be, in order to provide some continuity with what we have discussed earlier to bring you back to the data path data path control and so on, what I would do is I will just take one example of a double operand instruction. You just take add itself and then work out the full details of the sequence of the data path control for that, for which we have to know the architecture. So with reference to architecture, we will work out the details.

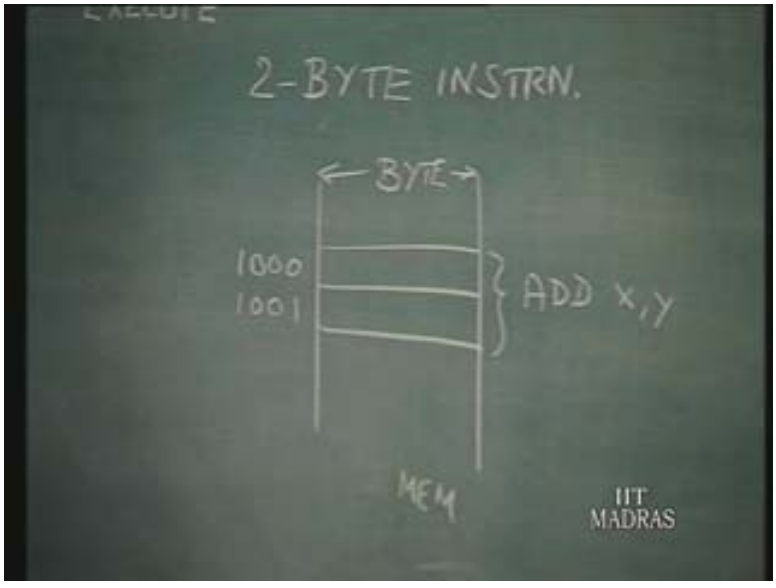
So any instruction is first fetched, and then it is interpreted or decoded, and then it is executed. In fact, we had discussed these quite at length and we also worked out full details about fetching of an instruction. While discussing that, I was mentioning we do not know much about the size of the instruction, and the size of the bus. If, for instance, an instruction is a 2-byte instruction and if the bus can accommodate only a 1-byte data, then that instruction must be fetched in two steps, 1 byte at a time, which would mean two machine cycles because we also discussed an instruction cycle consisting of machine cycle, and in each machine cycle there would be one bus access. So the bus access is actually to fetch an instruction, in this particular case, part of the instruction.

So what it does is it fetches the instruction. For instance, if we take add X Y; if this happens to be a 2-byte instruction, then in two machine cycles the instructions will be fetched. If it were a 3-byte instruction, that is, assuming the bus can accommodate only 1 byte, it will bring it in three machine cycles – this we had worked out earlier. Now let us go through this first instruction fetch now; that is the major phase. For fetching this instruction, now we have to assume a few things. I will assume here that there are two data, X and Y, of which X is already available in a register. The data X is available in a register, that is, already with the CPU, and Y will have to be fetched from memory. So now you can see that for fetching the instruction, you need some machine cycle or machine cycles and for fetching the data from the memory, the second one, you need some more – one or two – machine cycles; it all depends on the size of the data.

(Refer Slide Time 00:25:59)

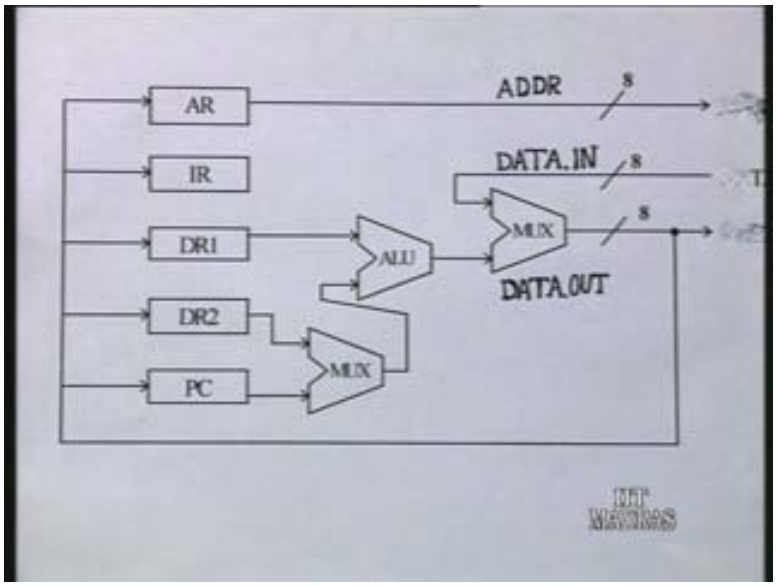
So what I am doing is I am assuming data X is available in register and data Y is available from memory; with this let us proceed. Now what is the size of the instruction? I will just say it is a 2-byte instruction; we have to assume a few things. Only then, we can proceed further. So it is a 2-byte instruction, which means, we assume that all the instructions are in the memory. In the memory, I will assume it as a byte organized memory, which means at any time we can fetch 1 byte from the memory, now two locations are needed for this particular add instruction. That is, add X Y is available in 2 bytes and I will also assume that this particular instruction is 1000 memory location; and memory address 1000 and 1001.

(Refer Slide Time 00:27:44)



Now how will this instruction be fetched? Something in the CPU must indicate that this instruction is in location 1000 and 1001. Let us take a look at the architecture, and that will give us some idea.

(Refer Slide Time 00:28:10)



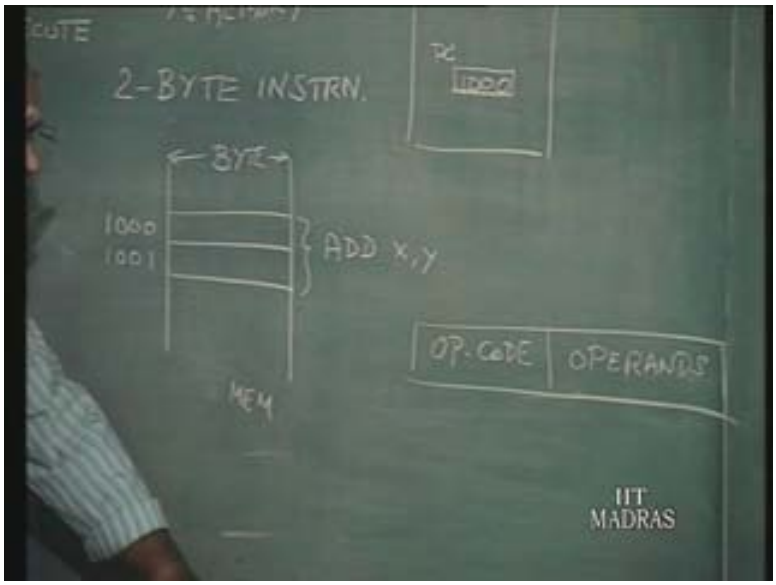
So here we have the program counter, two data registers DR1 and DR2, and one instruction register and one address register. From the address register we find the outgoing 8-bit address lines and we have the data bus part of it, the data input part of it, and the data output part of it, each of 8-bit width. Since we have assumed byte size for the data bus, obviously byte by byte, that is, 8 bits at a time, the instructions will be fetched. Since we assumed 2-byte instruction, 1 byte will come, it will be routed to the instruction register, and then in the second step, another byte will come; that also will be routed to the instruction register. Since we assumed a 2-byte instruction, obviously this instruction register will be of width 2 bytes.

There are a few assumptions that we are making; it may not really be so with reference to any specific processor; and I would like to bring your attention to another thing. I assumed only two registers in the architecture just to make life simple; there can be many registers there. If you have many registers, there must be a mechanism by which we can indicate which of the registers is used; we are avoiding those things. Since we have taken an instruction which assumes one data is already available in the register, and another one is in the memory, one of the data will have to be either in DR1 or DR2. Then take a look. This PC is a program counter; now what is the function of this? Essentially, the PC holds the address of the instruction always. Now in our case, since we have assumed that the instruction is in memory location 1000 and 1001, it is implied that PC, to start with, must hold the address 1000. That is the first point. That is, in the CPU, there are many things; that is, the entire data path is there and PC, to start with, must hold the address 1000. Why 1000? It is because that is where the specific instruction starts.

Now what is the first step? You may not work out all the steps for a few things at least. Now let us see the chart again. So PC holds 1000; now this 1000 must go to address register and then, on the address bus, the number 1000 must be placed, just the data path of it; then let us recall how this instruction is fetched. The control signal must be generated here; we are fetching the instruction, which means read control signal must be generated next and the processor will be reading it here. Then let us see how this sequence proceeds. PC content that is 1000; it must go to the address register. Now that is the first thing to do. The CPU places the address, indicates whether it is going to read or write. Now fetching means it is going to read in. Since we have assumed a 2-byte instruction, it is going to read in 1 byte and 1 byte comes over the data bus. Then it cannot immediately go to decode, decoding of instruction, because the entire instruction is not available. So what it will do is to place the next address again and bring the next. Now how will the processor know that it is a 2-byte instruction or a 3-byte instruction or a 1-byte instruction?

Obviously, whatever may be the size of the instruction, the very first byte of the instruction must indicate to the processor that another byte is there for that instruction or 2 more bytes are there in the instruction – this is important. Actually you remember I was talking about the instruction format and two fields, opcode and operand.

(Refer Slide Time 00:33:57)



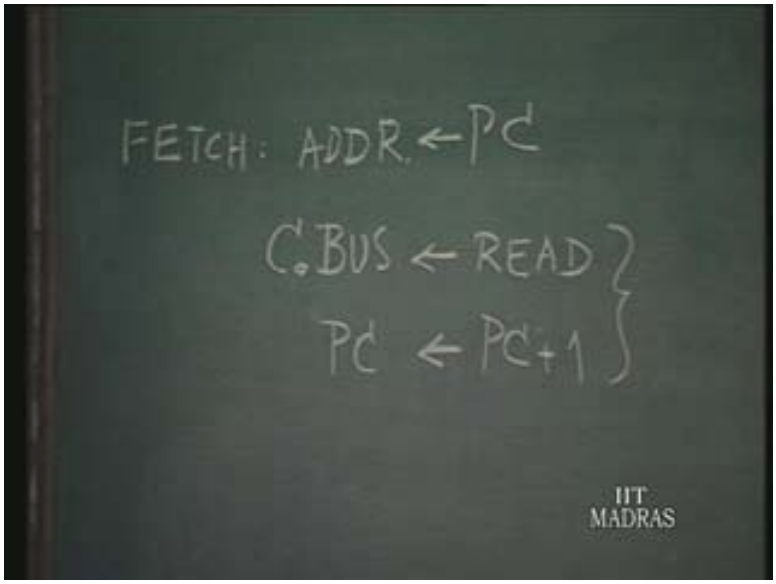
We did not really discuss the size of the opcode and also the other fields in this operand; we have not discussed that yet. So the very first byte of the instruction must include the opcode, and that will indicate the size of the instruction also. Otherwise, the processor will not know what to do. So the very first part of the instruction, that is, the very first byte of the instruction which comes in, will indicate the size of the instruction.

So the processor knows that it cannot immediately proceed to decode, that is, proceed with what exactly is the operation to be executed and so on. It will continue and fetch the second byte of the instruction also. I hope we have made certain assumptions without loss of generality. First the CPU places the address; then it indicates that it wants to read; and then the memory responds and CPU reads. Remember the four states we were talking about: T1, T2, T3 and T4?

Fetch, as we know, must consist of two things – what is that? We can refer to the chart again, the PC content, which is 1000, must be placed on the address bus. This is the first thing; PC contents must go to the address bus. In fact, we had worked out the details of this. Let us see the chart; the PC contents go to the address bus. The control signals must be such that whatever is placed on this leg of the multiplexer will be passed on to the output, and the ALU must be set up. The control for ALU must be set up in such a way that this input must be passed on as the output of ALU, and a multiplexer must be enabled such that whatever is passed on this leg goes here and then this address register will be enabled, so that it will accept what comes from here, and again, address register output must be enabled so that whatever comes must be placed on the address bus – we have worked out the details earlier.

So when you say PC goes to address bus, what it means is the data path of this content must be like this: through MUX, ALU, another MUX, and so on, and then to the AR and then to the output, on to the address bus. We worked out and so all these things are there. Now the entire thing can take place in one state so we have this in general. At the register transfer level, we are writing this as PC contents go to the address bus. Once it is on the address bus, the memory will have to get the appropriate control signal, so that is the next one. The next micro-instruction will be the one which we have not indicated in the data path and because we have not worked out the details of the control, we just write that the read signal must be placed on the control part of the bus. So I will just put it as control part of the bus or C bus. Now when the read control the controller generates the read signal and places it on the control part of the bus, simultaneously we can see that the program counter contents can always be incremented because there is no harm in incrementing, because this will be done in parallel.

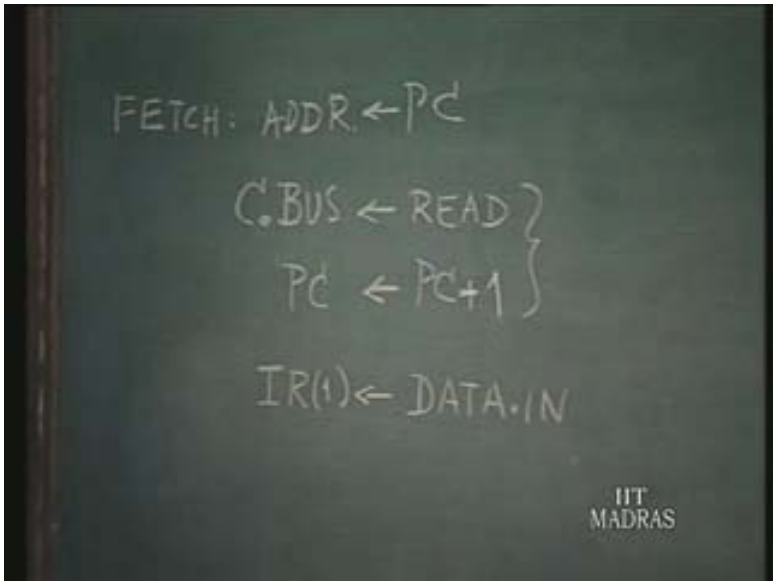
(Refer Slide Time 00:37:58)



So in another state, both these signals can be generated, so that with the same control signal we will call this as T2. The read control signal can be placed on the bus and the same control signal can also be used to increment the contents of the program counter. Now mainly I think we discussed this earlier; we normally increment the program counter. In fact that is why it is called a counter, not just a register. It is called a counter because normally the subsequent instruction or subsequent bytes of the instructions will be in successive locations. So you use a counter and keep telling the CPU to fetch from the next location and so on, and then at some point there may be a change. If there is a change, the a new address can be loaded onto the PC, or certainly you must have heard about branch in jump instructions. In those instructions, for instance, when the instructions come, some new address will be overwritten into PC; PC content normally will be incremented.

So in one state, the PC content goes to the address bus; in the next state read control is generated and simultaneously PC contents are incremented and then we are just allowing one state for memory; that is from the memory point of view. Let us work out only from CPU point of view, assuming that the memory is responding because memory wants two things. Memory wants to know from which address it has to generate data and what it should do with the data – whether you should put it out on to the bus or whether you should accept the data from the bus. Now, in this case, if it just accepts read, it will be right; we will come to that later. May be it will be part of this; even this instruction itself, may be the later half. The next is the CPU: at this point it is assumed. Assuming that the memory is ready with the data, then whatever is available on the data bus, which we have in the chart, you can see that over data dot in the data, the memory that has been placed will be coming in. Because this is an instruction, it must be routed to the instruction register.

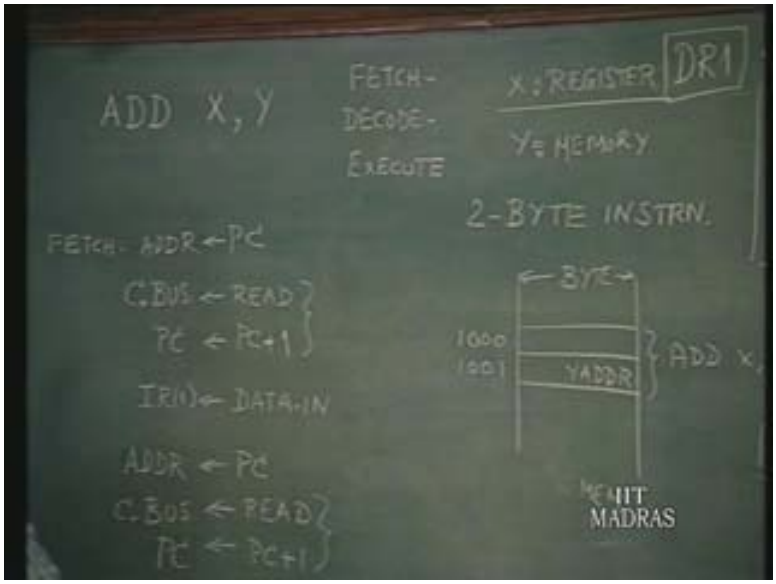
Now what did we assume? We assumed a 2-byte instruction, so obviously there must be two parts of the instruction. I will just put it as IR (1); there must be two parts of the instruction. So it will go to one part.
(Refer Slide Time 00:41:13)



Now at the end of these three steps, the CPU has got one part of the instruction; it is just the instruction. Actually some decoding of this particular instruction byte itself, that is, the first byte of the instruction itself will be done, which will indicate to the CPU that one more part must come in. So really the fetch phase is not yet over; it continues, and again the PC contents will be placed. Why are PC contents placed on the address bus or address part of the bus? That is because already PC is pointing to the next one and we had already incremented it.

After fetching 1 byte, PC content had been incremented, and so now PC contents, when placed on the bus – actually it means 1001 is placed on the address bus – it is the same thing as before. So I will just repeat read control will be generated; it will be put on the control part of the bus and, in parallel, PC contents will be incremented. There is nothing new about it and now, whatever comes over the data bus, we are assuming that the memory is ready. If memory is not ready, something more must be done here. So whatever is coming over the data part of the bus will now be routed to the second part of the instruction register.

(Refer Slide Time 00:43:16)



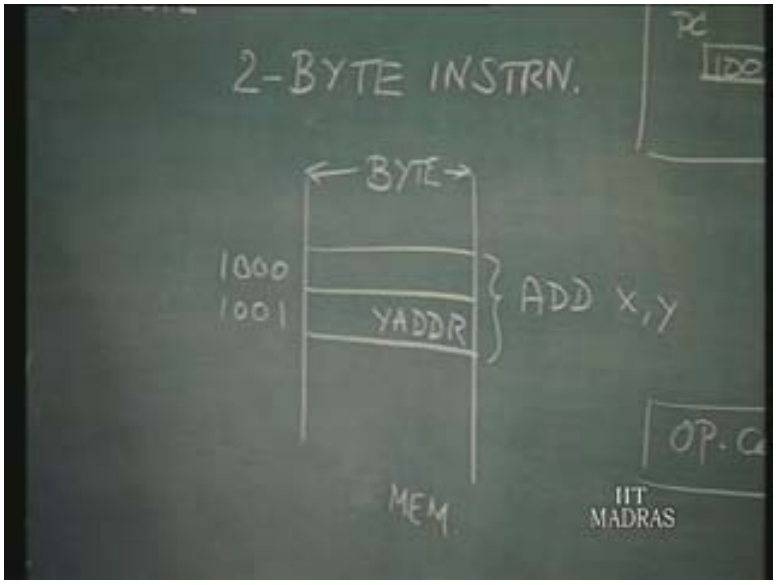
As I mentioned earlier, let us see the chart. What we are assuming is that the instruction register is of 2-byte width. So the 2 bytes of the instruction have come into the processor; now what is the next? The entire instruction is now available with the processor. So the instruction fetch part of it is over. Now full decoding of the instruction is possible, from which it will know whatever we assumed earlier. That is, we assumed that one data is available in the register, CPU itself, and another data is available in the memory, which means some part of this instruction will have to carry the memory address, that is, the address which stores Y. Some part of this instruction will also indicate that the second data, which is X, is already available in the register. Now see the chart – we have assumed only two registers; so what shall we assume?

Let us assume that this particular register, that is, X, is available in register DR1. That means, now this CPU has the whole instruction, and it also has the data X, because DR1 is there as part of the CPU; so the data is already available. From the memory address, which is available here, I will just put as Y address; some part of the instruction will be referring to the address of the second data Y.

(Refer Slide Time 00:45:03)

So that will be there – what is the next one? Now the processor goes through again, another fetch, but now this time it is data fetch or operand fetch; that is the phase.

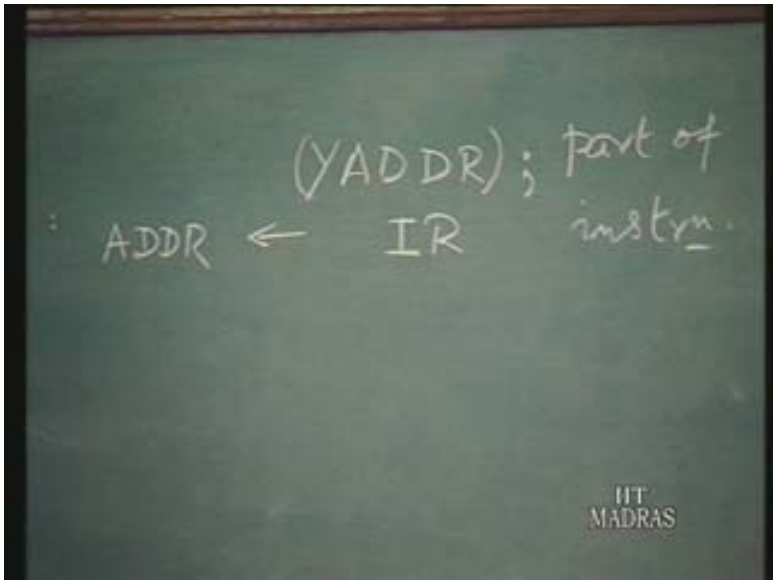
(Refer Slide Time 00:45:54)



The data fetch is similar to instruction fetch and the address will be placed on the address bus; instead, earlier, PC was pointing to the instruction. So PC confirms that place on the address bus; now the instruction points to Y address and so now, similar to instruction fetch, what it does is Y address is available as part of instruction. Let us note it.

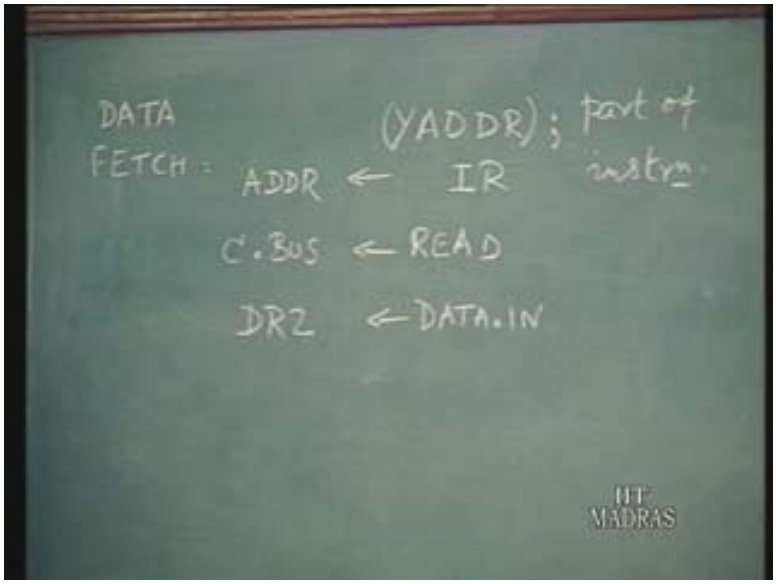
This is available as part of instruction and since we have assumed that instruction register holds the entire instruction, some part of IR, where the Y address is available, will be routed to the address bus. So let us see the chart – it is not shown here. IR will be holding that 2-byte instruction.

(Refer Slide Time 00:47:27)



From here the path is not shown, actually because we have not completed that this particular thing will have to get routed to AR and it will be placed on the address bus. Similar to the instruction fetch, it is again a question of reading. So we just have to repeat that: read goes to control part of the bus and in this case, there is no incrementing of PC because we are not dealing with the PC. Then assuming that the data is available, we will be having the actual contents of this Y address location, which is nothing but Y. That is, data Y is the data which we assumed is available in the memory. So memory will place that Y on data in, that is, data bus and that will have to be routed to DR2, because we have assumed DR1 holds the other data. So this will have to be routed to DR2.

(Refer Slide Time 00:48:56)



Let us look at the chart again; so the Y data, the second data Y is coming over data in, and through this multiplexer, it will get routed to DR2. So now, at the end of this, DR2 holds Y and DR1 holds X. That is what we have assumed earlier. Now the CPU has the full instruction; the CPU has both the data X and Y. Next is the execute phase. In the execute phase essentially the DR1 contents must be added with DR2 contents and the result, we will assume, goes to DR1; that is what must take place.

This is the actual operation – this operation has been indicated as part of the instruction; this instruction is available with CPU. So during decode or interpretation, the CPU knows that addition must be performed. That is what it is; let us go to the chart and see again. DR1holds X and DR2 holds Y and ALU must get an appropriate signal so that the contents of DR1 and DR2, which is reaching to the multiplexer, must be added and the output which is the sum through the multiplexer, has passed on to DR1. So that is the data path for that. So it will be done just in one go.

At this point, the instruction has been completely executed. Then normally there is another phase which we have not discussed, known as service phase; that is, you have an instruction fetch phase. In general, in fetch phase one part will be instruction fetch; another is data fetch; and then there is execute; and then service, because we have not yet discussed this in detail. After this, something further must be done during this phase; it will again go out and fetch the next instruction and go through the same thing again.

(Refer Slide Time 00:51:49)

So a simple add instruction, as we can see, consists of ten steps. So, ten states are needed for executing this instruction. That means, in the case of a micro-programmed implementation, ten micro-steps or ten micro-words will be there. Of course, hardwired logic is a different story.