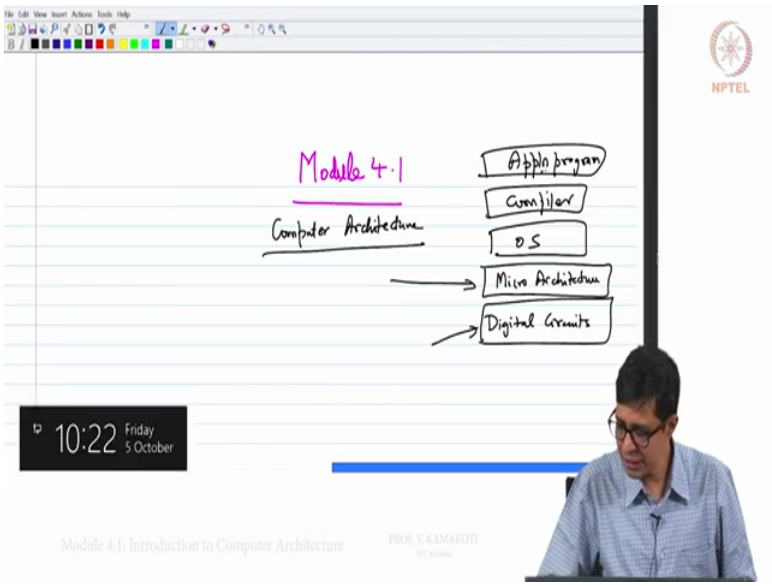
Foundations to Computer Systems Design.
Professor V. Kamakoti.
Department of Computer Science and Engineering.
Indian Institute of Technology, Madras.
Module 4.1.
Introduction to Computer Architecture.

(Refer Slide Time: 0:22)



So, welcome to module 4.1. What we have done so far, let us very quickly have a look at it. So, we started off with digital circuits, we made, we started with NAND gates, we made AND Gates, NOT Gates, muxes, demuxes, then putting all of them together we made an ALU and then after making that ALU, another are different components that we have done. We also made sequential circuits, so all these things we wanted to put together now, right.

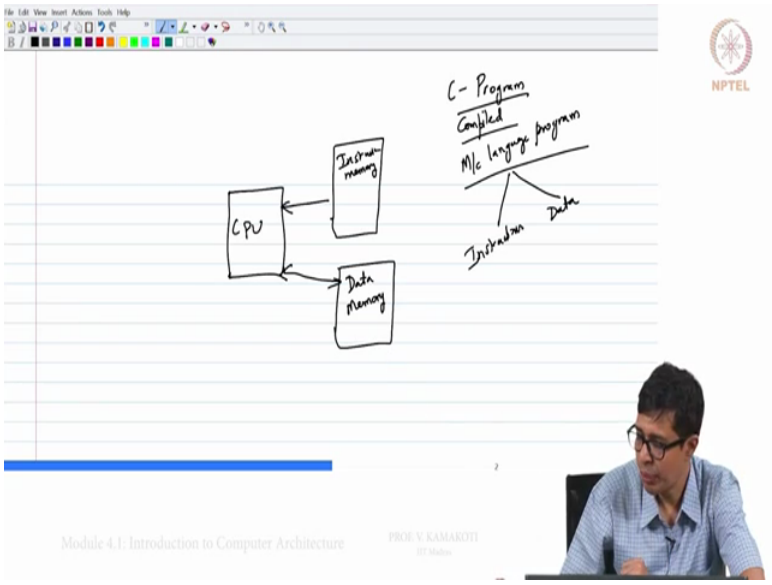
So the small small hardware components, the memories and all those things that we have made, we would like to assemble them together to make what we call as the microarchitecture. Before going to what this microarchitecture will look like, we wanted to understand what, what would be the expectation out of this microarchitecture. That was one of the important things that we did as a part of the module, lateral part of the module 3.

So we basically introduced reassembling which, the machine language, which had this A and C instruction, then we now go ahead and tell that this microarchitecture is expected to understand this A and C instructions that is given in binary to the microarchitecture and execute them on the digital circuits. So, we need to take all these small building blocks of this, the digital circuits that we have seen, put them together in such a way that that entire

piece that we have connected, the architecture that we have connected will now understand the C and A instructions that we have discussed, execute them and give back the results, right.

Now, the next step now for us in this big stack of OS, compiler and application programming, application programs, in this entire stack, in this entire stack of 5 layers, we will now go to the 2nd layer which we call it as computer architecture. So, in this module 4.1 to 4.6, we will be dealing more with computer architecture, essentially we will be telling as I just explained how do we use these small data circuits that we have assembled so far, both the combinational and sequential circuits, how are we going to connect them together so that they in turn connect that piece which we call as architecture or the microarchitecture will execute the C and A instructions that we have described earlier as a part of module 3.

(Refer Slide Time: 3:54)



And that is what we are going to do. So, how do we 1st understand computer architecture? Now what is the computer, computational model? Now, I have a program, a C program, this is compiled, finally you got the machine language program. We have seen some examples of this in the later part of module 3, wherein we took a C program and actually manually translated into the machine language program. This entire, this translation will be automated as we go in this course, we will see how it is getting automated.

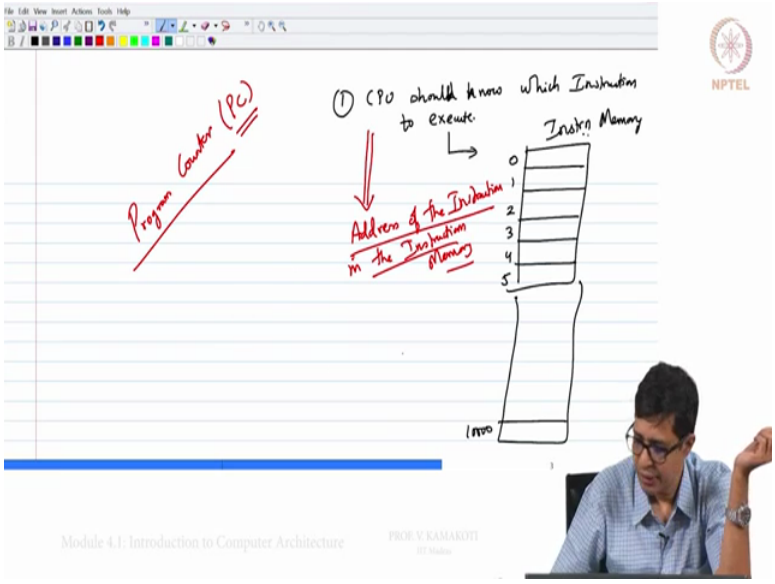
So there will be compiler which will take that C program and convert it for you as a machine language program. Now, this machine language program had 2 components, one was the success, another was that it is that this instruction will act upon always take and process. So, essentially there are 2 types of memory that we can see, one is called the instruction memory

and we have another version called the data memory. So, there is an instruction memory at the data memory.

In the instructions memory, the instructions will be loaded, in the data memory, the data which instructions will act upon will be loaded. Now, there will be a central processing unit that will be taking instructions from this instruction memory, according them one by one and as a part of this execution will be modified the data, right. So the CPU needs to take instructions one by one, execute them and for executing that instruction it will need data, that data it will be accessing from memory, right.

So, this is the computational model. Your hardware is expected to make, connect these components together in such a way that the CPU does this particular act as described now. So, drilling down a little further now, the CPU is expected to execute the instructions one by one. So, that means we should now understand what it means to execute one instruction by the CPU, right. So let us see that, what it means to execute one instruction by the CPU.

(Refer Slide Time: 6:33)



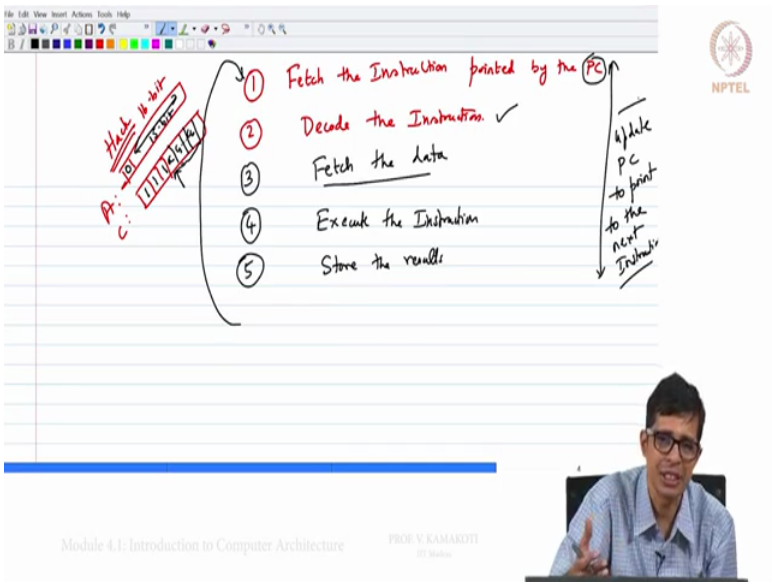
1st thing is the CPU should know which instruction to execute. The CPU should know which instruction to execute. Who will tell this? Who will tell a CPU this is the instruction that needs to execute? Now, one of the thing is we have an instruction memory, there is instruction memory, in this instruction memory they instructions are stored one by one and there is an address for each location 0, 1, 2, 3, 4, 5 say till some 10,000. Now, to tell the CPU which instruction it has to execute next, I can tell it by saying execute the instruction in address 5, execute the instruction in address 6.

So I can give the address of the instruction in the instruction memory, I can give the address in the section memory, the CPU can go and read from that address the instruction and start executing it. So to basically tell the CPU which instruction it has to execute, basically we need to tell the address of the instruction in the instruction memory, right. So which piece of architecture or which component in that architecture will tell the CPU the address of the instruction that it needs to execute.

That piece of hardware is what we call as a program counter. As a part of your project 3, you would have done a program counter, project 3F if I am correct. That program counter will be used to tell the CPU, that program counter itself is a 16-bit program counter and it will store the address of the instruction, address of the instruction in the instruction memory that the CPU has to execute. So, if the CPU has to execute next this instructions stored at address say 105, 105 will be stored in PC. So this program counter we will call it as PC, right.

So the 1st step is the CPU should know which instruction to execute and for that it will look into the program counter, the program counter will be having an address, that address it will take, it will go through to instruction memory and fetch the instruction for executing.

(Refer Slide Time: 9:55)



Right, so let us go and, so the 1st step is fetch the instruction pointed by the PC, pointed by the PC. Now what we do, after fetching the instruction, we want to decode the instruction. At this stage I will tell you something about decoding, but we will go much deeper into decoding as we proceed through this model, the lateral part of this module. Now what is decoding the instruction in our context, essentially the context of hack, the machine that we are designing,

there are 2 types of instructions, namely A and C. Now the decode unit has to quickly distinguish between whether it is an A instruction or C instruction.

And how do we do this? As we have mentioned in the previous module, the A instruction, all the instructions are 16-bit in length, the A instruction starts with the 0 while the C instruction starts with 111, right. So decoding the instruction at the 1st stage, we have to distinguish whether it is an A class instruction or C class instruction and that is got by inspecting the 1st one or 3 bits. So the CPU has to decode the instruction. After it decodes the instruction, then it has to find out what are the data necessary for this instruction to complete.

If it is C instruction, if it is A instruction, then it immediately has the data that is 15 bit following it. But if it is a C instruction, then, if it is a C instruction, then it notes what are the data that is needed, that is already, that is, that is there in the C structure. So there is computation involved, the reason a bit, small a bit here and then there is C1 to C6, the control bits. This A bit will tell us A and the combination of A and C1 to C6 will tell us where the source operands are there.

Foreign operation, suppose if I am adding I need to know which 2 numbers I need to add, that is called the source operands and we have already discussed source operands in the previous module. Now this 6, 6 to 7 bits in C will tell you what are the editor that is needed for the instruction to execute. So I will be fetching the data. And where will this data reside? This data will reside in the data memory. The instructions reside in the instruction memory, the data will reside in the data memory.

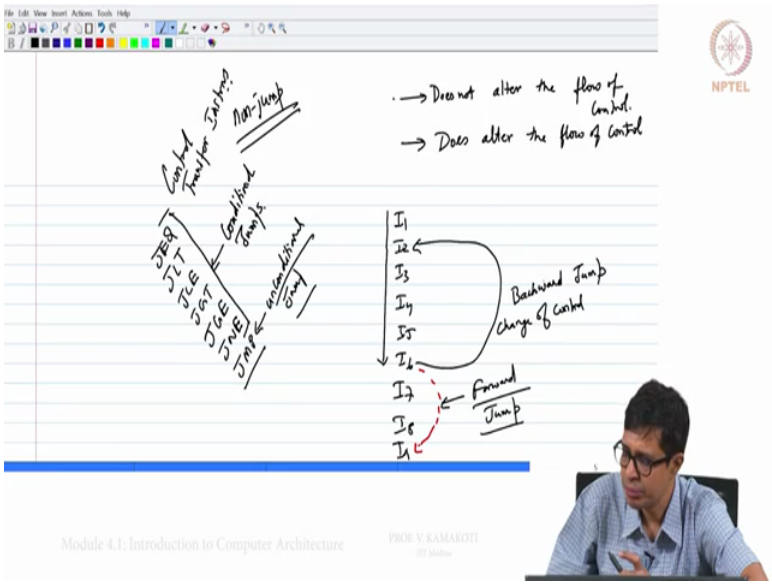
So I fetch the data from the data memory, then I basically execute the instruction. Then finally we store the results. In the case of, where are the results store, in the case of the C instruction, there are some destination bits and the destination bits will tell you where should I store the results. Right, I can store it at multiple places, I can store it in the D register, I can store it at the A register and so on and so forth. Other are many things that we need to see, so I have to store the results of this operation.

And after this, that is one thing, and I go and repeat this again and again and again. One thing that I need to know is that at this point the PC will always be pointed to the next instruction to be executed. So before I go back, I should also update PC somewhere. Somewhere in this line I have to update the program counter to point to the next instruction. So, these are the 5 stages, the instruction from the instruction memory, decoding the instruction, in our case

whether it is an A instruction or C instruction, then if it is C, little bit more depth of what is the destination operand, is it a jump instruction and what other source operands, depending on the A bits and the C bits that we have seen, right.

Then I need to fetch the data if necessary and then execute the instruction and store the results back. And then I have to go and do the next instruction. Again, so for that I keep updating the, I have to update the PC, okay. These are the 5 stages of execution, executing an instruction. Now, they will go and explain these 5 stages with respect to B with respect to the hack as we proceed in this module. But one of the things that we want to tell before we move forward is what is the challenge in updating the program counter.

(Refer Slide Time: 16:11)



What does the program counter do, the program counter points to the next instruction to be executed in the instruction memory. Now, we need to now tell you how, what are the challenges involved in updating the program counter. Now, there are 2 types of instructions that we see, one is an instruction that is just, there is one classification of instructions. One type of instruction does not alter the flow of control, another one does alter the flow of control.

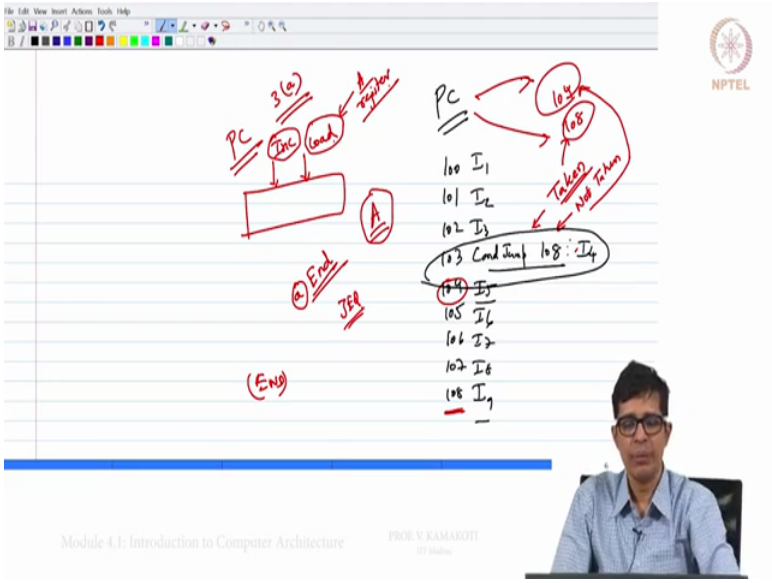
Thus, the instructions that does not alter the flow of control are the non-jump instructions, that means the flow will go, if these are the instructions I1, from I1 it will go to I2, then to I3, then to I4, then to I5, then to I6. But if I have a jump from I6 I can go back to I2, so this becomes a change of control. Sometimes it is called a backward jump, I go back, or I can go I7, I8, and from I6 I can even jump to say some I9. So this is called a forward jump.

So the moment I have jumped, the jobs that we saw in the previous module I could have J less than, J less than or equal to, J greater than, J greater than equal to, J not equal to, the normal jump JMP, right, okay, J equal to, so all these types of things, jumps are there and we can basically see. So these jumps, these 6 that we see here are basically conditional jumps. These jumps will be taken only when there is some conditions met. If the condition is not met, then these stuffs will not be taken.

But the JMP is an unconditional jump, it will be taken, irrespective of whether there is a condition, irrespective of any conditions. It will be surely taken, so this is called an unconditional jump. Now what is, what do these jumps do? It basically alters your flow of control. So when the flow of control is happening one after another, the instructions are basically doing, basically executing one after another, the presence of a jump will alter the control to some other instruction, rather than the succeeding instruction.

For example after I6, I7 should have been executed, but if I add a backward jump, then I will go back to I2, image to I9, if it is a case of forward jump. So these jumps are basically called control transfer instructions. They alter the control, they also transfer the control from one instruction to some other instruction. So, this makes, so when I am looking at the PC, so the, so the point that we are now discussing is how are we going to update the program counter, PC. Right.

(Refer Slide Time: 20:00)



So let me say, that there are one, there is say some 7 instructions, so I will write it as instruction 1, instruction 2, instruction 3. There is a conditional jump here to 108, so, this is

I4, which is a conditional jump to 108, so I have I5, I6, I7, I8, I9. Now the question will be when I am executing 103, after execution of I4, should I execute I9 or should I execute I5? Should I execute the instruction at 104 or should I execute the instruction at 108. This is a very very important question. And that depends upon the conditional jump.

If the conditional jump is, if the condition is true, that means the conditional jump is taken, then I have to go and execute 108. If the condition is false, that means the conditional jump is not taken, then I have to go and execute 104, that is the instruction that follows. So, then, when we talk of updating the PC here, right, when we are talking of updating the PC, should I update the PC with 104 or should I update it with 108, this is the question, right.

Should I update it with 103, after executing 103, should I update the PC with 104 or should I update it with 108, that depends upon this conditional jump. If the conditional jump is taken, then I have to put 10 it, if the conditions Is not taken, then I have to go back to 104. That is why if you carefully look at PC, that we did it as Project 3A, if you remember long back. It if you basically look at the program counter, in the program counter we also had a load input.

It is not just an increment, we also had an increment, we had load also. So if the, if it is not a jump, then PC will be just incremented to point to the next instruction. If it is a jump, if the current instruction is a jump, then if it is a unconditional jump, that the PC will basically go to that new address. So if it is, if it is an unconditional jump, let say 103 itself is an unconditional jump to 108. Then what will happen, after 103 the PC will be loaded with 108.

If it is a conditional jump, then if the condition is satisfied, then PC will be loaded with, say, this case was 08, if the condition is not satisfied, that the PC will be incremented. Right. So updating PC can be of 2 types, either increment PC, which normally happens, which, which essentially happens if the instruction is an unconditional, it is a non-jump are not controlled transfer instruction.

If it is a control transfer instruction and then the new address, if the control need to be transferred in the case of unconditional jump, it has to be transferred for sure, in the case of conditional jump, it needs to be transferred if the condition is satisfied, then you basically go to the new address. So, there you actually use this load to load not just incremented this address but to loaded with a new address. That is why when we designed this program counter, we had a facility to increment it and also to load it with a new value, right.

So what do you load, if I want to load a new address, from where do you load? Actually when we look at our assembly language, suppose I want to jump to some end, which is say Label end. So I will 1st put at end, that means this address of end is loaded into A registers and I say jump on some equal to or whatever. And if the jump has to be taken, where is the address, it will be the A register. So when I want to load something into the PC, because of control transfer instruction, that address will always come from the A register. Right.

So this is how we can basically update the PC. So to sum up what we have done in this module. We have seen the 5 stages of execution the instruction, fetching the instruction pointer by the PC, decode the data, fetch the data, execute the instruction and store the results. In addition to this we have also seen the PC needs to be updated and the PC updation in the case of non-control transfer instruction is incrementing.

And in the case of control transfer instruction, if there is a need to do a control transfer rather than just incrementing, then the new address should be loaded from the A register, then this is specific to our hack. But this is how any, any other system also works in practice, right. So what we need to build, going ahead what we need to build is a CPU which basically does the 5 functionalities. For every instruction that it is fetching, it has to do these 5 functionalities, store it and then go to the next instruction and so on and so forth.

So this is what we basically want to cover. So the remaining part of this module will basically deal with how a CPU is built, such that it execute these 5 operations one by one and how it basically updates the PC in terms of control transfer instructions also. So how do we build a hardware using the combinational and sequential building blocks that we have already made, how do we arrive at an architecture which will do these 5 operations for the A and C types of instructions that are going to be provided to it. And that is what we will see in this module as we proceed. Thank you.