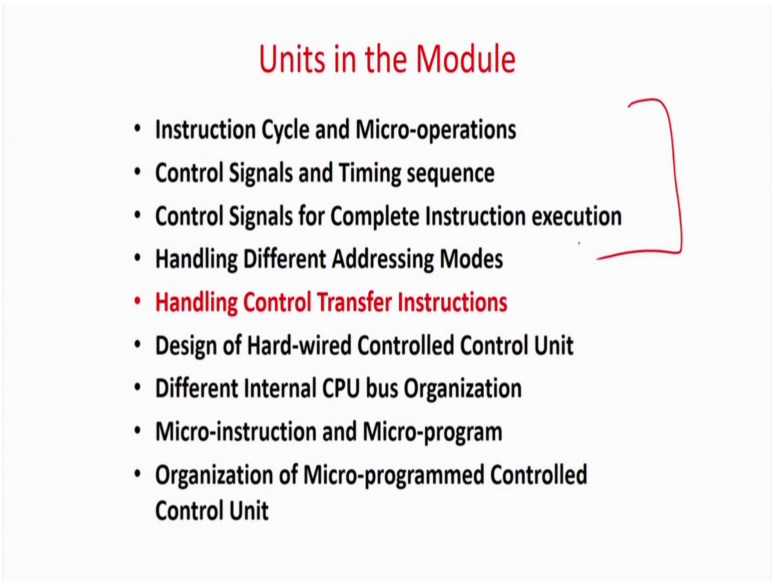
Computer Organization and Architecture: A Pedagogical Aspect

Prof. Jatindra Kr. Deka
Dr. Santosh Biswas
Dr. Arnab Sarkar
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 19
Handling Control Transfer Instructions

Welcome to the five unit of the module on control design. So, as discussed in the last unit that from today we will be discussing on the special type of instructions, what are the control signal involved in, and we will be mainly talking about control instructions which are of jump, handling of function calls etcetera which we call as transfers instruction. So, this unit basically we will be covering on control signals, microinstructions involving transfer instructions like jump, call etcetera.

(Refer Slide Time: 01:00)

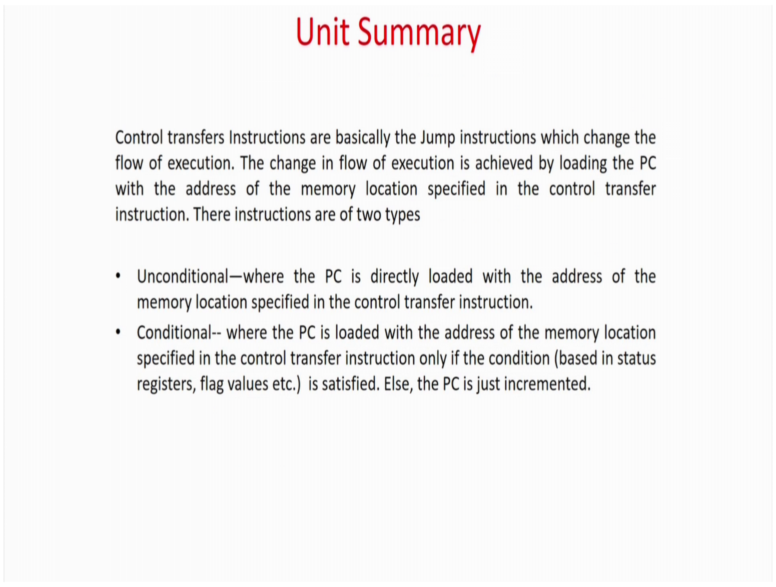


Units in the Module

- Instruction Cycle and Micro-operations
- Control Signals and Timing sequence
- Control Signals for Complete Instruction execution
- Handling Different Addressing Modes
- Handling Control Transfer Instructions
- Design of Hard-wired Controlled Control Unit
- Different Internal CPU bus Organization
- Micro-instruction and Micro-program
- Organization of Micro-programmed Controlled Control Unit

So, we will be talking after looking at the different type of control signals for general type of instructions. In today's unit, we are going to handle jump and jump call etcetera which come under the category of control transfer instructions.

(Refer Slide Time: 01:10)



Unit Summary

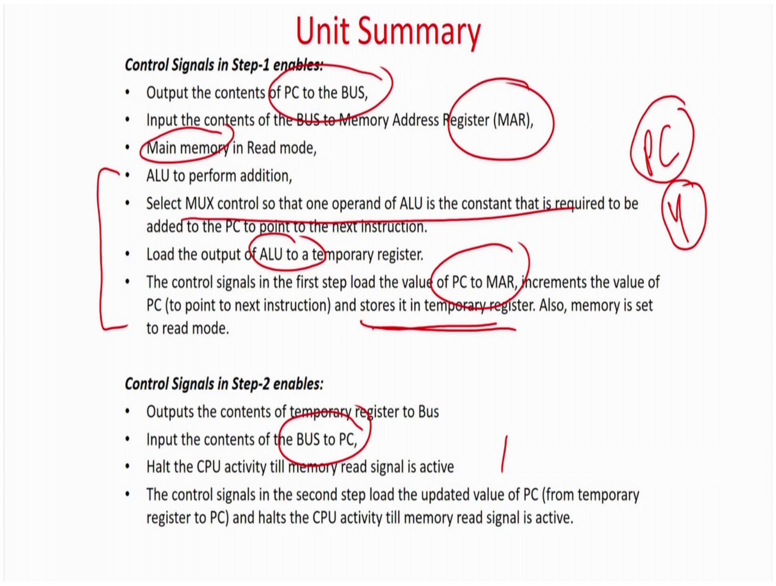
Control transfers Instructions are basically the Jump instructions which change the flow of execution. The change in flow of execution is achieved by loading the PC with the address of the memory location specified in the control transfer instruction. There instructions are of two types

- Unconditional—where the PC is directly loaded with the address of the memory location specified in the control transfer instruction.
- Conditional-- where the PC is loaded with the address of the memory location specified in the control transfer instruction only if the condition (based in status registers, flag values etc.) is satisfied. Else, the PC is just incremented.

So, basically in the units what is in the unit summary, we will first discuss about the unit summary before going into the depth of the unit. Basically as you already know the control transfer instructions are of two type conditional and unconditional sorry unconditional and conditional as the flow are in the thesis in the (Refer Time: 01:29). So, what is the unconditional jump instruction, an unconditional jump instruction is very simple like jump go to that memory location where the corresponding instruction is there maybe you call a function, so that is an unconditional jump.

And what is the conditional jump, when you look at these flags and decide what to do like jump on zero to some memory location 3030, where the next instruction is there. It will jump only if the zero flag is it, so that way we can differentiate I means control transfer instruction as conditional and unconditional. Basically now what are the basic steps involved or the basic control instructions or microinstructions involved or the control signals involved in each of these steps.

(Refer Slide Time: 02:06)



Unit Summary

Control Signals in Step-1 enables:

- Output the contents of PC to the BUS,
- Input the contents of the BUS to Memory Address Register (MAR),
- Main memory in Read mode,
- ALU to perform addition,
- Select MUX control so that one operand of ALU is the constant that is required to be added to the PC to point to the next instruction.
- Load the output of ALU to a temporary register.
- The control signals in the first step load the value of PC to MAR, increments the value of PC (to point to next instruction) and stores it in temporary register. Also, memory is set to read mode.

PC

Y

Control Signals in Step-2 enables:

- Outputs the contents of temporary register to Bus
- Input the contents of the BUS to PC,
- Halt the CPU activity till memory read signal is active
- The control signals in the second step load the updated value of PC (from temporary register to PC) and halts the CPU activity till memory read signal is active.

As we have already discussed in the last three units that basically the three steps that is the first three steps basically involve fetching of the instruction. So, in this case what happens in the step one basically we load the program into the bus, PC into the bus and we give it to the memory address register, so that the corresponding instruction can be fetched. We make the main memory in read mode and we get the value of the program or the instruction in the memory data register in this third stage basically.

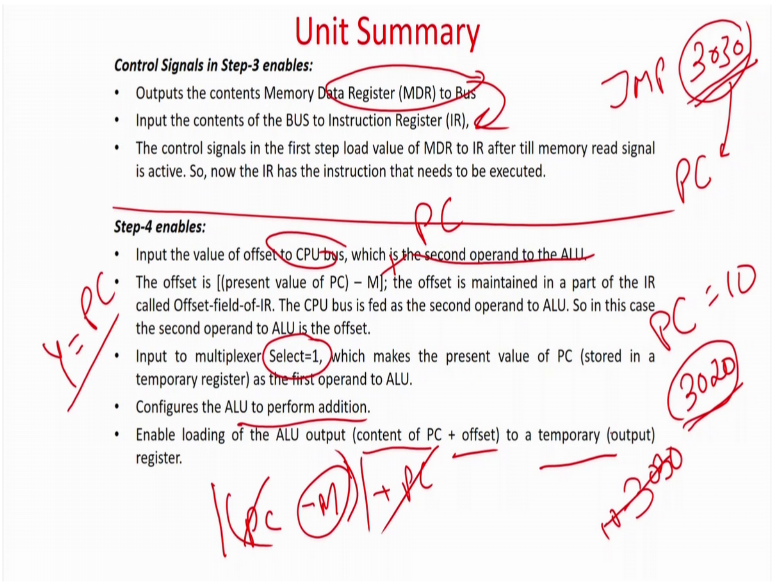
So, basically that is what is the thing and then we say that but here there are slightly difference in case of a jump instruction or can transfer instructions. In general type of instructions, we take the value of the program counter to the bus, put it in the address register, and then wait for the corresponding instruction to be obtained in the memory data register. And also what we do, we also try to increment the value of PC to the next value to point to the next instruction.

But in this case we do a slightly separate stop over here that what we do we actually also stop the incrementing of the PC here, but also we store as we will see the requirement what is the necessity. We also put the temporary value of PC into a temporary register Y. We will see that why it is required later because in the time at the same stage when you are trying to increment the PC as well as we will have to store the value of the PC to a temporal register. We will see that storage of the PC in a temporal register is something different which you are doing in transfer instruction. And we will see later see what is the importance of that.

Other things like select the MUX, so that the constant is added that is actually finding the next instruction that remains the same. Then basically what else load the output of the ALU to our temporal register that is PC equal to PC plus 1, all these things will be similar. But only one extra thing is basically the control signals in the first step load the value of the PC into the memory address register increments the value of PC that remains the same. Store it in a temporary register Y is a special extra thing, which we do for the control instructions all other part remains same. So, I am not elaborating.

The second stage is very similar basically bus value will go to PC that is PC equal to PC plus constant that is the next transfer, output the contents of the temperature to bus with a very seemingly simple. Halt the value of the PC, halt the value (Refer Time: 04:27) the memory says it is ok. Once it is done, you can go ahead, this is very similar to the second stage.

(Refer Slide Time: 04:32)



Unit Summary

Control Signals in Step-3 enables:

- Outputs the contents Memory Data Register (MDR) to Bus
- Input the contents of the BUS to Instruction Register (IR),
- The control signals in the first step load value of MDR to IR after till memory read signal is active. So, now the IR has the instruction that needs to be executed.

Step-4 enables:

- Input the value of offset to CPU bus, which is the second operand to the ALU.
- The offset is [(present value of PC) – M]; the offset is maintained in a part of the IR called Offset-field-of-IR. The CPU bus is fed as the second operand to ALU. So in this case the second operand to ALU is the offset.
- Input to multiplexer Select=1, which makes the present value of PC (stored in a temporary register) as the first operand to ALU.
- Configures the ALU to perform addition.
- Enable loading of the ALU output (content of PC + offset) to a temporary (output) register.

The third stage is also very similar. This third stage basically says that the memory read is ok. So, basically the output of the memory data register is going to the instruction register as simple as the fetch. So, up to this is more or less similar that is you fetch the instruction and in the instruction register; only one extra thing we do here you just keep it in mind that will store the value of program counter in a temporal register Y. Now, from fourth step on what is things to start changing basically.

Now, in this case, see previous case we have seen how to add, how to store from memory to the registers, but here actually it is something going to a jumps space. For example, let us take the instruction called jump unconditional to say 30 30, then we want to jump to that memory location that is what is the execution part here. So, in this case, what you have to do, you have to somehow load the value of program counter in the next stage to 30 30 that is what is the execution to be done. Because when you want to execute some instruction which is the memory location 30 30, what we have to do is to be simply load the value of PC to 30 30 and your job will be done. So, the execution part will take care of that.

So, what we do input the value of offset to CPU bus which is the second operand of the ALU now we will see what do you mean by an offset. The offset basically is a part of the instruction register, which actually goes to the CPU, CPU bus basically. So, what is an offset, offset is present value of PC minus M and you take the general is a positive value out of it. So, if you assume that the present value of the PC is equal to 10 for the time being let us assume that is equal to 10, so your offset will be 30 30 this 10 minus 30 30 the positive you are going to take it. So, it will be 30 20. So, this is what is actually your value of the offset 20, so that is what will be given by the instruction register. So, instruction register will generate the value of PC.

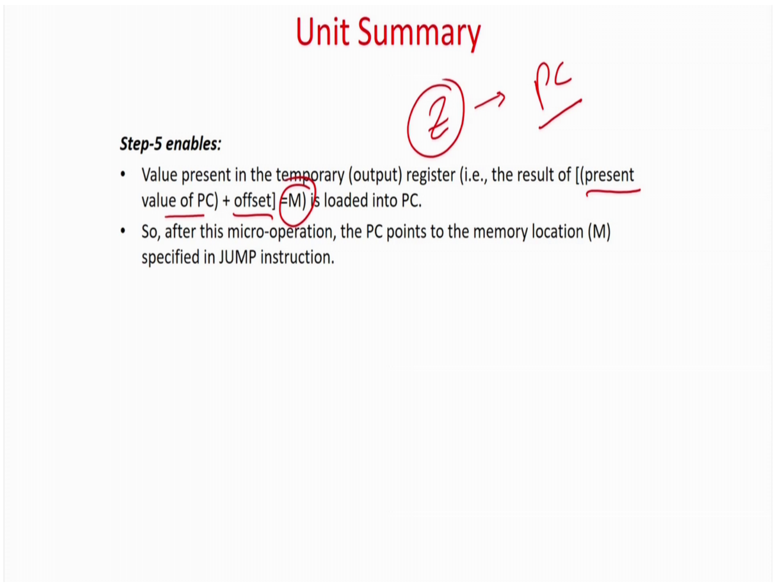
Next, what do you do the offset is set the offset value is this one and it is saying the input value of the CPU bus is the offset which is the second operand to the ALU. That means, what we are going to generate the jump address, and we are going to load it to the PC that is what is the job of the fourth step. So, what fourth step does fourth step is actually generated is offset, offset is nothing but the present value of CPU, a present value of the program counter minus M which is your memory location to be jumped that value is called the offset, the positive always take the positive value from it. So, in fact, so 30 20 is value of the offset in this example. Now, what do you do, so basically, so if you are considering single bus architecture, the 30 20 value will be an operant.

Next, what you do the input to the multiplexer is one. So, input the multiplexer is one, if you remember the previous lecture that is now not going to take any constant it is going to take the value from the temporary variable Y. So, what is the temporary variable storing here, we already discussed temporary variable Y is storing the value of PC. As I told you this is only extra thing that is happening in the first three instructions, when it is a jump instruction; in all other cases there is no involvement of any Y over here and the PC is directly updated. So, now, Y is another input to the ALU, it is the PC, and this is your offset which is nothing but present value of PC minus M. Then actually you configure the ALU to do addition operation.

So, finally, what you are doing we are doing the content of PC which is in Y plus and offset. So, basically what you are doing you are doing present value of PC minus M plus the value of PC. So, you are doing present value of PC minus m that value already is the offset plus PC, in fact, you are going to take the mode of it. So, basically what you are going we will cut out and you are going to get the value of m. So, which is in this case is 30 30 and which is actually loaded to a temporal register that is your Z. So, in fact, we use slightly a roundabout way means we are do not directly take the value of 30 30 and dump it into the CPU or shall dump it into the PC basically we take an offset value and then we add the value of PC to it.

These are actually help this helps for relocating programs, because if I say that if I write the value of 30 30 and say you are getting loaded at memory location some memory location called 700, so there is always an offset and relative addressing. To take care of the relative addressing, so how this roundabout way of generating the address is present. I cannot go into depth at this point on this part here because it is a part of something or assembly language programming. So, in that because to maintain the relativity and relative addresses, we go out a roundabout way rather they are not directly taking the value of 30 30 and loading it into the program counter. We just take a offset value first and then we again add the value of offset value to present value of program counter. So, they get canceled out, and you get the value of 30 30. So, it actually helps in relocating programs and some kind of other I means relative addressing more which you want to read in details, you can find out from the any kind any book on system programming.

(Refer Slide Time: 09:28)

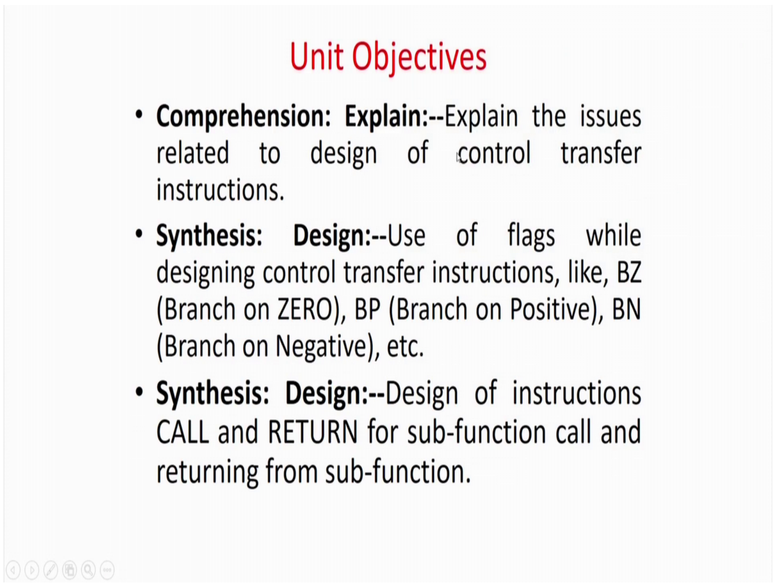

Unit Summary

Step-5 enables:

- Value present in the temporary (output) register (i.e., the result of [(present value of PC) + offset] =M) is loaded into PC.
- So, after this micro-operation, the PC points to the memory location (M) specified in JUMP instruction.

So, next is very simple. As I told you now the Z that is the output of the ALU stored in z which is nothing but the present value of PC plus offset which is nothing but equal to M. So, often that is already told that is nothing but value of PC minus M this is the (Refer Time: 09:40) when you get the M. And basically this one you will now update Z will update to program counter, so you are now jumping to that instruction and you are actually executing it. So, this is basically in summary what happens for the control signals and the microinstructions that takes place in a jump kind of an instruction. Anyway so that was a very brief summary that what happens and what are the control instructions, what are the flow that takes place for a jump instruction or any kind of control transfer.

(Refer Slide Time: 10:13)

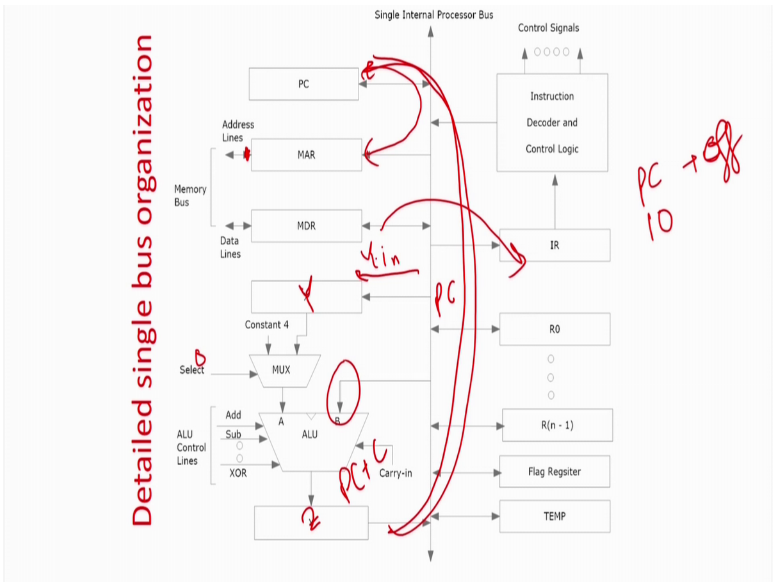


Unit Objectives

- **Comprehension: Explain:--**Explain the issues related to design of control transfer instructions.
- **Synthesis: Design:--**Use of flags while designing control transfer instructions, like, BZ (Branch on ZERO), BP (Branch on Positive), BN (Branch on Negative), etc.
- **Synthesis: Design:--**Design of instructions CALL and RETURN for sub-function call and returning from sub-function.

So, what are the objectives of this unit, first object is comprehension objective which tells explain the issue related to design of control transfer instructions. Like what are the special arrangement has to be made, if the PC has to be backed up what else extra instruction you require. Like here for example, the PC is also backed up at a temporary register Y and not only PC equal to the increment is stored into the PC, but along with that the updated value of PC is also stored in Y is something extra. Then design use of flags while design a control transfer instruction like branch on zero, branch on positive, branch on negative etcetera how the flag registers are used that explanation that synthesis will be able to do. And finally, the design objective design the instructions for call and return that is you will be able to design instructions which are involved in function call and return.

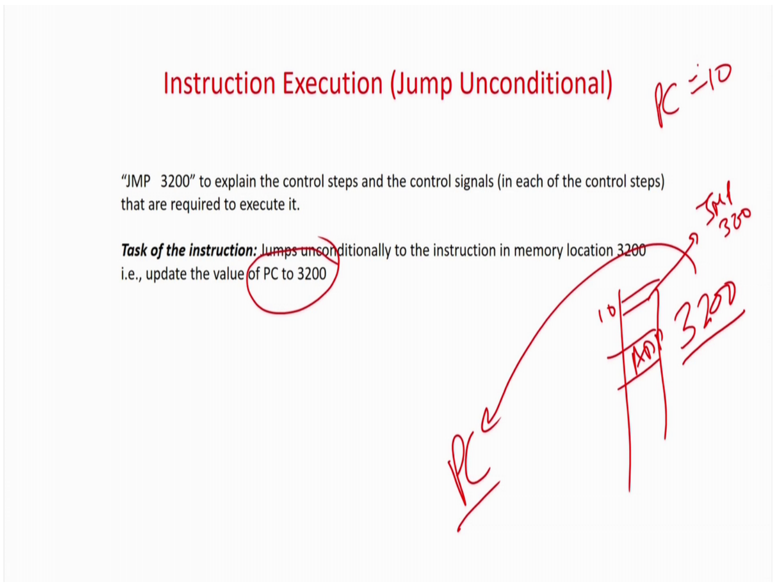
(Refer Slide Time: 10:55)



Now, before we start the module unit, we will have basically look at the single bus architecture, which is very similar you have the program counter, then you have the memory address register, memory data register very similar, then you have the ALU and there is a temporary register which call x which is an input to this. There is slight difference I mean this is in the previous thing you might have seen this as Y this we are calling it Y in the previous lecture, and this we are calling it Z.

So, just way of nomenclature let us keep it same for the ease of understanding or continuity that is Y that temporary register Z is the output of the ALU this is marks is a constant four, four actually just an example, it can be the size of the instruction. Then the ALU basically, this is the ALU which does all addition, subtraction. Then you have the temporary register all the tem all the temporary register this is your register set from zero to m, instruction register and control logic. One very important thing you have to note here is the flag register which is there, but actually I am again drawing it here specially because it takes a very important role in any kind of a control instruction. So, because zero, carry, jump on, overflow etcetera all depends on the flag value set. So, this flag resistor is playing a very important role in basically determining the control signal. So, that is why these are special emphasis we have put on this one, they the flag which is mainly shown over here.

(Refer Slide Time: 12:26)


Instruction Execution (Jump Unconditional)

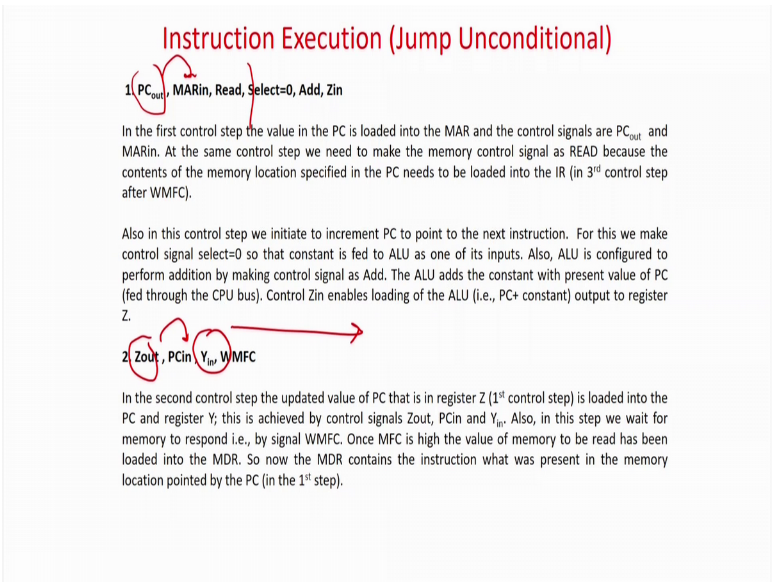
"JMP 3200" to explain the control steps and the control signals (in each of the control steps) that are required to execute it.

Task of the instruction: ~~Jumps uncon~~ditionally to the instruction in memory location 3200 i.e., update the value of PC to 3200

PC = 10
JMP 3200
10
ADD
3200
PC

So, now with this let us start taking some examples of instructions that exactly see what happens. The first instruction will be dealing it very details by looking at the bus diagram, bus architecture, how the signals are moving; and other instructions we can have a very quick overview. So, jump 30 unconditional jump; that means, the program counter should go to the value of 3200. So, if you look at the memory this way. So, maybe this is 3200 is the memory location some instruction may be there like say add or something, so that will be loaded to the PC and PC will do it. So, somehow you have to load the value of PC to this value. So, maybe there is an instruction or it can be having a hard instruction or whatever. The idea is that I want to jump this place, and I want to get this value out of this instruction or I want to execute this instruction. So, PC actually will have the value here. So, it will make a jump instruction. So, somehow I have to load the value of program counter to 3200 and your job is done, then actually programmed reveal execute from memory location 3200.

(Refer Slide Time: 13:35)



Instruction Execution (Jump Unconditional)

1. PC_{out}, MARin, Read, Select=0, Add, Zin

In the first control step the value in the PC is loaded into the MAR and the control signals are PC_{out} and MARin. At the same control step we need to make the memory control signal as READ because the contents of the memory location specified in the PC needs to be loaded into the IR (in 3rd control step after WMFC).

Also in this control step we initiate to increment PC to point to the next instruction. For this we make control signal select=0 so that constant is fed to ALU as one of its inputs. Also, ALU is configured to perform addition by making control signal as Add. The ALU adds the constant with present value of PC (fed through the CPU bus). Control Zin enables loading of the ALU (i.e., PC+ constant) output to register Z.

2. Zout , PCin, Y_{in}, WMFC

In the second control step the updated value of PC that is in register Z (1st control step) is loaded into the PC and register Y; this is achieved by control signals Zout, PCin and Y_{in}. Also, in this step we wait for memory to respond i.e., by signal WMFC. Once MFC is high the value of memory to be read has been loaded into the MDR. So now the MDR contains the instruction what was present in the memory location pointed by the PC (in the 1st step).

Now, we will see how it happens basically. The first stage is already discussed program counter out memory address in because why we already have discussed so much. So, let us think that this is memory location 10, somewhere it is executing. So, basically program counter is equal to 10, maybe the instruction called say jump 3200 is available at this place. So, what you have to do, you have to load the value of memory location 10 has the instruction jump 3200. So, already so many times I have discussed that this 10 value has to be load loaded into the memory address register.

So, basically program counter will be output it will be loaded into the memory address register, you have to make the memory in a read more, because you want to read the instruction. Select will be equal to 0 because you have select means that is the ALU must select. So, it will be adding a constant add and Z in basically the value will be written. So, it will quickly look at it, look at the structure what we are saying that value of program counter that is 10 will be fair to the memory address register. And memory data register after a certain amount of time we will get the instruction which is in that program counter memory address in the memory, it will be going to the instruction register that is what is a standard thing that happens.

Then and correspondingly PC out, memory address register in after the second stage it will be MDR out people go to instruction register, but before that also some this will actually happen up to certain amount of time because you have to give some time for the memory to respond. So, it will actually happen in the third cycle. But in between we prepare for to increment the PC, so we make it the value of 0, so that the constant can be added. So, in fact, program counter value is in fact, the PC value is over here in the bus because PC out is there it is going to the a memory address register as well it is available in the bus and constant this is equal to 0.

So, constant one or four that depends on the instruction size is there and we are making z in that means, here you have to you have PC plus constant that is four or whatever that value is ready. And whenever we say Z out means it is ready. So, in the second stage actually we will dump it over the piece that is going to happen. So, that is what program counter value will go to the memory address register read it, select zero means keep the value constant to be added, add is the mode, and Z in means output of the ALU will be stored in Z.

Next, what do you do PC in; that means, already the arithmetic logic unit has dump the value of PC plus constant in Z. So, you are making Z out, and you are going into the PC in that is PC has been incremented and we are waiting till the memory is ready, so that the instruction is dumped into the memory buffer register in third stage. Extra thing again I am repeating an extra thing that is Y in is very special in jump structure this was not there in the any kind of add load or store instruction. Now, we will see what is specialty about Y in that is Z out and Y in together; that means, we are having Z out z out is having nothing but it is having PC equal to PC plus constant that is the next value of the program counter. We always store it in PC that is standard, but again also we are storing in W, what is that extra and how it will help.

So, if you look at it program is constant basically it is going to PC we always store it because now PC is going to point to the next thing structure. But for the time being also we are dumping it to Y, so that is actually called Y in that is special that. PC equal to PC bus constant is dump as well as to the PC as well as to the temporary register, we will see why it will be required. I can just give you a very basic idea of y because the present value of PC may be say ten will be stored over here and after that we the offset will be given to this second block.

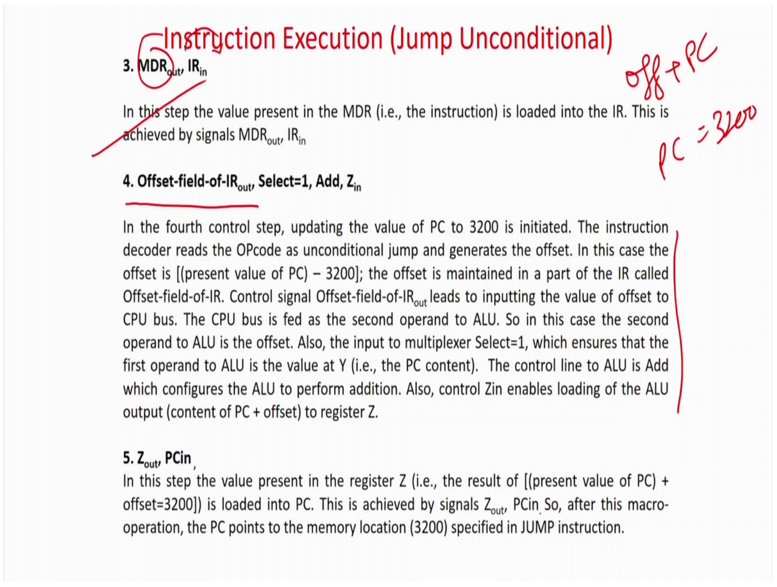
So, in that case, what happen this is the present value of PC plus offset will again give you the value offset which will coming which will come later the offset will be coming from the instruction register it will be added to the value of PC and you are going to get the value jump address. Already we have seen how to calculate the jump address by adding the present value of PC plus offset, because we have to have add the value of PC not be the constant, but this time with an offset, so we have to have the value of PC stored in a temporary register.

For all other cases, if you member the value for the first case that is you have to increment PC what you can do, you can load the value of PC in the bus and you can take me constant over here. So, one operand is constant and another operand is the PC, but in this case one operand is coming from the instruction register that is the offset and another one is actually your PC. So, how can I have two things going to the ALU together. You could have done it is that you could have taken the value of offset and dumped it in Y, and you could have taken the value of program counter in the bus that is also possible. That you take that you store the instruction register offset in Y and then you just float the value of PC in the bus and you can do it add it. And another way is that you store the value your program counter PC in Y and float the value of offset and that is actually a simpler way of doing it.

Because in this second stage you are anyway dumping the value of PC in the update value of PC in the program counter register. Same time, hand in hand you just take the value in y and storing after that you just take the value of IR in the bus and add it that is a simpler way of doing it rather than storing the PC first in the program counter on the updated value of PC in this one you are storing it. Then after some amount of time you do not go for this you remove this, you do not update the value of Y with this, you take in fact the offset and you load it over here and then again take the PC roundabout way. So, what we do we take a simplistic approach.

So, what are the simplistic approach the simply approaches, first load the value of program counter in the memory address register at this what we do, and make this is zero, so that the constant comes over here and the PC is already over here. So, this is equal to PC plus constant and (Refer Time: 19:20) z basically. So, Z in so the z will actually have the value of program update program counter, then you in the second stage you make Z out. So, it will go into the PC that is the updated value hand in hand also you dump the value of Y in the tempo temporary register Y you dump the value of program counter updated program counter. And next stage you take the value of IR in the bus and that is the offset from the add it. So, up to second stage, second stage, it is simple updated program counter value goes to PC in addition you are also storing the value of updated program counter it Y.

(Refer Slide Time: 19:54)



Instruction Execution (Jump Unconditional)

3. MDR_{out}, IR_{in}

In this step the value present in the MDR (i.e., the instruction) is loaded into the IR. This is achieved by signals MDR_{out}, IR_{in}

4. Offset-field-of-IR_{out}, Select=1, Add, Z_{in}

In the fourth control step, updating the value of PC to 3200 is initiated. The instruction decoder reads the OPcode as unconditional jump and generates the offset. In this case the offset is [(present value of PC) – 3200]; the offset is maintained in a part of the IR called Offset-field-of-IR. Control signal Offset-field-of-IR_{out} leads to inputting the value of offset to CPU bus. The CPU bus is fed as the second operand to ALU. So in this case the second operand to ALU is the offset. Also, the input to multiplexer Select=1, which ensures that the first operand to ALU is the value at Y (i.e., the PC content). The control line to ALU is Add which configures the ALU to perform addition. Also, control Zin enables loading of the ALU output (content of PC + offset) to register Z.

5. Z_{out}, PCin

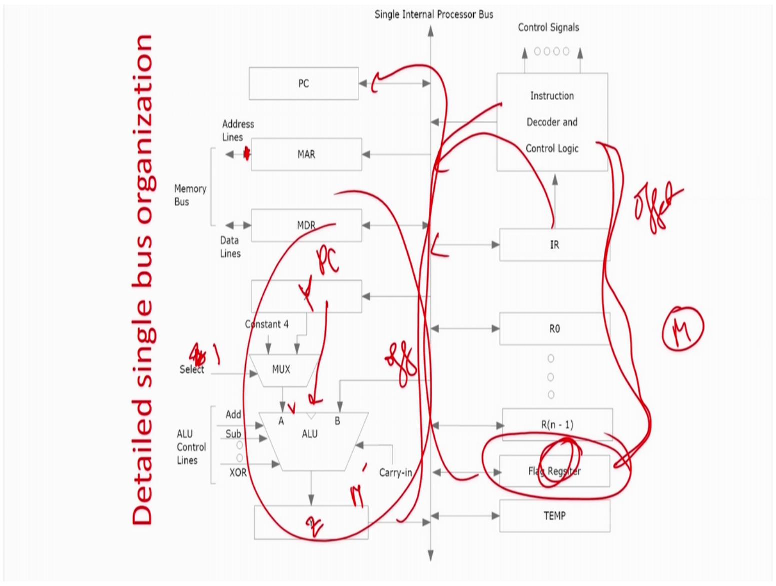
In this step the value present in the register Z (i.e., the result of [(present value of PC) + offset=3200]) is loaded into PC. This is achieved by signals Z_{out}, PCin So, after this macro-operation, the PC points to the memory location (3200) specified in JUMP instruction.

Now, let us see what happens. Then in the third stage is as simple as all other instruction that is your memory is ready. So, you dump the value of memory data register to the instruction register that is the instruction in this case jump 3,200 is loaded in to the instruction register that is same as all instruction. That is loading the value of the instruction from the memory to the instruction register. This is a new field new type of stuff which you are saying offset value of IR, because now you have to have the PC loaded with 3,200. And already seen how we can get it we can say that offset plus PC will actually give the value of 3200, because already we know offset is nothing but present value of PC minus 3200.

So, if you and the mode of it, so if you add the value of PC to it, you are going to get the value of 3200 that is actually elaborated in this write up you can read it from this slide I can just feel it. But again I am telling you this is of direct loading 3200 to the PC, we are taking up indirect way of doing it like by calculating the offset value and adding. So, this is actually going to help in many kind of re-locatable programs or relative addressing mode which you can read mainly in the see any system programming kind of book which can tell you what do you mean by the relative addressing etcetera.

So, for the time being, just you have to take in a black box that I am generating the value of 3000 in a roundabout manner that is calculating the value offset and then adding the value of PC. So, what I am doing I am saying that the offset value of IR equal to out that means, the instruction register will dump the value of this one so inbuilt way of generating the offset so it is out, select equal to one add and z in. Select equal to 1, in this case if you observe then you are not going to add again I am going to look at it.

(Refer Slide Time: 21:39)



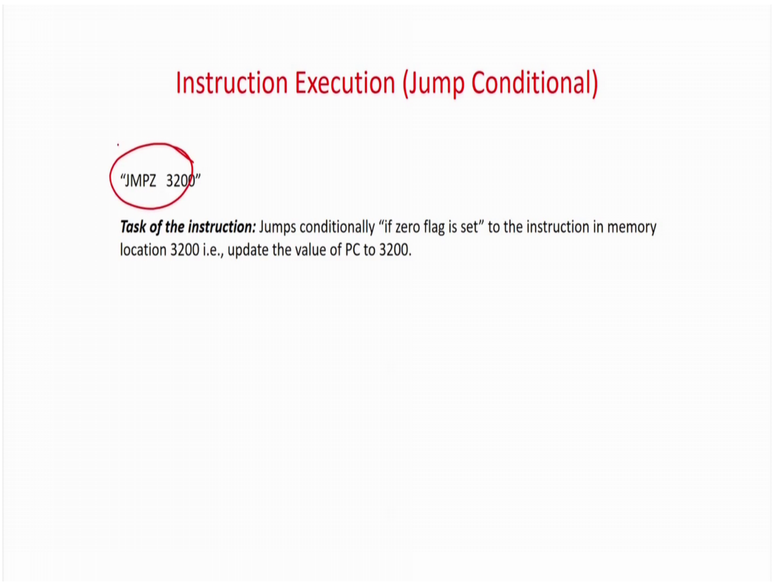
So, select will be now equal to 1, so that means, you are going to take the value from this Y register that is now but this is nothing but your recent value of PC. And also at that same time you are also dumping in this location let me erase the first part of the instruction, this one we are not erasing, now this is erased. Now, this PC is going over here that is PC equal to constant is outdated value of next value of PC, and this is coming over here.

And from instruction register basically through this, you are actually dumping the value of basically I should write it in this way, you are dumping the value of offset it is not the goal PC the resistor we have already told that with some things take a part of the IR register. So, in this case the IR is slightly multifunctional. So, he just calculates the offset and dumps the value over here. So, here you are getting the offset. So, if you offset you are adding with the program counter. So, basically you are going to get the value of M that is your jump instruction, so that is your M. So, M will be fed to Z, and it will be kept as input.

Next stage you will just dump the value of z to program counter, and your job will be equal to done, so that is what is done. So, in this case, we are keeping the value of offset as out select one that means, you are going to take the program counter input to the ALU plus offset and the value of Z in will be there which is actually equal to M. Next is very simple Z out equal to PC in. So, whatever will be the value of Z out that is nothing but M that is the jump instruction, jump locations PC and your job is done.

So, basically these tells that five microinstructions are required to basically to complete this whole unconditional jump instruction, and these are the control signals generated. And how they are actually play at all in a single bus architecture we have discussed it in details. In the next few similar type of instructions, we will not go in as details as I have told you, but we will just give a direction which will be more and more or less similar type of flow in the bus architecture, single bus architecture.

(Refer Slide Time: 23:37)



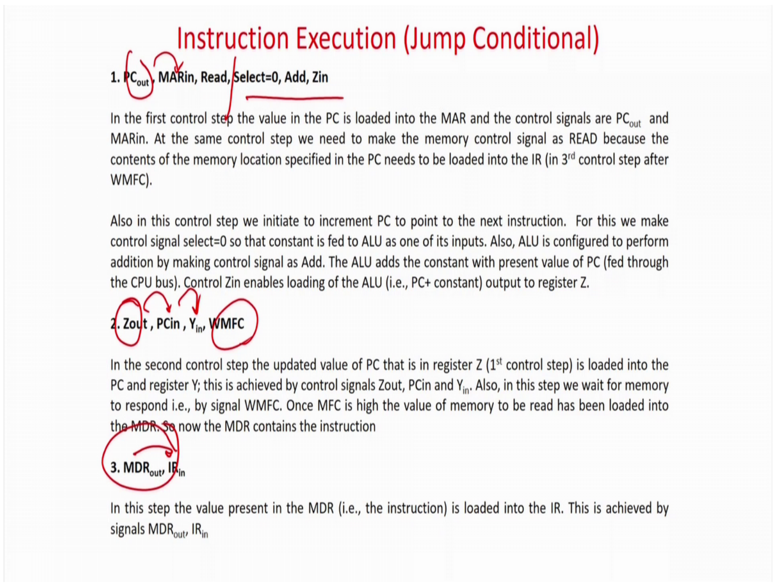
Instruction Execution (Jump Conditional)

"JMPZ 3200"

Task of the instruction: Jumps conditionally "if zero flag is set" to the instruction in memory location 3200 i.e., update the value of PC to 3200.

Let us slightly changing the instruction mode that is jump on zero that is conditional jump, you do over here.

(Refer Slide Time: 23:40)



Instruction Execution (Jump Conditional)

1. PC_{out}, MARin, Read, Select=0, Add, Zin

In the first control step the value in the PC is loaded into the MAR and the control signals are PC_{out} and MARin. At the same control step we need to make the memory control signal as READ because the contents of the memory location specified in the PC needs to be loaded into the IR (in 3rd control step after WMFC).

Also in this control step we initiate to increment PC to point to the next instruction. For this we make control signal select=0 so that constant is fed to ALU as one of its inputs. Also, ALU is configured to perform addition by making control signal as Add. The ALU adds the constant with present value of PC (fed through the CPU bus). Control Zin enables loading of the ALU (i.e., PC+ constant) output to register Z.

2. Zout , PCin , Y_{in}, WMFC

In the second control step the updated value of PC that is in register Z (1st control step) is loaded into the PC and register Y; this is achieved by control signals Zout, PCin and Y_{in}. Also, in this step we wait for memory to respond i.e., by signal WMFC. Once MFC is high the value of memory to be read has been loaded into the MDR. So now the MDR contains the instruction

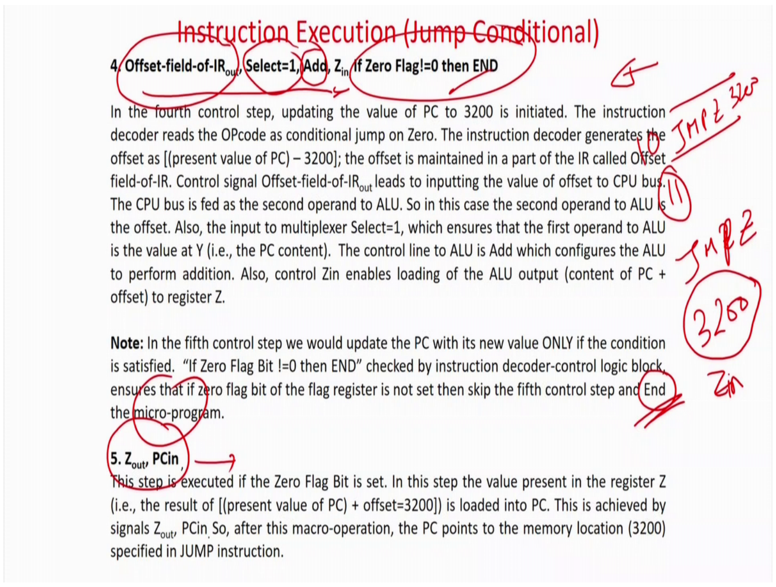
3. MDR_{out}, IR_{in}

In this step the value present in the MDR (i.e., the instruction) is loaded into the IR. This is achieved by signals MDR_{out}, IR_{in}

So, first instruction will be same program counter out, memory address register in, read, select at 0, then what you do that is the program counter output you will load it to the address register, so that you can get the value of the instruction in the third part have an instruction from in it because it is third instruction. So, the third microinstruction basically the memory will say that I am ready. So, the instruction which will be loaded into the memory data register will come to the instruction register as simple as that. And this already discussed is the same for all instructions select 0 add and Z in means you are preparing to add a constant program counter. So, it is done.

So, Z out in this case is nothing but program counter plus updated constant which will actually upload in program counter as well as we are a special case of jump instruction, we are storing it in Y in that job is done. Next by WMMC means your memory is now ready in the third stage what happens basically the memory data register will dump the value that is the instruction to the instruction register that job is done.

(Refer Slide Time: 24:38)



Instruction Execution (Jump Conditional)

4. Offset-field-of-IR_{out}, Select=1, Add, Z_{in}, If Zero Flag!=0 then END

In the fourth control step, updating the value of PC to 3200 is initiated. The instruction decoder reads the OPcode as conditional jump on Zero. The instruction decoder generates the offset as [(present value of PC) – 3200]; the offset is maintained in a part of the IR called Offset field-of-IR. Control signal Offset-field-of-IR_{out} leads to inputting the value of offset to CPU bus. The CPU bus is fed as the second operand to ALU. So in this case the second operand to ALU is the offset. Also, the input to multiplexer Select=1, which ensures that the first operand to ALU is the value at Y (i.e., the PC content). The control line to ALU is Add which configures the ALU to perform addition. Also, control Zin enables loading of the ALU output (content of PC + offset) to register Z.

Note: In the fifth control step we would update the PC with its new value ONLY if the condition is satisfied. "If Zero Flag Bit !=0 then END" checked by instruction decoder-control logic block, ensures that if zero flag bit of the flag register is not set then skip the fifth control step and End the micro-program.

5. Z_{out}, PCin

This step is executed if the Zero Flag Bit is set. In this step the value present in the register Z (i.e., the result of [(present value of PC) + offset=3200]) is loaded into PC. This is achieved by signals Z_{out}, PCin So, after this macro-operation, the PC points to the memory location (3200) specified in JUMP instruction.

Next will be very similar that you have to take the offset from the instruction register, add it; and in this case is important then you have to add it, select is equal to 1. So, this will be the output select equal to 1 that means, you are going to take the temporary value that is Y and not the constant and you are adding it that means, offset plus PC that is actually going to give you the M where you have to jump. And then you are actually save the value in Z; already this is very, very similar to the last unconditional jump instruction. Here is a very important thing that is coming up is called if zero if not flag zero then end.

I am not going into the internal details of this microinstruction corresponding to this, this is very simple, what happens basically this if you look at it this flag register is your flag register, it is a register with some special multifunction. So, whenever you are saying that the opcode phase that basically it is a jump on 0, so just you look at the flag which corresponds the flag bit which corresponds to basically your zero flag. So, if the zero flag is not zero, basically what it does it say that if zero flag is nonzero then end that very simple it will just take the value of the PC to a reset value or to the last value of the program counter and that microinstruction.

So, it says that in the fifth control step we will update the PC only if the condition is satisfied. If it is not zero then end. It is checked by the instruction decoder block and then it actually skips the next step or actually it ends the micro program in skipping is not a proper one in that sense basically. So, what happens that is zero instruction register in this pressure in this case will just check the zero flag. Actually, in fact, this is an input to the zero flag. In fact, this one is actually input to the instruction decoder or in fact we can have a connection over here I am not going into in a details because it will actually make it more complicated, but the idea is very simple.

So, what happens the output of this one is basically the instruction decoder which we flag bit and if the flag bit zero th flag bit is not set is not zero basically then what do you do you just reset the value of PC to the last address of the micro program. That is you are ending the micro program that is you reset the value or in this case you just return to the program which to the program to the instruction which is calling the microinstruction. That means you are ending the sequence of microinstruction here itself that means, you are actually trying to go to excite the next macroinstruction, so that is a reset of the PC that is very simple.

But if it is satisfied then what is going to happen is as simple as take the value of Z in and put it to the PC. That means, at this point of time when you are saying offset plus select one and add when you are doing and Z in that is these three parts basically there is offset value of IR select and add. In this case, you are adding y temporary register with the that is actually having the value of PC that is again let me just look at it. So, this one you are adding it, this is the offset, this is the PC, this part you are adding, this is the part you are adding and keeping it in cell, but whether we will upload the value of z to program counter will depend on whether what is the value of this flag register.

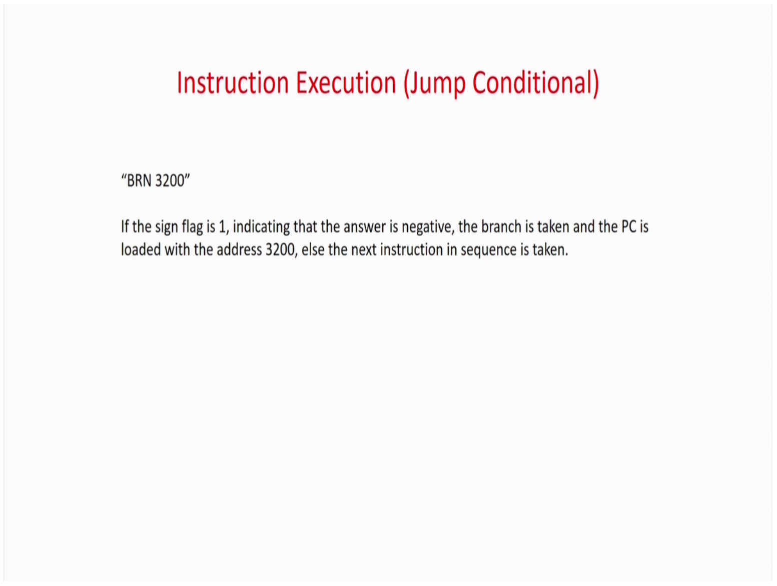
So, if the value of the zeroth flag in the flag register is satisfied that will be checked by the instruction decoder, then it will not do anything. It will just go to the fifth stage; fifth stage will just set Z out and PC. But if it is not set the zero flag is not set what it will do you know we just instruct the instruction decoder to reset the value of PC or actually in fact return to the macro instruction which actually call this sequence of microinstructions.

Now, in shortly see what is call and return that means, this for the time being I assume that is just somehow resets it, so that nothing more gets executed and basically your micro program ends. So, either there is two option; in this case either you end it that is you return to the macro instruction will go to the next instruction or you go to PC that means, says jump on zero jump on you know let me see an instruction. So, it is a jump on zero j m p 0. So, j m p 0 is the instruction opcode and you are going to see some location 3200.

So, till this point I am ready with this value in Z in, Z in as the value of 3000. Just I need to either update the program counter with 3200 that is simple like Z out PC in or I have to reset that is in the memories of memory location we are assuming it ten memory location have the instruction JMP0 3200. So, this one actually called the sequence of microinstructions, which we are discussing is five microinstruction. If in the fourth stage, you find that this zeroth flag is not satisfied, then I am not going to execute this fifth microinstruction. I will reset my PC in such a fashion that actually I will go to the eleventh instruction and you start from there itself; else you will reload it and the other set of equations you execute.

So, this actually explains how slightly if there is a conditional stuff then how slightly it changes. I am not discussing in details about how to check if zero flag is not equal to zero, very simple just it is kind of AND gate XOR gate based check. And just you have to reset the value of PC or return to the next macro instruction returning will be dealing later. But if it is zero then just we do not have to do anything, do not reset, you just go to the next microinstruction, it is very simple implementation.

(Refer Slide Time: 30:11)



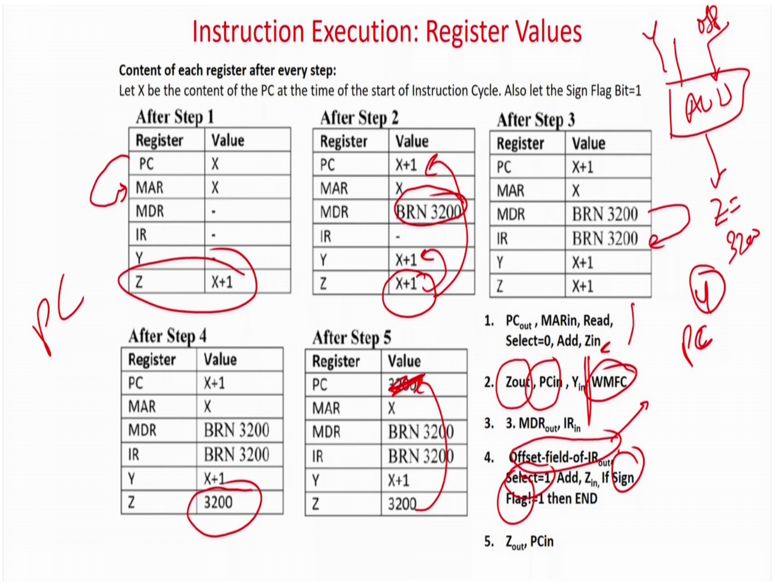
Instruction Execution (Jump Conditional)

"BRN 3200"

If the sign flag is 1, indicating that the answer is negative, the branch is taken and the PC is loaded with the address 3200, else the next instruction in sequence is taken.

Before going to jump and return, we have another simple instruction I am taking which is a sign flag. It is saying that branch to 3200 if the sign flag is one that means, it is a sign bit has been set. Same steps I am no difference, so in this case fetching the instructions and then in this case instead of your zero flag, you are going to check this sign flag. If they sign flag is set then it is fine; and it is not set you will actually go to the end that is we are going to reset.

(Refer Slide Time: 30:40)



Instruction Execution: Register Values

Content of each register after every step:

Let X be the content of the PC at the time of the start of Instruction Cycle. Also let the Sign Flag Bit=1

After Step 1

Register	Value
PC	X
MAR	X
MDR	-
IR	-
Y	-
Z	X+1

After Step 2

Register	Value
PC	X+1
MAR	X
MDR	BRN 3200
IR	-
Y	X+1
Z	X+1

After Step 3

Register	Value
PC	X+1
MAR	X
MDR	BRN 3200
IR	BRN 3200
Y	X+1
Z	X+1

After Step 4

Register	Value
PC	X+1
MAR	X
MDR	BRN 3200
IR	BRN 3200
Y	X+1
Z	3200

After Step 5

Register	Value
PC	3200
MAR	X
MDR	BRN 3200
IR	BRN 3200
Y	X+1
Z	3200

1. PC_{out}, MARin, Read, Select=0, Add, Zin
2. Zout, PCin, Y_{in}, WMFC
3. 3. MDR_{out}, IR_{in}
4. Offset-field-of-IR_{out}, Select=1, Add, Z_{in}, If Sign Flag!=1 then END
5. Z_{out}, PCin

But again I am just using this one to see what are the different register values in this example. So, we will be mainly concentrating at what are the register values for this instruction. So, what is the first command, first command is PC, memory addressing we select zero and this one that means, what I am trying to do over here I am going to add give the value of program counter to the memory address, so that we can get the instruction out of it. Then what happens basically after certain amount of halt etcetera the second stage says that Z out to the program counter in and Y in, Y in is very special.

So, in this case second instruction as we know so of course, this one corresponds to the incremental part, you are saying select 0, add and Z in. So, this thing means actually you have to add select 9 means it is the constant, add means the ALU will add an updated value of PC that is PC plus constant, in this example one we are taking an example will be give in the register z. So, said equal to z plus 1 PC has been implemented. Second stage basically Z out PC in the value of this updated program counter will be loaded into the PC, the PC is now updated.

And as I told you as a special case in case of jump addresses, the operand value of PC is also given to Y we have already seen Y because you have to add the value of updated value of PC to the offset. So, you require two operands. So, one operand we are storing in the tempo register Y. Then what happens basically then you have to wait for some amount of time in fact, it in greater than so in fact what happens this after certain amount of time after all this halt etcetera, we are putting the halt the instruction over here. After halting basically when everything is saturated the value of the instruction will be loaded over here that is the branch instruction will come to the memory data register that means, after Y in you should wait for WFMC when it is done it will come over here.

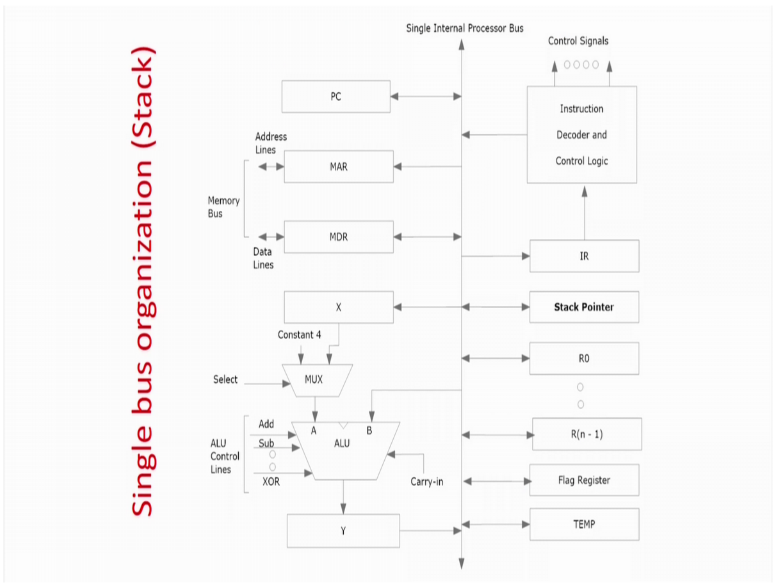
Next, what next I told you it is very simple the instruction from the memory data register we will go to the instruction register is a very standard operation. Now, what now four, now four here you are going to see. Now, what we are going to see in four instruction or that is your offset select equal to one over here and add that means offset plus offset in this is offset this is already we know what is the value of offset, and select equal to 1 means you are going to take the value from Y. So, y is having the value of program counter. Y is this is now y is actually having the value of program counter. And you are going to add if you see offset field IR out.

So, if you consider this is as your ALU already we see ALU is like this, it is coming from Y, because select is equal to 1, it is in add more value is going to Z register because of the Z in. And of course, the other input is coming from the offset because offset field of IR equal to out. So, this is your actually offset. So, we already have seen that Y is nothing but your PC if you are adding to offset. So, Z, it is will be not in be nothing but equal to 3200 over here - the jump address. We have already seen the calculation.

Then we are going to see so basically your Z is having the value of 3200 over here after doing the calculation in fourth. Now, importantly whether this fifth stage that is nothing but your Z will be loaded to the program counter or not will depend on if this sign flag. So, if the sign flag is set as required over here, we are going to the fifth stage and the value of Z will be loaded to the PC. So, the PC will start executing instructions in 3200 or if it is a false then basically the flag is not said as required so program counter value will not be 3200, it will be actually the address for the next macro instruction. The first macroinstruction was branch on zero 3200. So, it is not the case it will actually come out of this whole microinstruction cycle and it will go out to the next macroinstruction to be executed.

So, basically so we have taken another ex example in which case we have taken a branch on some sign and we have seen that for these steps basically what are the values which are loaded into the different registers. You can just look at the PC and calculate for yourself and you can get a very easy understanding. Now, we have seen about jump, conditional, and jump unconditional.

(Refer Slide Time: 34:47)

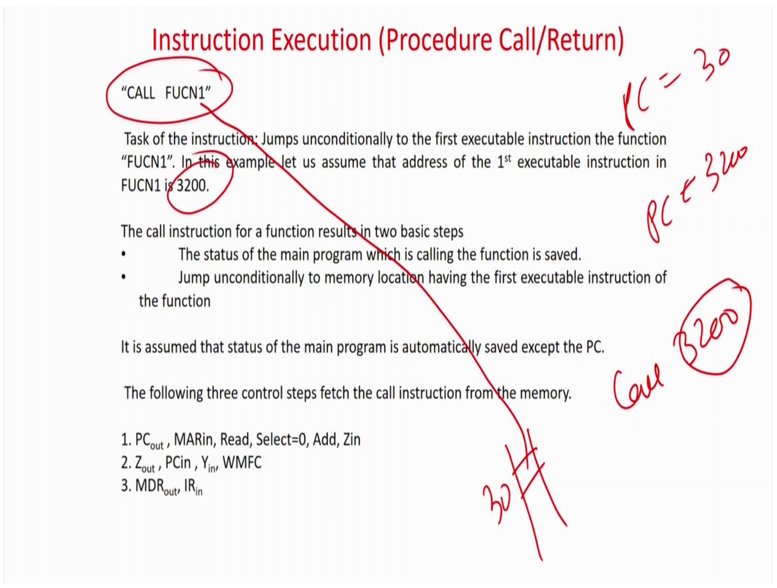


Next, this call and return that is one very important thing. We have also seen that when you were discussing about some kind of a subroutine call when you are looking at interrupts. So, every time we have seen that before we jump to a function or a sub interrupt sub routine service. So, what we do we save the value of the program counter we save the value of all the intermediate registers and all these stuff which we are doing in one function was temporary scratchpad or temporary memory, go and service the interrupt go or finish the prostitute and again come back and again pick up all the values of the temporary registers which we have saved and start executing.

Basically first very important if that save the PC. We save the PC because we will come back, we have to have to read out the value of the program counter and start executing from there. So, we have to always save the value of the PC in a stack. And along with the PC, we save all the temporary registers, but to make it the discussion bit simpler, we will assume that only PC we are saving, all others are saved default manner because that will actually make the decision bit simpler.

So, along the same single bus architecture, but with only one addition that is your stack pointer. If you look at it, I have put one stack pointer out over here. So, the stack pointer is very important because in this case you have to save the value of PC and where the value of PC will be safe is actually be defined by a stack pointer. To make it simpler as I told you, you are only showing for the PC, but along with the PC, you have to also save other stuff like your temporary registers, accumulators etcetera, which are the timing being we are assuming that will be set as default.

(Refer Slide Time: 36:20)


Instruction Execution (Procedure Call/Return)

"CALL FUCN1"

Task of the instruction: Jumps unconditionally to the first executable instruction the function "FUCN1". In this example let us assume that address of the 1st executable instruction in FUCN1 is 3200.

The call instruction for a function results in two basic steps

- The status of the main program which is calling the function is saved.
- Jump unconditionally to memory location having the first executable instruction of the function

It is assumed that status of the main program is automatically saved except the PC.

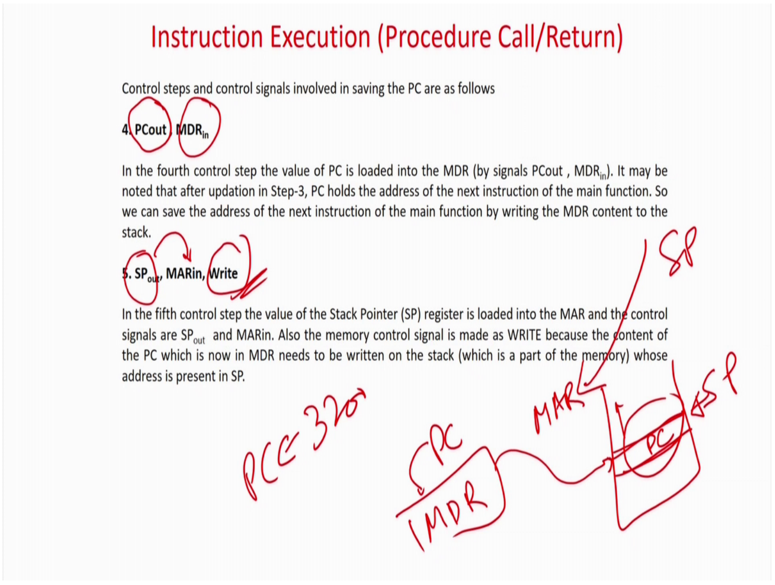
The following three control steps fetch the call instruction from the memory.

1. PC_{out}, MARin, Read, Select=0, Add, Zin
2. Z_{out}, PCin, Y_{in}, WMFC
3. MDR_{out}, IR_{in}

So, we will just go for function call and return. So, for example, we call is that we are calling function one and function one memory later set up 3200 that means, what we are calling the call 3200. 3200 memory location actually have the first instruction of that function. So, our job is to jump to 3200, but before that that means, you have to load the value of program counter to 3200, but before that is very important maybe you are at program counter is maybe at 30. So, at 30 memory location 30 if you are calling it as memory location 30, memory location number 30 is having the function call that will jump to 3200.

So, before executing this jump that is before making the program counter 3200, we have to store the value 31 that is the next location in a stack, because when I will be returning I have to start executing from 31 that is the idea. So, we will see quickly how it works basically. So, the first three set will be very very similar fetching the instruction and putting in it will the instruction register.

(Refer Slide Time: 37:24)



Instruction Execution (Procedure Call/Return)

Control steps and control signals involved in saving the PC are as follows

4. PCout , MDR$_{in}$

In the fourth control step the value of PC is loaded into the MDR (by signals PCout , MDR$_{in}$). It may be noted that after updation in Step-3, PC holds the address of the next instruction of the main function. So we can save the address of the next instruction of the main function by writing the MDR content to the stack.

5. SP$_{out}$, MARin, Write

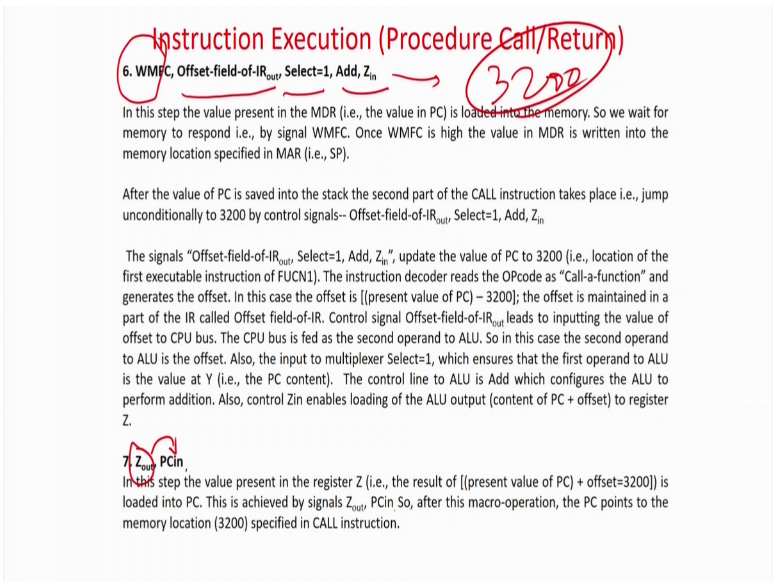
In the fifth control step the value of the Stack Pointer (SP) register is loaded into the MAR and the control signals are SP$_{out}$ and MARin. Also the memory control signal is made as WRITE because the content of the PC which is now in MDR needs to be written on the stack (which is a part of the memory) whose address is present in SP.

Now, already we know that when you are executing a microinstruction number three, program counter have been implemented equal to program counter plus constant. So, this has to be stored in a stack. So, what we are doing. So, basically program counter that is fourth state PC out that is the after that value of program counter we are putting it in memory data register that is simple, because whatever you want to write it to the memory we have to put it into the memory data register. Next, what, next we are making the memory to a write mode because the program counter has to be saved to a stack which is the memory now when I have to save that is very important that is this is your old memories there may be from here may be this stack starts.

So, now, what happens here store the value of program counter that is the update program counter program counter plus constant that is already updated in up to step number three. Now, this is your memory data register memory data register which we have already value of PC and we stored over here which has to be now dumped to this stack. So, the fourth stage PC out MDR in basically the value of program counter is stored for a MDR now the MDR has to be written to the memory. So, we make the memory in a write mode. And what will be the memory address register value it will be memory address register value will be nothing but your stack pointer. So, the pointer actually will grow upside or downside depending on the requirement. So, at present, we are assuming that we are growing up. So, this was the first place where the current PC has to be stored. So, actually the stack pointer knows this position. So, the stack pointer value will be related to the memory address register.

So, once it is done, of course, the value of program counter will be saved into them. So, basically these stage SP out memory data registered in address register in means the program counter stack pointer value will be saved to the memory address register, and there will be saving the value of the program come from as simple as that. Then next part it is so we have save the value of the program counter now we have put down to the value of program counter we have to dump the value of now 3200 because already we have saved the value of the program counter in this stack pointer.

(Refer Slide Time: 39:26)



Instruction Execution (Procedure Call/Return)

6. WMFC, Offset-field-of-IR$_{out}$, Select=1, Add, Z$_{in}$ $\rightarrow$ 3200

In this step the value present in the MDR (i.e., the value in PC) is loaded into the memory. So we wait for memory to respond i.e., by signal WMFC. Once WMFC is high the value in MDR is written into the memory location specified in MAR (i.e., SP).

After the value of PC is saved into the stack the second part of the CALL instruction takes place i.e., jump unconditionally to 3200 by control signals-- Offset-field-of-IR$_{out}$, Select=1, Add, Z$_{in}$

The signals "Offset-field-of-IR$_{out}$, Select=1, Add, Z$_{in}$", update the value of PC to 3200 (i.e., location of the first executable instruction of FUCN1). The instruction decoder reads the OPcode as "Call-a-function" and generates the offset. In this case the offset is [(present value of PC) – 3200]; the offset is maintained in a part of the IR called Offset field-of-IR. Control signal Offset-field-of-IR$_{out}$ leads to inputting the value of offset to CPU bus. The CPU bus is fed as the second operand to ALU. So in this case the second operand to ALU is the offset. Also, the input to multiplexer Select=1, which ensures that the first operand to ALU is the value at Y (i.e., the PC content). The control line to ALU is Add which configures the ALU to perform addition. Also, control Zin enables loading of the ALU output (content of PC + offset) to register Z.

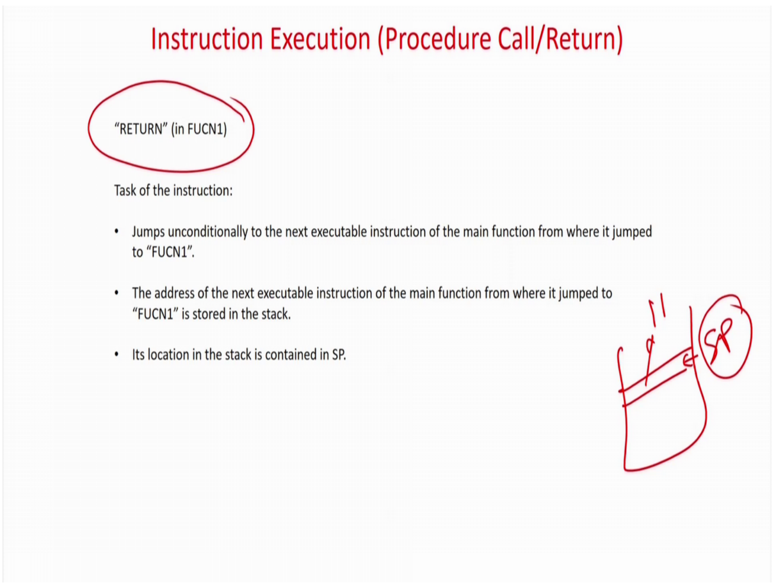
7. Z$_{out}$, PCin

In this step the value present in the register Z (i.e., the result of [(present value of PC) + offset=3200]) is loaded into PC. This is achieved by signals Z$_{out}$, PCin So, after this macro-operation, the PC points to the memory location (3200) specified in CALL instruction.

So, it is very simple already we have seen, we have to wait for some amount of time till the program counter value has been saved because before that if we update the program counter there may be a race condition or there may be a problem. Now, you wait till your memory has been written, so you are safe. Now, you are taking the offset value of the register select 1, add and Z in that is same stuff we are actually making the offset value added to the program counter which is nothing but it will give the value of the jump address in these cases we call address then Z out you give to the PC.

So, Z in Z actually in this case it is equal to nothing but 3200 that is the same way offset you are calculating and adding with the present value of the program counter, you are putting in the PC and you are going to jump, your job is solved. So, there is very similar to any other jump instruction. Only important part is this one that is you have to save the value of your program counter in the memory and data register and the stack pointer address has to be stored into the memory address register, so that the PC can be same and then you write the memory. And then you wait till you do the charm.

(Refer Slide Time: 40:29)



Instruction Execution (Procedure Call/Return)

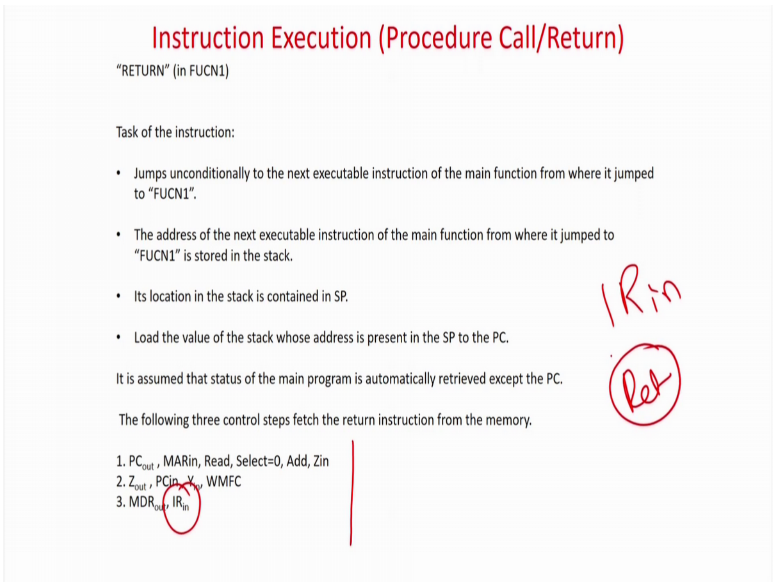
"RETURN" (in FUCN1)

Task of the instruction:

- Jumps unconditionally to the next executable instruction of the main function from where it jumped to "FUCN1".
- The address of the next executable instruction of the main function from where it jumped to "FUCN1" is stored in the stack.
- Its location in the stack is contained in SP.

So, before we end quickly have to look at what is the return, because once we have called you should also return. Return is very important as you have already seen if your condition is false you have to return, function is ending your return. So, return is as important as call. So, basically we will say return. In this case, but return will not specify basically end any address, but very will go basically let this be stack, stack will be pointing to some address, some content may be say 11, because we say jump from memory location 10. So, we will say return. So, return means you have to do nothing no address calculation basically of that shot, just you have to look at the memory of the stack pointer that whatever value is there would have to be loaded to the PC, so that the PC can start executing from that location.

(Refer Slide Time: 41:12)



Instruction Execution (Procedure Call/Return)

"RETURN" (in FUCN1)

Task of the instruction:

- Jumps unconditionally to the next executable instruction of the main function from where it jumped to "FUCN1".
- The address of the next executable instruction of the main function from where it jumped to "FUCN1" is stored in the stack.
- Its location in the stack is contained in SP.
- Load the value of the stack whose address is present in the SP to the PC.

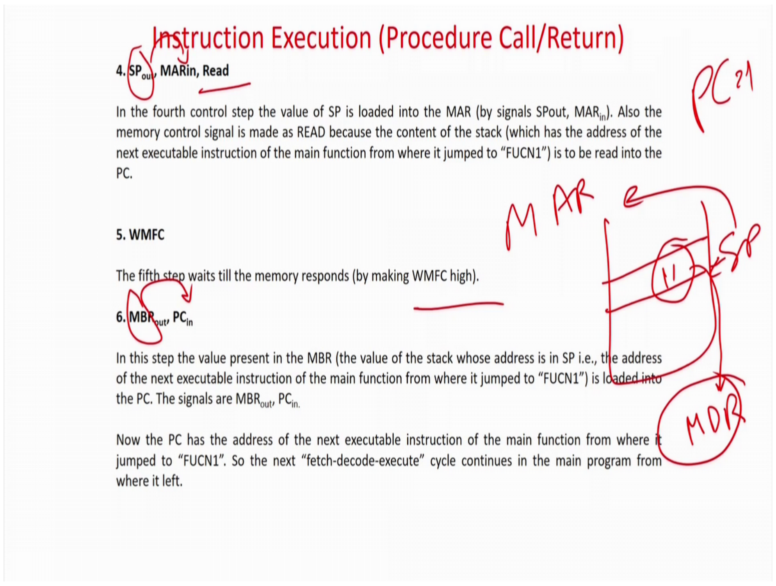
It is assumed that status of the main program is automatically retrieved except the PC.

The following three control steps fetch the return instruction from the memory.

1. PC_{out} , MARin, Read, Select=0, Add, Zin
2. Z_{out} , PCin, Y_{in}, WMFC
3. MDR_{out}, IR_{in}

So, again whenever the return instruction, the first thing will be extremely similar that you have to dump the value of register in R in will have the value of return that value will return instruction will come from there, all others will be similar.

(Refer Slide Time: 41:29)



Instruction Execution (Procedure Call/Return)

4. SP_{out}, MARin, Read

In the fourth control step the value of SP is loaded into the MAR (by signals SPout, MAR_{in}). Also the memory control signal is made as READ because the content of the stack (which has the address of the next executable instruction of the main function from where it jumped to "FUCN1") is to be read into the PC.

5. WMFC

The fifth step waits till the memory responds (by making WMFC high).

6. MBR_{out}, PC_{in}

In this step the value present in the MBR (the value of the stack whose address is in SP i.e., the address of the next executable instruction of the main function from where it jumped to "FUCN1") is loaded into the PC. The signals are MBR_{out}, PC_{in}.

Now the PC has the address of the next executable instruction of the main function from where it jumped to "FUCN1". So the next "fetch-decode-execute" cycle continues in the main program from where it left.

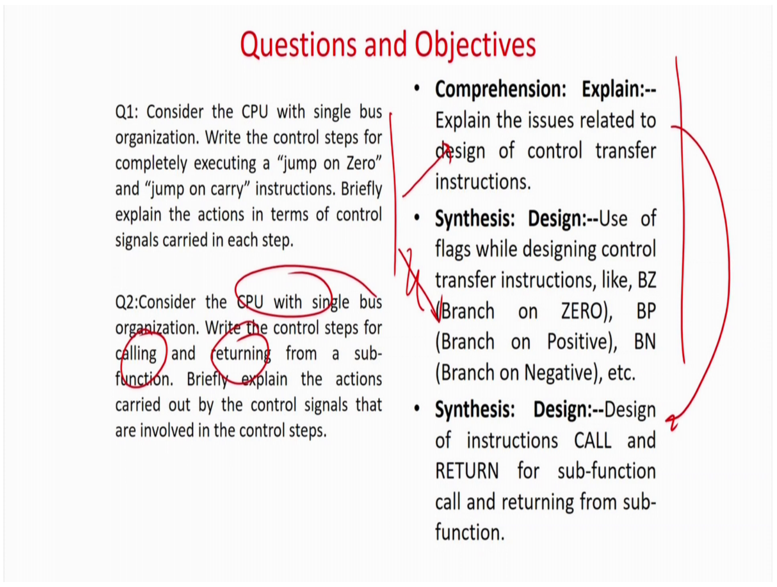
Then what now have to return. So, what I will do, I have to actually take both the value of the program counter, I have known update the value of program counter to the instruction when I have to start. I have stopped at one point in the main program, I have jumped, so again I have to re-go. Re-go start executing main program. So, if you remember, we started in the example we started from eleventh location up to tenth we had finished. The value 11, we have already stored in the stack pointer so in the stack, so basically that is what this stack. So, some stack pointer was pointing at this point and the value we have stored over it. Now, we have to read back the value of eleven and put in the PC.

So, what do you have to do? So, memory address register will have the value of PC now, because this stack pointer only knows where I have actually kept the last value of the PC. So, SP out memory address in study as simple as that because only the stack pointer knows where I have saved my last PC, and you have to make the memory read more because you will be reading about it. You wait for certain amount of time till your job is done.

So, whenever your reading is done then basically your memory data register of the buffer register will actually have the value from here. That means, by fourth and fifth state what you are doing you are actually writing the value of program counter, we are actually taking this stack address to the memory address register. That is you are going to take this address of this location to the memory address register and reading the value of 11, which is the program counter value from which will start executing after returning to the memory data register. After that only one thing remains, you take this value from the memory buffer register or the memory data register and put it in program counter. Now, your program counter will have the value of eleven and you start executing from where (Refer Time: 43:08) as simple as that.

So, call and return are very simple. Just it is a jump instruction, but before making a jump anything you have to store the value of the program counter in a stack pointer. And in return, it is more simpler than that fetch the return instruction, and go to the address of the memory with the stack pointer is pointing load that value in the program counter as simple as that and your job is done.

(Refer Slide Time: 43:33)



Questions and Objectives

Q1: Consider the CPU with single bus organization. Write the control steps for completely executing a "jump on Zero" and "jump on carry" instructions. Briefly explain the actions in terms of control signals carried in each step.

Q2:Consider the CPU with single bus organization. Write the control steps for calling and returning from a sub-function. Briefly explain the actions carried out by the control signals that are involved in the control steps.

- **Comprehension: Explain:--** Explain the issues related to design of control transfer instructions.
- **Synthesis: Design:--**Use of flags while designing control transfer instructions, like, BZ (Branch on ZERO), BP (Branch on Positive), BN (Branch on Negative), etc.
- **Synthesis: Design:--**Design of instructions CALL and RETURN for sub-function call and returning from sub-function.

So, with this actually we have gone through a control all the different types of control signals that are required and the sequence of microinstruction, and the corresponding control signals when you are going for any kind of a jump instruction, both conditional and conditional. And in the end, we have elaborately discussed that how it is actually it happens for jump it is actually a call and a return instruction for the special case also we have seen.

Now, as up for the pedagogical completeness, let us see what are typical questions and how the objective is satisfied. Like for example, in the first question we say consider a CPU with a single bus, execute an instruction jump on zero and jump on carry briefly explain the control steps in each step and show how they control sequence execute. So, of course, this is satisfying the requirement of explaining the issues related to sorry it will be actually satisfying the next two objectives use of flags because you are having conditions and basically on a single bus architecture use of flags while design control transfer.

So, basically this one will actually satisfy these instructions. And of course, this one is also is very important because we have to explain the issues related to control transfer. So, if you are able to design instructions, and the control instructions, and the microinstruction the control signals for jump on conditions like zero, carry, etcetera then you are actually satisfying these objectives.

Second question is that you are taking a single bus architecture briefly explain the actions in terms of control signals that are involved in returning and calling a subroutine. So, in fact, if that is the case then of course, you are satisfying the first and the last objective. Because the last objective is design of instructions, such as call and return from subroutines. So, with these questions after doing this lecture, we expect that you will be able to solve this question and meet these objectives.

So, with this we come to the end of this unit. And now till now what we have seen basically have seen what are the control sequence, what are the microinstructions, what are the control signals generated for all types of instructions like jump, load, store, defined modes of addressing and also in the end jump and control transfer instructions. Next important very important question is that who will generate these signals and how, so that is the heart of the whole control unit design. As we already discussed in the first while starting this module that there are two ways of generating the control signal based on taking the inputs from the registers, flags and all other parts from the control bus is a hardware you need, which will generate this.

This can be either done by a hard coded unit call the that is your hardware module you generate the controlling signals or you can ever some software approach like it is called micro program control, one is the hardware control, one is the micro program control. In the next few sequence of lectures in this unit module, we will be looking at how to generate these control signals both in a software manner, there is a micro program control manner, or as a hard coded hardware is the hardware, you need to generate all these controls. In the next few sequence of lectures, we will see how these control signals are generated.

Thank you.