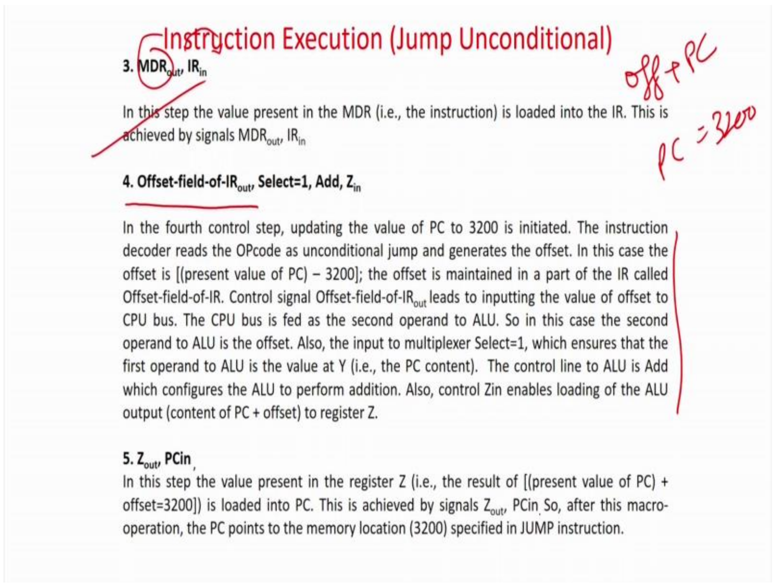
(Refer Slide Time: 19:54)



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

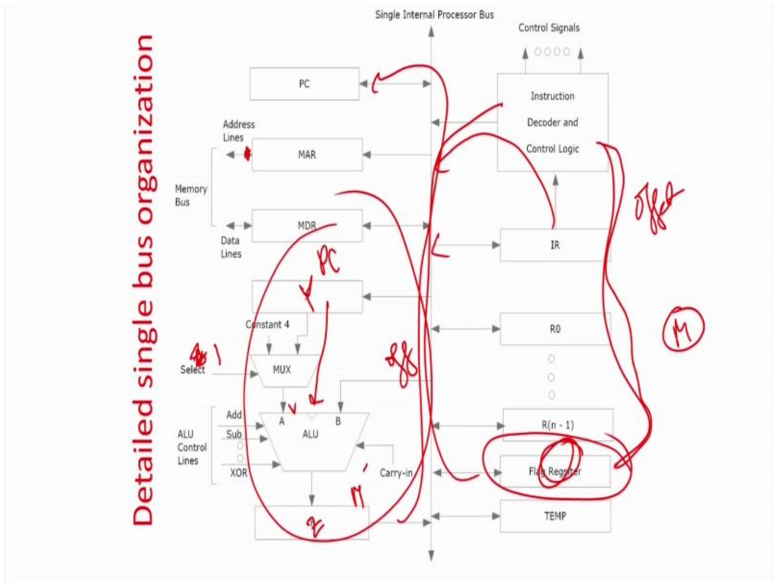
In this step the value present in the register Z (i.e., the result of [(present value of PC) + offset=3200]) is loaded into PC. This is achieved by signals Z_{out}, PCin So, after this macro-operation, the PC points to the memory location (3200) specified in JUMP instruction.

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 3200 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 we 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 $offset + PC$ will actually give the value of 3200, because already we know offset is nothing but present value of $PC - 3200$.

So, if you and the mod 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 the slide I can just clean it. But again I am telling you instead 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 relocatable 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 instruction register has a inbuilt way of generating the offset so it is out, $select = 1, add$ and Z_{in}. $Select = 1$, 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 updated value of next value of PC, and this is coming over here.

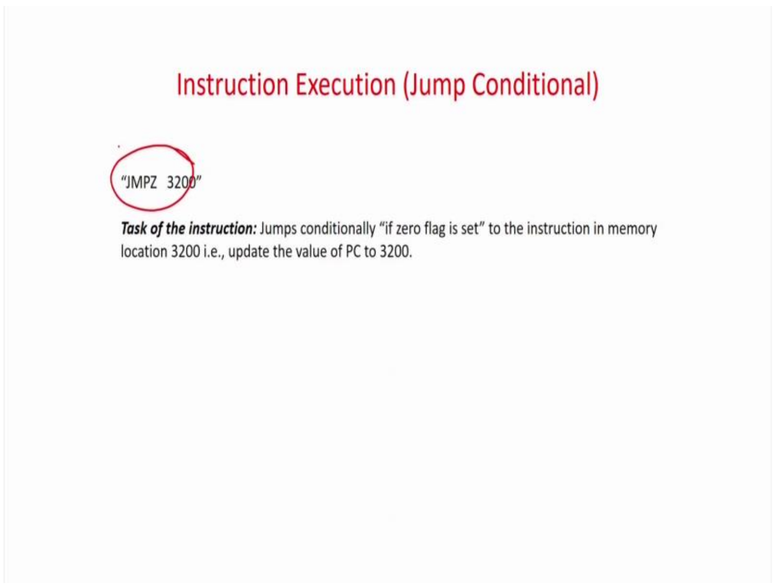
And from the instruction register basically through this, you are actually dumping the value of basically I should write it in this way, you are dumping the value of offset it is not the whole PC register we have already told that we sometimes take a part of the IR register. So, in this case the IR register 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 location of PC and your job is done.

So, basically this 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 a role 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)



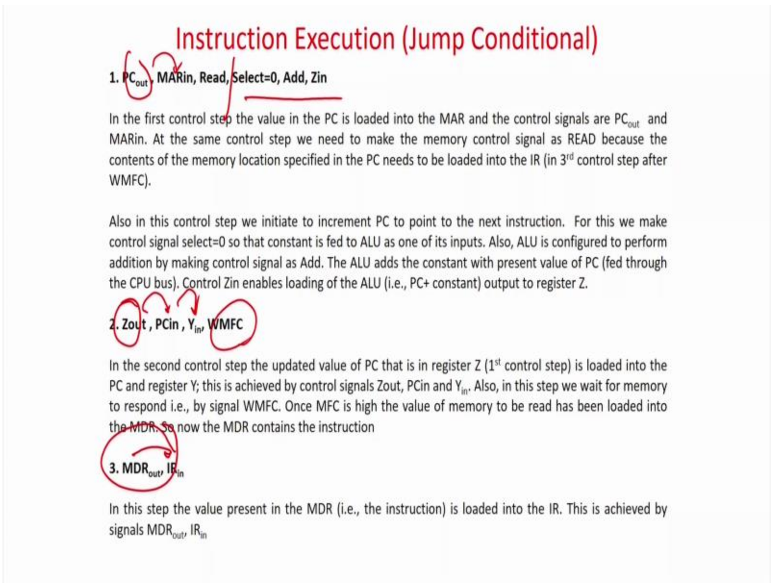
Instruction Execution (Jump Conditional)

"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's 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

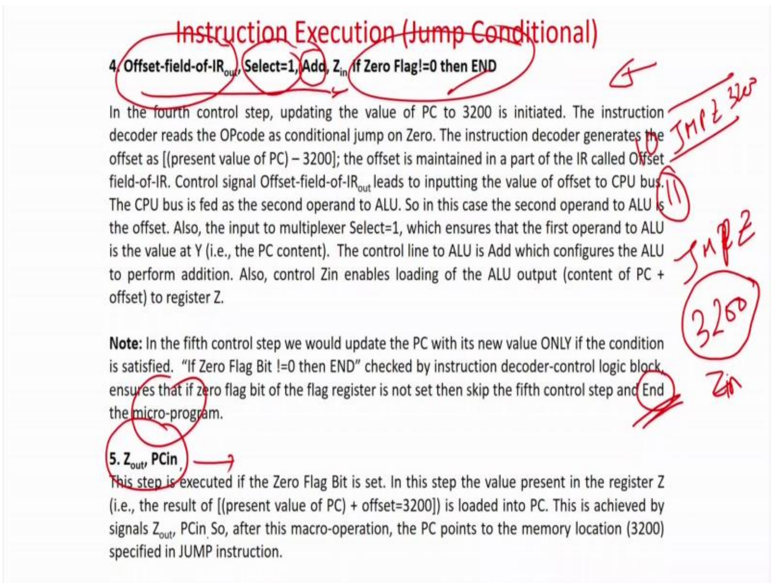
3. MDR_{out}, IR_{in}

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 third instruction from in it because it is third instruction or 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 to the 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 special case of jump instruction we are storing it in Y_{in} that job is done. Next by $WMFC$ 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 its 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 have actually saved 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's a register with some special multifunction. So, whenever you are saying that the opcode fetch that basically it is a jump on 0, so just you look at the flag which corresponds to 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 not zero 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 at 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. Skipping is not a proper word in that sense basically. So, what happens that is instruction register in this 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 inner 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 reads the flag bit and if the flag bit zeroth 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 as we will see just return to the program which 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 execute the next macro instruction, so that's the reset of the PC that is very simple.

But if it is satisfied then what is going to happen, 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 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 Z, but whether we will upload the value of Z to program counter will depend on whether what is the value of this flag register.

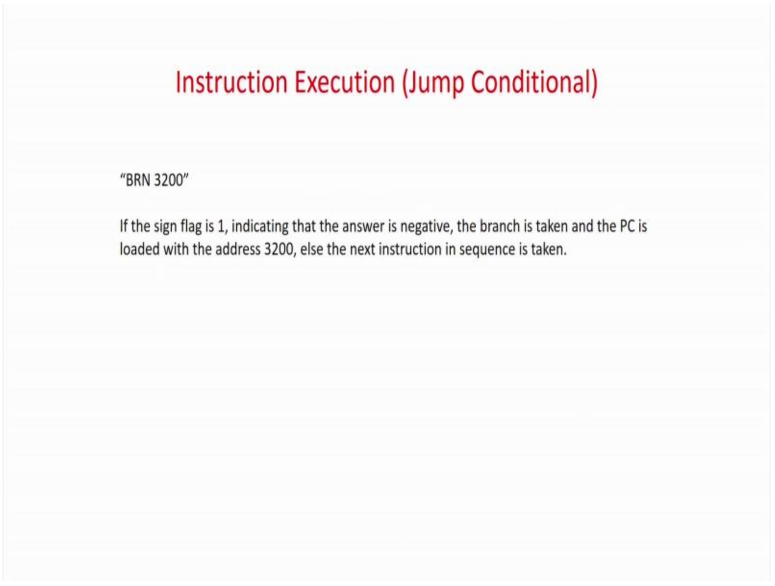
So, if the value of the zeroth flag in the flag register is satisfied that will be checked by the instruction decoder, then it will not do anything. It will just go to the fifth stage; fifth stage will just set Z_{out} and PC. But if it is not set the zero flag is not set what it will do you know just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

Now, in shortly we will see what is call and return that means, just for the time being assume that it 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 and go to the next instruction or you go to PC that means, it says that jump on zero jump on you know let me see the instruction. So, it is a jump on zero $JMP0$. So, jmp0 is the instruction opcode and we are going to say some location 3200.

So, till this point I am ready with this value in Z_{in}, Z_{in} has the value of 3200. 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 memory some memory location we are assuming it 10 memory location have the instruction $JMP0$ 3200. So, this one actually called the sequence of microinstructions, which we are discussing these 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 instructions will 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 a 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 we will be dealing it later. But if it is zero then just we don't have to do anything, do not reset, you just go to the next microinstruction, it is very simple implementation.

(Refer Slide Time: 30:11)



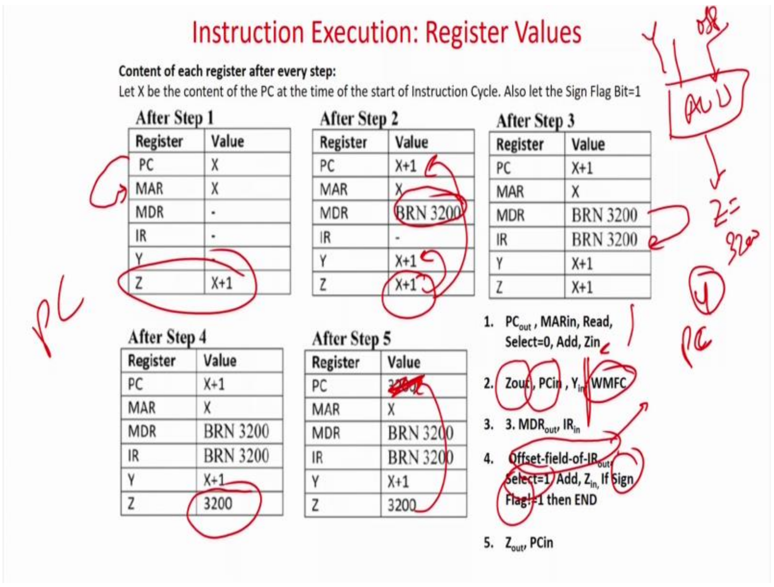
Instruction Execution (Jump Conditional)

"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's a sign bit has been set. Same steps 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 the 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 address in, Read, 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 $select$ 0 means it will be constant, add means the ALU will add and updated value of PC that is $PC\ +\ constant,$ in this example one we are taking an example will be in the register Z. So, $Z\ =\ Z\ +\ 1, PC$ has been incremented. 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 updated value of PC is also given to Y we have already seen Y because we 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 temporary register Y. Then what happens basically then you have to wait for some amount of time in fact, it is better that 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.

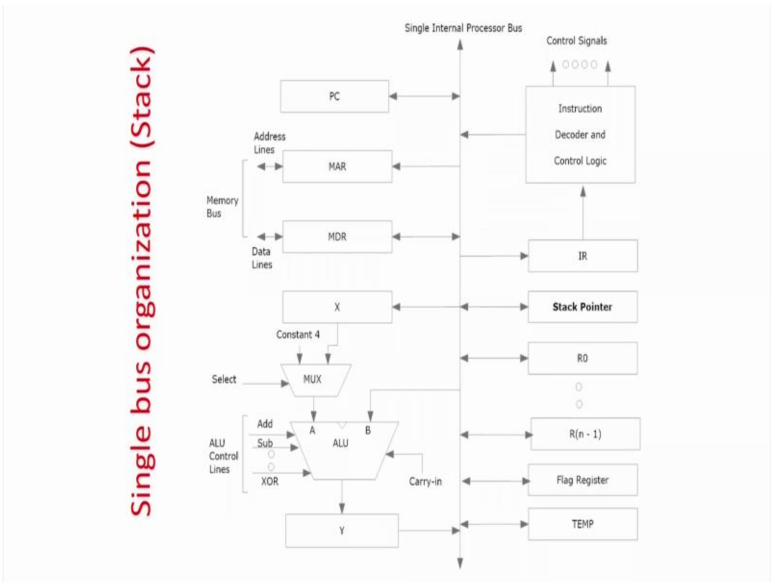
Next, what next as I told you it is very simple the instruction from the memory data register we will go to the instruction register it's a very standard operation. Now, what now four, now four here you are going to see. Now, what we are going to see in four instruction or that is your offset select equal to one over here and add that means offset plus offset is this is offset this is already we know what is the value of offset, and select equal to 1 means we are going to take the value from Y. So, Y is having the value of program counter. 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 have seen ALU is like this, it is coming from Y, because $select = 1$, it is in add mode 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 will 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 the 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 set as required so program counter value will not be 3200, it will be actually the address for the next macro instruction. The first macro instruction 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 macro instruction to be executed.

So, basically so we have taken another 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)

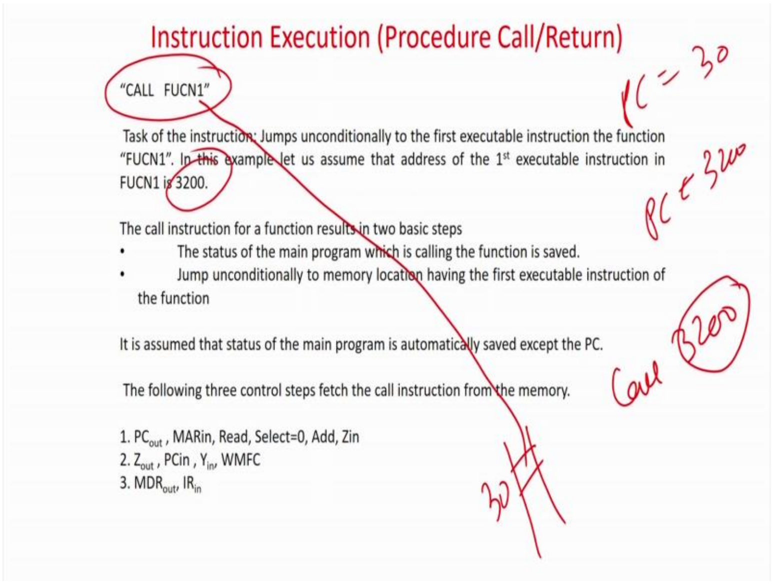


Next, is call and return that is one very important thing. We have also seen that when we were discussing about some kind of a subroutine call when we are looking at interrupts. So, every time we have seen that before we jump to a function or a sub interrupt subroutine service. So, what we do we save the value of the program counter we save the value of all the intermediate registers and all this stuff which we are doing in one function was temporary scratchpad or temporary memory, go and service the interrupt go or finish the procedure 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 is that we save the PC. We save the PC because when 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 in a 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 saved is actually will be defined by a stack pointer. To make it simpler as I told you, we 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 at 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.

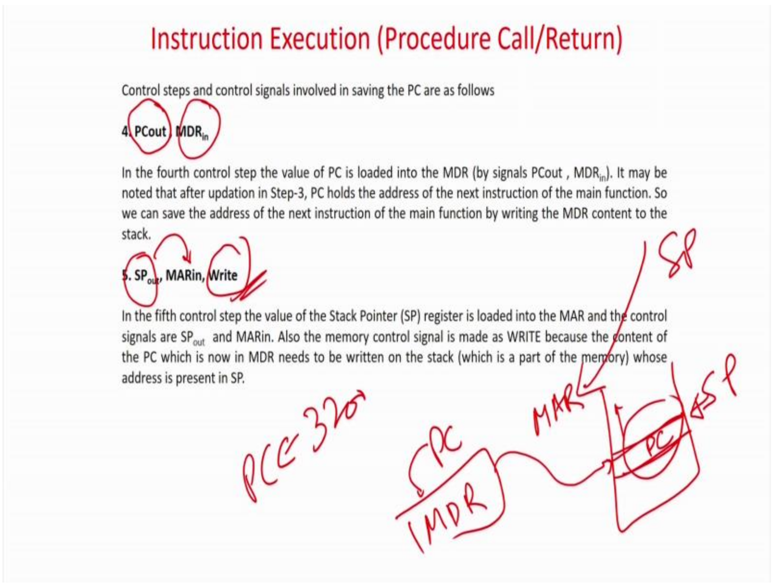
The following three control steps fetch the call instruction from the memory.

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 1 and function 1 memory let us say that 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 it 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 it in 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

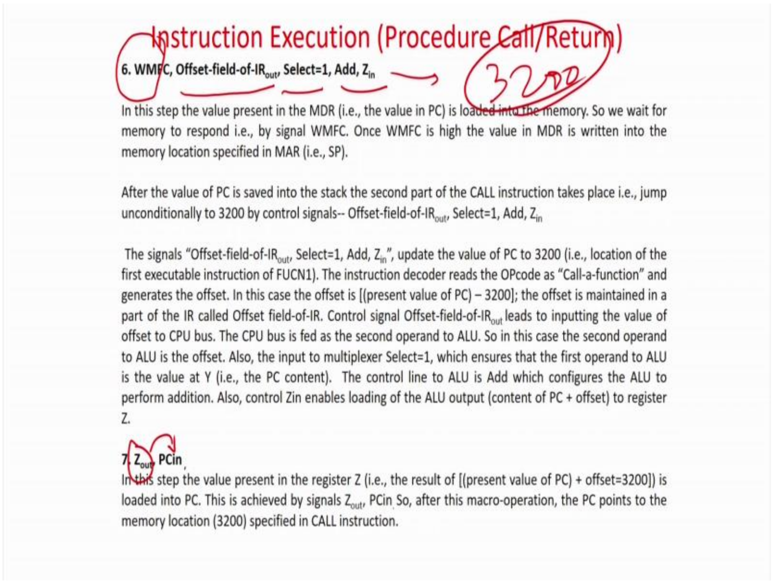
In the fifth control step the value of the Stack Pointer (SP) register is loaded into the MAR and the control signals are SP_{out} and MARin. Also the memory control signal is made as WRITE because the content of the PC which is now in MDR needs to be written on the stack (which is a part of the memory)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

Now, already we know that when we are executing microinstruction number three, program counter has been incremented 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 stage 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 where I have to save it that is very important that is this is your whole memory is there may be from here may be the stack starts.

So, now, what happens here I have to store the value of program counter that is the updated 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 the stack. So, the fourth stage PC_{out} MDR_{in} basically the value of program counter is stored over $MDR,$ 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 stack 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 loaded to the memory address register.

So, once it is done, of course, the value of program counter will be saved into the PC. So, basically this stage SP out, memory data register in, address register in means the program count stack pointer value will be saved to the memory address register, and there we will be saving the value of the program counter as simple as that. Then next part it is so we have saved the value of the program counter now we have to dump 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 the stack pointer.

(Refer Slide Time: 39:26)



Instruction Execution (Procedure Call/Return)

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

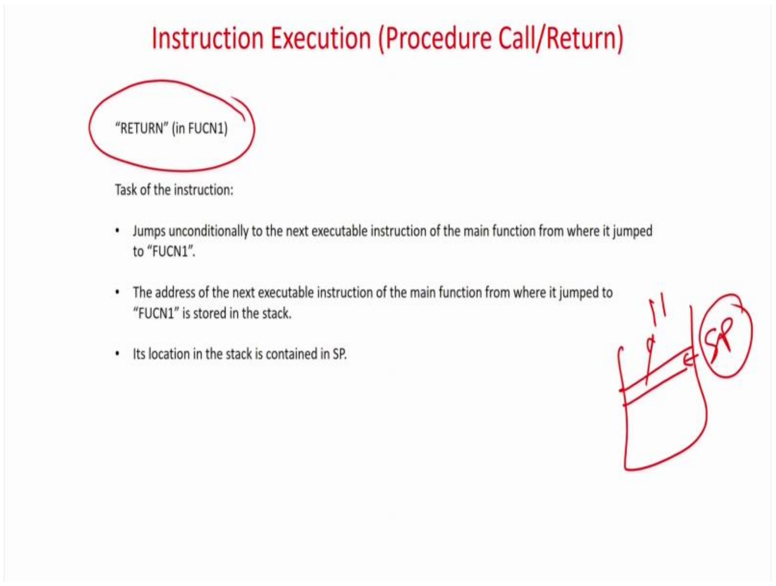
7. Z$_{out}$, PCin,

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 this case it is the call address then Z_{out}you give to the PC_{in}.

So, Z actually in this case 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 jump is solved. So, this is very similar to any other jump instruction. Only important part is this one that is we have to save the value of your program counter in the memory data register and the stack pointer address has to be stored into the memory address register, so that the PC can be saved and then you write the memory. And then you wait till you do the jump.

(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 we have to look at what is the return, because once we have call 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 you have to return. So, return is as important as call. So, basically we will say return. In this case, but return will not specify basically any address, but where it will go basically let this be the 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 sort, just you have to look at the memory or the stack pointer whatever value is there will have to be loaded to the PC, so that the PC can start executing from that location.