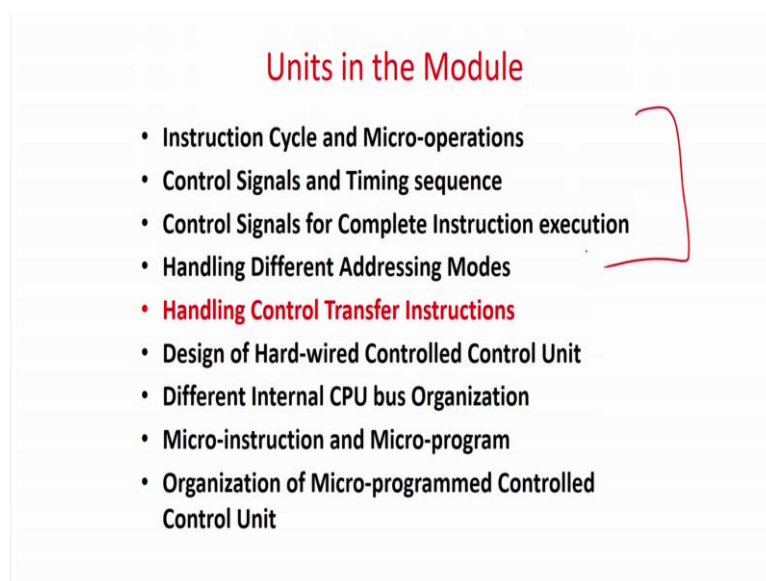


Computer Organization and Architecture: A Pedagogical Aspect
Prof. Jatindra Kr. Deka
Dr. Santosh Biswas
Dr. Arnab Sarkar
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 19
Handling Control Transfer Instructions

Welcome to the fifth 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 transfer 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)

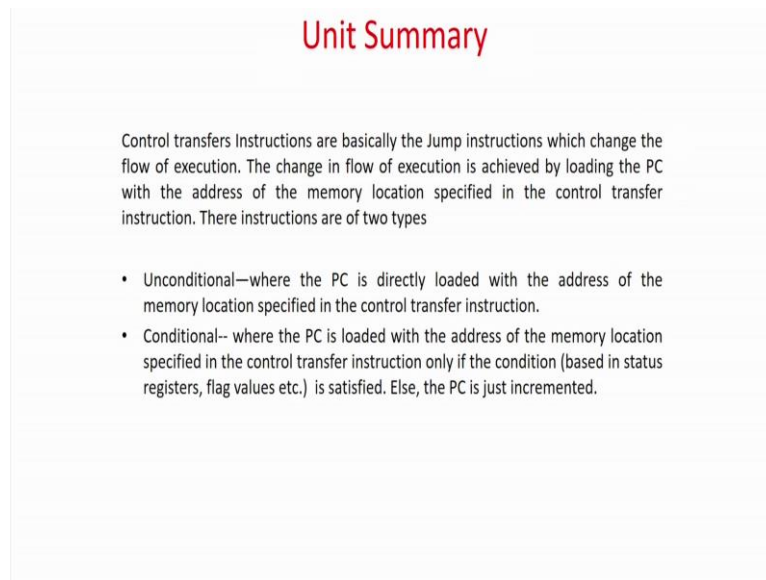


Units in the Module

- 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)

A presentation slide titled "Unit Summary" in red text. The slide contains a paragraph defining control transfer instructions and a bulleted list of their two types: unconditional and conditional.

Unit Summary

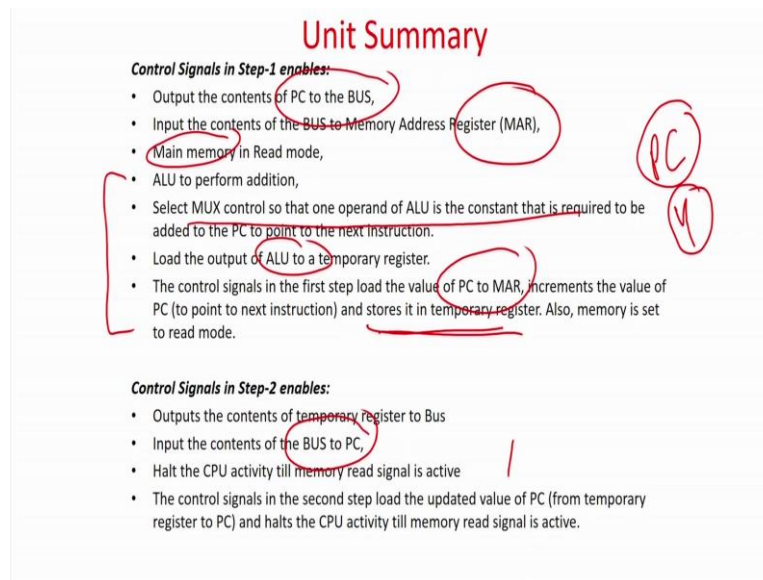
Control transfers Instructions are basically the Jump instructions which change the flow of execution. The change in flow of execution is achieved by loading the PC with the address of the memory location specified in the control transfer instruction. There instructions are of two types

- 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 we shown in the cases in the slide. 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 set, so that way we can differentiate the 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 the steps?

(Refer Slide Time: 02:06)



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.

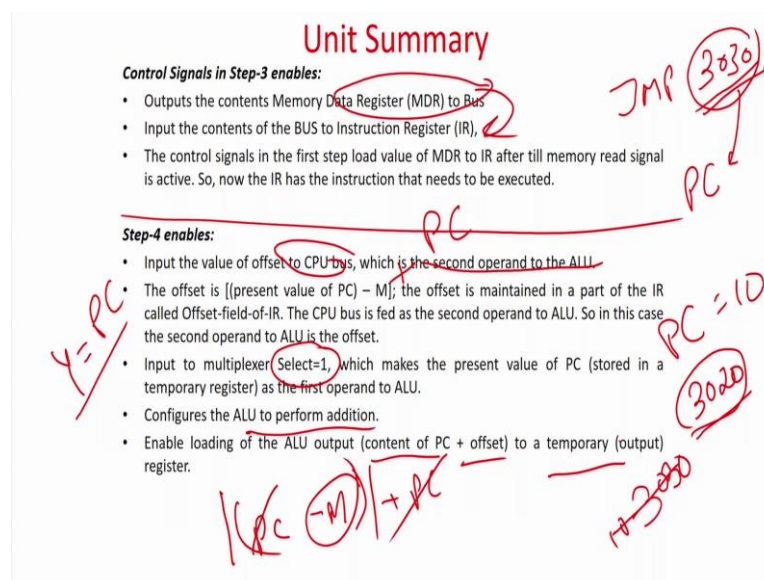
So, basically that is what is the thing and then we say that but here there are slight difference in case of a jump instruction or transfer instructions. In general type of instructions, we take the value of the program counter to the bus, put it in the address register, and then wait for the corresponding 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 step 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 temporary register. We will see that storage of the *PC_{in}* a temporal register is something different which we are doing in transfer instruction. And we will see later see what is the importance of that.

Other things like set the MUX, so that the constant is added that is actually pointing to the next instruction that remains the same. Then basically what else load the output of the ALU to a temporal register that is $PC = PC + 1$, 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 $PC = PC + constant$ that is the next transfer, output the contents of the temporary register to bus it is a very simple. Halt the value of the PC , halt the value till 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)



The third stage is also very similar. The 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 we will store the value of program counter in a temporary register Y . Now, from fourth step onwards 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 3030, then we want to jump to that memory location that is what is the execution part here. So, in this case, what we have to do, we have to somehow load the value of program counter in the next stage to 3030 that is what is the execution to be done. Because when you want to execute some instruction which is the memory location 3030, what we have to do is to simply load the value of PC to 3030 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 we 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 $PC - M$ and you take the general the 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 3030 this $10 - 3030$ the positive you are going to take it. So, it will be 3020. 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 ok.

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 generate it's 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 3020 is value of the offset in this example. Now, what do you do, so basically, so if you are considering single bus architecture, the 3020 value will be an operand.


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's 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 $PC - M$. Then actually you configure the ALU to do addition operation.

So, finally, what we are doing we are doing the content of PC which is in Y plus an offset. So, basically what you are doing you are doing present value of $PC - M$ plus the value of PC . So, you are doing present value of $PC - M$ that value already is the offset plus PC , in fact, you are going to take the mod 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 3030 and which is actually loaded to a temporal register that is your Z . So, in fact, we use slightly a roundabout way means we do not directly take the value of 3030 and dump it into the CPU or 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 3030 and say you are getting loaded at memory location some memory location called 700, so there is always an offset and the 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 than not directly taking the value of 3030 and loading it into the program counter. We just take an 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 3030. So, it actually helps in relocating programs and some kind of other means relative addressing mode 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

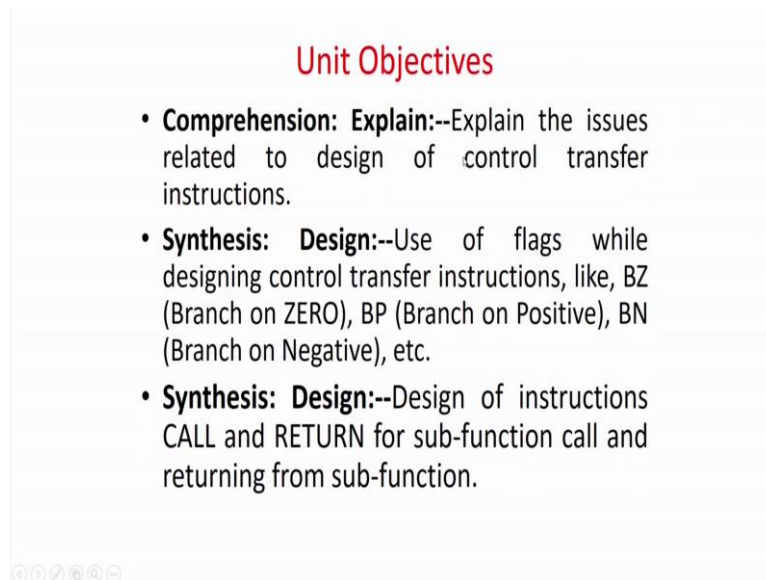


Step-5 enables:

- Value present in the temporary (output) register (i.e., the result of $((\text{present value of } PC) + \text{offset}) - M$) 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, after that is already told that is nothing but value of $PC - M$ this gets cancelled and 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)

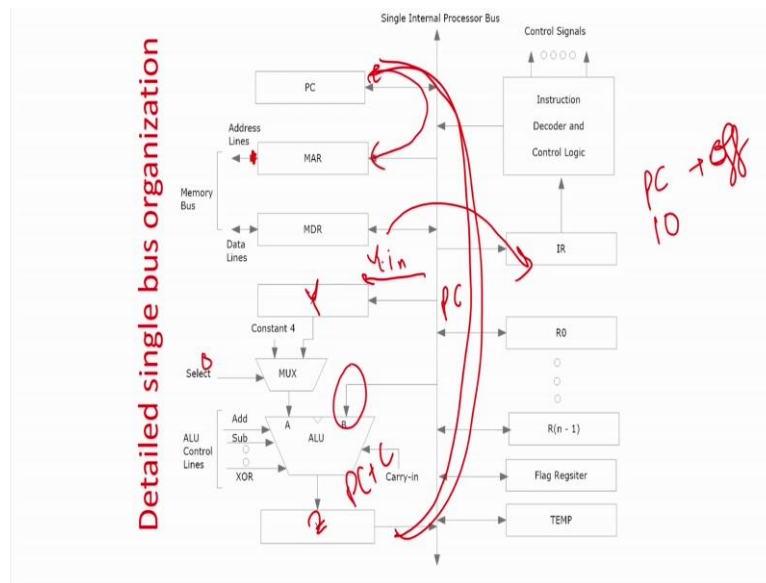


Unit Objectives

- **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 objective is comprehension objective which tells explain the issues 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 as 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 designing 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 you 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.

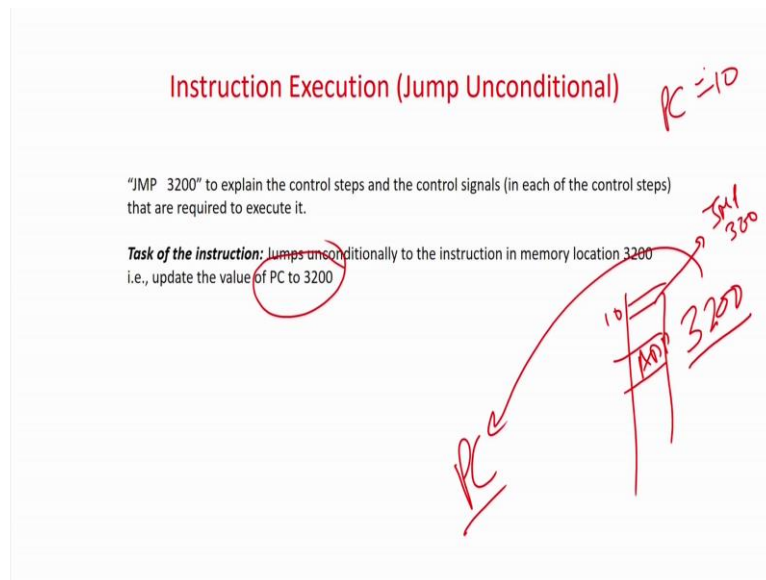
(Refer Slide Time: 10:55)



Now, before we start the unit, let's 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 .

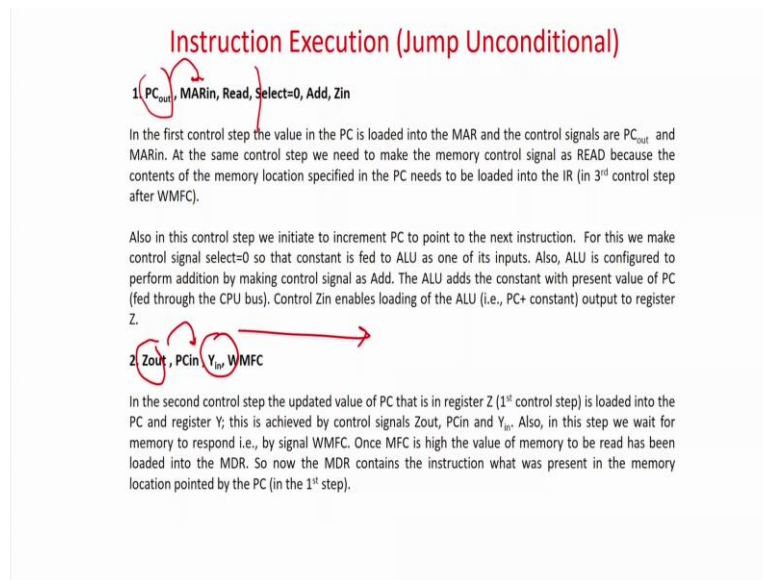
So, just way of nomenclature change let us keep it same for the ease of understanding or continuity that is Y that temporary, register Z is the output of the ALU, this is mux is a constant four, four actually just an example, it can be the size of the instruction. Then the ALU basically, this is the ALU which does all addition, subtraction. Then you have the temporary register, this is your register set from zero to n , 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 register 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, the flag register which is mainly shown over here.

(Refer Slide Time: 12:26)



Ok so, now with this let us start taking some examples of instructions that exactly see what happens. The first instruction will be dealing in 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 program counter will execute from memory location 3200.

(Refer Slide Time: 13:35)



Now, we will see how it happens basically. The first stage is already discussed program counter out memory address in because why already we 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 mode, because you want to read the instruction. Select will be equal to 0 because you are select means that is the ALU mux select. So, it will be adding a constant add and Z_{in} basically the value will be written. So, if we quickly look at it, look at the structure what we are saying that value of program counter that is 10 will be fed to the memory address register. And memory data register after a certain amount of time 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} it will 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 memory address register as well it is available in the bus and 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 $PC + constant$ 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 PC 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 dumped the value of PC plus constant in Z . So, you are making Z_{out} , and you are giving it to 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 instruction 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 $PC = PC + constant$ 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 instruction. But for the time being also we are dumping it to Y , so that is actually called Y_{in} that is special, that $PC = PC + constant$ is dumped 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 why 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 the second block.

So, in that case, what happen this is the present value of $PC + offset$ 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 $PC + offset$, because we have to have add the value of PC not with a 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 remember 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 the 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 load the value of PC_{in} the bus and you can do it add it. Another way is that you store the value of program counter PC_{in} , Y and load the value of offset and that is actually a simpler way of doing it.

Because in the second stage you are anyway dumping the value of PC_{in} the updated value of PC_{in} the program counter register. Same time, hand in hand you just take the value in Y and store it 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 that is 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 approach is, first load the value of program counter in the memory address register that 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 $PC + constant$ and this was our Z basically. So, Z_{in} so the Z will actually have the value of program updated 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 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 IR and 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 in Y .