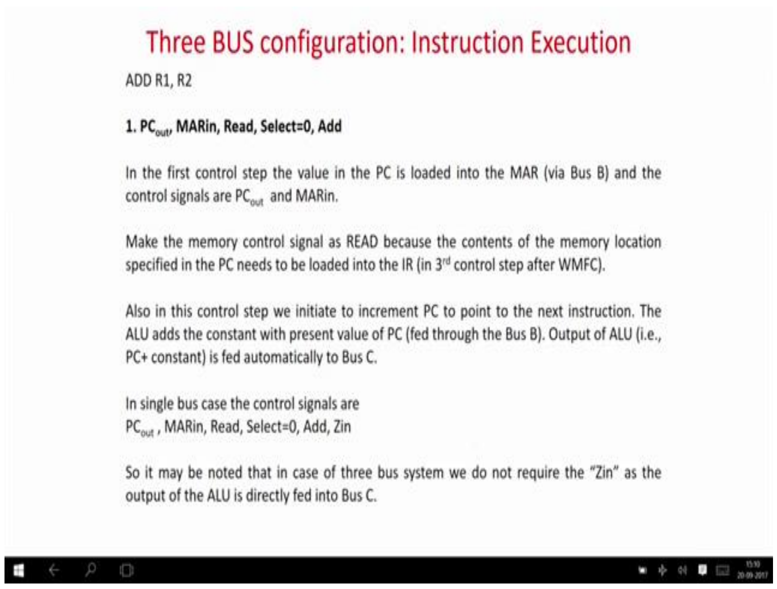
(Refer Slide Time: 36:52)



Three BUS configuration: Instruction Execution

ADD R1, R2

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

In the first control step the value in the PC is loaded into the MAR (via Bus B) and the control signals are PC_{out} and MARin.

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. The ALU adds the constant with present value of PC (fed through the Bus B). Output of ALU (i.e., PC+ constant) is fed automatically to Bus C.

In single bus case the control signals are
PC_{out} , MARin, Read, Select=0, Add, Zin

So it may be noted that in case of three bus system we do not require the "Zin" as the output of the ALU is directly fed into Bus C.

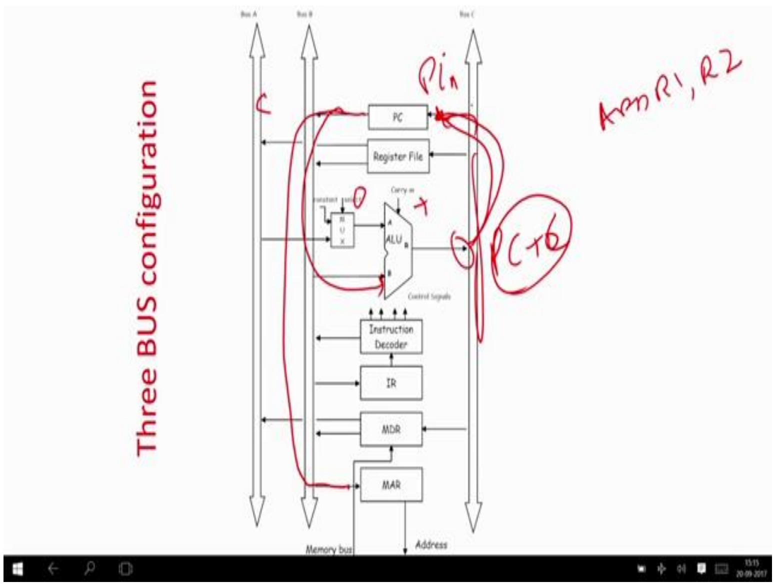
Now, we are going to take two examples; in one example we will show that what are the advantages of having three buses and another case we will show that we do not get so much advantage, if you are considering a multiple bus architecture. Two extremes that means two different instructions we will take and show, but for most of the cases we are always going to have an advantage, because that is very obvious because if you have multiple buses things will go parallelly, but for one or two stray examples we can see where the advantage is not there in fact, you are having more hardware, but still the number of stages are not reducing.

So, the first case we are going to take is add R_1 into R_2. So, what is the thing? So, two variables already available in R_1 and R_2 and then one you have to do it. So, first is you have to fetch the instruction. So, how do you fetch the instruction? Basically program counter output value will go to memory address register in, that is as simple as for single bus architecture then you put the memory in read mode here you select 0; that means, you want to add the constant and increment program counter and add. So, what this step does? Already I have discussed so many times in some previous units, that program counter value is the address where the instruction is there that you put in the memory address register, make the memory to read mode, select is 0 that you have to add the constant and make it to add mode.

But if you look at a single bus architecture, we had another signal that is called Z_{in}. Because the output of program counter plus constant has to store in a separate temporary register which we call it Z or Y and we have to wait till this step is over then only you can write the value to the program counter. But in this case as we have already seen we do not require any kind of a temporary register. So, we do not have anything called Z_{in} or Y_{in} something like that to hold the value.

So, what happens we will go to the figure and see what happens basically? So, the program counter is going to feed to the memory address register, this is the first step which can be done directly, which we are going through let me let me just erase it basically we are doing it through bus B.

(Refer Slide Time: 38:58)



So, PC out is we are going we are basically doing add R_1 and R_2.

So, program counter out is going to get MAR in that is done then we are going to select this equal to 0, so that the constant goes over here and program counter of course, if you see we are also using it to give it to the value here. So, this is equal to PC. So, if you can see the program counter is connected to bus A, program PC value is going to bus B and we are feeding it to memory address register and also we are feeding it to the second component of the ALU.

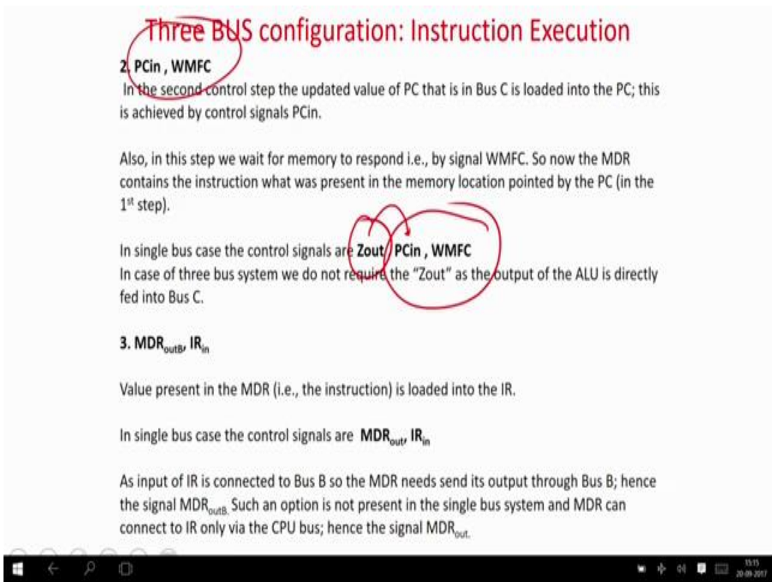
So, therefore, output of the PC is connected to bus B and memory address register is taking the value from bus B. So, the as I was saying that slightly sometimes you have to think that if you are instead of doing this if you are taking the value from bus A that will not be a very good design, because most of the time the program counter value is taken as the input to the memory address register. So, therefore, we will whichever output bus you are dumping the value of program counter, that should be considered as an input to the memory address register.

So, even if I can take the input from here, but we are not doing it, because the PC is writing through bus B; but it can be done if you are sorry it can also be done, but if you have to you have to take the output for program counter through bus A, you have to disconnect this line and you have to disconnect this line.

So, any option you can take. So, as I told you this is a very flexible architecture you can have your own design in this case we are taking this. So, the program counter value is going to come to the memory address register, same time it is going to B and you are taking $select = 0$ memory is in add mode and here is equal to PC plus constant is already there, and you can see and it is already feeding back to the program counter. So, there is no need to store anything.

So, program counter out, you will select 0, memory address register in and at the same amount of time sorry and same amount of time you can say that the addition is being done. So, just you can update the value of PC through bus C. So, in a single step we are able to do it. So, this is; what is the stage PC Program counter out, memory address register in, read the memory, select 0, because PC plus constant and add it, there is no concept of any temporary register to hold the value of $PC = PC + 1$.

(Refer Slide Time: 41:15)



Three BUS configuration: Instruction Execution

2. PCin , WMFC

In the second control step the updated value of PC that is in Bus C is loaded into the PC; this is achieved by control signals PCin.

Also, in this step we wait for memory to respond i.e., by signal WMFC. So now the MDR contains the instruction what was present in the memory location pointed by the PC (in the 1st step).

In single bus case the control signals are **Zout, PCin , WMFC**
In case of three bus system we do not require the "Zout" as the output of the ALU is directly fed into Bus C.

3. MDR_{outB}, IR_{in}

Value present in the MDR (i.e., the instruction) is loaded into the IR.

In single bus case the control signals are **MDR_{out}, IR_{in}**

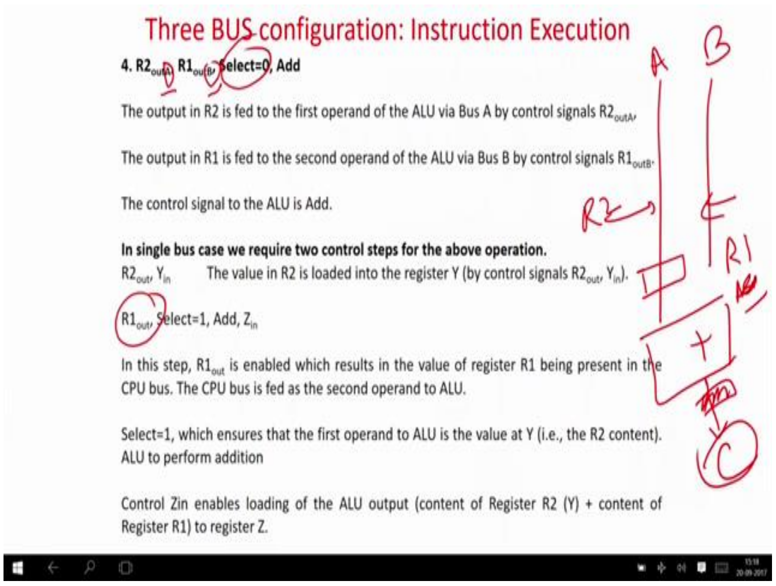
As input of IR is connected to Bus B so the MDR needs send its output through Bus B; hence the signal MDR_{outB}. Such an option is not present in the single bus system and MDR can connect to IR only via the CPU bus; hence the signal MDR_{out}.

Next stage is simple you have to make PC in read the value of output to the PC, and wait till the memory response. And if you look at the single bus architecture, these things were very similar you have to read the value of PC, WMFC, but also you had a sin single instruction called Z_{out}. Z_{out} will go to the value of PC in, but here PC in will not require any Z_{out}, because already bus C is carrying the value of the new value of the program counter. So, again I will look back the figure and then it will be clear.

So, in this case what happens we are not having any temporary variable out which is going to the PC in, here directly you just make PC in PC in. So, whatever the value is in C will directly go over here. So, one control instruction is saved in this case and of course, WFMC same in both the case, because you want to read from some memory you have given the value to the memory address register. So, sometime you have to wait till the value comes from the memory bus and it will be dumped to the memory address register. That is the instruction add R_1 and R_2 will come to memory data register after a wait of some amount of time.

Then next what is going to happen? The memory register data will be out and it will go to the instruction register and this step will be very very similar to the single bus architecture. So, this is your point, memory data register out, instruction register in it will be also similar for a single bus architecture, because we are not handling as I told you we have a single memory and we are not having handling any kind of multiple instructions together.

(Refer Slide Time: 42:47)



Three BUS configuration: Instruction Execution

4. R2$_{out(A)}$, R1$_{out(B)}$, Select=0, Add

The output in R2 is fed to the first operand of the ALU via Bus A by control signals R2$_{outA}$,

The output in R1 is fed to the second operand of the ALU via Bus B by control signals R1$_{outB}$.

The control signal to the ALU is Add.

In single bus case we require two control steps for the above operation.

R2$_{out}$, Y$_{in}$ The value in R2 is loaded into the register Y (by control signals R2$_{out}$, Y$_{in}$).

R1$_{out}$, Select=1, Add, Z$_{in}$

In this step, R1$_{out}$ is enabled which results in the value of register R1 being present in the CPU bus. The CPU bus is fed as the second operand to ALU.

Select=1, which ensures that the first operand to ALU is the value at Y (i.e., the R2 content). ALU to perform addition

Control Zin enables loading of the ALU output (content of Register R2 (Y) + content of Register R1) to register Z.

Now, let us see we have to now do the real addition? So, if you look at it. So, what is the addition? So, we are assuming that the two registers R_1 and R_2 already has the value, and the instruction that is $R_1\ add\ R_1, R_2$ is going to the instruction register from instruction register it goes to the instruction decoder decoding has been done, and it will have to generate the signals. So, what it has to generate? Register value R_2, register value R_1, they have to be dumped to two different buses they have to be added and the value will be out. So, very simple R_{2out}, R_{1outB}.

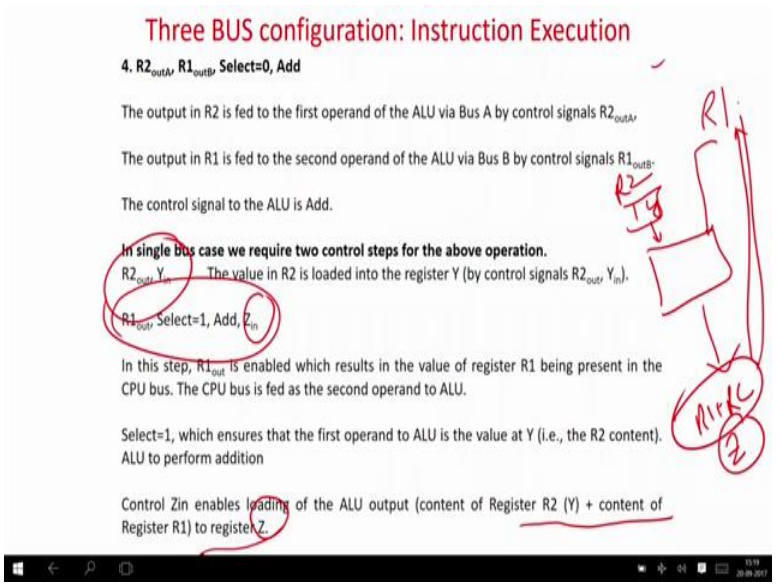
So, now that means, what as I told you here we have to observe that basically here the signals are R_{2outA} and R_{2outB}, unlike in a single bus architecture you had something called R_{out} R_{1out}, R_{2out}, R_{3out}, here we have R_{outA} and R_{outB} now why they are different because the register R_2 can give the value to two different ports port A and port B there are two different buses.

In this case R_2 is giving the value at A and R_2 is giving the sorry R_2 is giving the value at A and R_2 at A and R_1 at B; that means, there are two wires that is A and B this is two buses basically R_2 is giving at A and R_1 is giving at B. So, simultaneously two datas are available there and these are basically if you look at the figure they are going to the two different ports of the ALU and you make $select = 0$ that is very obvious, because in this case you are not taking PC increment, but you are taking the operand 0 and you have a add.

So, you have to very nicely you have to observe that we do not have to store the output; this is directly it will directly go to bus C. So, you need not have to store any temporary variable over here because we have directly two buses available which can give input to the ALU, and also the output that is equal to R_1 plus R_2 can directly go and feed bus C. So, you also do not require any kind of a temporary registers over here. So, those things are not required.

Now, so, this is the same thing we are doing, we are taking two wires A and B, and dumping the values of R_2 and R_1 and making the add operation. Now if you look at a single bus architecture, it would be slightly more complicated.

(Refer Slide Time: 45:10)



Three BUS configuration: Instruction Execution

4. $R2_{outA}$, $R1_{outB}$, Select=0, Add

The output in R2 is fed to the first operand of the ALU via Bus A by control signals $R2_{outA}$,

The output in R1 is fed to the second operand of the ALU via Bus B by control signals $R1_{outB}$.

The control signal to the ALU is Add.

In single bus case we require two control steps for the above operation.
$R2_{out}$, Y_{in} The value in R2 is loaded into the register Y (by control signals $R2_{out}$, Y_{in}).

$R1_{out}$, Select=1, Add, Z_{in}

In this step, $R1_{out}$ is enabled which results in the value of register R1 being present in the CPU bus. The CPU bus is fed as the second operand to ALU.

Select=1, which ensures that the first operand to ALU is the value at Y (i.e., the R2 content). ALU to perform addition

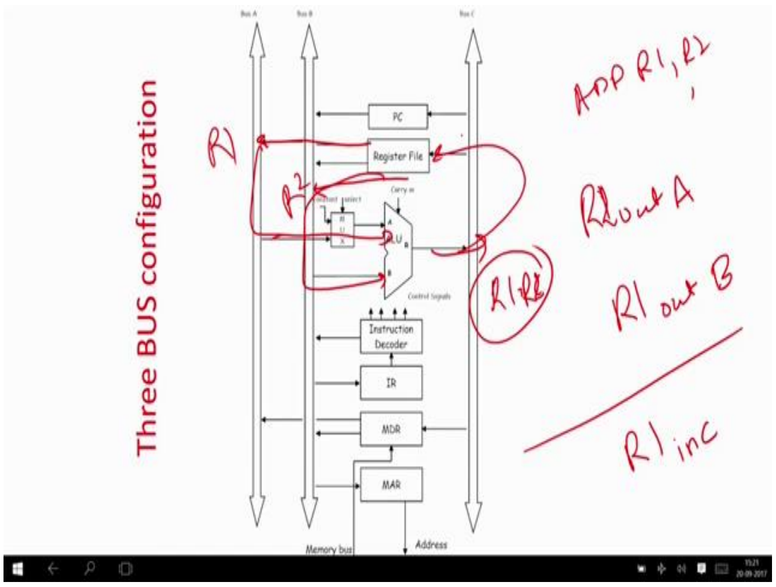
Control Zin enables loading of the ALU output (content of Register R2 (Y) + content of Register R1) to register Z.

So, what we have to do? We have to take R_{2out} and Y_{in}. So, what was that? We have to take the value of R_2 and store in a temporary register that is actually equal to Y if you remember it is a temporary register Y we are now having the value of R_2 over here one step is gone.

Next step you are connecting R_1 here directly to the bus then here you are going to have the answer that is $R_1 + R_2$. But then again we require to store this in a temporary register and in third stage only you can write back the value using the bus this is the single bus. So, first one you will store the value, then the second stage you do the add and you write in the temporary variable and finally, only after this one finally, you have to make sorry finally, basically the content of R_2 and content of will be stored in the register Z_{in}, and then this Z this temporary register Z, the third stage can only dump the value to wherever required.

But in this case if you look at there is nothing called such type of any registers, you dump the value of v and R_1in bus A and bus B and just add it the value will be available in the memory register C. So, again let me just again look back the figure. So, what I am doing. So, may be consider this as register R_1. So, one register R_1 will dump the value here that is R_{1outA}. So, that is may be saying R_{1outA} or R_{1outB}; that means, in which bus they are going to give the output.

(Refer Slide Time: 46:25)

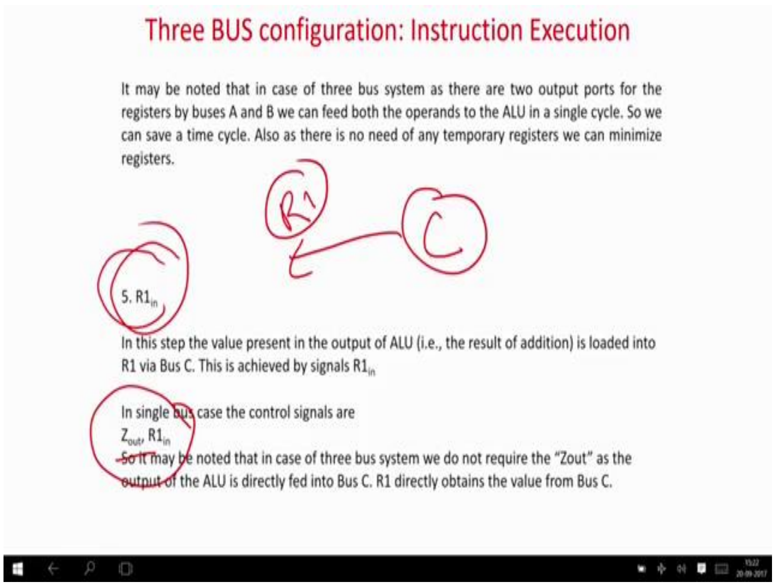


So, we can also have R_{1outA}, R_{1outB}; that means, in that case both the A and B will have the value from the register same register that also can be done, but in this case R_1 out A may be the instruction another will dump the value of here. So, in our example I think it was two anything is ok. So, R_{1outB}. So, in this case 2 and then this is going to have the value of register R_1 this is going to have the value R_2 and if you look at. So, this one is going over here, B is going over here both the operands are here and the output $R_1 + R_2$ is likely available the output.

So, now the instruction was add R_1 R_2. So, very simple the value is already available at C, just write the next control instruction will be that is will be registers R_1, and you have to make in port C or in that is this value will be directly going to register R_1 in this port thus after this one more inst control signal required will be R_{1inC} and basically you are done.

So, if you look at it ok. So, this was the case. So, we have selected addition is being done, but in case of single bus more number of stages as we have seen more in for more number of intermediate registers and finally, you just put R_1 in because this is a single R_1.

(Refer Slide Time: 47:47)



Three BUS configuration: Instruction Execution

It may be noted that in case of three bus system as there are two output ports for the registers by buses A and B we can feed both the operands to the ALU in a single cycle. So we can save a time cycle. Also as there is no need of any temporary registers we can minimize registers.

5. $R1_{in}$

In this step the value present in the output of ALU (i.e., the result of addition) is loaded into R1 via Bus C. This is achieved by signals $R1_{in}$

In single bus case the control signals are

Z_{out}, $R1_{in}$

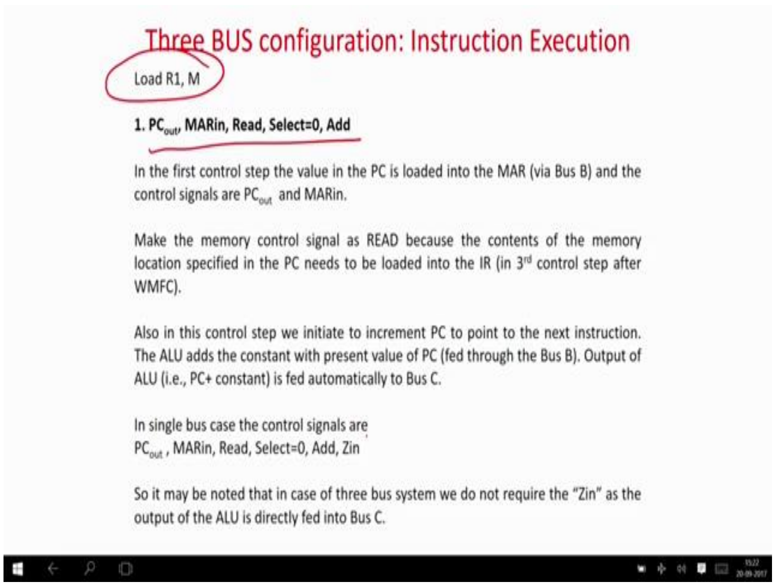
~~So it~~ may be noted that in case of three bus system we do not require the "Zout" as the ~~output~~ of the ALU is directly fed into Bus C. R1 directly obtains the value from Bus C.

So, R_{1in}; that means, the value of bus C will go to basically your register R_1.

But in case of a single bus architecture, you will be Z_{out} plus R_{in}; that means, now the intermediate value was stored in Z. So, it has to be dumped to R_{in}. In this case the value is available in already bus C and it can be given. So, if you study here we will save actually one control step for addition and of course, the overall steps will be reduced at the same time we do not require any kind of internal CPU registers explicitly some temporary registers are avoided and so the less number of control signals are generated.

So, this instruction shows a very explicit advantage of using a multiple bus architecture for most of the designs you will for most of the instructions we will find out there are advantages. You can easily try out on your own, but what I am going to show you now is one case where the advantages are not there, that is same number of instruction time or timelines will be required of course, you will save in the number of intermediate registers that of course, will be there, but I will show you where the timeline is similar.

(Refer Slide Time: 48:52)



Three BUS configuration: Instruction Execution

Load R1, M

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

In the first control step the value in the PC is loaded into the MAR (via Bus B) and the control signals are PC_{out} and MARin.

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).

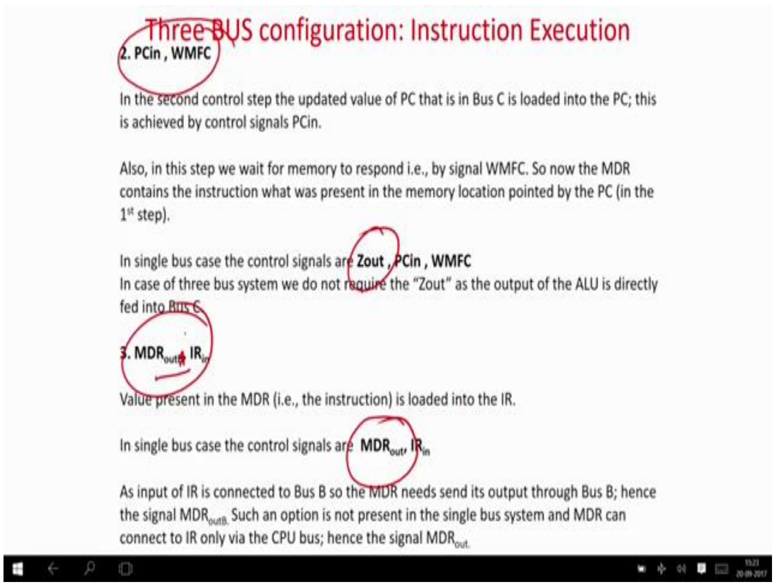
Also in this control step we initiate to increment PC to point to the next instruction. The ALU adds the constant with present value of PC (fed through the Bus B). Output of ALU (i.e., PC+ constant) is fed automatically to Bus C.

In single bus case the control signals are
PC_{out} , MARin, Read, Select=0, Add, Zin

So it may be noted that in case of three bus system we do not require the "Zin" as the output of the ALU is directly fed into Bus C.

So, what is the instruction? The instruction says that load some value from memory location M to R_1. So, the first stage is very similar program counter out memory register in, read, select 0 and add and of course, as again I told you there is nothing called Z_{in} or something because in this case program counter values are available directly in the bus. So, that is the difference already we have seen this one is avoided already we have discussed because as I told you in some previous lecture, that instruction fetch concept is similar for all the instructions. So, that will be very similar.

(Refer Slide Time: 49:31)



Three BUS configuration: Instruction Execution

2. PCin , WMFC

In the second control step the updated value of PC that is in Bus C is loaded into the PC; this is achieved by control signals PCin.

Also, in this step we wait for memory to respond i.e., by signal WMFC. So now the MDR contains the instruction what was present in the memory location pointed by the PC (in the 1st step).

In single bus case the control signals are **Zout , PCin , WMFC**
In case of three bus system we do not require the "Zout" as the output of the ALU is directly fed into Bus C.

3. MDR_{outB}, IR_{in}

Value present in the MDR (i.e., the instruction) is loaded into the IR.

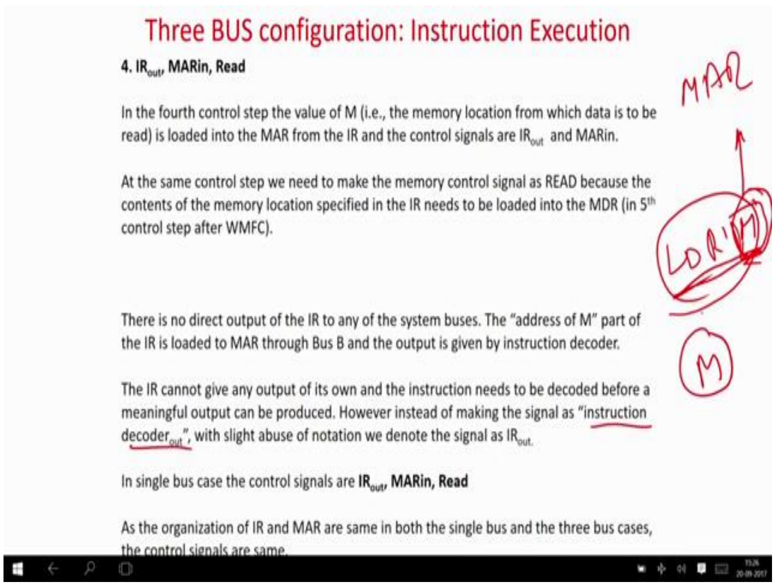
In single bus case the control signals are **MDR_{out}, IR_{in}**

As input of IR is connected to Bus B so the MDR needs send its output through Bus B; hence the signal MDR_{outB}. Such an option is not present in the single bus system and MDR can connect to IR only via the CPU bus; hence the signal MDR_{out}.

Similarly, you have to wait for program count in and wait till the data or the instruction in this case comes to the memory data register. Of course, there will be no Z_{out} already discussed and finally, the memory data outB will go to R_{in} there is a slight difference here. So, generally in the when you are having a single bus architecture we have memory data out, but here we are having memory data outB that is the memory data should be giving the output to memory bus B.

So, in this case as you have multiple buses. So, we generally make it explicit, otherwise because in our architecture we could have also avoided it because out memory data register if you look at is just connected to basically single output.

(Refer Slide Time: 50:03)



Three BUS configuration: Instruction Execution

4. IR_{out}, MARin, Read

In the fourth control step the value of M (i.e., the memory location from which data is to be read) is loaded into the MAR from the IR and the control signals are IR_{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 IR needs to be loaded into the MDR (in 5th control step after WMFC).

There is no direct output of the IR to any of the system buses. The "address of M" part of the IR is loaded to MAR through Bus B and the output is given by instruction decoder.

The IR cannot give any output of its own and the instruction needs to be decoded before a meaningful output can be produced. However instead of making the signal as "instruction decoder$_{out}$", with slight abuse of notation we denote the signal as IR_{out}.

In single bus case the control signals are **IR_{out}, MARin, Read**

As the organization of IR and MAR are same in both the single bus and the three bus cases, the control signals are same.

So, the memory data register we are connecting to both A and B. So, in this case you have to explicitly specify that where you have to go connect basically because we have to go to instruction register and instruction register is basically connected to B. So, in this case we are saying MDR_{outB}.

So, as I told you we can it is up to you, you can also have slight changes also you can also have something like then it will be slightly different you can take the program counter to B instruction register will come over here and then MDR can load over here. So, slight flexibilities you can do. So, in this architecture as you have seen, so the instruction register is connected to bus B. So, you have to tell MDR_{outB}. But in a single bus architecture MDR_{out} means MDR_{out} right. So, we were here. So, basically now v is going to the instruction register.

Now, now it is done. So, the now the instruction that is a $load\ R_1, M$ that has come to the instruction register now things will basically be different from the previous instruction we have considered. Now what? Now basically your instruction decoder has to tell the address of M and it will has again has to go to the memory address register. That is why we are saying that IR out is going to the memory address register in, now it will read the data.

Here one thing we have to know that basically instruction register actually contains the whole thing, but with slight abuse of notation, we are just saying that the instruction decoder is going basically we are saying that IR_{out} the whole IR means the whole thing, but actually want to just look at this M. So, the instruction register decoder basically gives this value to the memory address register this part, but we are not explicitly writing it over here, because then you should have written something like instruction decoder out and that also for this part.

So, that is that can be very easily implemented, but for ease of notion similar type of notation I am keeping the things simple, we just write an instruction register out and memory address register in. In fact, it is basically instruction decoder out which is actually considers only the M part it. Anyway with this notation simplicity let us take it through. So, the instruction register out that is the value of M will go to the memory address register in and then you have to read the memory and you have to wait for some amount of time.

In case of a basically a single bus architecture, this would remain the same instruction register out memory in and read. Basically as I again emphasize here because we are having a single memory system and at the same amount and then and at the same time basically you are also not having multiple memories, and a single instruction execution at a time. So, when you are taking a data from the memory or instruction from the memory, this type of instructions the control signals will be similar both for three bus as well as a single bus architecture right.

So, in this case the value will be read from the instruction register to the memory address register.