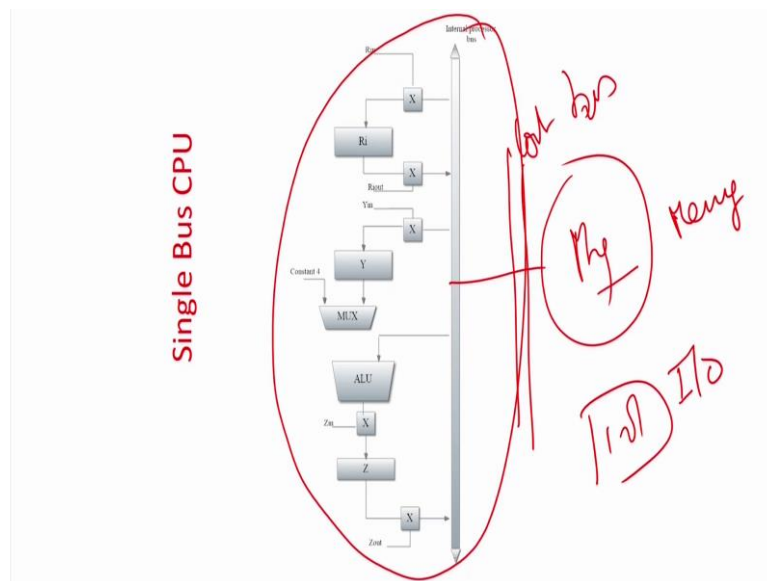


As I told you for example, if I am using this mouse then when I am making a mouse click then your control signal will be read from the control bus by the CPU, it will find out that the mouse click is there then we will it will give command for display. So, whenever the I/O device is involved, memory device is involved, which is out of the CPU then the control bus comes into picture which is taking signals in and out from the control unit.

(Refer Slide Time: 20:34)

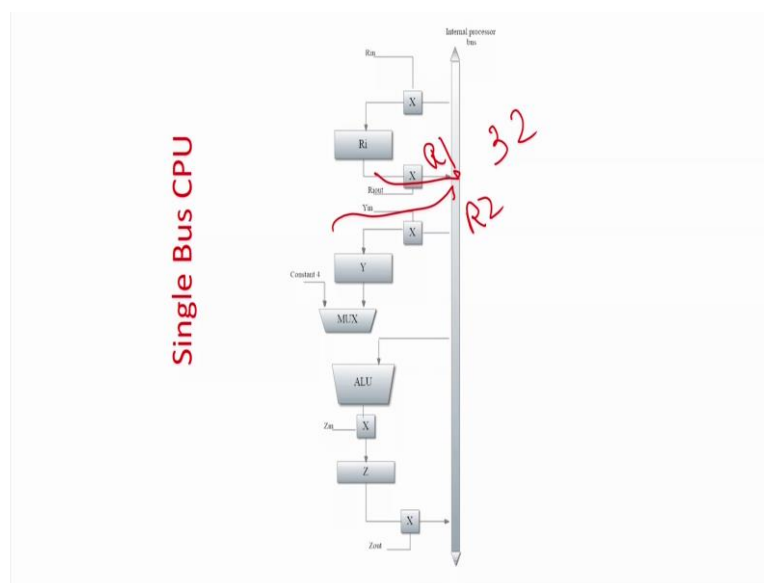


Now, very important thing that is we are going to look at what is a basic architecture for a single unit bus. So, let me zoom it. So, if you look at it, it is basically again let me escape. So, if you look at in a broad picture, so this is a single bus. So, in one part of the bus this side, you can have your, you can assume that there will be an internal bus, there will be some control buses etcetera, there will be your memory, there will be your I/O. So, all these devices will be there and it is an internal and of course, there can be some control bus and several other buses which we are at present we are not talking about. So, internal control bus means this is this part is basically nothing but your CPU.

And this part is your memory, these part is your I/O devices, this part is your memory devices and in fact this is a control bus to for the synchronization. Now, what we will do now initially we look into details on the internal CPU bus because for the external devices when you will be covering entire modules on I/O, there will be entire modules on memory, so we will be handling that in details.

So, for example, for the time being let us just look at the details of the internal bus. So, there are some registers R_1 to R_{32} , R_{64} how many registers you have. So, if you want to take from any input from the register from the internal bus, then what actually you have to do you have to make R_{in} enable that is $R_{in} = 1$. If R_{in} is $= 1$, whatever data is available in the internal processor bus will be fed to R_1 . If it is $R_1 R_2$ we have just drawn it in a single R_i , but in fact if there are 32 you have to replicate this part in 32 ways. But one very important thing is that so like for example, if I want to get the value of output of R_i to this processor bus, I have to make $R_{iout} = 1$.

(Refer Slide Time: 22:17)



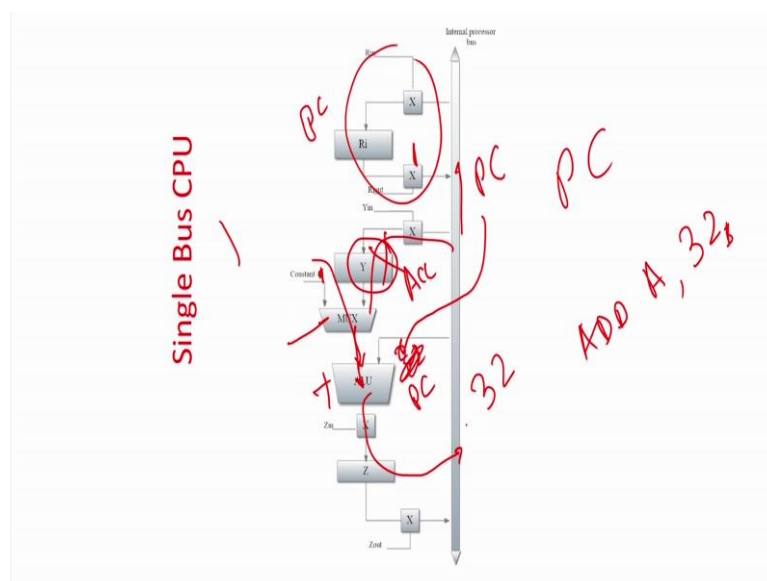
So, what happens see if for example, I have got the value 32 in the bus. Now, what I want to do I want to read this 32 into R_1 , and R_2 and R_3 . So, what you have to do R_{in} one for R_1 has to be made 1, R_{in1} for R_2 has to be made 1, and R_{in2} for R_3 has to be made 1. So, in this case all the registers will be enabled in a read mode. So, 32 will go to R_1 , R_2 and R_3 this is fine.

But we have to be very, very careful that R_{iout} cannot be more than one for any block which is giving output with register. For example, if I say that somehow I make R_{iout} that is $R_{1out} = 1$, and $R_{2out} = 1$. What will happen the data from register R_1 will also go to the output and somehow in this case some R_1 will also go to the output R_2 will also go to the output, there will be a contention so that we cannot have.

So, while giving any output to the control unit sorry what output to the internal CPU bus, we have to be very, very careful that only one register or one ALU or one memory buffer register

etcetera is loading into the internal bus. Multiple parties cannot output at a single go in the internal CPU bus that is very, very important. Now, who takes care the control unit, because the control unit will generate the signals $R_{1in} = 1$, $R_{2out} = 1$ something like that.

Now again if I zoom this next part of it. So, you can see that is basically second part is an ALU. So, either you can get the value from Y , so that is means whatever this is an input from the control bus sorry it is from the internal bus where you can get the data values. So, either you can get the data value Y_{in} , so it is a multiplexer to the ALU. So, you can get one as I told you ALU basically does all the mathematical and logical operation. So, there are two operands for this. So, one operand basically comes from this CPU bus, and the operand basically Y is the register, so sometimes you may have to hold the value temporarily.



value that is what we are seeing that *ADD 32* if you see, so it can be fed over here by this line direct connection because of the controller will set in such a fashion, so the value of 32 will be loaded over here. The control signals also will make the ALU to be *ADD* mode for example, when I am going for going to execute this command *ADD accumulator 32 immediate*.

So, it will be converted to *ADD* mode by the signal for the signals of the control unit 32 will be loaded in the bus, so you will have the value of 32 over here. But previous value of accumulator has to be added to this. So, in fact, actually the accumulator will be loaded in the previous cycle value of accumulator will be loaded to *Y* or in some sense you can also think that in my case *Y* is an accumulator. So, basically sometimes this is what is the idea.

So, in the first go, the value of accumulator will be or if the accumulator has if you have to load some particular value the accumulator. In first case, you will be loading the value of the accumulator from memory or some block to *Y*, I am assuming *A* as an accumulator. And in the second unit clock, you will actually *ADD* this accumulator with 32, but now you can see a MUX over here, because sometimes we require to *ADD* some constants also like program counter. Program counter will be program counter plus 1. So, in this case, I have kept the value 4, because I am assuming in this figure that assuming that the number of instruction length is 4, basically I can also say that the sorry I can make this constant as one.

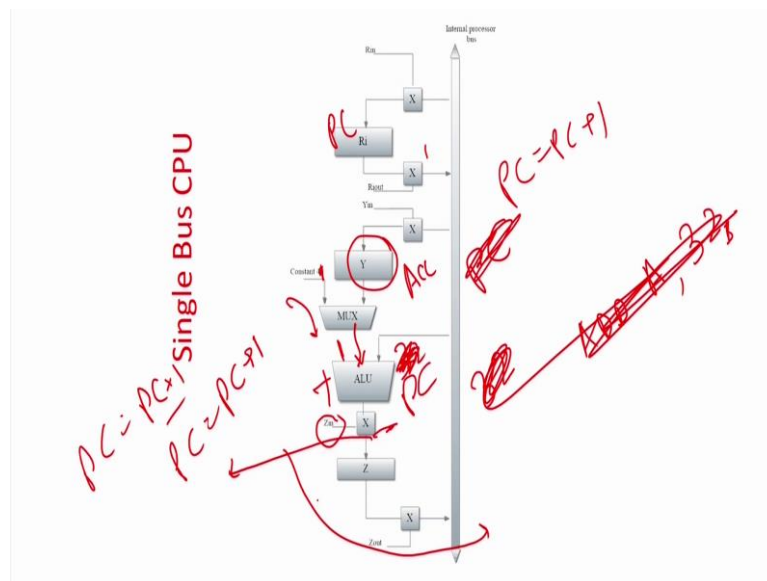
So, why I am making this constant as one; basically sometimes when I have to increment the *PC*. So, in this case what happens I will make the control unit will make MUX set as this part that is so for example, if this was about the *ADD*, so if I go for a so, for example, if I say that I want to go for a *PC* increment mode. So, in the *PC* increment mode, what is going to happen, you have to go for an increment of the *PC*. So, in this case I will this is the flow from the accumulator, so in this case I will not go by this path. So, what I will try to do is that is the constant. So, the multiplexer will be set in such a fashion by the control unit that this signal will enable this to go to the ALU.

And in fact, so this one will be added. Now, the value of *PC*, so you are going to add it to 1, and then the *PC* will be say for example, one because this is the chunk of registers. So, for the time being assume it to be the *PC*. So, in this case, what happens you will enable in such a fashion that this is enable will be 1. So, in this part, now the value of *PC* will be there. So, the *PC* will be connected over here. So, now, it is not 32, it is the value of program counter, and

program counter will be program counter plus 1. So, now, after that you can dump the value output here, and you can again feed it back. So, that is again let me clean the whole picture.

So, now, again what let me do it in steps? So, what is the first step, the first step is basically for *ADD*, actually I go by this configuration that the configuration is this, but now for *PC* to be incremented, let us assume this is to be *PC*.

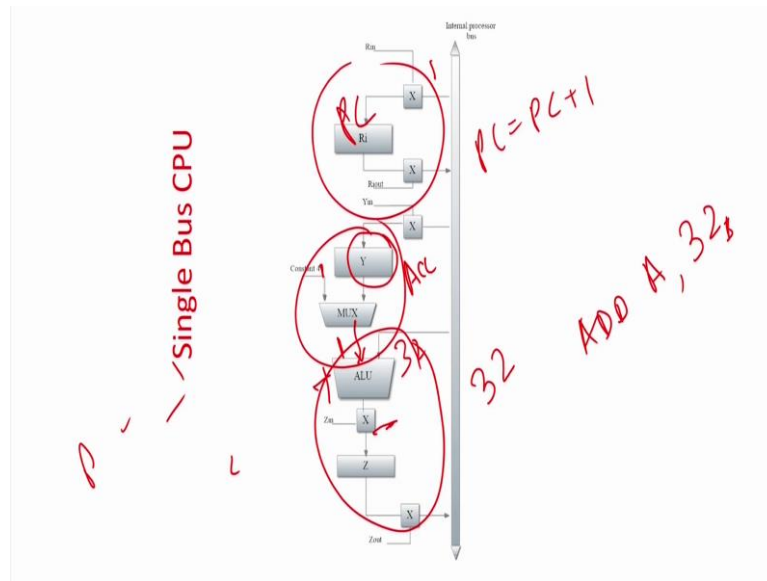
(Refer Slide Time: 27:58)



So, first we are going to go for $PC = PC + 1$. So, what we are going to do, for adding basically we are for adding two numbers or some other operation, we are going to take this path, but actually for the *PC* because it is to be incremented by a constant. So, what we are going we are going to take this path. So, the mux is set in such a fashion. So, this one is one or constant one in case if the size of the instruction is one, it may be 4, 8 or 2, but the size of the instruction that value will be fed over here, accumulator will in plus mode. So, in this case, we are adding a value of 1, but now it should not be the value 32.

So, it is not the case you should have the value of the *PC* fed over here. So, what I will do assuming that this is the *PC*, I will make this $PC_{out} = 1$. So, the value of the *PC* will be in this bus. So, now, the value of *PC* is fed over here. And addition is done and then what happens this is the Z_{in} plus will be made one or in other words this is the path to control the output of the ALU to the bus. So, now, it is added $PC = PC + 1$, and it is now waiting over here. Now, here it is $PC = PC + 1$. Now, after some time what you will do you will actually now it is having the value of $PC = PC + 1$, because now the output is fed over here.

(Refer Slide Time: 29:17)



Now, after that what you will do now you will make this as 1, because now $PC = PC + 1$. So, now, it will be loading the value of PC and PC will be incremented. So, in this fashion basically we require a multiplexer for that. So, these are basically if you want to understand what happens basically in this case, so this is basically the architecture of single bus. So, we have some registers which have input and output control that is register can be fed in, register can be fed out from the control bus.

Similarly, this is an ALU, ALU is also the facility you can load, and also you can write out the value of the ALU to the control bus. And we have a multiplexing arrangement. So, in one path of the multiplexing arrangement, you can take some operand from the control bus; and the other operand is a constant like increment, increment by one, but some other constant value. So, this is in a nutshell a very broad idea of a single bus CPU.

So, now we will tell you whatever I have discussed is given in the text, so I am saying that for each register including the program counter, memory buffer register etcetera there are two signals R_{in} and R_{out} . Basically if R_{in} is 1, the register is going to read from the bus; if $R_{out} = 1$ then it is going to write into the bus, but you have to be very very careful that we should not basically enable two R_{out} 's at a time then there will be contention. But there can be multiple R_{in} 's.

(Refer Slide Time: 30:35)

Working of Single Bus organization

- For each register (shown as R_i) two control signals (R_{in} and R_{out}) are used to place the contents of the register on the bus or to load the data on the bus to the register.
- To elaborate, the input of register R_i is connected to the bus via control signal R_{in} and output of R_i is connected to bus via R_{out} . When R_{in} is 1 then data on the bus are loaded into R_i and when R_{out} is set to 1 then contents of the register are placed on the bus. It is assured that at a given time no two registers can have their control signal R_{out} as 1; this ensures that there are no conflicts.
- Example: In case of instruction `MOV R2, R1`, first we set R_{1out} to 1 so that data of $R1$ is placed on the bus and then by setting R_{2in} to 1 (and R_{1out} to 0) data from the bus is loaded to $R2$.

$R2_{in} = 1$
 $R1_{out} = 1$

For example, if you are saying that move $R1$ and $R2$ then what will happen basically $R2$, $R1$ is feeding to $R2$. So, what is the control unit is going to generate the control unit is going to generate that $R_{out} = 1$, because R_{out} is going to give the value to the bus, and R_{in} $R2$ register is reading. So, $R2$ in should be equal to 1. So, $R2_{in} = 1$, and $R1_{out} = 1$. So, R_{out} $R1_{out}$ is giving the value to the bus, and $R2$'s reading. So, of course, other things has to be made 0, because $R1$ out is equal to in this first case. So, $R1_{out}$ is = 1. So, $R1$ $R1_{in}$ has to be made 0 that is the case. For example, $R2$ is reading, so basically $R2_{in}$ is 1 and $R2_{out}$ has to be basically set to 0 that is what is very simple logic.

(Refer Slide Time: 31:22)

Working of Single Bus organization

"MOV R1, 32"

- IR_{out}, MAR_{in} , Read
- MDR_{in} , WMFC (control signal that causes processor to wait for MFC signal)
- $MDR_{out}, R1_{in}$

- First, the address of the memory which is to be read (32 in this example) is loaded into the MAR. This is achieved by making signals IR_{out} as 1 and MAR_{in} as 1. As the contents of MAR are always on the address bus, when a new address is loaded into MAR it will be on the memory address bus at the start of next cycle.
- The memory control signal is made READ, which results in the data in the corresponding memory location to be loaded in the MDR. So, in the second cycle MDR_{in} is set. However, the system needs to wait for the delay involved in memory reading (indicated by MFC signal). After the delay (i.e., WMFC becomes 1), the data from memory is loaded into MDR.
- In next clock cycle MDR_{out} is made 1 and $R1_{in}$ is made 0 which transfers data to register $R1$ from MDR.

$R1$
 $R1_{in} = 1$

Now, we will take some more important basic ideas, other instructions to make it more clear. So, we are taking an instruction called *MOV R1, 32*. So, basically what is the steps, what happens. So, let us assume that this is your control bus then what happen then this is a register *R1*. So, as I told you, you can read from the register and also the register can be making an output this is your control bus. So, either you can read or you can write. So, in this case, the 32 has to move, in fact, here 32 is a memory location given the assumption. So, now, what happens 32 is not an immediate operand, it's a memory location. So, you have to read the value whatever is available in memory location 32 to register *R1*.

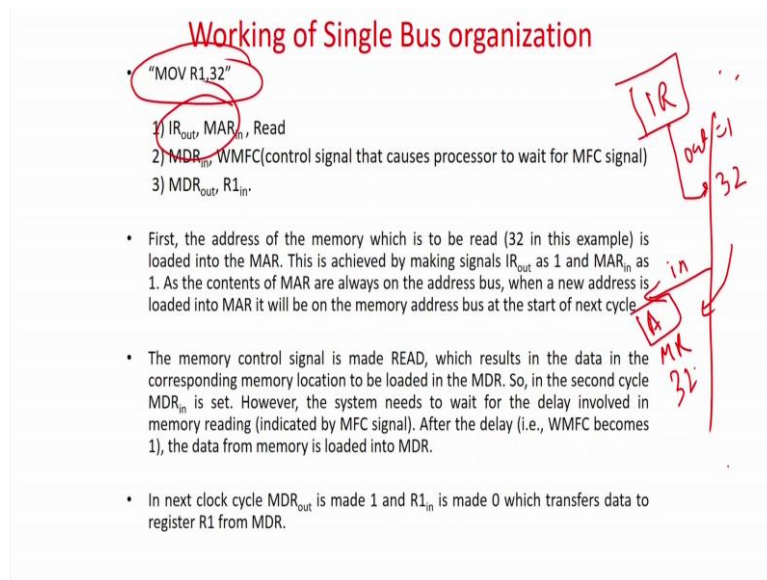
So, of course, register *R1* in should be equal to 1, because we are going to read it and then basically other steps will follow. So, what are the steps, first step is that the instruction register should give the command or the microinstructions following that. So, what are the microinstructions for that? For example, the value of 32 has to be first loaded into the memory address register, why, because the memory address register is going to give the value to the memory, and it will say that I want to read the value from 32, and it will dump the value in memory data register or memory buffer register.

And then finally, what is the first step, you have to put the value of 32 in memory register, memory address register make the signal read that is the first signal. Then what happens then the memory will see the memory address register is 30, whatever value is memory location 30 will be dumped to memory data register or memory buffer register and you have to wait for some time as I told you this signal is called *WFMC*.

Control signal that causes the processor to wait for the *MFC* signal. And it will have to wait for some time, when the memory says that I have write written the value what was the memory location 32 to the memory data register or the memory buffer register it will become 1 then you know the memory has given the data now everything is stable. Now, the memory buffer register will dump the value to *R1*.

So, these are the three microinstructions. The first microinstruction will lead to basically loading the value of 32 in the memory address register, second thing it will read the value in the memory data register, third microinstruction will load the value of memory location in the *R1*. So, what are the steps, let us write down the signals and the registers.

(Refer Slide Time: 33:46)



So, basically let us assume this is a control bus. So, first is the instruction register. As I told you, it is very, very important; the instruction register is the heart. So, what it will load, it will load the value of 32 in the memory address register. So, as I told you this is a memory address register - *MAR*.

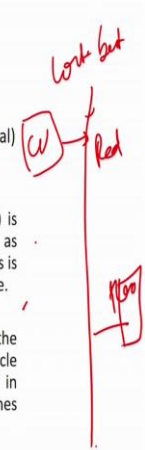
So, in this case, memory address register in will be one and instruction register out will be equal to 1, because instruction register will dump the value of 32 to the memory address register. So, in fact these are the two internal signals that is $R_{out} IR_{out} = 1$, so instruction register will dump the value control bus; and $MAR_{in} = 1$, so it will read the value of 32. Now, the 32 has gone to the memory address register. These are the two internal control signals.

(Refer Slide Time: 34:31)

Working of Single Bus organization

- "MOV R1,32"
- 1) IR_{out}, MAR_{in} , Read
- 2) MDR_{in} , $WMFC$ (control signal that causes processor to wait for MFC signal)
- 3) $MDR_{out}, R1_{in}$

- First, the address of the memory which is to be read (32 in this example) is loaded into the MAR. This is achieved by making signals IR_{out} as 1 and MAR_{in} as 1. As the contents of MAR are always on the address bus, when a new address is loaded into MAR it will be on the memory address bus at the start of next cycle.
- The memory control signal is made READ, which results in the data in the corresponding memory location to be loaded in the MDR. So, in the second cycle MDR_{in} is set. However, the system needs to wait for the delay involved in memory reading (indicated by MFC signal). After the delay (i.e., $WMFC$ becomes 1), the data from memory is loaded into MDR.
- In next clock cycle MDR_{out} is made 1 and $R1_{in}$ is made 0 which transfers data to register R1 from MDR.



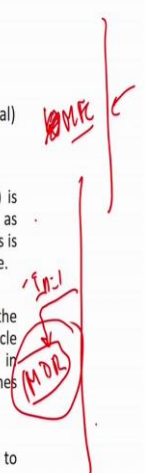
Now, next what next we have the control bus, this is your control bus which is an external control bus. So, the control unit will generate the value of read to the memory, because this control bus is connected to the memory, this is the memory which is an external, but the first two signals are internal, this is an external signal. Now, let us go to the second signal. So, in the second signal what happens it is saying that now after some amount of time, the data will come to the memory data register.

(Refer Slide Time: 35:05)

Working of Single Bus organization

- "MOV R1,32"
- 1) IR_{out}, MAR_{in} , Read
- 2) MDR_{in} , $WMFC$ (control signal that causes processor to wait for MFC signal)
- 3) $MDR_{out}, R1_{in}$

- First, the address of the memory which is to be read (32 in this example) is loaded into the MAR. This is achieved by making signals IR_{out} as 1 and MAR_{in} as 1. As the contents of MAR are always on the address bus, when a new address is loaded into MAR it will be on the memory address bus at the start of next cycle.
- The memory control signal is made READ, which results in the data in the corresponding memory location to be loaded in the MDR. So, in the second cycle MDR_{in} is set. However, the system needs to wait for the delay involved in memory reading (indicated by MFC signal). After the delay (i.e., $WMFC$ becomes 1), the data from memory is loaded into MDR.
- In next clock cycle MDR_{out} is made 1 and $R1_{in}$ is made 0 which transfers data to register R1 from MDR.



We know that you know already the memory address has been set so some data will be coming to the memory data register this is an internal part or a memory buffer register whatever. So, as the data from memory location 32 is going to come over here this in signal I have to make it one, this an internal signal. But then if you wait for some external control bus, you have to wait for some of the signal which is coming from the memory that is called wait for *MFC* this signal is *MFC* we are waiting for *MFC*. Whenever *MFC* is going to be one at that point of time basically already *MDR* is equal to one input. So, whenever $MFC = 1$ then you know that there is a valid data over here, I have already made it 1.

So, whatever is the memory buffer register is coming over here, but I cannot immediately read the value in the memory data register, from the memory data register to *R1*. I have to wait for some amount of time. Whenever *MFC* is 1, which is an external signal then again the third micro cycle instruction control signals come up. Now what, so now basically *MDR* is already one signal has come. So, now, what is going to happen already memory data *MDR* has already read.

(Refer Slide Time: 36:07)

Working of Single Bus organization

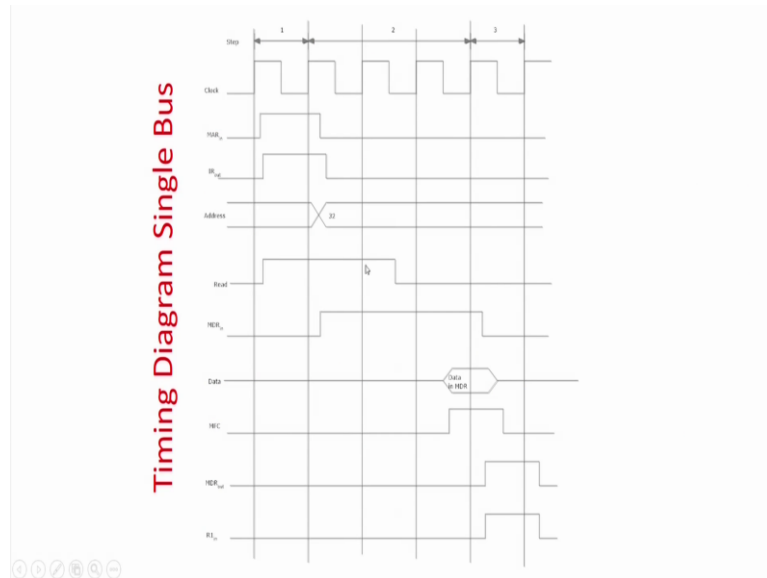
- "MOV R1,32"
- 1) IR_{out}, MAR_{in} , Read
- 2) $MDR_{in}, WMFC$ (control signal that causes processor to wait for MFC signal)
- 3) $MDR_{out}, R1_{in}$

- First, the address of the memory which is to be read (32 in this example) is loaded into the MAR. This is achieved by making signals IR_{out} as 1 and MAR_{in} as 1. As the contents of MAR are always on the address bus, when a new address is loaded into MAR it will be on the memory address bus at the start of next cycle.
- The memory control signal is made READ, which results in the data in the corresponding memory location to be loaded in the MDR. So, in the second cycle MDR_{in} is set. However, the system needs to wait for the delay involved in memory reading (indicated by MFC signal). After the delay (i.e., $WMFC$ becomes 1), the data from memory is loaded into MDR.
- In next clock cycle MDR_{out} is made 1 and $R1_{in}$ is made 0 which transfers data to register R1 from MDR.

So, now if this is your *MDR* it has already read. So, now, initially it was in was one because it was reading from the memory, now it has to make it out. So, my memory data out signal will be 1, so out signal is equal to 1, because already we know that the memory has dumped the value in the memory data register and it will read to *R1*. So, this in signal has to be 1. So, the memory data will be given over here. So, again in this case, these are the two control signals

generated, $MDR_{out} = 1$ and $R_{in} = 1$. And both are registers basically they are all internal signals. So, this is how in a single bus memory operation that is reading from 32 into $R1$ works. So, basically what I have told is written in the text over here which you can go through.

(Refer Slide Time: 36:51)



Now, we are going to look at in a timing, as I told you that will be all discussing it in terms of timing sequence. So, let us look at the timing sequence. So, this is your clock and we are doing everything in the positive edge or in the positive edge of the clock. So, this is the positive edge, this is the positive edge and so forth. So, first as I told you first one read $R1$, 32. So, what you have to do, first you have to read the value of 32 from the instruction register to the memory address register. So, what I am doing just after the first clock edge I am making $MBR = 1$ and $R_{out} = 1$.

So, what is going to happen R_{out} that is instruction register which is containing now the memory location 32 will be dumped to memory address register. So, I have made both the control signals equal to one. So, in the next clock edge, what is going to happen that is the synchronization. At this clock is what is going to happen the value of 32 from instruction register will go to the memory register that is at this rising clock edge, the value of 32 will go from instruction register to memory data register, it will happen at this clock edge.

So, now again let me erase this, and let us study the future in future what happens basically that is the first clock edge. Secondly, so already we have seen that in this clock edge the value will be dumped. So, we have actually given the value of address 32 is has gone to the memory data

register. So, in this clock edge has got this address of 32 is now loaded into this memory buffer register sorry memory address register. Now, already we know that we have to give a read signal. So, anyway I have given the read signal over here, we could have also given the read signal over, I have given the read signal initially. So, now, at this clock edge, these 32 is loaded in the memory address register as well as the read signal is the one.

So, now the memory is configured that in this case in the first in this clock edge, the instruction register has given the value of 32 to the memory address register which is this case, read signal was already one. So, now, the memory knows very well what it has to do and in fact, already I have, so this is happening in this clock edge. So, by this clock edge the memory data should come, because in this case I have sent the value 32 to the address register in this clock edge that works.

And again in this clock edge, the value of 32 will be taken in by the memory and the read signal is also one at this place. So, in fact, now it will start dumping the value of the memory data register the memory will start dumping the value whatever is in the 32 that will actually happen in the next clock in this clock, it should start happening. Because in this clock edge 32 is loaded in this clock edge because everything happens in positive clock edge.

So, if you find in the first clock edge nothing happens in the second clock edge basically in this edge the instruction register value has gone *MAR* address register, and in this clock edge 32 is loaded in that case and read instruction is also 1. So, in this memory, in this clock edge the memory is fully configured to deliver the value of memory location 32 the memory data register, already read signal was there. And if you see *MDR_{in}* was already set, so now, in this clock pulse if you compare the value will be after this clock pulse that is this clock pulse the value will be fed to the memory data register because the signal is already one.

And only after this the *MFC* will signal will be set as high. So, what is this *MFC* signal, only after this clock pulse? The *MFC* signal will be high means it will say that now I have dumped the value of memory location 32 in the memory data register in a very peaceful manner everything is stabilized. Now, you can read, so that is the signal, so that will happen in this positive clock edge. And after that has happened that *MFC* has been given as out, so what you will do this says that I have done or I have given everything which was in memory location 32 to the memory data register.