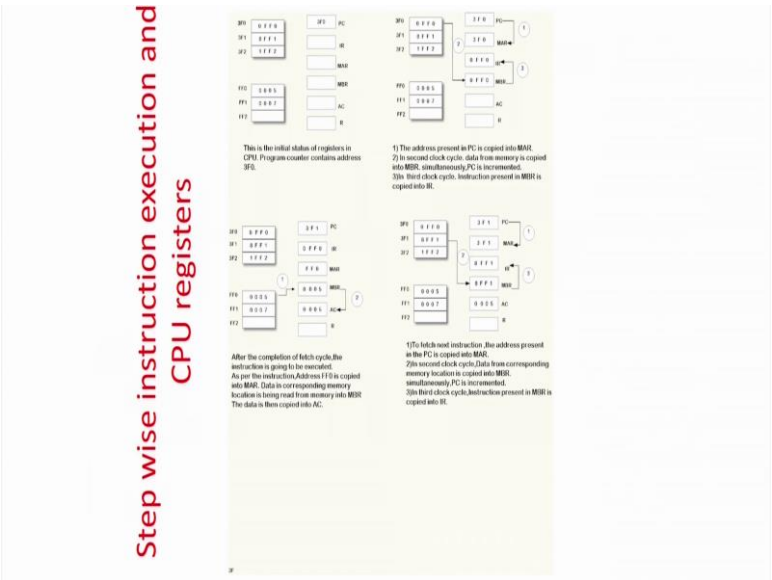
So, the threes codes for LDA store and load add and store are this one. That is the opcode is 000 means that it is a load instruction add means 1000 and 0001 for store. And then this is where you have to load the value is FF0. So, this one is going to be the binary. So, if somebody erases this and say this is your first line of code, 000 very difficult to read and understand therefore, we always keep the memories. So, in this case the instruction size is 4 instruction size is 4 + 12 that is 4 × 4 16 bits.

So, it is a 16 bit instruction size that you can also think that a memory in this case is a 16 word bit is the word size. So, in all the instructions mainly in this case whatever is discussing for this example are loaded in a single word. So, it is easy to fetch decode and execute next it add FF1. So, same case now you can see that this format has means the code has changed where this correspond to add, and this one is the case similarly for storing this one. So, the idea is that if I write only in these 3 binary numbers it is very difficult to understand. So, you always go for the mnemonics and as it's again a single of the 3 instructions are written in a single address format. So, the last 2 are basically data transfer and this is the arithmetic.

(Refer Slide Time: 17:20)



Now, we see step wise basically what happens, now we will again deal with we have already discussed a similar example beforehand, but now we will see in more depth of the different instructions, even registers and the formats. So, as I told you. So, this is the first instruction to be executed. So, the PC is going to have the value of this one value of the memory location of the first instruction. Then what happen is that. So, in this case. So, this instruction no as I told

you we are assuming that this is a 16 bit size. So, each of this is memory location has a 4 instructions

So, only one word can be taken to the memory buffer register or the instruction register and your job is done for example, in this case this is a single address instruction. So, assuming that if it had been a 2 word instruction the double 2 address instruction. So, in this case you might have taken a longer size. So, maybe 0 FFF and in this case maybe the other part of the other address would have been there.

So, in that case you would first required to bring this, taken to IR, maybe the IR would also have been elongated over here then this part of the instruction will be taken from memory buffer register to the IR it will be a longer case and then only you will be able to after 2 memory read operation you will be able to understand the meaning of the instruction. If it would have been something like say I allow add memory location FF0 memory location FF1. So, in this case 0FF0 would have been here and in this case it will be the FF1. So, you have to first read this then read this part this memory location of course, it would have been have been there merge it and then do. So, it will be more difficult to do that, but your instruction you would have required only 1 or 2 instruction to solve it, but now we require 3 instructions because we are taking a single address format.

But single address these things are very simple because every memory location has a single instruction. So, what is the case? So, if FF0 this has to be fetched. So, 3 is a load operation sorry 0 is a load operation from where I have to load? So from FF0; that means, it is saying to load the value whatever is available in FF0 the value of 5 to accumulator.

So, first F what happens the PC is pointing to this, this instruction is going to the memory buffer register and as we know that because of this addressing format each memory location has a single instruction. So, need not worry directly take the value of memory buffer register to instruction register, it will decode it and it will find out that it is asking to load the value of FF0 to accumulator. So, that is what is being done. So, it is loading the value of FF0 to the memory buffer register and then to the accumulator and the value of PC will be incremented by 1 next what happens. So, what is the value of value of FF1?

So, it says that add value of accumulator to whatever value is in FF1. So in fact, again as I told you one word is enough to take a full instruction one address format. So, it will be loaded in the memory buffer register it will go to instruction register decode it and it tells that 8 FF1

means whatever value is in FF1 has to be added to the accumulator that has store back to the accumulator. So, this one is done, memory buffer register it is the instruction register, it is decoded and it knows what to do, and you can understand that program PC is actually program counter has changed to next instruction to be executed.

Now so, as I told you the next step says that you have to add, this one was the step that you have to add the value of the accumulator to FF1. So, 5 was initially in the accumulator now 7 has been in the memory buffer register, you add these 2 value there is 7 + 5 is 12 that is C and will be again stored back to the accumulator, this is the job which is done by the second instruction and finally, now what happens is that next you have to store.

So, this is the program counter has gone to a third instruction, it is getting loaded over 1FF2 to the instruction register. So, what it tells that whatever is in the value of accumulator because 1 means store to FF2. So, accumulator value will go to memory buffer register and it will be stored over here. So, this is the final execution. So, what we have seen in this case.

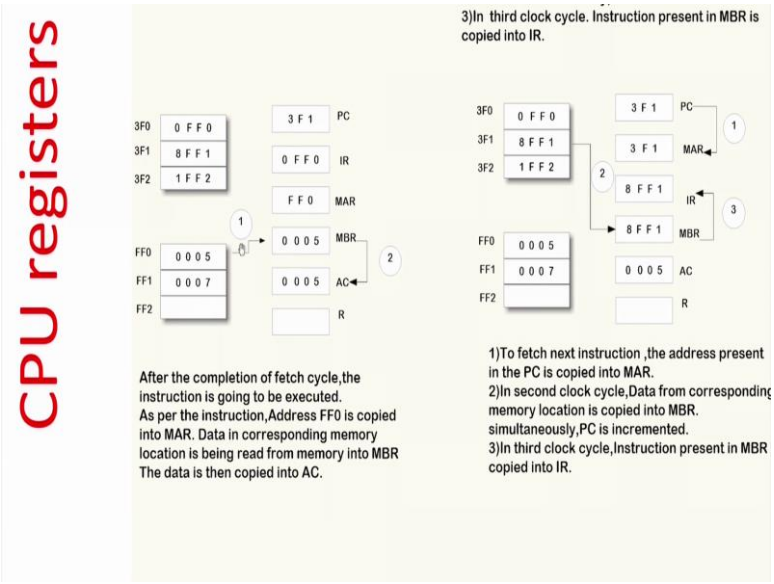
So, in this case we have seen that as the instructions where a single address instruction and the size of the memory was we have assumed to be 16 or something. So, it fitted and very easily one instruction could be fetched and it could be decoded then the job done. But you could have taken 2 number of instructions at a time. So, sorry double 2 address then we could have saved on the number of instructions, but in this case you have to take this word and this word join them in the instruction register decode and do it.

So, that would have been a more complex way of solving the problem and again we have seen that what happens basically. So, in this first was the memory operation memory data transfer operation. So, in this case the data is transferred from the main memory to the memory buffer register and then it goes to the accumulator or it will go to the instruction register depending on the case. The next was a address arithmetic operation in this case what happens.

So, it takes some values from the accumulator and the memory location, and it actually operates, it uses an adder to add it and store back the results that is happening in the third step, which is again a I/O operation. Now basically now we will. So, that was more or less whenever you talk about arithmetic operation a ALU means arithmetic and logic unit is being brought. Let us store an I/O.

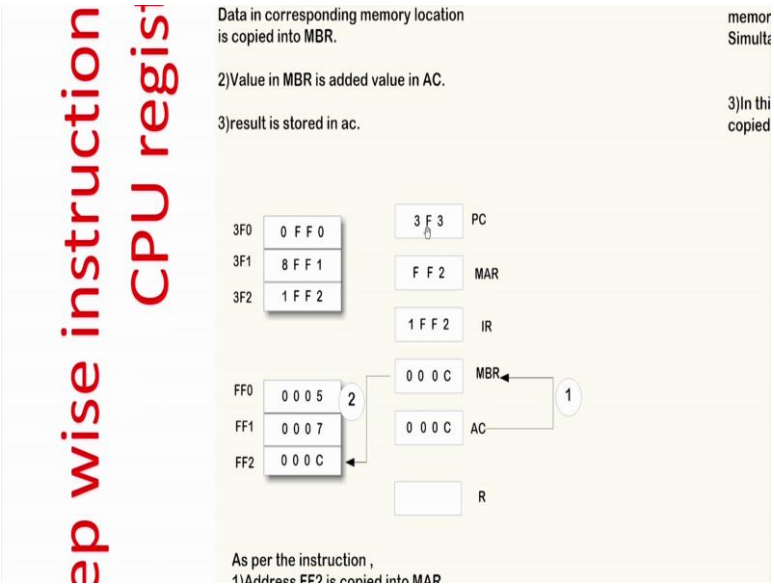
So, control means what? It's very simple control means based on the value of some of the jump instruction the value of the PC will be changed like for example. So, if you look at it. So, what happened?

(Refer Slide Time: 23:04)



So, the first case the value of program counter was this, the next the value of program counter is next 3F1 and after that the value of program counter becomes 3F2 that is it goes step by step. Now what it can happen that if there is a conditional instruction then instead of going incrementing like 3F0, 3F1 at some point of time it will change.

(Refer Slide Time: 23:13)



So, that's simple. So, if you have a conditional instruction the value of PC gets changed, if you have a arithmetic instruction what happens? An adder or a multiplier or subtractor is used. Important are with lot of complicated registers and control signals are used are the memory operation and the I/O operations. So, the memory operations basically will be dealing with here in details, and for the I/O operations as I told you, we have a separate module on this because I/O is a more complex operation because in that case you will have a separate device which is connected to the CPU. It may be a camera, it may be a keyboard, it may be a mouse.

So, you have to first understand whether the mouse is correct or operationally fine, whether it is ready to give the data and so forth. Like it's in case of a keyboard it says that I need a data, then you have to wait till a person presses the keyboard, then I press the keyboard then again the data comes. So, a lot of synchronization issues. So, a whole module is dedicated to that, but right now we will see what is the memory fetch operation how it interacts with the memory, what are the addresses involved, what the registers involved, and what are the controls signals. So, memory fetch means memory read because the other class of instructions like data, data transfer as well as your control are quite simple. So, what happens whenever you want to fetch a memory location, you have to give the value in the memory address register.

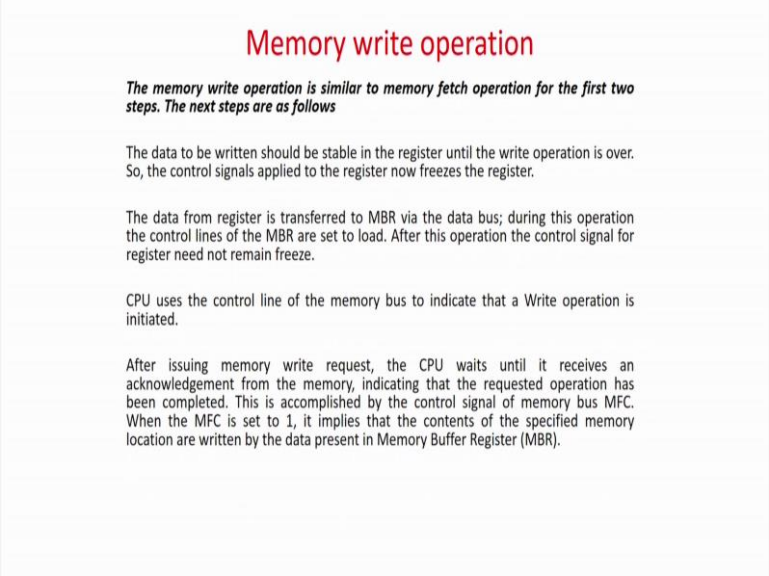
So, whatever value is given in the memory address register is the location from where the data has to be read. The MAR is connected to the memory bus and hence the address is required for this one to main memory. So, MAR basically is connected to the, memory address register is

connected to the basically address bus of the memory and so, it is therefore you directly memory will get the value of the address from there. Next the CPU has to tell that it's a read operation, because we are at present discussing about memory read. So, there is a control line we have already told, it's read or write it tells the control line says that it's a read operation. Now actually there is a difference between the CPU speed and the memory speed and the I/O speed. CPU is the fastest, memory slightly slower and I/O is extremely slower.

Because I/O means myself, you and some human being is handling, memory is a slower device compared to a CPU that we will see later, but for the time being you can consider this hierarchy. So, therefore, there is a synchronization issue. That whenever I say that I read from the memory, but you are never assured that immediately I will get the answer. So, there is a synchronization issue that is a handshaking that you say that I want to read it read some memory location. So, I give the value in the memory address register, and then the memory address register via the address bus is connected to the memory, then you say that read so the read signal is made, but how much time I should wait before I take the value from the memory buffer register. Because based on your read and the address in the MAR, the data will be saved in the memory buffer register. Though there is actually the CPU waits till this is an acknowledgement from the memory that is memory function completes MFC. So, MFC is a very very important control signal. So, what it says is that whenever a job is done that is the read is done data is dumped to the memory buffer register it will make the MFC signal high or maybe it will say that it's enabled. So, now, what will happen? You have to read the value from the memory buffer register and you can freeze the location therefore. So, that freezing etcetera will come later.

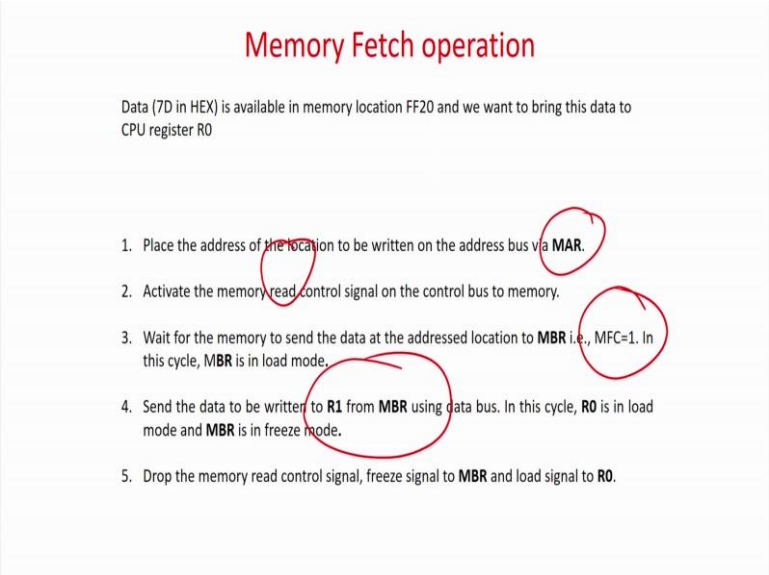
But for the time being whenever you said I want to read from the memory, and the data is available in the memory buffer register, and then you have to wait till the memory function completely set to 1. Once it is done you know that the value is stable in the memory buffer register, now it can be read to the accumulator it can be read to the instruction register and so forth. So, next is basically the memory write.

(Refer Slide Time: 26:57)



But let us first this is the memory write we will come to it later.

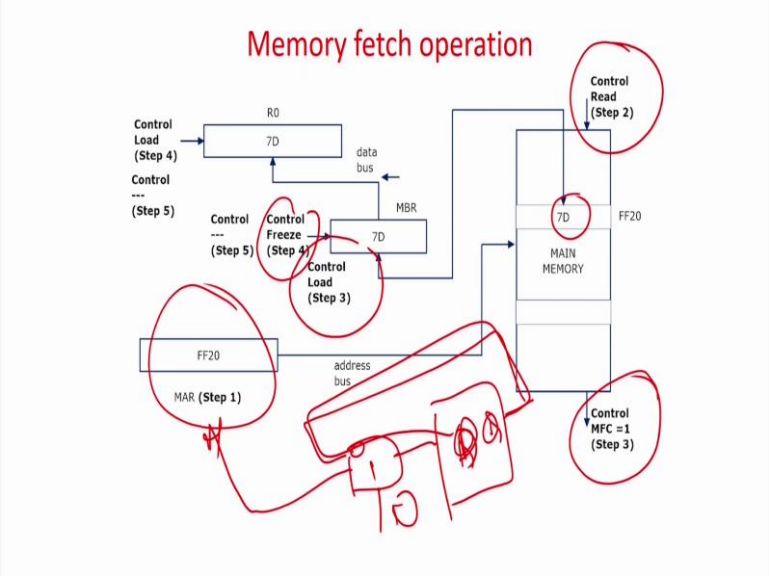
(Refer Slide Time: 26:58)



First let us see basically a memory fetch operation in the steps as well as the pictorial representation. As I told you first it is in the memory address register, then read, then you wait if the memory function complete is ready, then when it is done then it can be read to the memory buffer register or it can be register 1 accumulator or wherever, and then only then actually the memory buffer can be is ready you read it from this one and then you can go ahead. I will better take it a figure and explain it. So, what was the example? The example they are saying that data

7 hex which is available in memory FF20 we want to bring it to CPU register $R0$ that is the example. And typically this is a flow whether you want to take it to accumulator or any register this flow will more or less will become similar if it's the accumulator the memory buffer will write to the accumulator, if it is $R0$ the memory buffer register will write to $R0$.

(Refer Slide Time: 27:54)



So, now let us see step one. So, I want to read from this memory location FF20. So, memory address register will have the value of FF20, this is connected to the address bus it is going then I want to read this basically. So, now, next step is read, read signal is given now you have to wait how you have to wait till memory function complete is one. So, once the memory functions complete is 1, it means that the value of this one is written to the memory buffer register and it is stable.

So, once that is done in step 4 you have to freeze it, many times I was saying what is freeze. So, freeze is a very simple operation if I take a D flip flop, D Q is the D flip flop, if I connect D back to Q. So, if I can connect D back to Q therefore, there will be no change in the value of D even if I apply a clock. So, that is actually a freeze operation. So, if I make a permanent connection like this. So, your memory D flip flop will always have the same values. Actually what happens is that there is a basically a multiplexer and this is D and this is Q sorry this is D and this is Q, maybe 1 port will be free another port will be loaded and this will be a mux. So, if you make mux = 1 then from the port it will read to D and your flip flop can be set and reset and if I make this mux value equal to 0 then the Q will be feedback to Q sorry Q will be

feedback to D, and there will be a loop back and it will be in a freeze position that the value will never be stabilized.

So, this is actually multifunction register, so you can read over it from the digital fundamentals. So, basically in step 4 after the MFC is ready, you freezed it the memory buffer register is freezed, the value should not change over here. That is even if you give a new memory address you say read MFC 1 or whatever the value of memory buffer register do not change and then in the step 4 you freeze it and load the value in the register $R0$. So, once it is done in step 5 you can release the freeze. So, what happened step 1: I give the address, step 2: I see read then I have to wait for synchronization that the memory function is complete, after it's complete you read the memory buffer register.

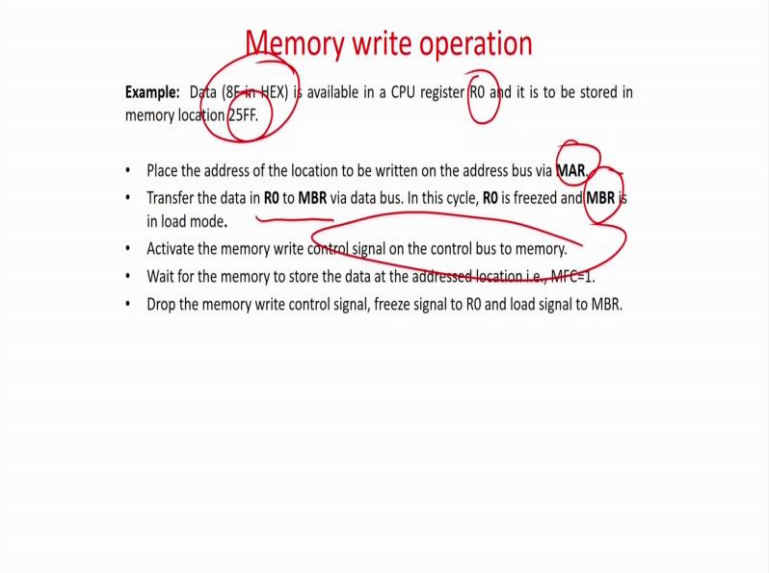
So, at the same step when you are reading it before they are actually after reading it you have to freeze it or you can see just after step 3 when the MFC has been 1 you freeze the memory buffer register that is you do not allow any more changes to write over there from the neighbor; that means, even if I given a new address line and if I give a new step or whatever this value should not be disturbed and then in this step after freezing it you read the value of memory before register to a your required register which is $R0$ in this case, it can be an accumulator and then after step 5 you defreeze this and again the step repeats once again.

So, that is what I was saying that in this case generally registers have freeze mode all these commands are there. So, after basically the memory sets that MFC. So, generally we freeze the in case of read we will freeze the memory data register or also for many cases we will freeze the write and read registers from the memory. So, the data corruption because of race conditions do not happen basically. So, we are discussing in details for that memory data transfer operations from the memory because it involves many more critical steps as we have seen it is not just like giving the address and getting the value there is lot of synchronization involved. Similarly the memory write register is also very similar, but in this case what should happen is that in this case you have to give a write command, first you have to give the memory location value then you have to give the write and then you have to again wait till the memory function is complete; that means, the memory says that I have been able to successfully read the value from the memory buffer register, but again before after you put the so, what are the steps. So, step is that first to give the value in the memory buffer register then you say that you give the memory address or you can change the phase then you say write, before you give the write signal your data to be written to the memory has to be stable in the memory buffer register and

then you have to wait, then the memory says that memory function complete, that I have already read the values from the memory buffer register.

But once the memory starts reading you have to again freeze the memory buffer and then only you can release the freeze only when after it says that the memory has been read from the memory buffer register. So, there is again a synchronization involved that is whenever the for the time for which the memory is reading from the memory buffer register it has to be freezed and you have to wait till it says that the memory function complete is equal to 1 then you can again de freeze and go for the next value.

(Refer Slide Time: 32:09)

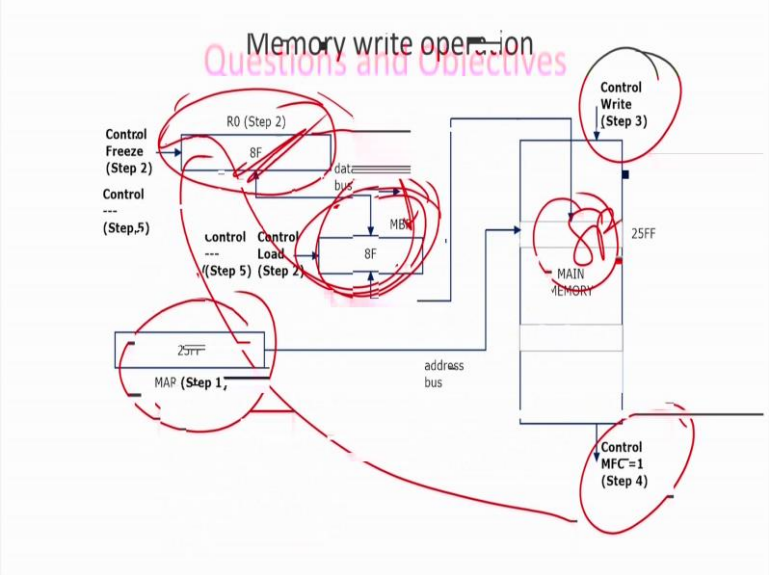


So, again this is the same example it says there is a data 8F in hex which is the available register 0 and you have to store in memory location 25FF. So, place the address in memory buffer register then transfer $R0$ to MBR and now memory buffer register is to be freezed that is what first you locate in the memory address register you put the value of 25FF that is I want to handle this memory location, then you put the value of R0 register or accumulator whatever may be the case in the memory buffer register. And then you stop the memory buffer then you freeze it, because now the memory will start reading from the memory buffer register.

Here is now the memory buffer changes, then there can be always a race condition and some garbage value may go into the memory. So, you have to have a freeze control signal. So, you freeze the memory buffer register and then you activate write, and then you have to wait till the memory says that MFC is 0; that means, I have read back read the data correctly from the

memory buffer register, now you can remove the control signal you can de freeze everything and go for the next operation.

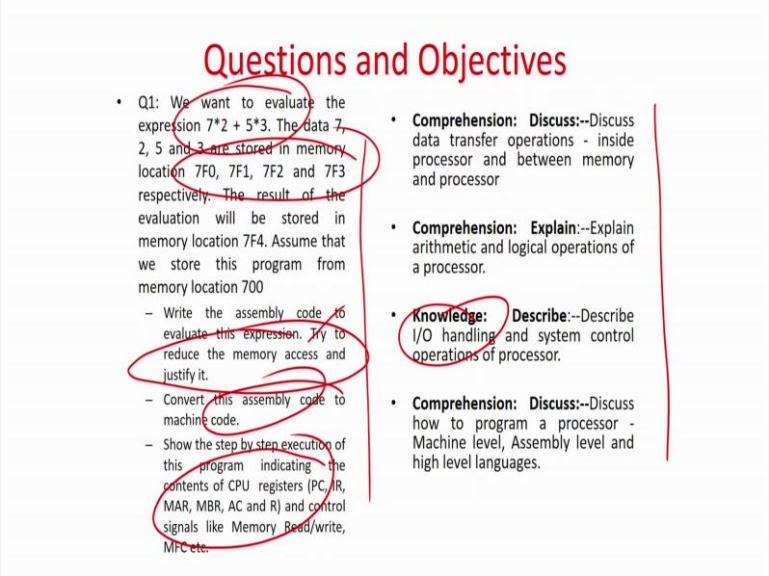
(Refer Slide Time: 33:07)



So, this is the example. So, always first step in general any memory operation will be the value of the memory address register, then in step 2 you bring the value of from register the 8F to the memory buffer register and freeze, that is very important. This one you put over there and you freeze this whole step, you can you basically freeze the memory buffer register don't allow any changes there. Now you say give write signal then write signal means it will start reading from the memory buffer register.

But note here that the memory buffer register is now freezed, so the memory buffer is freezed know whatever may be the changes over here it will not be changed and memory will very nicely read the value that is 8F will come over here and after 8F will come over here, it will say that memory function is complete. Once memory function is complete you can de freeze everything and go for the next memory cycle ok.

(Refer Slide Time: 33:53)



So, in a nutshell these was a very small module in which we have showed you the classification of different type of instructions, based on the functionality and then one of the most integrated form of instruction that is the data transfer and I/O. So, I/O will be dealt in a detailed manner, but in the current unit we have shown you in a very detailed manner, that basically how a input output operation happens what are the synchronization signal involved and on that class of instructions, how the memory and the CPU is synchronized and what are the detailed memory involvement and the registers involvement in them. Ok so, as I told you these were the basic objectives which we are targeting in this unit that, you have to basically describe what are the different type of instructions.

What are logic and different type of arithmetic operation, control operations etcetera? A very simple question if I say that I want to evaluate some expression and I say that the values are available over this memory location, then I ask you to write assembly language code and very precisely show how the memory type of operation happens, write an assembly language code for that then very explicitly show that in different type of modes, how the memory is accessed, how the synchronization happens and also show the different type of register values.

So, after listening to this unit and the units prevails to that you will be able to solve this problem. Here you will be able to the design the first the assembly language code then maybe translate it into the machine language, but anyway that is not very easy to read. So, we keep the mnemonic version, then you can see that there will be lot of memory read and write operations.

So, how they will interact with the memory, so basically you can talk about the I/O handling in this case actually I/O means a memory in this case basically because I/O we are not directly dealing with we have just given you the idea. So, basically different type of logic operations or arithmetic operations will be happening over here. So, you can have get a great deal of if you are able to solve this problem.

So, most of these knowledge based and explanation based comprehension based objectives will be solved I mean; obviously, it can be met basically. So, with this we come to the end of this 2 basic modules or basic units sorry basic 2 units in which case we have seen an instruction type, components of an instruction, and how they can be classified.

So, next unit will be a more dedicated unit and it will be elaborate unit on which we will show that as I told you is add instruction. Add instruction can be very different type add can be of immediate it can directly take from a memory; it can take from one from a memory and one for a register. So, how the different type of instructions how different type of instructions for a same operation happens basically, there is in more depth of how the instructions can be curtailed so that it helps you to solve the more same problem in a different in a more efficient manner.

Like add the data can be from the instruction itself and if you will require a very wide data or a very precise data you have to store it in multiple locations in the memory. So, same instruction we will be doing the same operations, but in a different manner. So, that is different mode of addressing modes will be there. So, next unit will be more dedicated on or more rigor analysis of different type of instructions.

Thank you.