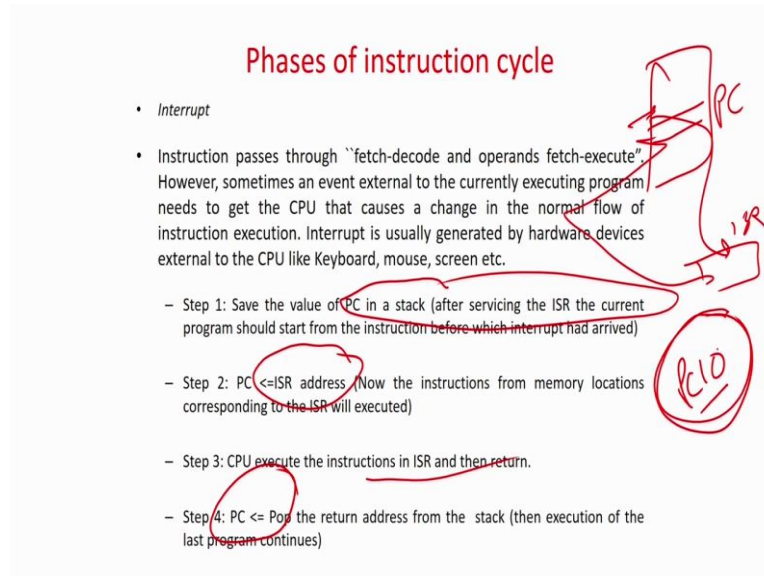


(Refer Slide Time: 23:27)



Now, we are coming to the indirect phase of instruction execution that is a interrupt. As we again discuss that interrupt is basically a normal flow of code is going on, then some hardware or some IO devices interrupt which has to be serviced in urgent manner then basically the instruction starts.

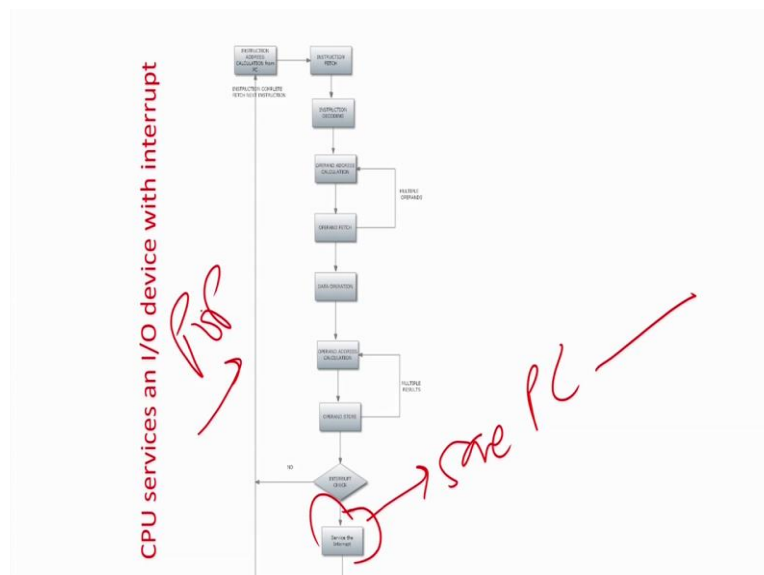
So, what happens? So, after no interrupt can offer where the instruction is being executed, instruction 1 2 3 4 5 6 7 8 9 10 is going on it between no interrupt can come. But after an instruction have been executed it will check whether there is an instruction interrupt, if there is a interrupt it has come then what you do you save the value of *PC* in a stack. Why? Because a may *PC* may be 10 now when the interrupt has occurred; that means what, after executing instruction 9 *PC* has become 10 and then interrupt has occurred or we have detected the interrupt. Then while coming back you have to again restart your code form 10th location or what or that is the *PC* value it is stored location.

So, what we will do? You will save the value of program counter all the registers intermediate values in a stack and then you will go to the instruction service routine who will instruction service routine nothing but another code itself or a code module with the some instruction is a jump. So, how can you jump? Now the instruction service routine address will be loaded into the *PC*, but the code actually was to execute to a 10, but now instruction has interrupt has started so you have to service the interrupt that is a new set of code you can think it to be a function it will jump and again come back.

So, I save the values of $PC=10$ and save all other intimidator registers and then I put the address of the interrupt service routine to PC . Now the PC will start pointing out to the instruction which is in the interrupt service routine, maybe this is your memory where your PC 10 it has to be executed, but some interrupt occurred then your PC is jumping to here that is the ISR, it will start operating from here and after it is finishing that it should again come back to 10. So, how it is done? Again the value after you complete the ISR you put pop back the value of PC from the stack, so now again PC will have the value of 10 and again you will restart everything. So, that is what is the idea of an interrupt service routine.

Again I will just show you the flow in a zoom manner.

(Refer Slide Time: 25:40)



So, you can see. So, address calculation whatever is done instruction you fetch, address calculation for PC . So, the instruction is fetched from what is the value of the from the memory in the PC instruction is fetched, then instruction is decoded, calculate the address of the operands; we have already seen it can be in a direct, indirect, immediate fetch operand if there multiple you have to do multiple times operation fetch. But I told you that very now less large number of operands in single instruction is not a very good idea. Keep on doing it, then after all the operands has been fetched you do the data operation that is in fact, it may this your logic or and arithmetic operation that is the operand that is your execution of the instruction then finally, again you have to find out where the answer has to be stored for that also some operand address calculation is required.

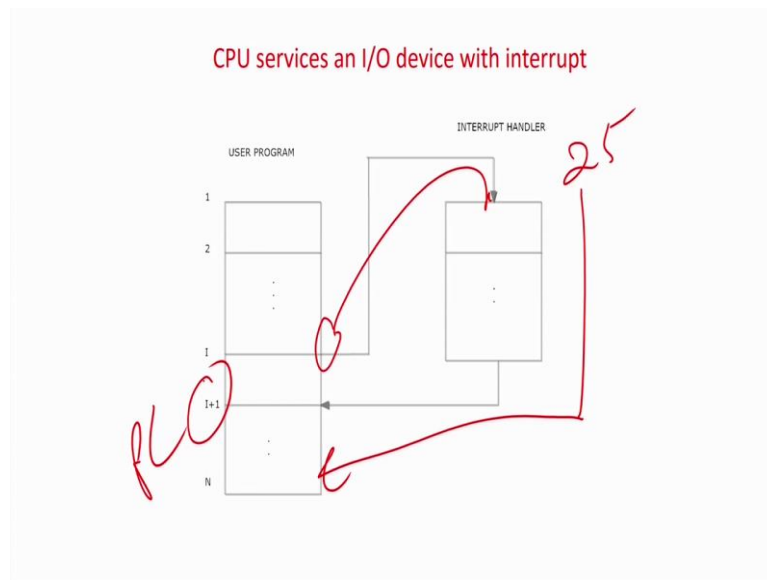
Like for example, if you said that add $a + b$ and store it to some place or instruction may be simple like store something to something. So, in that case the data operation will be nothing to store the instruction. So, you have to also go for operand address calculation where you want to store the result, you store the result and you go on back. But sometimes your operand address calculation may be very trivial line add accumulator with memory location 32; that means, whatever the data is in a memory location 32 has to be added to the accumulator and stored back into the accumulator itself. Accumulator is a register, so operand calculation in this case trivial that is the accumulator itself, but you still you have to do it store the result you have to keep on doing it.

Now, this completes actually one set of instruction execution after storing then you will check whether interrupt has done has arrived or not. We do not check any interrupt in between because if you do it you may go into a deadlock mode that one instruction is not completed you start another instruction and so much lot of cumbersome stuffs may happen.

So, we stop after the instruction has been completed then we check whether the interrupt is there or not. If no, again you if PC has been already incremented you keep on in fetching the instruction decoding it executing and go on, but if it is not then you have to service the interrupt. Servicing the interrupt means you have to basically here actually before you service the interrupt here you have to save all the result of PC intermediate values and etcetera and after you service the interrupt you have to again reload the value of PC that you save everything.

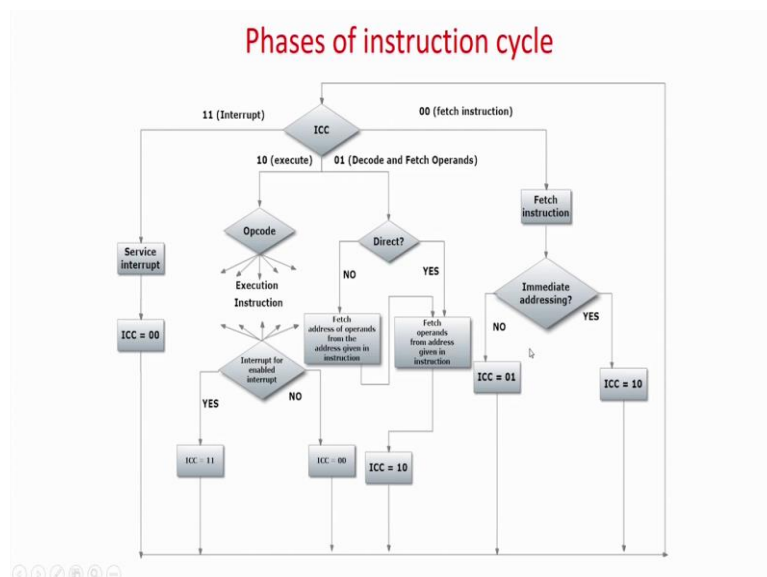
So, at this point you save PC or the program status word, after servicing the interrupt again you pop and do your job. So, it's very simple fetch decode execute. Say if there is an interrupt if there is interrupt save everything, service the interrupt, again come back and keep on doing that is what is the idea of a whole life of an instruction.

(Refer Slide Time: 28:15)



So, this is actually the figure. So, user program you check there is an interrupt you go to the interrupt handler that is interrupt service routine. So, PC value should be $I + 1$ or something, but now it will be changing to may be some 25 or arbitrary value, service the interrupt again get back to the value of $I + 1$ and again start execution that is what is the pictorial representation.

(Refer Slide Time: 28:37)



Ok now, this something very interesting. Now basically there is something called a ICC that is nothing but instruction cycle code that is how basically the cycle code that is at what stage of

instruction you are in, is it fetch, decode, interrupt or execution. How it is determined that, what stage instruction is.

(Refer Slide Time: 28:43)

Phases of instruction cycle

- The four phases can be identified using a code called Instruction Cycle Code (ICC). ICC is a two bit code and designates which part of the phase the CPU is in.
- The codes assigned to each phase is as follows
 - 00: Fetch phase
 - 01: Decode and Operand phase
 - 10: Execute phase
 - 11: Interrupt phase

So therefore, there is a special code called instruction cycle code is a two bit code and it is used to determine phase. Now we will see how basically it is very interesting and how basically this codes are changed based on phase to phase, phase to phase. So, 00 is the fetch, 01 is the decode and operand phase, 10 is the execute phase and 11 may be interrupt if it comes. As I told you generally we talk fetch decode execute decode means operand is also fetched is in that cycle. So, let's start over here. So, if you see.

So, what happens? So, let me zoom it over here. So, if you zoom it over here you can see what happens basically. So, first ICC value is 00 initially. So, instruction fetch it comes over here. So, if its 00 means its instruction fetch. So, it will come to this part. So, instruction is fetched. Now if you study it's an immediate addressing or a non-immediate addressing.

So, if it an immediate addressing means what, the value of the operand itself is available in the instruction itself. So, add accumulator 32 immediate, I have to say whether 32 is the memory or it's an immediate. So, I say immediate; that means, value of what is value is present in the accumulator has to be added with 32 and get the result back in the accumulator, but sometimes we say the 32 H is a memory location in that case you have to go to the memory location get the value and then add it.

So, immediate addressing means yes then I set the code of ICC as 10, let us go by this flow and then we will see, immediate addressing then what we have to do? then it's 1 0 and if you come back to 10 you have note that 10 is nothing but execute; that means, if you fetch the instruction if it is a immediate addressing immediate you make ICC = 10 that means immediately you can execute. But if it is immediate addressing then you have to get the value of the operand from the memory. So, you make the ICC value equals to 01. So, what is the value of ICC 01 what will happen it will come over here ICC value is now basically from here is 01.

So, ICC is the 01; that means, you have decode and fetch the operand because the instruction is a instruction which is non-immediate that is the value is available in the memory location. Then you come over here then you check it's a direct or indirect. Direct is very simple the value of where the operand is available that address is available in the instruction itself. So, if it is yes fetch the operands from the given instruction address because see that I have said add accumulator 32 memory location.

So, the value what I have to add is available in the memory location 32. So, directly you fetch the operands in the instruction and then make ICC as 10, ICC as 10 means directly you can go for instruction execution. If it is no then indirect, so what was indirect add 32 if I say that it's an indirect instruction then I say add some accumulator 32; that means, the 32 memory location that in 32 memory location also I don't have the value of the operand. 32 memory location has some value that value is again an address where the value of the operand is exactly present.

So, it says fetch the operands from the given address in the instruction; that means, whatever is an indirect one. So, the indirect one means say for example, 32. So, memory location 32 now is an indirect instruction. So, you fetch the value from 32. So, what is 32? That is again an address where the operand is specified. So, you give the value of the value of memory location 32 to this and then you fetch it. So, again you can easily understand. So, if I say immediate 32 sorry direct 32 that means you fetch the operand from the given address in the instruction directly you can get the value of the operand from the memory. But here is the indirect, so you go to 32 fetch the memory location value and then you give it to this one. So, first you get go to 32 memory location get the address of the operand and then again fetch it then once in a indirect manner or direct manner you get the operand make ICC = 10, ICC = 10 means you directly you can execute.

So, now, all the fetching of the instructions are done, if it is immediate ICC you can directly make 10, bad execution no 01, no means you have to find out the values. So, if it is direct you can directly fetch the value from the address which is available in the instruction make it ICC 10 means execute, if it is no go in an indirect step and then make ICC = 10. So, now, ICC has been executed. So, once the ICC is 10 you execute it.

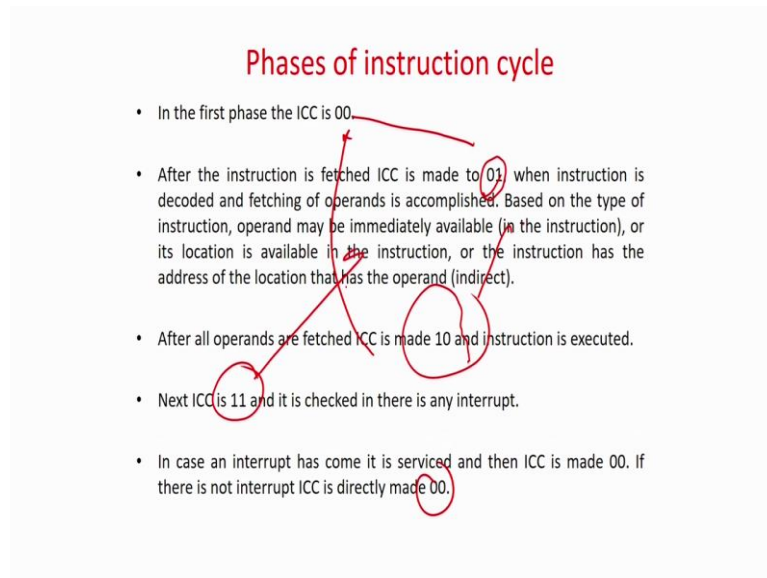
After it has been executed you have to check for interrupt it interrupt is no make ICC = 00 means next instruction will be fetched, if it is yes then make ICC = 11, ICC = 11 means again it will servicing the interrupt and after servicing the interrupt you again make ICC = 00, so basically this is the cycle.

So, in a nutshell you start with the ICC 00 code is instruction fetch, fetch it if it is available in the in instruction the data is available or the operand available immediately, make ICC = 10; that means, directly execute if it is a not an immediate make 01, 01 means they will be. So, in that case you will be executing this block and in this block you will be executing it in that case means immediate means what directly value is available need not do anything directly execute it, not available means you have go to this middle block. So, in the middle block what happens in the middle block if look at it is being transferred from 01? So, it is being transferred to this block.

So, in this block it is a direct or a indirect way of may fetching the instruction and then after fetching the instruction you go to the execute phase that is 10 and after a 10 you have fetched all the instruction and then sorry 10 means it is the instruction execution. So, after you get the value of 10 you execute and it is 0 1; that means, it is not for execute you have to get the data, to get the data you are going to go for that middle cycle and once it is done you get make the value of ICC 10. So, you execute it after execution you check the interrupt if it is the interrupt you make 11 that is the interrupt phase, service the interrupt and again make it 00 after that you go to the fetch cycle.

So, this diagram basically shows the cycles in terms of instruction cycle code, you start with 00 if everything is available in the code itself you directly make the code 10 execute it, if not make it 01 get the data from the memory or indirectly from the memory and then again go back to 00 if there is no interrupt if there is a interrupt make the code as 11 and again after servicing the interrupt make it 00 now that is what is the idea.

(Refer Slide Time: 35:28)



So, again what I have told you is written over in this slide. So, first is 00 then ICC is made equal to 01 based on what is the type of instruction. If it is immediate then you need not do anything and otherwise you have make it 01 and after all the operands are fetched you make it 10, that is now it's ready to execute 10 means ready to execute 01, means you have to fetch the data.

So, after data has been fetched you make it 10 and then execute it, if there is a interrupt you make the code 11 otherwise if there is a no interrupt you directly make it 00 and keep on doing it. So, basically the cycle is 00, 10, 01, 10 and back. Sometimes after this, this may come, but otherwise it's 00, 01, 10, 00 interrupt means it will come basically.

(Refer Slide Time: 36:13)

Simple assembly language program execution

- Instruction Fetch
- The Program Counter (PC) has the address of the instruction to be executed. So, value of PC is loaded into the Memory Address Register (MAR) and the control signal to the memory is Read.
- The content of the memory location (specified in the MAR) is read into Memory Buffer Register (MBR).
- The data from the MBR is transferred to the Instruction Register (IR)
- Following these data transfers, the PC is incremented by 1, so that it now contains the address of the next instruction to be executed.
- It may be noted that if the current instruction is a Branch or Jump then the PC gets the value of the address specified in the Jump or Branch instruction.

So, anyway this discussion is already I have done. So, instruction fetch, so what happens in that cycle, then instruction decode and instruction execute.

(Refer Slide Time: 36:22)

Simple assembly language program execution

- Instruction Execute
- Based on the type of instruction, execution may involve data transfer between the CPU and the I/O module. Also, arithmetic and logical operations may be performed on the data, as well as some instructions such as jumping to another location involve updating the PC to the address of the jumping location.
- The example given below illustrates the execution of a simple code.
- *We need to add two numbers that are in memory locations FF0 and FF1, and finally store the result in memory location FF2.*

I will now it is better basically without going into the theory you can read over this theory. So, it is taken in a nutshell it is represented program counter then how it is brought etcetera and then basically how it is executed. So, now, it is better that we take an example rather than taking so much about theory we will take an example that in a memory location there are two memory location FF0 and FF1 and FF2 is the place where I have to store the result. So, some data is

present in FF0 some data is present in FF1 I have to add these two numbers and store the value in FF2. So, this is what I want to do and I am going to show you with an example.

Now, these are the data FF0 and FF1 are the two locations where data is present and you have to write the value in FF2. So, this corresponds to the data memory of one Neumann architecture, but then also some where the instruction should be present that instruction which will do the adding for you.

(Refer Slide Time: 37:10)

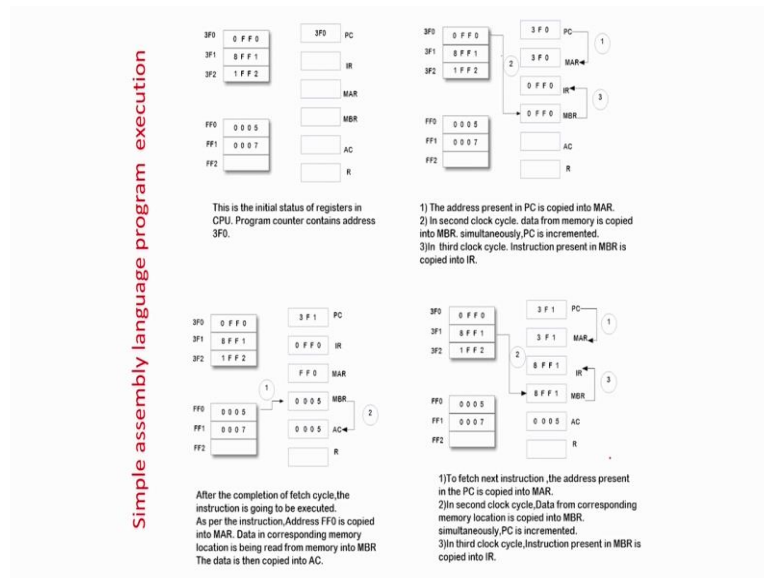
Simple assembly language program execution

- The assembly language program is as follows:
- 1st Instruction **LDA FF0**: The contents in memory location FF0 are loaded into accumulator. After the instruction is executed accumulator stores value 5.
- 2nd Instruction **ADD FF1**: The contents in memory location FF1 is added to accumulator. The final result is stored in accumulator. So 5+7 addition is performed and result 12 is stored in accumulator.
- 3rd Instruction **STA FF2**: This instruction stores the contents of accumulator in memory location FF2.
- If we assume that the first instruction is store in location 3F0, the following diagram demonstrates step by step execution of this code.

So, let us assume that there is this is the code right, this is the code I will tell you and that some where the code also has to be placed in the memory. So, we are assuming that the memory location for the starting of the code is 3F0. So, what is the first instruction it will say that load FF0 that is, so first data is FF0 so you load the value of FF0 in the accumulator, then you add the add FF1. So, what does it mean? It means that whatever is the memory value available in FF1 that you add with accumulator. Now accumulator has the value present in FF0. So now, we will have FF0 + FF1 and the value will be stored in the accumulator.

Finally, STA FF2 means store the value of accumulator in the memory location FF2. So, these are the 3 instruction LDA FF0, ADD FF1 and STA FF2. So, this 3 instructions also will be stored in the memory because it's a von Neumann architecture and the number it starts from FF0. So, this is basically your simple code and the memory architecture, I will now go step by step.

(Refer Slide Time: 38:02)



So, this is your architecture. So, you see the 3F0, 3F1 and 3F2, these are the data memory first instruction is 0FF0, 0 is the code as I told you for fetching an instruction from the memory location this is the memory location this is the memory location FF0 in the accumulator. So, 0 is the op code here, FF0 is the memory location that is a direct memory direct instruction, that is not an immediate not an immediate one because FF0 is not a data, FF0 is basically points to that memory this one actually pointing if you look at it, it is basically pointing to this memory location that is FF0.

So, it's a direct addressing. So, if you just find out what is the value of this location that is 5 will be loaded. 8 is the opcode for add, add accumulator; that means, whatever value is in the accumulator you add whatever is the value of FF1 memory location and store it back to the accumulator. 1 is again the opcode for right back what is store. So, whatever value is stored in the accumulator you store to this FF2. So, FF2 to be stored.

So, now, this is the basic memory configuration important registers are program counter, instruction register, memory address register, memory buffer register, accumulator and some normal register is available to us. So, program counter is always pointing to the first instruction that is FF 3F0. So, program counter is pointing to this.

Now, next, next what happens? So, the program counter is now pointing to 3F0 that is this memory location and this code will be 0FF0 will go to the memory buffer register because we want to read it; that means, whatever value in the PC that value will go to the address register

address bus of the memory and this location whatever is value over there will first go to the memory buffer register and add is an instruction it will go to the instruction register simple.

Program counter value will go to memory address register memory address register is 3F0, so this is the value of the 3 a register it will be fetched over here memory buffer register. Till now we don't know whether it is an instruction or data, but from the memory register I will go to instruction register because we always start with the instruction because nobody can start with the data an instruction than an instruction. So, first instruction, so it's an instruction. So, instruction decoder has now the value of this. Simple again repeating program counter has the value of 3F0 that is the memory location where the first instruction is there, it is loaded to memory address register this value memory gives the value to the memory buffer register, first instruction or the first memory access of a code that is always an instruction. So, it is going to the instruction register that is FF0.

Now the instruction will fetch over the code it will find out 0. So, what is that first 0 means? It is the opcode. So, it means that you have to fetch the operand from the memory location FFF0. So, if I assume that it is a 16 bit word, 4 4 4 4 the 16 bit word. So, only first 4 bits are reserved for opcode and the other 3 bits are reserved for the memory location address. So, in this case I can address a memory whose size is 2^{4+4+4} that is 2^{12} . If you want to go for higher this one then you have to go for a indirect addressing, then it will be continue over here and then the address will be of this one will be able anyway that is not a concern for us right now.

So, next stage what happens. Now, all this story has been done program counter is immediately incremented by 1, it is now start pointing to 3F1. Then what? Name instruction register already knew that I have to get the value from memory location number FF0. So, FF0 will be the value of FF0 will be fetched to the memory buffer register. Now there is difference between instruction fetch and a data fetch. So, this was the first instruction which was fetched that is 0FF0. So, it is going to instruction register. Now, the instruction register knows that instruction already be in there now I have to get the data. So, where the data is there? So, FF0 the value of FF0 is in the memory address register the memory will give the value of whatever is available in FF0 that is 0005 to the memory buffer register. Now it will not go to instruction register because this is a data which is already what we have to do has been decoded by 2 it will go to accumulator, 0 means load the value from the memory location to accumulator. So, accumulator has the value 5.

Now next see what happens now the program counter has gone to this one. So, 8 FF1, this is the new instruction. So, 3F1 means. So, 3F1 is the memory address this is the value 8 FF1. So, this from this memory location it will go to the memory buffer register and already in the last instruction here fetched the data, so this is a instruction. So, first for the instruction next cycle we got a data. So, now, again it is a new data, but is an instruction. So, that is 8 FF1. So, instead of going to the accumulator or anywhere else it will put the value of 8 FF1 into the instruction register. Now, 8 stands for adding, adding of where, whatever is in the accumulator that is 5 with whatever is the data available in FF1. So, it will add 5 to the data available in FF1 and store it accumulator back.

So, let us go to the next step. So, even let me zoom it over and this is your step. So, now, you see what I told you. So, again now instruction *PC* is now pointing to the next instruction, but currently we are executing this. So, it is having the value of instruction register 8 FF1. So, what it tells that I have to add, add what, whatever value is available in the accumulator that is the previous value, with whatever is the value available in memory location FF1 and I have to store it in some register or some accumulator in this case right.

So, now the memory buffer register will have the value of memory address register will have the value of FF1. So, FF1 will be memory address register; that means, this address and whatever value is available in FF1 that is 7 will go to memory buffer register that one will be added with the accumulator and stored back to the accumulator that is what is being done by this opcode 8. So, know the value of this one is $7 + 5 + 7$ is hex C, I get it right into the accumulator.

Now, the last instruction; the *PC* is pointing to this one. So, now, just previous to that access the memory for data. So, again I will access the memory for instruction. So, again the value of *PC* that is 3F2 will be the memory address. So, this is 3F2 this data will be put to memory buffer register so in fact, it's an instruction for the last instruction was a data access. So, now, this is an instruction access. So, now, the instruction will be going over here, so 1FF2. So, it will be decoded what 1 stands for whatever is in the accumulator please write back to the memory location that is specified over here.

So, memory location accumulator sorry accumulator has now the value of 000C that is the asset of the output where it has to be stored, 1 is saying that whatever is put in the accumulator you have to write back where I have to write back I have to write back in FF2; so again the last

instruction how it is executed? So, *PC* has now gone to the next instruction anyways not there for us, but it is incremented. So, now, the instruction was 1FF2, 1 says that whatever is in the accumulator write back and FF2. So, what is FF2? FF2 is this location. So, this FF2 will be copied to memory address register, but now the memory will be in a write mode, in all other steps it was in a read mode. So, it will be in a write mode. And what is the address? The address is FF2 it is address register and what I have write is the value of the accumulator. So, I will write the value of the accumulator in memory buffer register.

In all other cases it was happening the other way round what was happening what was available in the memory buffer register I was writing it to the IR if it is or the instruction what I was writing to the in a accumulator if it is was a or sorry or I was writing to the accumulator if it is was a instruction or in other way round means if it was an instruction I was writing to the IR and if it is some kind of data I was putting it to the accumulator.

But now I have to write back to the memory. So, what I am doing I am taking the value of accumulator and writing to the memory buffer register and the memory buffer register will write the value in FF2 that is C. This completes the execution of these 3 steps you read, add and again store back and this is actually done. So, after that the *PC* is calculating the value of 3F3, but that is again the next instruction which is not at all concerned for us.


So, this in this unit we have shown basically what is an instruction, basic idea, what are the components it have, how they are executed, how they are fetched, what is the life cycle of instruction. And with a very nice example we have seen that how an instruction is accessed, how it is fetched from the memory, then how it is executed, how operands are fetched from the memory, how they are operated and again written back to the memory or this completion of the instruction.

So, again towards the end we again also see some of the questions and how it meets the objectives. So, consider an instruction fetch cycle of this one, instruction fetch, execute, decode explain the purpose of each of the 4 phases.

(Refer Slide Time: 46:45)

Questions and Objectives

- Q1: Consider an instruction cycle consisting of fetch, decode and operands fetch, execute and interrupt phases. Explain the purpose of these four phases.
- **Comprehension: Explain:--** Explain the Fetch and Execute cycle of an instruction.
- **Comprehension: Explain:--** Explain the use of indirect cycle to access data during instruction execution.
- **Comprehension: Describe:--** Describe the use of IO devices and incorporation of interrupt cycle in an instruction cycle.




So, we easily go for the objective which says that explain instruction fetch execute decode cycle. Explain the use of indirect cycle that is another objective. Briefly explain using an example how to provide CPU services using an IO interrupt that is as I we have already explained discussed how interrupt is done, what is interrupt service routine. So, we after doing the unit obviously, an answering this question you will be able to meet the third instruction, third objective. This, write the use of IO devices and incorporation in the instruction cycle. So, these two objectives actually match question number 2.

(Refer Slide Time: 47:23)

Questions and Objectives

- Q3: Give a scheme to identify the four phases in an instruction (fetch, Decode and operators fetch, execute and interrupt).
- **Comprehension: Explain:--** Explain the Fetch and Execute cycle of an instruction.
- **Comprehension: Explain:--** Explain the use of indirect cycle to access data during instruction execution.
- **Comprehension: Describe:--** Describe the use of IO devices and incorporation of interrupt cycle in an instruction cycle.



Give a scheme to identify the 4 phases in an instruction that is using the ICC. So, it can match the instruction, that is the objective of explain the fetch and executes cycle of an instruction and explain using a simple example how an assembly language code is executed. Again the first and the second and third instructions are basically objective are clarified over here; that means, if your assembly language have some interrupts then of course these two inter interrupts objectives are met otherwise the first objective is met. So, in fact we have now studied how basic instructions what is the basic instruction, how it looks and how this unit actually encompasses these 3 objectives.

The next week, next unit we are going to see how we can design instructions on a specific canonical format. At that time we have kept this in a very generic manner that is the opcode, this is the instruction, this is the part of data, this is the part of operands how can we make it more formal as suitable for our computer architecture. That is what we are going to study in the next unit.

Thank you.