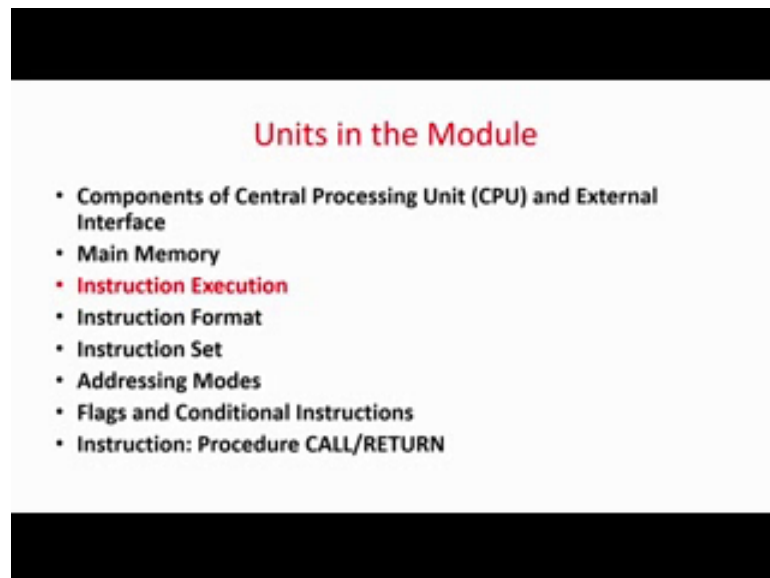


Computer Organization and Architecture: A Pedagogical Aspect
Prof. Jatindra Kr. Deka
Dr. Santosh Biswas
Dr. Arnab Sarkar
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Addressing Modes, Instruction Set and Instruction Execution Flow
Lecture - 09
Instruction Execution

So, welcome to the module on addressing modes instruction set and instruction execution flow, lecture number 3, in the unit number 3.

(Refer Slide Time: 33:00)



Since the last 2 units we have basically seen what is the basic structure and the components of a CPU and external interface and we have seen basically the black box architecture of a memory. So, with this we have built enough background to understand the basic stuff which we are going to cover in this module that is, what is an instruction, how it is executed, what are the different types and formats instruction set design and what are the typical use of instructions to call functions and return procedure.

So, with the background build now we are going to see the first step that given a very simple instruction how it is executed. So, that is this unit is dedicated to understanding the execution of an instruction furthers the basic idea required for the memory architecture as well as for the how the CPU is organized is means suffice.

(Refer Slide Time: 01:25)

Unit Summary

- The operations or tasks that must be performed by CPU for executing an instruction :
- **Instruction Address Calculation (IAC):** Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number to the address of the previous instruction.
- **Instruction Fetch (IF):** Read instruction from the memory location into the processor.
- **Instruction Operation Decoding (IOD):** Analyze instruction to determine type of operation to be performed and operand(s) to be used.
- **Operand Address Calculation (OAC):** If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
- **Operand Fetch (OF):** Fetch the operand from memory or read it from I/O.
- **Data Operation (DO):** Perform the operation indicated in the instruction.
- **Operand Store (OS):** Write the result into memory or out to I/O.

So, as I told you that for a pedagogical perspective. So, we are first going to see what is the summary of this unit. So, in this case we are first going to see what is the performance or what is the functions which are performed by a CPU for executing an instruction.

So, basically as it is a Von Neumann architecture so the instruction as well as data are in the memory. So, first what happens? So, whenever you want to execute an instruction, first we have to calculate the address of the instruction and in which memory location and at what address the instruction is there. So, that is the first step. Secondly, the instruction will be fetched, now it will be fetched and loaded into a special register call a instruction register IR. So, now, instructions will be decoded that what he has to do sometimes, it may be very simple like shift 2 numbers or add 2 numbers or do bitwise shifting and sometimes it can be very complicate like a matrix multiplication. So, that is actually called the instruction fetch.

Then you have to find out that is the operation decoding that what is the instruction expected to do that is all the instruction basically has 2 minimum characters, minimum feels one is the opcode that is the operations it has to do that is again binaries because all the instructions are basically representing binary. So, one part of the address will be dedicated for instruction decoding that is, that is what the instruction is expected to do and it has to be decoded. Then based on that it may have it may operate on 2 numbers, it

may operate on 2 operands it may operate on single operand. Like for example, if the instruction just compares whether number is greater than 0. So, it is just a single operand instruction or so add, but if it adds 2 numbers then it is 2 operand instruction.

So, next you have to find out that what we you find out all those things when you decode an instruction, after decoding you have to find out whether you want one operand to be fetched, whether you want 2 operands to be fetched or whether the operand is given itself with the instruction. For example, I may have an instruction that is called immediate addressing mode where the data is given in the instruction itself. For example, I may want to compute 5 plus 5. So, this 5 plus 5 is the given in the instruction itself.

So, you need not go and bring the data from the memory itself so; that means, you have to go for operand address calculation, but sometimes I may see that I want to add 3 with the data which is available in the memory location may be the variable of the memory address is 0003 h. So, in that case we have to go for operand address calculation and then we have to fetch the operand from the memory.

So, after that is done you have to do the operation on the data and then you have to store back the data in the memory, if it is required. So, in a nutshell basically we store fetch the instruction, decode the instruction and then find out whether some operands has to be fetched from the memory and then fetch the operands do the operation and store it in a memory location.

So, generally we call this whole thing in a very few steps instruction fetch, instruction decode, instruction execute. So, in decoding we generally involve decoding this instructions as well as bringing the values from the memories or if the instructions are immediate addressing mode then take the values from the instruction itself that is decoding instruction fetch decode. In decode we do all the stuff and execute means simply we execute the instruction, but in a details are given over here, so again.

(Refer Slide Time: 04:32)

Unit Summary

- From the point of view of the user program, an interrupt is just that: an interruption of the normal sequence of execution. If an interrupt is pending, the processor does the following:
 - It suspends the execution of the current program being executed and saves its contents. This means saving the address of the next instruction to be executed (current contents of the program counter) and any other data relevant to the processor's current activity.
 - It sets the program counter to the starting address of an interrupt handler routine.
 - After the interrupt handler routine is complete the control returns to the current program at the point from where the interrupt was called. All the data that were saved are re-loaded to the appropriate locations so that the execution of the current program may continue.

So, generally from the users perspective we see that instruction 1, then instruction 2, then instruction 3 it will keep on going if there is a jump instruction you will jump to the position again come back and so forth.

But there is also very special stuff which is actually called the interrupt, sometimes based on requirements you may want to interrupt the existing flow of instruction, that will be based on requirement may be a code is executing at that time you want to move the mouse. So, in that case the program will be interrupted and your mouse misplace will be displayed and again the code will execute. So, interrupt so an interrupt is there. So, basically suspense the normal code flow and immediately it has to service the interrupt for example, the code is executing I am moving the mouse. So, the mouse is been changed.

So, what happens, so the basic say for example, I am listening to music and at the same time I am moving the mouse. So, for as (Refer Time: 05:22) limited fraction of time will not in a; is in a very high speed processing in a module PC you do not get it time difference, but what happens is that after is executing of the current instruction always the system or the CPU checks whether there is a interrupt, for example, now I have moved the mouse.

So, after execution of the instruction it will check the mouse has been moved it has to be displayed. So, that is actually called the interrupt service routine that is displaying the

movement of the mouse in the appropriate place, then before servicing the interrupt what are the current state of this code like for example, a code was adding two plus 3 plus 4 plus 5 in 3 instruction.

So, 2 plus 4 has already been added then the mouse movement was there is interrupt then again you service the interrupt that is the mouse movement has been show, then again you will come back to the normal code then again you have to recollect what I have already done that is already I have adding 2 plus 3 that I have to remember recollect that and then do the final calculation and keep on addressing.

So, before addressing the interrupt or servicing interrupt you have to store the present status of the code like a program counter, the status register etcetera which is in a stack. So, this is actually called saving the current program status (Refer Time: 06:26) then you run the interrupt service routine, service the interrupt and again come back and recollect what has been already return that is use pop of the values from the stack for the corresponding registers and again then go ahead and stack executing the normal instruction.

In a nutshell after every instruction is executed if check whether a interrupt has come, if an interrupt has come save all the stuff like program counter, instruction register values and some register values of that is the intermediate stuff of your code in a stack then service the interrupt again come back from where we have left and an that point of time you again recollect everything back and go ahead.

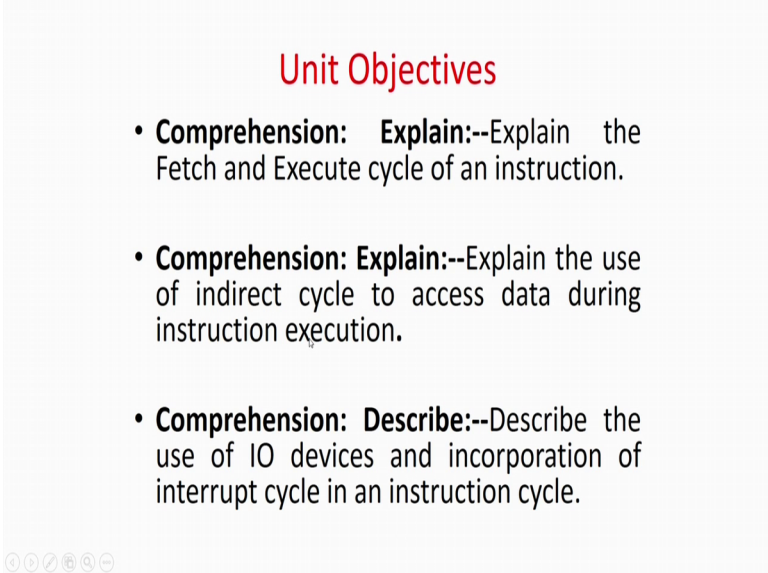
So, therefore, interrupt means not only servicing interrupt you have to store back everything. So, after coming back you can start from where you have left. So, that is actually a basic idea of instruction execution that is you fetch decode and execute and. In fact, of the instruction interrupt then you have to service the interrupt service and then again come back.

So, what are the objectives of this unit is the comprehension explain the fetch and execute cycle of an instruction, that we will be able to explain what is the fetch and execute cycle of an instruction or a how a instruction is fetched decode and executes, then there is a direct mode as well as a indirect mode. So, we will be also able to express indirect mode of a instruction execution direct mode is some operands will be the values of your operands will be either in instruction or you can directly finding the value of the

instruction opp is of the operates the operands of the memory, but indirect means you will be re directed to a memory location and in that memory location we will not have the data.

But in that memory location you will have a another pointer which will go to the data that is a indirect job, sometime if I see that load something from memory location 3030. So, the data is expected into the memory location 30 30, but in indirect mode 30 30 will have another memory location and that memory location will have the data. So, that is actually indirect way of doing, it is what is the benefit etcetera we will see in coming words and finally, describe the type of IO devises and interrupt cycle.

(Refer Slide Time: 08:27)



Unit Objectives

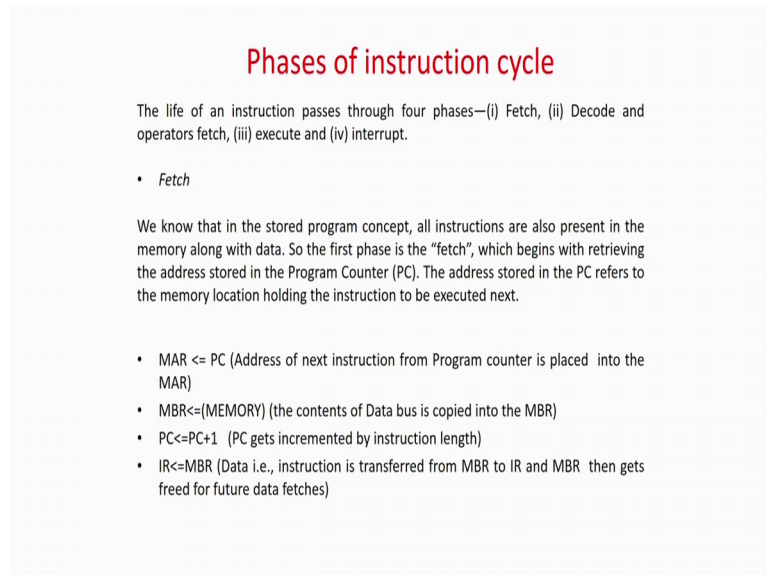
- **Comprehension: Explain:**--Explain the Fetch and Execute cycle of an instruction.
- **Comprehension: Explain:**--Explain the use of indirect cycle to access data during instruction execution.
- **Comprehension: Describe:**--Describe the use of IO devices and incorporation of interrupt cycle in an instruction cycle.

So, generally interrupts basically happened into IO, as I said like I am moving a mouse. So, the interrupt has to be serviced. So, for example, I do not do that then what happens you are running a long code and I am moving the mouse, I am not allowing the interrupt. So, what is going to happen the whole code will run and then only we will able to see the mouse movement, then what is going to happen then the main problem will be like for example, I am showing the PPT a code is been executed and I do not allow the mouse interrupt.

Then what will happen only after the whole PPT has been shown then only the mouse movement will be displayed. So, this is not going to survive (Refer Time: 08:58). So, therefore, that every time the PPT is running means some code are executing and this

been displayed in the screen. So, when I move a mouse the servicing the instruction by interrupt service routine and after servicing is coming back. So, we will be also able to describe what is a interrupt service routine and how its handles for IO device.

(Refer Slide Time: 09:18)



Phases of instruction cycle

The life of an instruction passes through four phases—(i) Fetch, (ii) Decode and operators fetch, (iii) execute and (iv) interrupt.

- *Fetch*

We know that in the stored program concept, all instructions are also present in the memory along with data. So the first phase is the "fetch", which begins with retrieving the address stored in the Program Counter (PC). The address stored in the PC refers to the memory location holding the instruction to be executed next.

- $MAR \leftarrow PC$ (Address of next instruction from Program counter is placed into the MAR)
- $MBR \leftarrow (MEMORY)$ (the contents of Data bus is copied into the MBR)
- $PC \leftarrow PC + 1$ (PC gets incremented by instruction length)
- $IR \leftarrow MBR$ (Data i.e., instruction is transferred from MBR to IR and MBR then gets freed for future data fetches)

So, now we will see the details of the instruction cycle. So, the instruction cycle basically has fetch, decode an operation fetch operate sorry operands fetch execute an interrupt, if interrupt if it is there. So, basically as I told you decodes and operand fetch sometimes we call as a decode itself and then execute. So, fetch decode execute, fetch decode execute these are the 3 terms you always use decode means operators fetch is also coming in the decode. So, fetch is fetch that is a memory we have already seen memory, there are cells and you can access one cell at a time.

So, first whatever the instructions is available over there is fetch. So, how it will be fetched, there is special registers which is actually called the PC that is called the program counter. So, the program counter will initially have the value of the memory location where the instruction starts, for the time being let us assume that the instruction first instruction is located at 0000 H memory location. So, PC value will be 000 x. So, PC means program counter is having the value of 0000 x. So, immediately that instruction will be fetched and that will be exe decode and executed. So, whatever value of the program counter that particular instruction is fetched.

So, how it is fetched? So, as you already knowing a one architecture both the instruction as well as the data are in the instruction is in the memory, we do not differentiate that way. So, fetching an instruction or fetching an operand of the data is the same way. So, first we say that memory address register will have the value of PC because you have to get the value of the instruction to a form the memory.

So, is already one (Refer Time: 10:47) you have to give the address of that instruction, where is the address of that instruction the address of the instruction is in the PC. So, memory address register will now have the value of the PC, next 1 g b is get given and the registers is in read for the memories in read mode. So, memory that value in that memory location which is mentioned in the PC will be located to memory buffer register that is saying the 0 0 0 0 memory location which was in the PC is now being fetch that is rate and the memory will be dumb the value of the instruction that is present in memory location 0 0 0 0 two memory buffer register.

So, memory buffer register know will have the instruction which was in the memory location 0000 that was pointed by the program counter, after doing this program counter is incremented by 1 because now it has to in the next instruction cycle it has to take the next instruction and then the memory buffer register that is the memory where is your got the instruction right now will be given to the special register which is call the instruction register that is the (Refer time: 11:44).

So, basically what happens? The instruction is taken from the 0eth memory location via memory buffer register and it is put into a special purpose register call the instruction register which will now handle the instruction. So, it now knows first instruction has come which was pointed by the program counter. Now I have to decode and execute the instruction, specially it has to be noted that now the program counter will be incremented by 1 because now I want to execute the next instruction in the next cycle then the next stage.

(Refer Slide Time: 12:15)

Phases of instruction cycle

Decode and Operands Fetch

- Next the instruction is decoded. Instruction is partitioned into two parts—(i) operation to be performed i.e., determined by the OP Code and (ii) fetching the operands on which the operation is to be performed. Based on the mode of addressing used in the instruction the steps (and path) of fetching the operands differ.
 - Immediate:** The operands are specified in the instruction itself.
 - Direct:** In this case the memory location of the operand is specified in the instruction. This is a direct fetch phase and operands are brought to the location (e.g., user registers, accumulator, MBR etc.) as per the instruction.
- It involves the following steps
 - Step 1: MAR \leftarrow address of operand in IR
 - Step 2: MBR \leftarrow memory cell whose address is given in MAR

Now, the instruction is already presented the instruction register. So, now, what we have to do we have to decode and fetch operation.

Sometimes some people actually say decode and operand fetch as 2 different parts because you decode the instruction and then based on the requirement you have to fetch it, but sometimes again you club them together it depends. So, of course, a instruction at least has 2 parts, one is the operation to be perform that is defend by the opcode because a truth table what I have to do that is the job of instruction.

So, instruction must always have a specific part which is actually call the half code of the one operation it has to do. So, may be if the I assume that there only 3 instruction in my computer. So, may be 0 0 for add, 0 z 0 1 for fetch and 0 2 for right. So, I will add bring some numbers and the half code will be 0 1 then I added it. So, the half code will be 0 0 and one by one divided by the half code will be 1 1, so 000 1 1 1. So, these are 3 two bits required and the 3 combinations if the instructions of a computer has only 3 instructions.

So, there is some part of the instruction which is used code at (Refer Time: 13:18) tells these computer what to do. Then depending on that you have fetch the operand, operand, operands. That is very important. Because the instruction means you have do something and do something on some operand operands. So, the operand values will be also this specified in the instruction. So, depending on the instruction side type there can be two operands, 3 operands 4, operands. In fact, theoretically speaking n number of operands

given a instruction. But if you have such long instructions I think you have already seen what is going to happen then your memory width will be in huge. So, it do not want to do that it you may have memory whose width may be 1 0 2 8 or 1 0 2 4 you do not like to do that. Therefore, you always restrict our instruction let to certain limit.

Generally you may have two operate operands in that case. So, in that case phases 32 bits instruction will suffice. So, now, there is something called addressing mode. So, there are different types of addressing mode. Now what is an addressing mode? The addressing mode means it will tell you that when of the value of the operands specified. The most simplest word is immediate. We can always say that these one instruction call increment I N C R increment and the value may be 3 this is one of the most simplest instruction it test that increment had varies what you have to increment whatever value of the integer which is given in the instruction itself. So, this is actually call the immediate instruction.

In that case operand fetch is not required at all. That is the in operand of the operand operand is available in the instruction itself. But sometimes it is not very easy because you will the idea is something like that this is a memory cell.

Now, 1 1 instruction, some part will taken by the off code that is of the n code what the instruction will do the remaining part is be limit will be giving you to the right the operands. So, for examples, this may be it may 8 bit assume then what happens maybe I could 3 bits to have the number of operations. So, there are 8 operations. So, 3 bits one for that. Now we have only 5 bits remaining; so, if have 5 bits remaining. So, whatever data you can put can be maximum of 2 to the power 5. So, 0 to 2 the power 5 that is 32 numbers can already represented, but if I want to add 100 plus 100 then you cannot write instruction like this. So, this not going to sup have may purpose.

So, better I can do I need a longer space to put the operands, but longer space to put the operands means I have to make the instruction like longer either memory not very good. So, what I will try to do? I will have direct addressing mode or a indirect addressing mode. So, direct addressing mode means here I will put the value of the memory may be I will write 30, 6 bits. Now this is going to point to a memory location and that memory location you have the exact value. So, you can assume that the whole memory may be say 32 bits correct and let us assumes that this is 32 bit and we have let me re-draw properly.

So, the main problem is that immediate addressing is very easy to handle, but then you will be limited by the range of data for this is the memory weight this 32 and let me have a instruction where I assume 10 bits (Refer Time: 16:27) may be 1000 instructions were 10 bits are result for the instruction type and then 22 is for the operands.

(Refer Slide Time: 16:26)

Phases of instruction cycle

Decode and Operands Fetch

- Next the instruction is decoded. Instruction is partitioned into two parts—(i) operation to be performed i.e., determined by the OP Code and (ii) fetching the operands on which the operation is to be performed. Based on the mode of addressing used in the instruction the steps (and path) of fetching the operands differ.
 - Immediate: The operands are specified in the instruction itself.
 - Direct: In this case the memory location of the operand is specified in the instruction. This is a direct fetch phase and operands are brought to the location (e.g., user registers, accumulator, MBR etc.) as per the instruction.
- It involves the following steps
 - Step 1: $MAR \leftarrow \text{address of operand in IR}$
 - Step 2: $MBR \leftarrow \text{memory cell whose address is given in MAR}$

So, may be what I can do is that. So, this 22 bits are result for the operands. So, what I can do is that I can put some value directly here then my range will be 2 to the power 22, but. In fact, generally we may not have a single kind of thing single operand instruction will not have we may have 2, so one operand here one operand here. So, may be now 11 and 11, 22 plus (Refer Time: 16:59).

So, the range is 2 to the power 11, 2 to the power 11 and 10. So, 10 bits are result for operation type if first operand is 11 bits second operand 11 bits. So, you cannot have 2 lakhs and 2 lakh number range kind of an operation because you are limited by 2 to the power 11 that is nothing, but $2^{10} \cdot 2^4$. So, range is $2^{10} \cdot 2^4$ range is $2^{10} \cdot 2^4$. So, two operands range is a 0 to $2^{10} \cdot 2^4$ if you want a have negative numbers then again it will be happen.

So, what is the better of doing it? I actually put some memory location. So, now, the range is 2 to the power 11, 2 to the power 11 you can access memory who sizes 2 to the power 11 or 2 to the power 11. So, let me have a memory like this whose size is 2 to the power 11. So, it will tell whatever memory location you will have, you will have point

one memory. If the memory reads you 32, so the your data is present over here. So, now, what is the width of the data? Width of the data is 32 bits.

So, is the huge number to the 30 or 32 huge number you can do very high position calculations. So, therefore, you always go for direct or there is non immediate mode of instruction that is in the instruction you will give an address of a memory where the data will be store. So, you can have a wider range of numbers should be represent. So, that is actually called direct instruction. So, in that case you have to fetch the operand. So, where the operand will be specified? The address of the operand will be specified in the instruction itself.

(Refer Slide Time: 18:19)

Phases of instruction cycle

- *Decode and Operands Fetch*
- Next the instruction is decoded. Instruction is partitioned into two parts—(i) operation to be performed i.e., determined by the OP Code and (ii) fetching the operands on which the operation is to be performed. Based on the mode of addressing used in the instruction the steps (and path) of fetching the operands differ.
 - Immediate: The operands are specified in the instruction itself.
 - Direct: In this case the memory location of the operand is specified in the instruction. This is a direct fetch phase and operands are brought to the location (e.g., user registers, accumulator, MBR etc.) as per the instruction.
- It involves the following steps
 - Step 1: MAR ← address of operand in IR
 - Step 2: MBR ↔ memory cell whose address is given in MAR

Handwritten annotations: "ADD 32 x 32H", "32", "32H", "1", "2 11 2 11".

So, what will have to do do that? Some sub steps will be required. So, again you have take the value of the operand which is that is the address of the operand from the instruction put it in the memory address register and again the memory that is now the data of part of the memory. So, again the memory buffer register will have the value. Like for example, I may have say ADD is an instruction and then I say 32 H then 32 H then I act this side is not possible 33 x and 33 x; that means, there is a memory and I am referring to 32 H location over here and I am this is locating 33 H, but now this is 32 bits.

So, I can add two 32 bits numbers together. So, this one this will now go to the address bus the address bus will point over here and this one will bethis data will be fetched, next

this data will be fetched and you can add this two numbers. So, this is actually call decode and fetch.

(Refer Slide Time: 19:14)

Phases of instruction cycle

- Indirect: In this case, the memory location of the operand is specified in another memory location whose address is present in the IR. So this is called indirect fetch phase.

It involves the following basic steps:

- Step 1: $MAR \leftarrow IR \text{ address}$ (IR address has the address of a memory location that has the address of the operand)
- Step 2: $MBR \leftarrow \text{memory cell whose address is given in MAR}$
- Step 3: $IR \text{ address} \leftarrow MBR$ (IR is now in same state as if direct addressing had been used, and the next steps are that of direct addressing)

There is another instruction the benefits etcetera will show in the later units, but that is something a call indirect mode.

So, what is an indirect mode? So, in an indirect mode what happens you can find out that indirect mode, what indirect mode basically what happens that is in the last instruction what will have seen? In the last instruction we said that the data will be present the address of the data will be present in the instruction. So, now, you can go to a memory location and in the memory location that data will be presented indirect means basically there is one more step which is going over here.

So, in indirect mode what happens it is just one more re direction from the direct mode that is this is an instruction this is your opcode this is referring to a memory location, in that memory location in the direct mode we have the data itself, but now it will again have some address to the another memory location were the data will be there. Now, what are the advantages disadvantages will cover later in later the units and modules. But idea is that double the whole that is you point from here to here, here also your data will not be there, but the data will be in another memory location whose address is given over here.

So, again it actually represents more wider range of memory or wide of you can excess more wider range of memory over here. So, for example, as we are taking there 11 bits and there 11 bits, so what is the range of memory you can excess? Only 2 to the power 11, but say your memory is an level of Giga bytes, 2 to the power 11 is already in the level of kilo bites. So, I cannot have very wide address over here so, but if I have the address of the address here then again it is a 32 bit. So, you can excess a 32, 2 to the power 32 size memory by this method.

So, if you have taking the direct approach which was in the previous case. So, the address of the operand was present directly over here you go to the memory location you find it. But early this sizes of the memory that can be excess 11, 2 to the power 11. But if I have a larger size memory then you will going to have the indirect one, that is from here if point f errors of the data will not be there here will have the address of the data. So, here is again 32 bit. So, now, it is 2 to the power 32. So, again full fledged memory can be accessed over here. So, that is the again the advantage of indirect mode, but more details will see in the later units here is what is the idea.

So, here the steps will be slightly indirect. So, what will happen?

(Refer Slide Time: 21:44)

Phases of instruction cycle

- Indirect: In this case, the memory location of the operand is specified in another memory location whose address is present in the IR. So this is called indirect fetch phase.

It involves the following basic steps:

- Step 1: $MAR \leftarrow IR$ address (IR address has the address of a memory location that has the address of the operand)
- Step 2: $MBR \leftarrow$ memory cell whose address is given in MAR
- Step 3: $IR \text{ address} \leftarrow MBR$ (IR is now in same state as if direct addressing had been used, and the next steps are that of direct addressing)

First the memory address register will have the instruction register. Instruction register have the address of a memory location that have the address of the memory operand (Refer Time: 21:51). Memory address register initially will have that first whatever is 2

to the power 11 whatever register address you give it will be there, it will point to the memory, memory will give the data, but again that memory here have again some address that will be again fit to the memory buffer register and again then the data will be coming out over here. So, that is multiple steps. So, memory address register is first will on the instruction register first will give the value to the memory address register, memory buffer register will have the memory cell whose address is in the MAR.

So, now it is actually again this value will be again fed to memory address register and finally, this value will be given then actually indirect address. Now you have a wider indirect means wider range of memory can be accessed.

(Refer Slide Time: 22:39)

Phases of instruction cycle

- *Execute*
- The instruction is executed based on the OP Code on the operands
- The results of the execution of instructions can be classified as
 - **Data transfer:** Examples are data transfer from a memory location to a register, or vice versa, Read and write data from hardware devices etc.
 - **Arithmetic and logic operations:** Examples are addition, subtraction, multiplication, negation of each bit, comparison etc.
 - **Control flow operations:** Examples are Branch to another location in the program, conditionally branch to another location if a certain condition holds etc.

Now, instruction has been fetched it has been decoded what to do and the operands are also been fetched now your phase executing. So, this is actually the job of the CPU of the processing unit will have to do the job that is simple. So, based on the off code you have to either do any 3 of these things that is data transfer, either you read from the memory write from the memory arithmetic and logic operation that is add subtract multiple logical or not or sometimes you have to branch. That is based on the answer of instruction either you will be the next instruction or will jump (Refer Time: 23:10) 20 steps ahead.

So, basically there are 3 type of instructions data transfer, arithmetic and logical, and finally, control that is if then else kinds of a statement. Arithmetic means plus minus,

data transfer means it scan f, free f. So, scan f, free dividing in the C if the machine or in the architecture version it will be load and store.

(Refer Slide Time: 23:27)

Phases of instruction cycle

- *Interrupt*
- Instruction passes through "fetch-decode and operands fetch-execute". However, sometimes an event external to the currently executing program needs to get the CPU that causes a change in the normal flow of instruction execution. Interrupt is usually generated by hardware devices external to the CPU like Keyboard, mouse, screen etc.
- Step 1: Save the value of PC in a stack (after servicing the ISR the current program should start from the instruction before which interrupt had arrived)
- Step 2: $PC \leftarrow \text{ISR address}$ (Now the instructions from memory locations corresponding to the ISR will be executed)
- Step 3: CPU execute the instructions in ISR and then return.
- Step 4: $PC \leftarrow \text{Pop}$ the return address from the stack (then execution of the last program continues)

The diagram illustrates the flow of an interrupt. It starts with a normal instruction flow, then an interrupt (ISR) occurs. The PC value is saved in a stack. The CPU then executes the ISR. After the ISR, the PC value is restored from the stack, and the original program continues. Handwritten red annotations include a box around 'PC', a circle around 'ISR', and a circle around 'PC10'.

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

So, what happens? So, after no interrupt can offer where the instruction is been instituted, instruction 1 2 3 4 5 6 7 8 9 10 is going on it between no interrupt can come. But after a instruction have been executed it will check with the there is an instruction manner 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 of when the interrupt has be occurred; that means what, after executing instruction 9 PC has become 10 and then interrupt has occurred. I have deducted the interrupt then while coming back you have to again restart your code form 10th location or where or that is the PC value it be stable location.

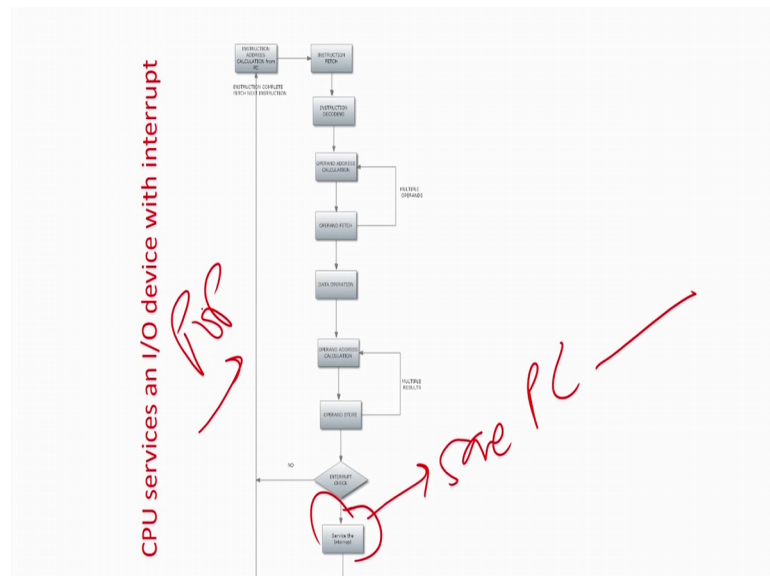
So, what we will do? You will save the value of program counter all the registers intermediates value in stack and then you will go to the instruction service routine who will instruction service routine nothing, but another code itself or a code module with the some instruction is a jump. So, how can you jump? Now the instruction service routine address will be loaded into the PC, but we 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 of to be a function it will jump and again come back.

So, I save the values of PC equal to 10 and save will other intimidator register and then I put the address of the interrupt service routine to PC. Now this is will be start pointing out to the instruction which is in the interrupt service routine, may be this is your memory where your PC 10 it has to be executed, but some interrupt occur then your PC jumping to here that is the ISR insert 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 (Refer Time: 25:23) 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 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 you have already seen if be in a direct, indirect, immediate fetch operand if there multiple you have to do multiple times operation fetch. But I told you that vary now less large number of operands in single instruction is a 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 be your logic or an 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.

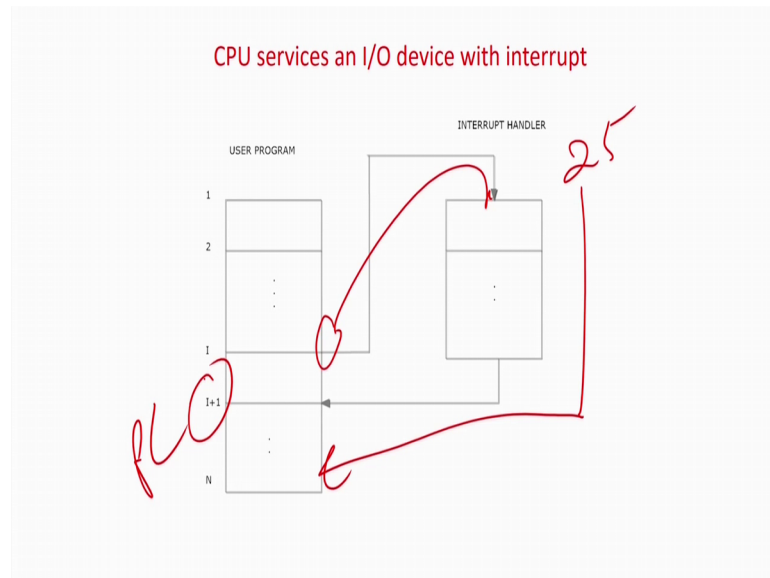
Like for example, if you said that add a plus b and store it to some place or instruction may be simple like store something to something. So, in that case the data operation will be nothing will store of the instruction. So, you have to also go for operand address calculation where you want to store the result, you store the result and you go on back. But sometimes your operand address calculation may be very trivial like add accumulator with memory location 32; that means, whatever the data is in a memory location 32 as to be added to the accumulator and stored back into the accumulator itself. Accumulator is a register, so operation 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 as arrived or not. We do not check any interrupt in between because if you do it you may go into a dead lock state that one instruction is not completed you start another instruction and so much lot of troubles on 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 implemented you keep on in fetching the instruction decoding it executing and go on, but if it is not then you have to service 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, ALU is etcetera and after you service the interrupt if the again reload the value of PC you save everything.

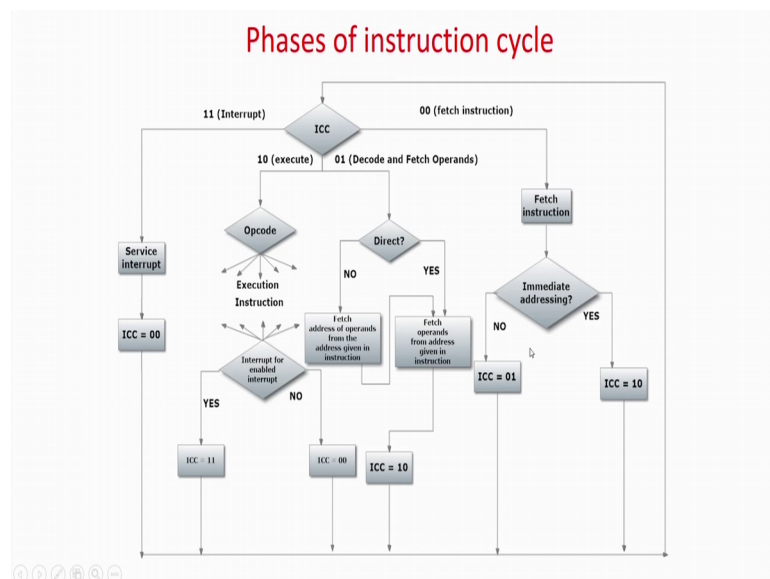
So, at this point you save PC all program state after servicing the interrupt again you pop and do your job. So, it is very simple fetch decode executing. So, keep there is an interrupt if there is interrupt save everything service the interrupt again come back and keep on doing that is what is the idea 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 plus one or something, but now it will be changing to may be some 25 or r bit value service the interrupt again get back to the value of I plus 1 and again start execution that is what is the vectorial representation.

(Refer Slide Time: 28:37)



Now, this something very interested. Now base basically there is something call 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 determine 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 call instruction cycle code is a two bit code and it is used to determine (Refer Time: 29:02). Now we will see how basically it is very interesting and how basically this code for chased based on face to face, face to face. So, 0 0 is the fetch, 0 1 is the decode and operand fetch, 0 1 is the execute face and 1 1 may be interrupt if it come. As I told you generally we talk fetch decode execute decode means operand is also fetched is in that cycle. So, let us 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 0 0, ICC value is 0 0 initially. So, instruction fetch it comes over here. So, if its 0 0 means its instruction fetch. So, it will come to this part. So, instruction is fetched. Now if you starting with the immediate addressing or 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 address with the 32 with the memory or it is 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 when I said the code of ICC as 1 0 lets go by this flow and then will see immediate addressing then what we have to do then it is 1 0 and if you come back to 1 0 you have note that 1 0 is means nothing, but execute; that means, if you fetch the instruction if it is a immediate addressing immediate you make IC equal to 1 0 then means immediately you can execute. But if it is a immediate addressing then you have to get the value of the operand from the memory. So, you make the ICC value equals to 0 1. So, what is the value of ICC 0 1 what will happen it will come over here ICC value is now basically from here is 0 1.

So, ICC is the 0 1 is instructed; that means, you have decode and fetch the operand because the inst instruction is a immediate 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 is a direct or indirect direct is very simple the value of where the operand is available that you address is available in the instruction itself. So, if it is yes fetch the operands from the given instruction address because say 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 1 0, ICC as 1 0 means directly you can go for a instruction execution. If it is no then indirect, so what was indirect add 32 if I say that is an indirect instruction then I say add some accumulator 32; that means, the 32 memory location that in 32 memory location also I do not have the value of the operand 32 memory location have some value that value is again an address where the value of the operand is exactly present.

So, it says fetch the operand from the given address in the instruction; that means, whatever is a indirect one. So, the indirect one means say for example, 32. So, memory location 32 now is a indirect instruction. So, you fetch the value from 32. So, what is 32? That is again an address when we when we operands is specified. So, you give the value of the value memory location 32 to this and then you fetching. So, again you can easily understand. So, if I say fetch immediate 32 sorry direct 32 whatever you fetch the operand from the given address in the instruction direct 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 value. So, first you get go to 32 memory location get the address of the operand and then again fetch then once in a indirect

manner or direct manner you get the operand make IC equal to 10 IC equal to 10 means you directly you can execute.

So, now, all these fetching of a instructions are done, if it is immediate IC you can directly make 10, bad execution no 0 1, no means you have to find out the values. So, if it is direct you can directly face the value from the address which is available in the instruction make it IC 1 0 means execute if it is no go in an indirect step and then make IC equal to 1 0. So, now, IC has been executed. So, once the ICC is 1 0 you execute it.

After it has been executed you have to check for interrupt it interrupt is no make ICC equal to 0 0 means next instruction will be fetched if it is yes then make IC equal to 1 1, IC equal to 1 1 means again it will servicing the interrupt and have to servicing the interrupt you make an ICC equal to 0 0, so basically these is the cycle.

So, in a nutshell you start with the ICC 0 0 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 equal to 10; that means, directly execute if it is a not an immediate make 0 1, 0 1 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 did 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 transfer from 0 1. So, it is being transfer 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 fetch that is 1 0 and after a 1 0 you have fetched all the instruction and then sorry 1 0 means it is the instruction execution. So, after you get the value of 1 0 you execute and it will 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 IC 1 0. So, you execute it after execution you check the interrupt if is the interrupt you make 1 1 that is the interrupt face service the interrupt and again make it 0 0 for that you go to the fetch cycle.

So, this diagram basically shows the cycles in terms of instruction cycle code, you start with 0 0 if everything is available in the code itself you directly make the code 1 0 execute it if not make it 0 1 get the data from the memory or indirectly from the memory

and then again go back to 0 0 if there is no interrupt if there is a interrupt make the code as 1 1 and again after servicing the interrupt make it 0 0 now that is what is the idea.

(Refer Slide Time: 35:28)

Phases of instruction cycle

- In the first phase the ICC is 00.
- After the instruction is fetched ICC is made to 01 when instruction is decoded and fetching of operands is accomplished. Based on the type of instruction, operand may be immediately available (in the instruction), or its location is available in the instruction, or the instruction has the address of the location that has the operand (indirect).
- After all operands are fetched ICC is made 10 and instruction is executed.
- Next ICC is 11 and it is checked in there is any interrupt.
- In case an interrupt has come it is serviced and then ICC is made 00. If there is not interrupt ICC is directly made 00.

So, again what I have told you is written over in this slide. So, first is 0 0 then ICC is made equal to 0 1 based on what is the type of instruction. If it is immediate then you did not do anything and otherwise you would make it 0 1 and after all the operands are fetched you make it 1 0, that is now is ready to execute 1 0 means ready to execute 0 1, means you have to face the data.

So, after data has been (Refer Time: 35:52) you make it 1 0 and then executed if there is a interrupt you make the code 1 1 otherwise if there is a no interrupt you directly make it 0 0 and keep on going it. So, basically the cycle is 0 0, 1 0, 0 1, 1 0 and back. Sometimes after this, this may coming, but otherwise is 0 0, 0 1, 1 0, 0 0 interrupt means it will coming 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 decodes 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 the (Refer Time: 36:28) it is represented program counter then how it is broad etcetera and then basically how it is executed. So, now, is better that will take an example where then taking so much of all theory will take an example that in a memory location there are two memory location FF0 and FF1 and FF2 is the place for I

have to store the result. So, some data is present in FF0 some data is present in FF1 I have to add this two numbers and store the value in FF2. So, this is what I want to do and I will show 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, these 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

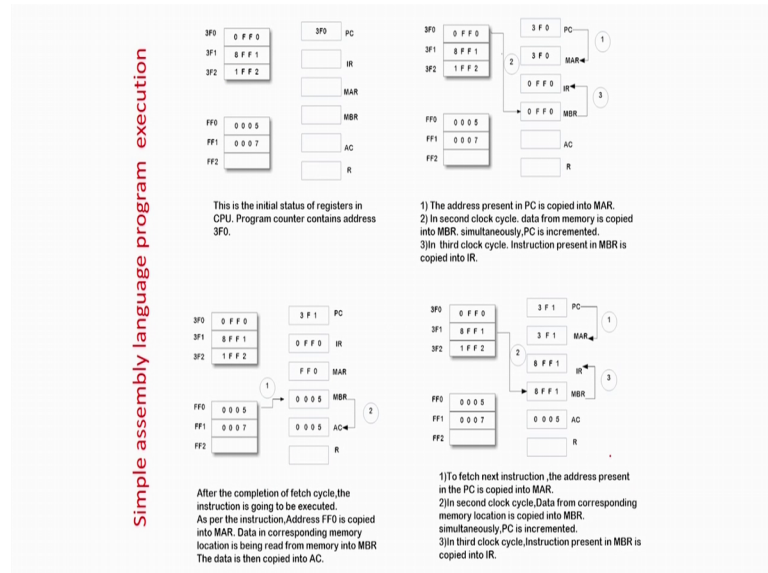
- The assembly language program is as follows:
- 1st Instruction **LDA FF0**: The contents in memory location FF0 are loaded into accumulator. After the instruction is executed accumulator stores value 5.
- 2nd Instruction **ADD FF1**: The contents in memory location FF1 is added to accumulator. The 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 said that load FF0 that is, so first data is FF0 so you load the value of FF0 in the accumulated 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 accumulated. Now you accumulated has the value present in FF0. So now, we will have FF0 plus FF1 and the value will be stored in the accumulated.

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 instruction also will be stored in the memory because is a von Neumann architecture and the number is

starts from FF0. So, this is basically your simple code are the memory architecture, I will go now step by step.

(Refer Slide Time: 38:02)



So, this is your architecture. So, you have see the 3F0, 3F1 and 3F2, these are the data memory first instruction is 0 FF0, 0 is the code as I told you for fetching an instruction from 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 just memory location that is FF0.

So, is an direct addressing. So, if you just find out what is the value of this location that is 5 will be (Refer Time: 38:50). 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 stored to this FF2. So, FF2 to this stored.

So, now, this is the basic memory conflict is a 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 0 FF0 will go to the memory buffer register because you want to read it; that means, whatever value in the PC that value will go to the address register address buff of the memory and this location whatever is value over there will first go to the memory buffer register and add this 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 faced over here memory buffer register. Till now where do not 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 then an instruction. So, first instruction, so it is 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 of the memory buffer register, first instruction or the first memory excess of a code that is always an instruction. So, it is going to the instruction register that is FF0.

Now the instruction will special with 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 F FF0. So, if I assume that it is a 16 bit word, 4 4 4 the 16 bit word. So, early first 4 bits are result for a oppcode and the other 3 bits are result for the memory location address. So, in this case I can address a memory who sizes 2 to the power 4 plus 4 plus 4 that is 2 to the power 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 concerned 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 fetch to the memory buffer register. Now there is difference between instruction fetch or the data fetch. So, this was the first instruction which was fetch that is 0 F F 0. 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 0 0 0 5 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 which whatever is the data available in FF1. So, it will add 5 with 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 not pointing to the next instruction, but currently you 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 store back to the accumulator that is what is being done by this opcode 8. So, know the value of this one is 7 plus 5 plus 7 is H 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 excess the memory for data. So, again I will excess 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, is an instruction for the last instruction was a data excess. So, now, this is an instruction excess. So, now, the instruction will be going over here, so 1 F F 2. So, it will be decoded what one 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 not the value of 0 0 0 C that is the asset of the output weight has to be stored, one 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 therefore, us, but is an intimated. So, now, the instruction was 1 F F 2, 1 taste that whatever is in the accumulator write back and FF2. So, what is FF2? FF2 is this location. So, this FF2 will be copy to memory address register, but now the memory will be in a write mode in all other step 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 a address register and what I have write I have to what is the value of the accumulator. So, I will write the value of the accumulator in memory buffer register.

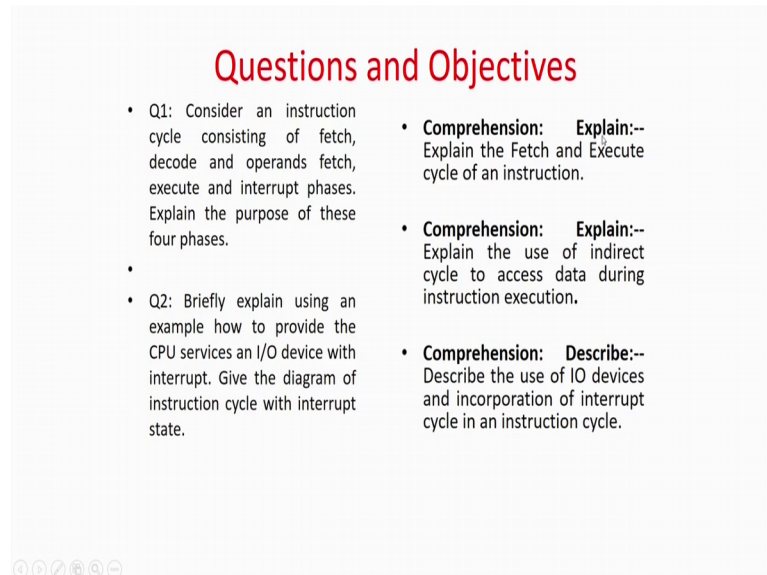
In all other cases it was happening the other way round what was happening what was available in the memory buffer register I was writing it to the IR if it is or the instruction what I was writing to the in a accumulator if it is was a or sorry for I was writing to the accumulator if it is was a instruction or in other way round means if it was an instruction I was writing to the IR and it will 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 write the value in FF2 that is C. These completes the execution of these 3 steps you reach 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, these in these unit we have shown basically what there 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 the a very nice example we have seen that how a instruction is accessed, how it is fetched from the memory, then how it is executed, how operands are face from the memory, how they are operated and again return back to the memory or this completion of the instruction.

So, again towards the end we now again we have to see some of the questions and how it meets the objectives. So, consider an instruction fetch cycle of this one, instruction fetch execute the de 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.
- Q2: Briefly explain using an example how to provide the CPU services an I/O device with interrupt. Give the diagram of instruction cycle with interrupt state.

- **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 provides CPU services using an IO interrupt, but is as I we have already at (Refer Time: 47:02) discuss how interrupt is done, over the 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 an incorporation in the instruction cycle. So, these two of the, that 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).
- Q4: Explain using an example how a simple assembly language program is executed.
- **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 instruction that is using the ICC. So, it can match the instruct, that is the objective of explain the fetch and executes cycle of an instruction and explain using a simple example how a assembly language code is executed. Again the first and the second and third instructions are basically objective of clarified over here; that means, if your assembly language have some interrupts then off course these two inter interrupts objectives are made otherwise the first objective is made. So, in fact we have now studied how basic instructions what is the basic instruction, how its looks and how this unit actually an come process this 3 objectives.

The next week, next unit where 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.