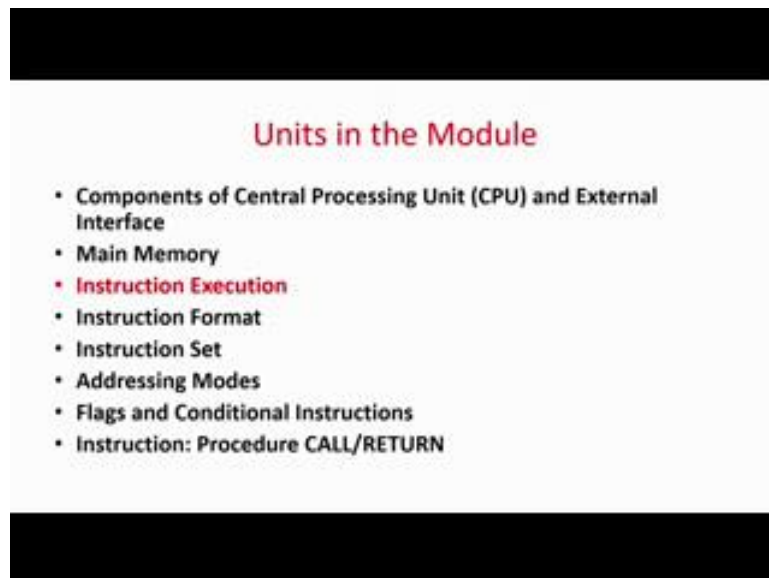


Computer Organization and Architecture: A Pedagogical Aspect
Prof. Jatindra Kr. Deka
Dr. Santosh Biswas
Dr. Arnab Sarkar
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

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, so that is 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 built 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, for that the basic idea required for the memory architecture as well as for the how the CPU is organized is will 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 you 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 called instruction register IR. So, now, the instructions will be decoded that what it 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 complicated 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 fields one is the opcode that is the operation it has to do that is again binarized because all the instructions are basically represented in 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 a 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 a 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 $5 + 5$. So, this $5 + 5$ is 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 you have to go for operand address calculation and then you have to fetch the operand from the memory.

So, after that is done you have to do the operation of 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 enough 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 requirement 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 place will be displayed and again the code will execute. So, interrupt so an interrupt is there. So, basically it suspends 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 has been changed.

So, what happens, so the basic say for example, I am listening to a music and at the same time I am moving the mouse. So, for a maybe little fraction of time will not in a in a very high speed processing in a modern *PC* we won't get a time difference, but what happens is that after execution 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 ok the mouse has been moved so 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 $2 + 3 + 4 + 5$ in 3 instructions.

So, $2 + 4$ has already been added then the mouse movement was there, it was an interrupt then again you service the interrupt that is the mouse movement has been showed, 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 added $2 + 3$ that I have to remember recollect back 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 word then you run the interrupt service routine, service the interrupt and again come back and recollect what has been already done that is pop of the values from the stack for the corresponding registers and again then go ahead and start executing the normal instruction.

In a nutshell after every instruction is executed you 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 or that is the intermediate stuff of your code in a stack then service the interrupt again come back from where we have left and at that point of time you again recollect everything back and go ahead.

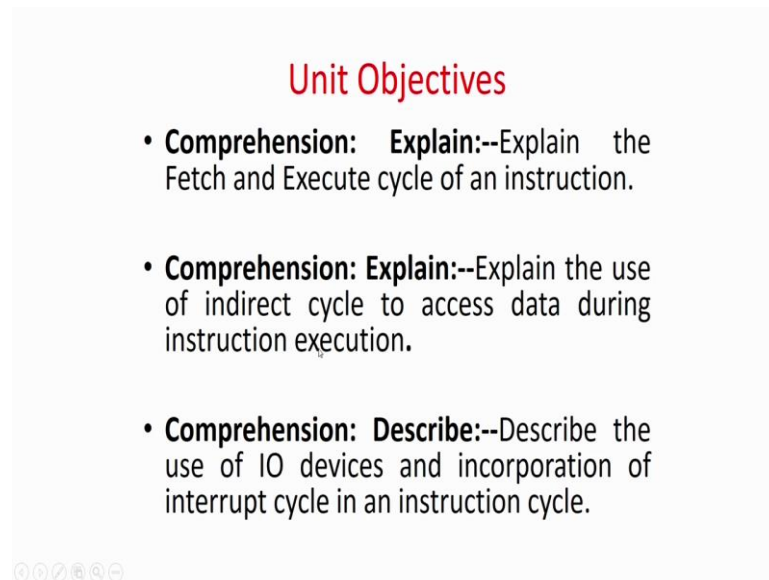
So, therefore, interrupt means not only servicing interrupt you have to store back everything. So, after coming back you can start from where you have left. So, that is actually a basic idea of instruction execution that is you fetch decode and execute and in fact, if the instruction interrupt then you 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 you will be able to explain what is the fetch and execute cycle of an instruction or a how an instruction is fetched decode and executes, then there is a direct mode as well as indirect mode. So, you will be also able to express indirect mode of instruction execution, direct mode means some operands will be the values of the operands will be either in the instruction or you can directly find the value of the instruction operands in the memory, but indirect means you will be redirected 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 an indirect job, sometimes if I say that load something from memory location 3030. So, the data is expected in the memory location 3030, but in indirect mode 3030 will have another memory

location and that memory location will have the data. So, that is actually indirect way of doing it. What is the benefit etcetera we will see in coming units and finally, describe the type of IO devices 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 happen due to IO, as I said like I am moving a mouse. So, the interrupt has to be serviced. So, for example, I don't 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 be 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 being executed and I don't 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 serve our purpose. So, therefore, that every time the PPT is running means some code are executing and it's being displayed in the screen. So now when I move a mouse the servicing the instruction by interrupt service routine and after servicing it is coming back. So, we will be also able to describe what is an interrupt service routine and how it's handled for IO devices.

(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 and operation fetch operate sorry operand 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 we 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 instruction is available over there is fetched. So, how it will be fetched, there is a special register 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 0000 hex. So, *PC* means program counter is having the value of 0000 hex. So, immediately that instruction will be fetched and that will be decoded and executed. So, whatever value of the program counter that particular instruction is fetched.

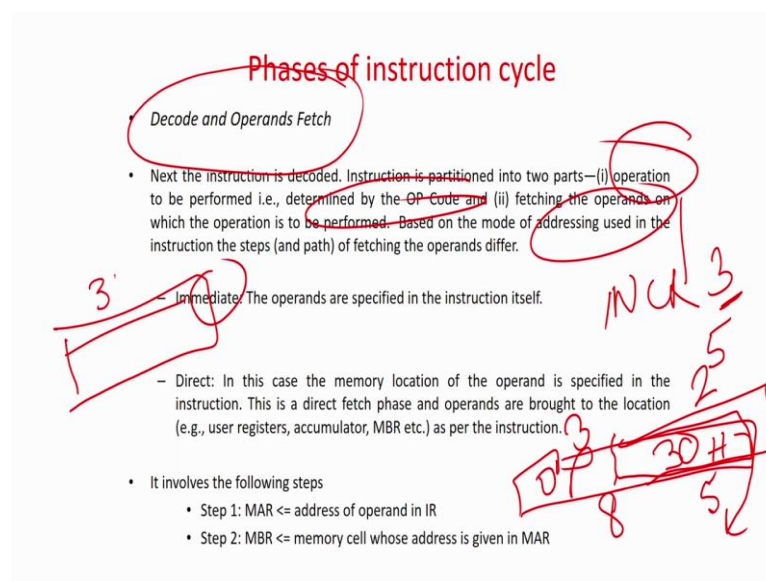
So, how it is fetched? So, as you already know in a Von Neumann architecture both the instruction as well as the data are in the instruction is in the memory, we don't differentiate that way. So, fetching an instruction or fetching an operand or the data is the same way. So, first we say that memory address register will have the value of *PC* because we have to get the value of the instruction to a from the memory.

So, already on the way 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 once it is given and the register is in read or the memory is 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 0000 memory location which was in the *PC* is now being fetched that is read and the memory will be dump the value of the instruction that is present in memory location 0000 to memory buffer register.

So, memory buffer register now 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 you have got the instruction right now will be given to a special register which is called the instruction register that is the *IR*.

So, basically what happens? The instruction is taken from the 0th memory location via memory buffer register and it is put into a special purpose register called 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, especially 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 now, the instruction is already present in the instruction register. So, now, what we have to do we have to decode and fetch operation.

(Refer Slide Time: 12:15)



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

So, instruction must always have a specific part which is actually called the opcode or the operation it has to do. So, may be if I assume that there only 3 instructions in my computer. So, may be 00 for add, 01 for fetch and 02 for write. So, I will add bring some numbers and the opcode will be 01 then I add it. So, the opcode will be 00 and when I want to write back the opcode will be 11, so 00, 01, 11. So, these are the 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 that will tell this computer what to do. Then depending on that you have to fetch the operand, operand, operands. That is very important. Because the instruction means you have to do something and do something on some operand operands. So, the operand values will be also be specified in the instruction. So, depending on the instruction type there can be two operands, 3 operands 4, operands and in fact, theoretically speaking n number of operands given an instruction. But if you have such long instructions I think you can already see what is going to happen then your memory width will be in huge. So, you don't want to do that if you may have memory whose width may be 1028 or 1024 bits we don't like to do that. Therefore, we always restrict our instruction length to certain limit.

Generally we may have two operate operands in that case. So, in that case suffice 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 where are the value of the operands specified. The simplest one is immediate. We can always say that there is one instruction called increment INCR and the value may be 3 this is one of the simplest instruction it tells that increment had various what you have to increment whatever value of the integer which is given in the instruction itself. So, this is actually called the immediate instruction.

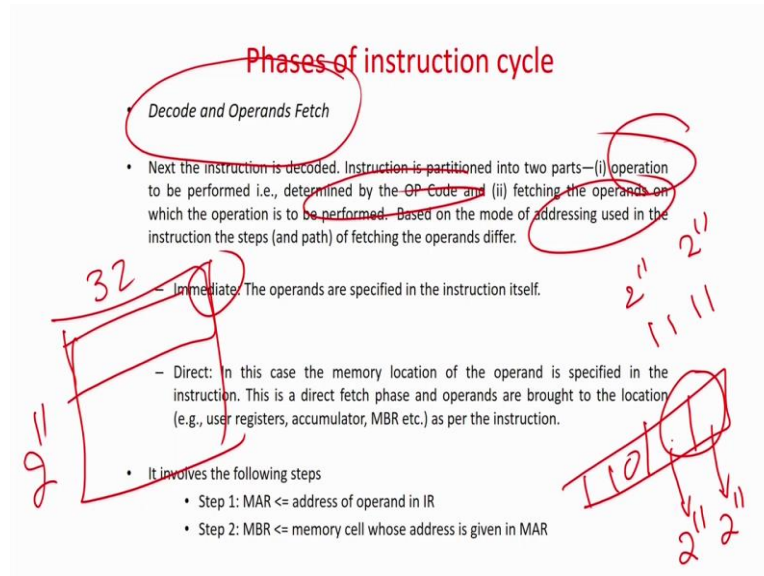
In that case operand fetch is not required at all. That is the operand is available in the instruction itself. But sometimes it is not very easy because you will the idea is something like that this is a memory cell.

Now, 11 instruction, so some part will be taken by the opcode that is we have to encode what the instruction will do, the remaining part is the limit will be giving you to write the operands. So, for example so this may be it may 8 bits assume then what happens maybe I put 3 bits to have the number of operations. So, there are 8 operations. So, 3 bits are gone for that. Now we have only 5 bits remaining; so, if we have 5 bits remaining. So, whatever data you can put can be maximum of 2^5 . So, 0 to 2^5 that is 32 numbers can only be represented, but if I want to add $100 + 100$ then you cannot write an instruction like this. So, this not going to serve my purpose.

So, better what I can do I need a longer space to put the operands, but longer space to put the operand means I have to make the instruction length longer, wider memory not very good. So, what I will try to do? I will have a direct addressing mode or an indirect addressing mode. So, direct addressing mode means here I will put the value of the memory may be I will write 30 H six bits. Now this is going to point to a memory location and that memory location will have the exact value. So, we can assume that the whole memory may be say 32 bits correct and let us assume that this is 32 bit and we have let me redraw it properly.

So, the main problem is that immediate addressing is very easy to handle, but then we will be limited by the range of data. So this is the memory width is 32 and let me have an instruction where I assume 10 bits it will have may be 1000 instructions where 10 bits are reserved for the instruction type and then 22 is for the operands.

(Refer Slide Time: 16:26)



So, may be what I can do is that. So, this 22 bits are reserved for the operands. So, what I can do is that I can put some value directly here then my range will be 2^{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, maybe now 11 and 11, $22 + 10$ bits.

So, the range is 2^{11} , 2^{11} and 10. So, 10 bits are reserved 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^{11} that is nothing, but 2048. So, range is 2048 range is 2048. So, two operands range is a 0 to 2048 if you want to have negative numbers then again it will be half.

So, what is the better way of doing it? I actually put some memory location. So, now, the range is 2^{11} , you can access memory whose size is 2^{11} . Now so, let me have a memory like this whose size is 2^{11} . So, it will tell whatever memory location you will have, you will now point to a memory. So the memory width is 32, so your data is present over here. So, now, what is the width of the data? Width of the data is 32 bits.

So, it's a huge number 2^{30} or 32 huge number you can do very high precision 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 to be represented. 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

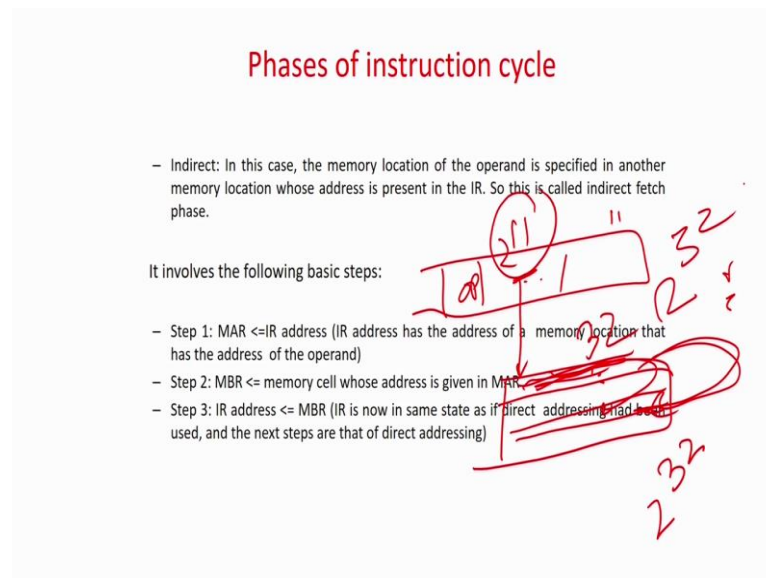
- *Decode and Operands Fetch*
- Next the instruction is decoded. Instruction is partitioned into two parts—(i) operation to be performed i.e., determined by the OP Code and (ii) fetching the operands on which the operation is to be performed. Based on the mode of addressing used in the instruction the 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 notes on slide:
 ADD 32H 33H
 32
 32H
 1
 1111
 1
 2112

So, what we will have to do, to do that? Some sub steps will be required. So, again you have to 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 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 add this sorry it is not possible 32 H and 33 H; 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 be this data will be fetched, next this data will be fetched and you can add these two numbers. So, this is actually called decode and fetch.

(Refer Slide Time: 19:14)



There is another instruction the benefits etcetera we will show in the later units, but that is something called 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 we 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 redirection 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 another memory location where the data will be there. Now, what are the advantages disadvantages we 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.

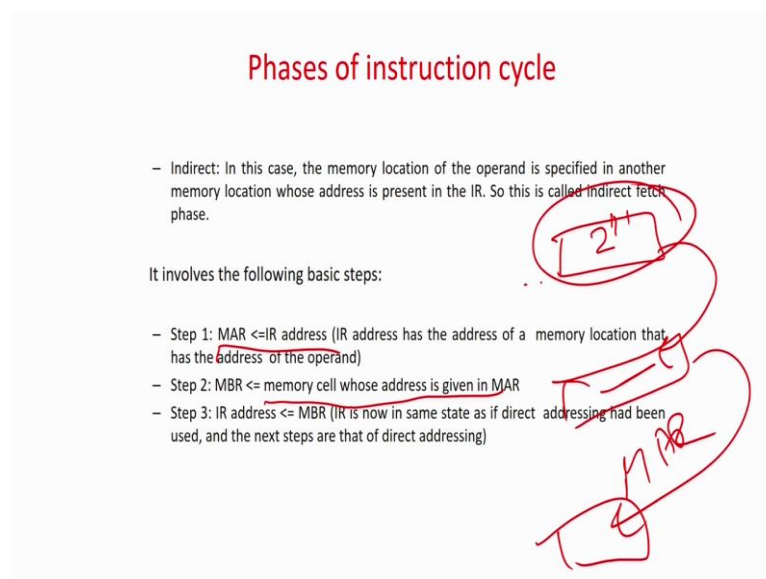
So, again it actually represents a wider range of memory or wide of you can excess wider range of memory over here. So, for example, as we are taking their 11 bits, so what is the range of memory you can access? Only 2^{11} , but say your memory is an level of Gigabytes, 2^{11} is already in the level of kilobytes. 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 access a 2^{32} size memory by this method.

So, if you are 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 only this size of the memory that can be accessed is 2^{11} . But if I have a larger size memory then we are going to have the indirect one, that is from here you point here, here also data will not be there here we will have the address of the data. So, but here is again 32 bit. So, now, it is 2^{32} . So, again a full-fledged memory can be accessed over here. So, that is again the advantage of indirect mode, but more details we will see in the later units that is what is the idea.

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

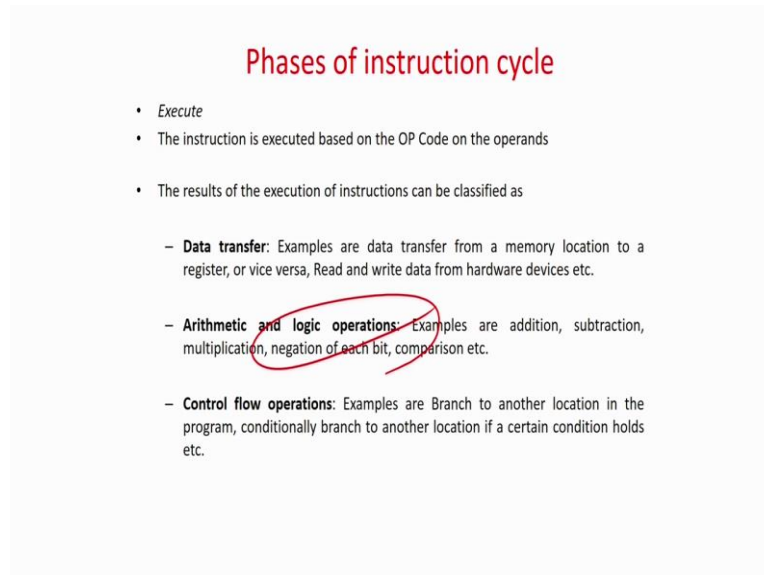
(Refer Slide Time: 21:44)



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 operand and a jump. Memory address register initially will have that first whatever is 2^{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 fed 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 or 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 we 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.

Ok now, instruction has been fetched, it has been decoded what to do and the operands are also been fetched now you have to execute it. So, this is actually the job of the CPU or the processing unit will have to do the job that is simple. So, based on the opcode 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 we will jump later on 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 scanf, readf. So scanf, readf are in the C if the machine or in the architecture version it will be load and store.