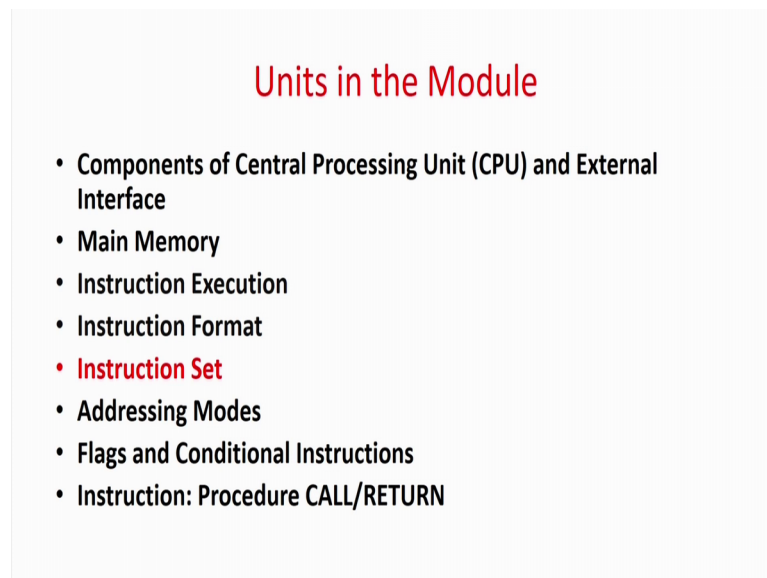


**Computer Organization and Architecture: A Pedagogical Aspect**  
**Prof. Jatindra Kr. Deka**  
**Dr. Santosh Biswas**  
**Dr. Arnab Sarkar**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology Guwahati**

**Lecture – 11**  
**Instruction Set**

So welcome to unit number 5 of the module on addressing mode instruction set and instruction execution flow.

(Refer Slide Time: 00:34)



**Units in the Module**

- Components of Central Processing Unit (CPU) and External Interface
- Main Memory
- Instruction Execution
- Instruction Format
- **Instruction Set**
- Addressing Modes
- Flags and Conditional Instructions
- Instruction: Procedure CALL/RETURN

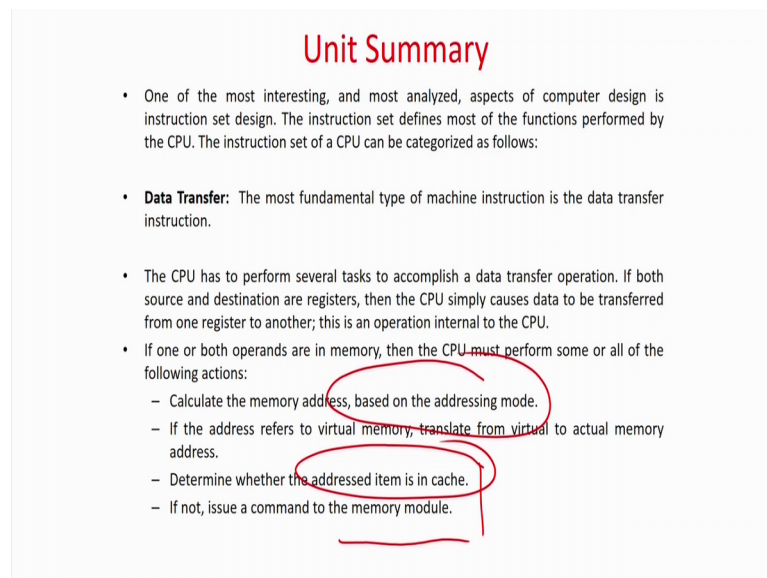
So, till now basically we have mainly concentrating on how what is the basic instruction how it looks like and how it basically executes, from now onwards we are trying to go in more depth of basically; how instruction works, how it is designed, how it can be clusters, what are the different type of sets in which you can club them etcetera.

So, in the last unit we basically saw about what is the basic instruction format that what it has it has 4 different maybe depending on the instruction format it can have 1 address 2 address or even 0 address; now in this unit basically non instruction set we are going to see in more depth of how the instruction can be club based on their functionalities. So, this is a small module sorry it is a small unit of the module in which case we will try to categorize the say instructions based on their functionalities, many times we have

discussed while discussing in this module that it can be of advatic type, logical type or data transfer type.

But in this case we will look into more depth of those classes and basically what if you are executing an instruction, what are the different registers that are set, so slightly in more depth will go in this ok.

(Refer Slide Time: 01:46)



The slide is titled "Unit Summary" in red. It contains a bulleted list of points. The first point is a general statement about instruction set design. The second point is "Data Transfer: The most fundamental type of machine instruction is the data transfer instruction." The third point states that the CPU performs several tasks for a data transfer operation, depending on whether source and destination are registers or memory. The fourth point lists actions for memory-based operations: calculating the memory address, translating virtual to actual memory address, determining if the item is in cache, and issuing a command to the memory module if not. Red handwritten circles and lines highlight the phrases "the CPU must perform some or all of the following actions:", "Calculate the memory address, based on the addressing mode.", "If the address refers to virtual memory, translate from virtual to actual memory address.", and "Determine whether the addressed item is in cache." in the list.

### Unit Summary

- One of the most interesting, and most analyzed, aspects of computer design is instruction set design. The instruction set defines most of the functions performed by the CPU. The instruction set of a CPU can be categorized as follows:
- **Data Transfer:** The most fundamental type of machine instruction is the data transfer instruction.
- The CPU has to perform several tasks to accomplish a data transfer operation. If both source and destination are registers, then the CPU simply causes data to be transferred from one register to another; this is an operation internal to the CPU.
- If one or both operands are in memory, then the CPU must perform some or all of the following actions:
  - Calculate the memory address, based on the addressing mode.
  - If the address refers to virtual memory, translate from virtual to actual memory address.
  - Determine whether the addressed item is in cache.
  - If not, issue a command to the memory module.

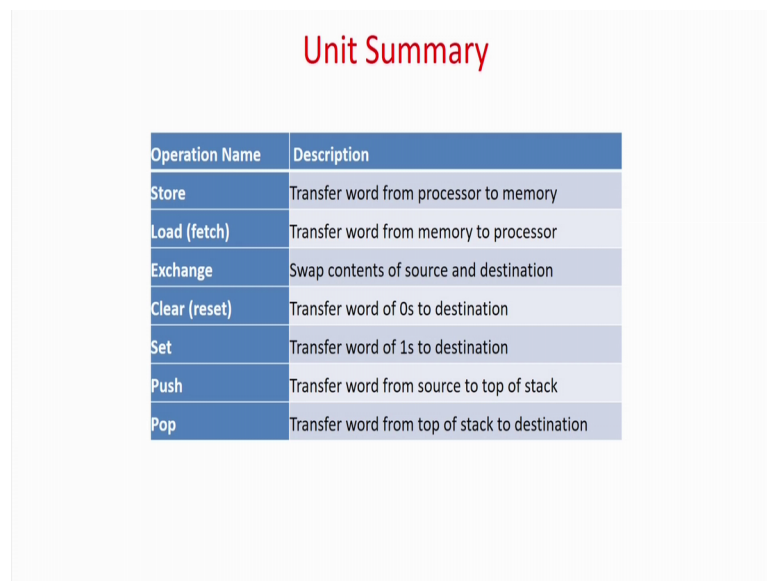
So, in this case basically what is this unit summary about. So, in this unit basically we will be classifying the instruction based on the functionalities and in each class what are the different type of instructions available we will be looking in depth. So, basically first is the data transform, we all know that if the some instructions like load store etc; transfer data from basically based on 1 memory location to other the memory location can be a register; it can be a accumulator, it can be a memory location in the main memory it can be a cache. So, anyway we are not concentrating on the cache that way because, already we have know told you and in between the registers and the main memory these are element of memory called cache, in which case wherever you want to execute some code a part of that main memory; where you are going to executive or the temporal data are loaded into the cache.

So, you can easily access them because the cache inside the processor, but anyway that will take in more details in our future module on memory. So, basically for us right now the classification of instruction of data transfer means you have to transfer data from 1

memory location to other and any will ask in unit, we have seen that the how many such operations can be done in an instruction depends on the number of addresses.

If it is a 3 address then we can have more number of data transfers corresponding to a 2 addresses and so for. So, basically what happens in this case so it has to calculate the address of the operands based on the addressing mode; so if the addresses then actually those things will be detailed out in a future module then basically you first find out whether the value of these memory location is available in the cache and if it is in the cache you can directly fetch it and if it is not in the cache then you have to read from the memory module; but in fact, this will be more dealt in more details in a future module as we have discussed just told right now. But in our as of now if we just understand that it is a transfer of data from 1 memory location to another.

(Refer Slide Time: 03:40)



The image shows a slide titled "Unit Summary" with a table listing various operations and their descriptions. The table has two columns: "Operation Name" and "Description".

Operation Name	Description
Store	Transfer word from processor to memory
Load (fetch)	Transfer word from memory to processor
Exchange	Swap contents of source and destination
Clear (reset)	Transfer word of 0s to destination
Set	Transfer word of 1s to destination
Push	Transfer word from source to top of stack
Pop	Transfer word from top of stack to destination

So, as I told will be going into depth of basically or will in look into the more details of what are the exact type of memory, such type of data operation data or means what I say that is their transformation operations. So, basically have stored so transfer 1 word of the transfer word from transfer to memory that is some operations you have already done in the processor and you store the result to a memory.

So, that memory is generally a main memory sometimes it can also be a register, then something called load fetch that is you actually transfer the word from memory to a processor; that the pause the register it can pause the register and accumulator. Exchange

such type of instructions are also there in which you can swap the contacts clear reset set and reset also very important operations that there are some instructions which you can make all the values of memory location 0 or 1.

So, set and reset are also a class of operations which fall under this set, push and pop as I told you that you have to push a value to a register me stack and pop a value from the stack is also false in the class of data transfer based or data operation or data movement based instruction. So, that is data transfer based instruction. So, as I told you that push and pop basically corresponds to a stack based machine or a 0 address based instruction; next is basically the arithmetic.

(Refer Slide Time: 05:00)

Unit Summary	
Operation Name	Description
Add	Compute sum of two operands
Subtract	Compute difference of two operands
Multiply	Compute product of two operands
Divide	Compute quotient of two operands
Absolute	Replace operand by its absolute value
Negate	Change sign of operand
Increment	Add 1 to operand
Decrement	Subtract 1 from operand

**Arithmetic:** Most machines provide the basic arithmetic operations like add, subtract, multiply, divide etc. The execution of an arithmetic operation may also involve data transfer operation to provide the operands to the ALU input and to deliver the result of the ALU operation.

So as I told you that all have been discussing throughout in many of the units over here, that there are 3 types of operable basically like mainly heart of all the computation is arithmetic and logic; that is you have to add 2 numbers you have to multiply 2 numbers there is all the mathematical operations like add subtract multiply divide absolute negate increment decrement. So, whatever up things are whatever we know about standard mathematical operations the all the instruction sets or the instructions that it get to it will be called as arithmetic equation.

But again as I highlighted in the last unit that add can be of several types the that add immediate; that means, you will have to add the value of 1 operand will be available

instruction itself, add to memory locations value of the 2 memory locations will be loaded then it can be adding indirect.

So, as I idea is that even for add subtract multiply at each particular also particular operation also; there can be lot of variations as simply add immediate and add from memory immediate means the operand value will be will be given in the instruction itself. So, each can have a lot of different variations itself increment 1 increment decrement increment by 1 increment by 2 they are quite standard like negate absolute, they do not have much variations like add subtract multiply you have lot of variations.

(Refer Slide Time: 06:16)

**Unit Summary**

Operation Name	Description
AND	Performs the logical operation AND bitwise
OR	Performs the logical operation OR bitwise
NOT	Performs the logical operation NOT bitwise
Exclusive OR	Performs the specified logical operation Exclusive-OR bitwise
Test	Test specified condition; set flag(s) based on outcome
Compare	Make logical or arithmetic comparison Set flag(s) based on outcome
Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control etc.
Shift	Left (right) shift operand, introducing constant at end
Rotate	Left (right) shift operation, with wraparound end

**Logical:** Most machines also provide a variety of operations for manipulating individual bits of a word or other addressable units.

Next is basically logical 1 logical means they are mainly basically beat wise operation so like and or not exclusive or then actually very important these are the standard ones, but there are some important ones like left sheet right sheet compare that is this test and compare actually these things are very important as we will see more on the in the future module, future unit will be looking at the jump instruction or conditional instruction execution.

So, in that case actually test and compare this will be very important that whenever some mathematical operations are done basically some flag values are set. So, you can test those flag values that whether the 0 flag is set then you take a jump instruction, you compare to arithmetic operations and then some set some values. So, if you compare 2

values like comma a comma b, that means compare the register value a register value b if they are equal some flags will be set.

So, this type of instructions basically fall under the category of a logical instruction, like and or not our basic logics left shift right shift are very simple test and compare are very important logical instructions which will be used for execution of jump instructions mainly; then some control variables can be set for some kind of protection purposes like interrupts etc, which will be dealt in later on the nutshell these are some of the instructions basically which fall into the class of logical instructions that is till now.

We were saying that and or not are mainly the logical instructions, but apart from the basic ones these are other very important logic instructions.

(Refer Slide Time: 07:46)

### Unit Summary

Operation Name	Description
Input (Read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
Output (Write)	Transfer data from specified source to I/O port or device.
Start I/O	Transfer instructions to I/O processor to initiate I/O operation.
Test I/O	Transfer status information from I/O system to specified destination

**Input/ Output :** Input/ Output instructions are used to transfer data between input/output devices and memory/CPU register.

And then there are some instructions for I O generally many of the cases we say that the I O is a part of the data transfer operation, but for many classes we can also classify them as the input output; basically you read from some port you write from some port that is the input output devices are available. So, there will be a whole module on I O which will be taught by the other faculty members who are dealing with the courses. So, in that case it is more or less data transfer like input is read output is write, but in this case it will not be exactly from a memory.

So, it will be from some output devices like it can be a mouse it can be a keyboard etc. So, there we have full unit dedicated to that maybe I can say start of I O; that means, say I want to read from the keyboard. So, there will be given instruction for that test an I O. So, whether the device from where I am going to read or write is functionally proper or not. So, there is and the full sector of the operations or instructions basically which are dedicated to I O sometimes mean till now.

It means as a separate unit dedicated for I o. So, till now we are not discussing much about input output, because sometimes in a very broad sense of an abstract sense. We can also call it the data transfer operation because you are going to read you are going to write start can be a control instruction test can be again a control instruction or it can be a in fact logical operation you can tell like that in read write from a memory is very simple data transfer operation. But when you are talking to some input output devices it will be a I O instruction like start and test I O can be control or they you can think of logical, but when you are talk about specifically about connection twin into other devices apart from memory then actually it is an I O instruction. So, these part will be delta more details in a future module and I O.

Then in the last part actually of this classification in the control instructions, as I told you so generally the instruction goes in sequence, but based on some conditions of an operation some flags may be set based on the value of the flag you can take, the next instruction or some other instruction that is the conditional instructions.

(Refer Slide Time: 09:48)

### Unit Summary

Operation Name	Description
Jump (branch)	Unconditional transfer, load PC with specific address
Jump conditional	Test specific condition; either load PC with specific address or do nothing, based on condition
Jump to subroutine	Place current program control information in known location; jump to specific address
Return	Replace contents of PC and other register from known location
Skip	Increment PC to skip next instruction
Skip Conditional	Test specified condition; either skip or do nothing based on condition
Halt	Stop program execution

Important conditional instructions are branch there is unconditional load, this specific address wherever you want to jump conditional means you have to check the value of the flag; if it is true you take that location either you is continuous as procedural. Next step jump to subroutine basically if there subroutines in the code you jump to that procedural return, means after completing these subroutine or the interest subroutine ISR you go back to from where the procedure was called, so in that case all the values of the PC etc; has be brought back from the stack and it has to go some other instructions has keep conditional, that is next instruction may be skipped next instructions may be skipped on some conditions.

So, basically these are in a nutshell the broad classification of conditional instructions, jump conditional jump to route subroutine return from the subroutine skip, an instruction skip an instruction based on some condition and halt is also a very important control instruction, where you stop the code again as we are discussing everything for pedagogical perspective.



(Refer Slide Time: 10:42)

## Unit Objectives

- **Comprehension: Discuss:**--Discuss data transfer operations - inside processor and between memory and processor
- **Comprehension: Explain:**--Explain arithmetic and logical operations of a processor.
- **Knowledge: Describe:**--Describe I/O handling and system control operations of processor.
- **Comprehension: Discuss:**--Discuss how to program a processor - Machine level, Assembly level and high level languages.

So, again this is basically a recall and a knowledge based kind of an objectives mainly in this case, you will be able to after doing this unit you will be able to discuss the different type of operations inside a processor mainly between a processor and a memory that is data transfer operation, you will be able to explain about the arithmetic and logical operations.

You will have some idea and you will describe about I O handling system and control operations of a processor you will be able to describe the basic idea and in fact, a separate module we will discuss in depth and comprehension means here; you will be able to discuss how to program a processor in a machine lever assembly language high level languages.

Basically the idea is that after doing this unit and also on the knowledge is of some of the previous units, you will be able to tell that given a code a high level code, how what will be the assembly language level, what will be the instruction at the assembly level, what will be is binary version of the machine level and what will be the corresponding high level; that means, given a high level code you will be able to translate it into assembly language and a machine language and discuss how it basically executes.

(Refer Slide Time: 11:45)

### Step wise instruction execution and CPU registers

Example: The content of memory location FF0 is 5 and FF1 is 7. We want to add these two numbers and store the result in memory location FF2. Assume that we store this program from memory location 3F0.

To perform this task, following operations have to be accomplished.

- Contents of memory location FF0 have to be loaded into accumulator. As it is given that 5 is present in FF0, it should be loaded into accumulator.
- Contents of memory location FF1 have to be read and should be added to value in accumulator. Result of the addition should be stored in accumulator. 7 has to be added and the result should be stored in accumulator.
- The result of the addition which is stored in accumulator must be written into memory location FF2. i.e., result of addition (12) have to be written into memory location FF2.

So, without mean as I told you this unit is basically you can think as a second part of the last unit on whatever of the instruction set, basically we are discussing on the instruction format. So, this last 2 units that is you so in fact if you look at it. So, instruction format and instruction set you can call that at the 2 small units there one is covering about the different formats and this 1 this unit is to talking about the basic transification based on some operations so.

In fact, that is why without going into much more theory in this unit let me give you in more details by some examples, that will set the that will make you help in understanding more on this units; basically because most of the theory related to this unit has been discussed in the last unit, anyway discussing about the instruction formats, because formats means what are the basic structures or what are the basic components and here we are discussing on the instruction set how to classify based on it is functionality like data transfer I O control or arithmetic logic right.

So, we take a very simple example like there is 2 memory locations FF0 it has 5 and FF1 has the value of 7, we have to add these 2 numbers and the result then has to be written in anywhere location FF2.

So, already sustain program we have discussed in a previous lecture previous unit, but here we are going to look at it more on a more from the perspective of the instruction set and instruction format as well as also we will look at how the memory is handled and

how internal registers are handled and what are the control signals; it will bill a same type of program now in more depth from the architectural concept as well as the instruction format and addressing mode concept. So, we are saying that the job is simple to memory location have 2 values you have to add them.

(Refer Slide Time: 13:33)

### Step wise instruction execution and CPU registers

*Assembly language program for calculating 5+7 is as follows:*

```
LDA FF0 // The contents in memory location FF0 are loaded into accumulator. Now accumulator stores value 5.
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.
STA FF2 // Accumulator contains the result of addition and the result have to be written into FF2. This instruction stores the contents of accumulator in memory location FF2.
```

So; obviously the first instruction and yet we are taking a format what is the machine thing, the example you are solving here is a single address instruction. So, they told you the same good we have seen maybe in a previous unit, but now we are going to focus more on it from the last 2 unit the current unit and last unit perspective, that is on the addressing mode and the instruction set. So, the first 1 is l d that is load accumulator FF0 so; that means, what whatever is value is memory location FF0 will be loaded to the accumulator. So, this is a data transfer instruction as well as it is a single address instruction the single addresses because as I already told you single address instruction means, 1 of the operator or the operand is basically a sorry the operand basically accumulator. So, it is done and then as I told you whenever 1 thing I miss to tell you basically.

So, whenever you are writing a code you have to assume what is the instruct what are the machine type. So, for example, in this case we are assuming that the architecture of the CPU or the machine type is a 0 sorry single address format. So, in this case 1 is F2 a

accumulator then we are saying add FF1. So, what is it is a arithmetic operation arithmetic instruction.

So, it says that whatever is in the value of accumulator that is value 5 which was FF0 will be added to the value which is available in FF1 it will be stored back to register finally, you have to store the value of s d FF2 that is the value of the accumulator you have to store it to FF2 memory location. So, now accumulator has 5 plus7 so it will be stored over there. So, to do this operation we have 2 I O operations sorry 2 data transfer operation and 1 is the arithmetic operation and this machine is a single address machine.

(Refer Slide Time: 15:16)

**Step wise instruction execution and CPU registers**

Converting the assembly code to machine code :

Let the Opcode for LDA instruction is: 0000  
 Let the Opcode for ADD instruction is: 1000  
 Let the Opcode for STA instruction is: 0001

Hence,

LDA FF0 is equivalent to 0000 1111 1111 0000 in machine language.  
 LDA F F 0

ADD FF1 is equivalent to 1000 1111 1111 0001 in machine language.  
 ADD F F 1

STA FF2 is equivalent to 0001 1111 1111 0010 in machine language.  
 STA F F 2

*Handwritten notes: A circled '16' and '16 bits' with arrows pointing to the opcode fields. A '2' is written next to the opcode definitions.*

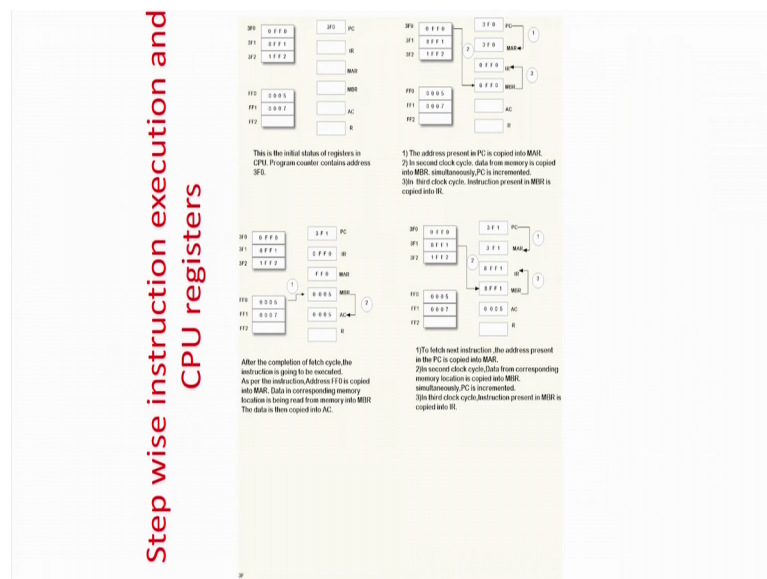
Now, as I told you so that now if you look at the machine code. So, as I told you machine code is always a binary code, but generally what happens we do not write it too much in the textual literature of the books or the slides, because it becomes very difficult to read and understand. So, we will feel guilty so what it says it says load FF0. So, single so this is for the opcode and this is for the operand single address. So, another is defector the accumulator which is not mentioned. So, maybe I am telling you that maybe this machine has 3 instructions. So, I think 2 bit codes were enough to do this, but let us assume that the machine has some more it has 16 operations to do. So, they have kept for bit as the size of the opcode.

So, the three codes for load and store and load add and store are these. There is the opcode is 000 means that it is a load instruction add means 1000 at zero 01 first 2. And then

this is the of where from where you have to load the value is FF0. So, this one is going to be the binary. So, if somebody erases this and say this is your first line of code, 0 0 0 very difficult to read and understand therefore, we always keep the memories. So, in this case the instruction size is 4 = plus 2 1 that is twenties sorry 16 sorry 16 is 4 into 4 16 bits.

So, it is a 16 bit instruction size that you can also think that a memory in this case is a 6 6 16 word bit is the word size. So, in will be all the instructions mainly in this case whatever is discussing for this example are loaded in a single word. So, it is easy to fetch decode and execute next it add FF1. So, same case now you can see that this format has means the code has changed where is correspond to add, and this one is the case similarly for storing this one. So, the idea is that if I write only in these 3 binary numbers it is very difficult to understand. So, you always go for the mnemonics and as it is again a single of the 3 instructions are written in a single address format. So, the last 2 are basically data transfer and this is the arithmetic.

(Refer Slide Time: 17:20)



Now, we see step wise basically what happens, now we will again deal with will we have already discussed a similar example beforehand, but now we will see in more depth of the different instructions, even registers and the formats. So, as I told you. So, this is the first instruction to be executed. So, the PC is going to have the value of this one value of the memory location of the first instruction. Then what happen is that. So, in this case.

So, this instruction no as I told you we assuming that this is a 16 bit size. So, each of this is memory location has a 4 instructions

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

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

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

So, first F what happens the PC is pointing to this, this instruction is going to the memory buffer register and as we know that because of this addressing format each memory location has a single instruction. So, need not worry directly take the value of memory buffer register the instruction register, it will decode it and it will find out that it is asking to load the value of FF 0 to accumulator. So, that is what is being done. So, it is loading the value of FF 0 to the memory buffer register and then to the accumulator and the value of PC will be incremented by 1 next what happens. So, what is the value of value of F 1?

So, it says that address value of accumulator to whatever value is in FF 1. So in fact, again as I told you one word is enough to take a full instruction 1 address format. So, it will be loaded in the memory buffer register it will go to instruction register decode it and it is that FF 8 FF 1 means whatever value is in FF 1 has to be added to the accumulator that has told back to the accumulator. So, this one is done memory buffer register it is the instruction register it is decoded and it knows what to do, and you can understand that program PC is actually program counter has changed to next instruction to be executed now.

So, as I told you the next step says that you have to add, this one was the step that you have to add the value of the accumulator to FF 1. So, 5 was initially in the accumulator now 7 has been in the memory buffer register, you add these 2 value there is 7 plus 5 is 12 that is c and will be stored back to the accumulator this is the job which is done by the second instruction and finally, now what happens is that next you have to store.

So, this is the program counter has gone to a third instruction, it is getting loaded over 1 FF to the instruction register. So, what it tells that whatever is in the value of accumulator because 1 means 2 to FF 2. So, accumulator value will go to memory buffer (Refer Time: 21:22) stored over here. So, this is the final execution. So, what we have seen in this case.

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

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

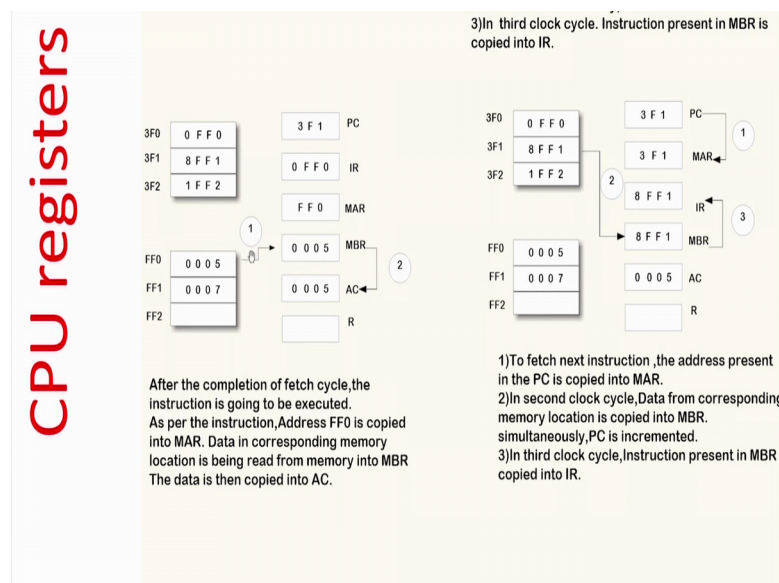
So, it takes some values from the accumulator and the memory location, and it actually operates it uses an adder to add it and store back the results that is happening in the third step, which is again a I O operation. Now basically now we will. So, that was more or less whenever you talk about arithmetic operation a (Refer Time: 22:40) with arithmetic and logic unit is thing let us stood a I O.

(Refer Slide Time: 22:44)

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

So, what happened? So, the first case the value of program counter was this, the next the value of program counter is.

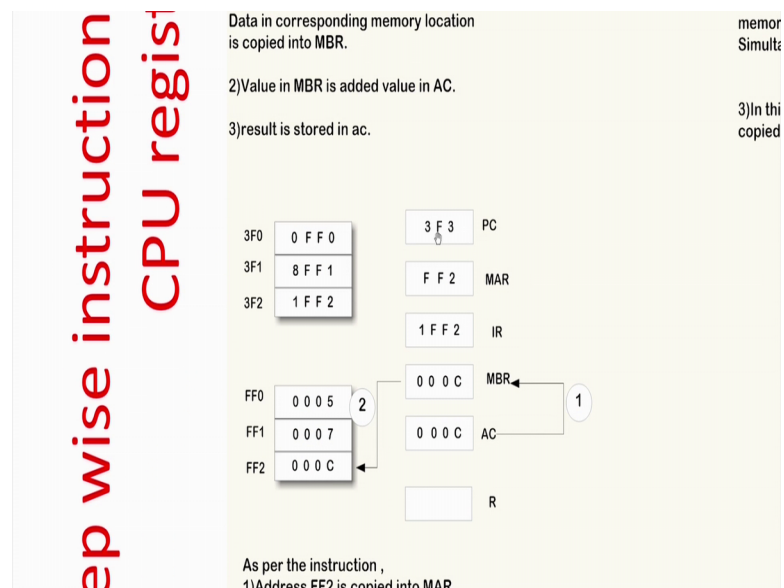
(Refer Slide Time: 23:04)



Next 8 F, 3 F 1 and after that the value of program counter becomes 3 F that is it goes step by step now what it can happen that if there is a conditional instruction then instead of going incrementing like 3 F 0, 3 F 1 at some point of time it will change.



(Refer Slide Time: 23:13)



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

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

So, whatever value is given in the memory address register is the location from where the data has to be read. The MAR is connected to the memory bus and hence the address is required for this to a main memory. So, MAR basically is connected to the memory address register is connected to the basically address bus of the memory and. So, it is there for you directly memory will get the value of the address from there. Next the CPU has to tell that it is a read operation, because you are at present discussing about memory read. So, there is a control line we have already told, it is read or write it tells the control line says that it is a read operation. Now actually the difference between the CPU speed and the memory speed and the I O speed. CPU is the fastest memory slightly slower and I O is extremely slower.

Because I O means myself you and some human being is handling, memory is a slower device compared to a CPU that we will see later, but for the time being you can consider this hierarchy. So, therefore, there is a synchronization issue. That whenever I say that I read from the memory, but you are never assure that immediately I will get the answer. So, there is a synchronization issue that is a handshake that you say that I want to read it read some memory location. So, I give the value in the memory address register, and then the memory address register via the address bus is connected to the memory, then you said that reads.

So, the read signal is me, but how much time I should wait before I take the value from the memory buffer register. Because based on your read and the address in the MAR, the data will be saved in the memory buffer register. Though there is actually CPU waits till this is an acknowledgement from the memory that is memory function completes MFC. So, MFC is a very very important control signal. So, what it says is that whenever a job is done that is ready is done data is done, to the memory buffer register it will make the MFC signal high or maybe it will say that it is enable. So, now, what will happen? You have to read the value from the memory buffer register and you can freeze the location therefore. So, that freezing acceptor will come later.

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

(Refer Slide Time: 26:57)

## Memory write operation

*The memory write operation is similar to memory fetch operation for the first two steps. The next steps are as follows*

The data to be written should be stable in the register until the write operation is over. So, the control signals applied to the register now freezes the register.

The data from register is transferred to MBR via the data bus; during this operation the control lines of the MBR are set to load. After this operation the control signal for register need not remain freeze.

CPU uses the control line of the memory bus to indicate that a Write operation is initiated.

After issuing memory write request, the CPU waits until it receives an acknowledgement from the memory, indicating that the requested operation has been completed. This is accomplished by the control signal of memory bus MFC. When the MFC is set to 1, it implies that the contents of the specified memory location are written by the data present in Memory Buffer Register (MBR).

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

(Refer Slide Time: 26:58)

## Memory Fetch operation

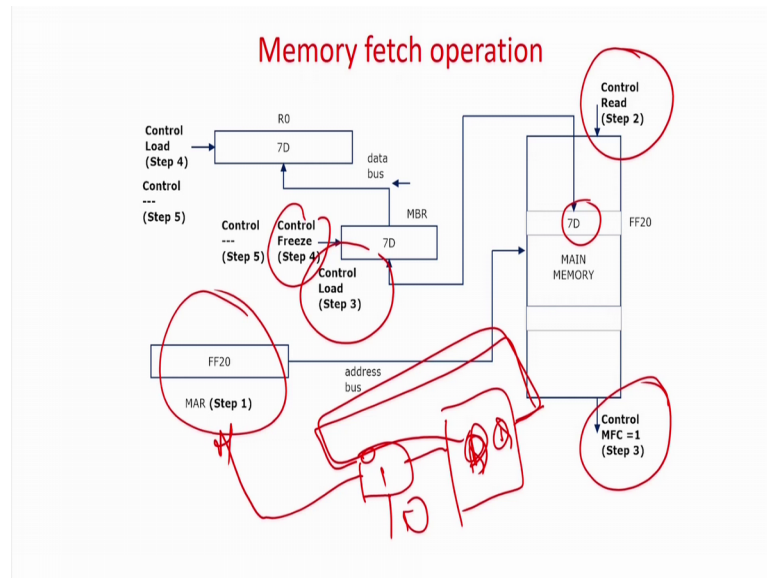
Data (7D in HEX) is available in memory location FF20 and we want to bring this data to CPU register R0

1. Place the address of the location to be written on the address bus via **MAR**.
2. Activate the memory read control signal on the control bus to memory.
3. Wait for the memory to send the data at the addressed location to **MBR** i.e., MFC=1. In this cycle, **MBR** is in load mode.
4. Send the data to be written to **R1** from **MBR** using data bus. In this cycle, **R0** is in load mode and **MBR** is in freeze mode.
5. Drop the memory read control signal, freeze signal to **MBR** and load signal to **R0**.

First let us basically a memory fetch operation in these steps as well as the pictorial representation. As I told you first it is in the memory address register, then read, then you wait that the memory function complete is ready, then when it is done then it can be read to the memory buffer register or it can be register 1 accumulator or wherever, and then only then actually the memory buffer can be is ready you read it from this one and then you can go ahead. I will (Refer Time: 27:26) take it a figure and explain it. So, what was

the example? The example they are saying that data 7 hex which is available in memory FF 2 0 we want to bring it to CPU register R 0 that is the example. And typically this is a flow whether you want to take it to accumulator or any register this flow or more or less will become similar is the accumulator the memory buffer will write to the accumulator it is R 0 the memory buffer we registered write to R0.

(Refer Slide Time: 27:54)



So, now let us see step one. So, I want to read from this memory location FF 20. So, memory address register will have the value of FF 2 0, these try to the address bus it is going then I want to read this basically. So, now, next step is read, read signal is given now you have to wait how you have to wait till memory function completely is one. So, once the memory functions complete is 1, it means that the value of this one is, it is written to the memory buffer register and it is stable

So, once that is done in step 4 you have to freezy, may time I was saying what is freeze. So, freeze is a very simple operation if I take a D flip flop D Q is the D flip flop, if I connect D back to Q. So, if I can connect D back to Q therefore, there will be no change in the value of D even if I apply a flop. So, that is actually a freeze operation. So, if I make a permanent connection like this. So, the your memory D Q will always have the same branch actually what happen is that there is a basically arranged multiplexer and this is D and this is Q sorry this is D and this is Q, maybe 1 port will be free another port will be low and this will be a mass. So, if you make mass is equal to 1 then from the port

it will read to the and your flip flop can be set and reset and if I make this mass value equal to 0 then the Q will be feedback to Q sorry Q will be feedback to D, and they will loop back and it will be the freeze position that the value will ever be stabilized.

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

So, at this same step when you are reading it before they are actually after reading it you have to freeze it all you can see just after step 3 in the MFC has been 1 you freeze the memory buffer register that is you do not allow any more changes to write over there from the neighbor; that means, even if I given a new address right and if I give a new step or whatever this value should not be disturbed and then in this step after freezing it you read the value of memory before register to a your required register which is r 0 in this case it can be an accumulator and then after step 5 you defreeze this and again the step repeats once again.

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

So, what are the steps. So, step is that first to give the value in the memory buffer register then you say that you give the memory address or you can change the value then you say right before you give the right signal your data to be written to the memory has to be stable in my memory buffer register and then you have to wait then the memory said that memory function complete, that I have already read the values from the memory buffer register.

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

(Refer Slide Time: 32:09)

**Memory write operation**

**Example:** Data (8F in HEX) is available in a CPU register R0 and it is to be stored in memory location 25FF.

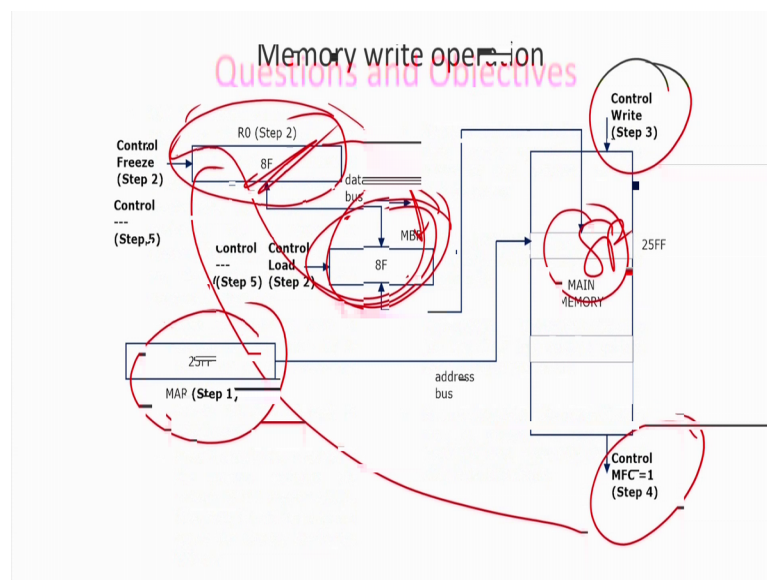
- Place the address of the location to be written on the address bus via MAR.
- Transfer the data in R0 to MBR via data bus. In this cycle, R0 is frozen and MBR is in load mode.
- Activate the memory write control signal on the control bus to memory.
- Wait for the memory to store the data at the addressed location i.e., MFC=1.
- Drop the memory write control signal, freeze signal to R0 and load signal to MBR.

So, again this is the same example it says there is a data a F in hex which is the available register 0 and you have to store it memory location F F. So, place the address in memory buffer register then transfer R to MBR and now memory buffer register is to be frozen that is what first you locate you the memory address register you put the value of 25 FF that is I want to handle this memory location, then you put the value of R naught register or accumulator whatever may be the case in the memory buffer register. And then you

stop the memory buffer that you freeze it, because now the memory will start reading from the memory buffer register.

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

(Refer Slide Time: 33:07)



So, this is the example. So, always first step in general any memory operation will be the value of the memory address register, then in step 2 you bring the value of from register the ATF to the memory buffer register and freeze that is very important. This one you put over there and you freeze this whole step, you can you basically freeze the memory buffer register do not allow any changes there. Now you say keep write signal then write signal is if you start reading from the memory buffer register.

But note here that the memory buffer register is now freezed said the memory buffer is freezed know whatever may be the changes over? Here it will not be changed and memory will very nicely read the value that is ATF will come over here and after ATF will come over here, it will say that memory function is complete. 1 memory function is complete you can defreeze everything and go for the next memory cycle ok.

(Refer Slide Time: 33:53)

### Questions and Objectives

- Q1: We want to evaluate the expression  $7*2 + 5*3$ . The data 7, 2, 5 and 3 are stored in memory location 7F0, 7F1, 7F2 and 7F3 respectively. The result of the evaluation will be stored in memory location 7F4. Assume that we store this program from memory location 700
  - Write the assembly code to evaluate this expression. Try to reduce the memory access and justify it.
  - Convert this assembly code to machine code.
  - Show the step by step execution of this program indicating the contents of CPU registers (PC, IR, MAR, MBR, AC and R) and control signals like Memory Read/write, MFC etc.
- **Comprehension: Discuss:**--Discuss data transfer operations - inside processor and between memory and processor
- **Comprehension: Explain:**--Explain arithmetic and logical operations of a processor.
- **Knowledge: Describe:**--Describe I/O handling and system control operations of processor.
- **Comprehension: Discuss:**--Discuss how to program a processor - Machine level, Assembly level and high level languages.

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

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

So, after listening to this unit and the units provides to that you will be able to solve this problem. Here you will be able to the design the first the assembly language code then



maybe translate into the machine language, but anyway that is not very easy to read. So, we keep the mnemonic version, then you can see that there will be lot of memory read and write operations. So, how they will interact with the memory, basically you can talk about the I O handling in this fact actually I O means a memory in this case basically because I O we are not directly dealing with they have just given you the idea. So, basically different type of logic operations of arithmetic operations will be happening over here. So, you can great a great a deal of if you are able to solve this problem.

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

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

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

Thank you.