# 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**

## Control Unit
## Lecture – 16
## Instruction Cycle and Micro-operations

Welcome to the next module on control unit. So, in the course in the last module we have basically it see that what is the basic architecture of a computing system? Like we have a control unit, then we on the other side we have the memory, we may have some peripherals and then on very theoretical level we have seen what are the component inside like a memory block, then we have also seen what are the registers in the control unit, then we have seen an ALU, how they are connected and also we have seen in a very theoretical notion and have a code get exsiccated.

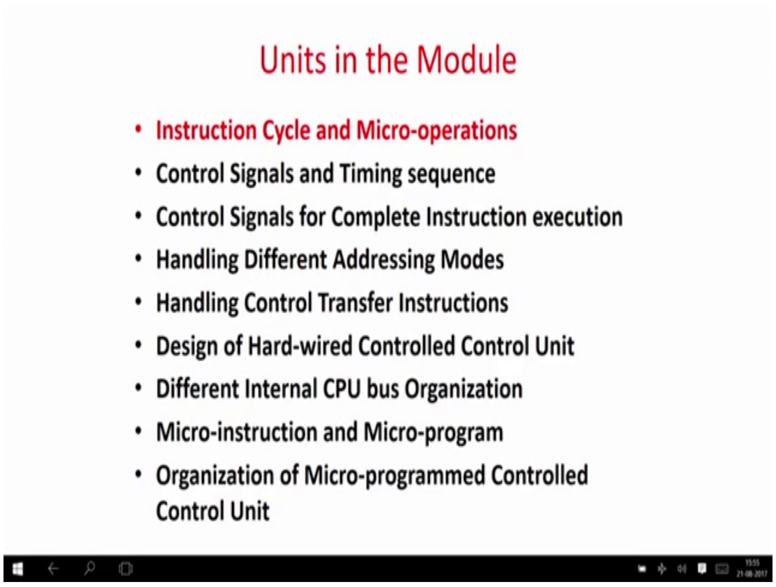
But you might have thought that such a complicated situation that you instruction is taken, then just the next instruction taken after that it is decoded, fetched and so many (Refer Time: 01:05) steps happen.

And they also happen in the hardware where everything in terms of bits. So, then means; that means, there should be lot of control signals or there should be lot of control integrities which happens, because which helps for a code to execute in such a integrated environment that is on a basic computing system.

So, the last module was on the basic architecture of that system. In the next module that is in the control unit what we are going to see we are going to look at basically how those, how those units.

Like for example, your registers, your memory, your CPU, your busses, how they are basically interconnected and what are the signals and what are the exact signal flows that is require and how the signals are generated. So, that a code executes in a current fashion. So, in this mod module we will be mainly concentrating on how a control unit is generated? What are the required signals? How the signals are generated? And how the sequence of signals actually maintains a proper flow of the core distribution? So, that is the basic idea of this module.

(Refer Slide Time: 02:08)



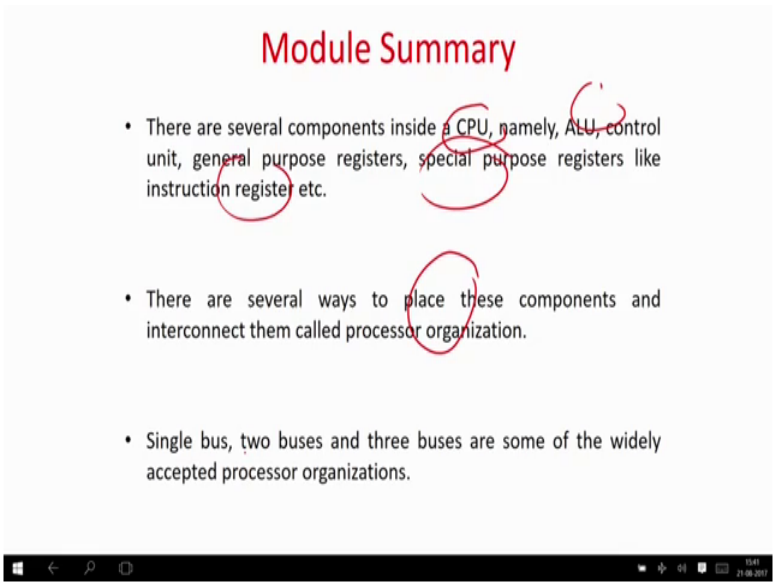
## Units in the Module

- Instruction Cycle and Micro-operations
- Control Signals and Timing sequence
- Control Signals for Complete Instruction execution
- Handling Different Addressing Modes
- Handling Control Transfer Instructions
- Design of Hard-wired Controlled Control Unit
- Different Internal CPU bus Organization
- Micro-instruction and Micro-program
- Organization of Micro-programmed Controlled Control Unit

So, in this module basically we will be making mainly looking at the instruction cycle and the micro operation inside that then you will be making mainly looking at control signals and timing sequence and so, forth. You can re go through all the different units in this module. So, this will give you a feel that basically this module focuses on how what are the instructions? How they are dividing to micro instructions? And how automatically some control signals are generated, which actually are required for the control flow of the data in the register or CPU busses etcetera and how basically what are different ways are actually generating the sequence in the control unit.

So, a code executes in a code and interrupts and if you have for example: jump instruction. So, how basically a code executes and what are the required control signals for this generation and how actually timing is organize and what is the timing issues etcetera, which will be learning in different units of the module. So, the details of the modules are available in this slide.

(Refer Slide Time: 03:08)



## Module Summary

- There are several components inside a CPU, namely, ALU, control unit, general purpose registers, special purpose registers like instruction register etc.
- There are several ways to place these components and interconnect them called processor organization.
- Single bus, two buses and three buses are some of the widely accepted processor organizations.

So, as we are going by a pedagogical aspect. So, let us discuss on the module summery. That what is going to be covered in this module so? In fact, as we already discussed in the last module that a basically a computing element has CPU, ALU, special purpose registers instruction registers etcetera.

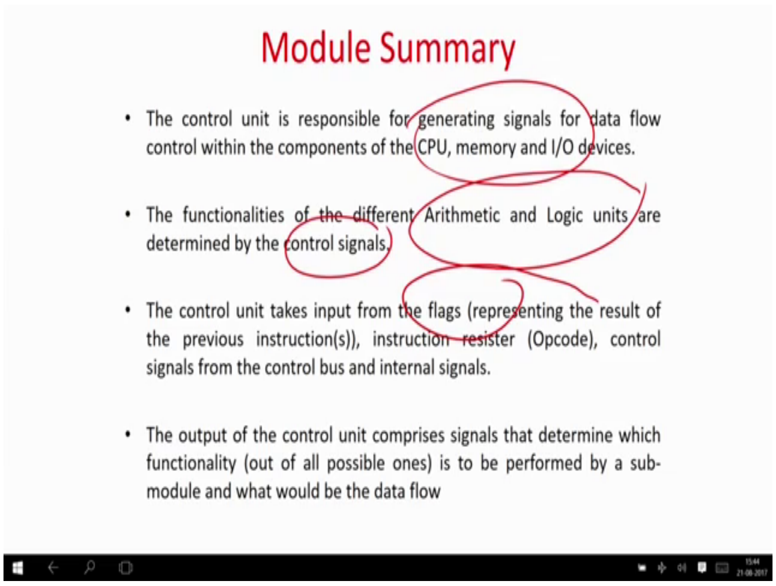
And basically there are several ways to interconnect it that, whether you have a single bus architecture, whether you have double bus architecture, whether you have three bus architecture whether you have some local memory for a local CPU etcetera.

So, we will study in this module how basically different ways of connecting the components like as I told you two bus three bus and three bus organization. So, we will have a look at this look at the idea of such interconnects of such units like register CPU etcetera and then we will see for different such configurations how are signal generated? And how actually the flow of code moves?

Because it is very obvious that if you have three bus system the execution of code will be faster compare to a single bus system, because in if your single bus system is there then there is to be lot of multiplexing because there is a one resource which is a bus and it is only depended on of all data transfers.

But for example, if you have a three bus system then actually some bus can be dedicated for register to registered flow some bus can be used for memory to registered flow and so forth. So, therefore, the control signals will also be different in the latter case compare to the formal case if the three bus system the code execution will be faster because we have now three busses to solve the problem and so forth. So, we will be looking at how control signals are generated and in different processor organizing; organization configurations.

(Refer Slide Time: 04:40)



## Module Summary

- The control unit is responsible for generating signals for data flow control within the components of the CPU, memory and I/O devices.
- The functionalities of the different Arithmetic and Logic units are determined by the control signals.
- The control unit takes input from the flags (representing the result of the previous instruction(s)), instruction resister (Opcode), control signals from the control bus and internal signals.
- The output of the control unit comprises signals that determine which functionality (out of all possible ones) is to be performed by a sub-module and what would be the data flow

Then basically as I told you whatever your code moves and how a code sequence a code is sequenced or if I say that you fetch a code or fetch a instruction or you fetch a data that mean some signals has to be generated. Like we have already seen that if you are looking for a memory, you should generate whether it is a read write signal, we have to generate some values for the memory buffer register and when the memory buffer register has already all the all the data is already given to the memory buffer register from the memory, you should generate a signal saying that the data is now ready and it has been sent to the buffer you can read it.

For example you want to write something to the memory, then you write it into the memory buffer register and then you have to wait for some time all this handshaking signals. For example, if there are 0 flag. So, the 0 flag is set; that means, the control signal will be generated. So, all such different signals how they are generated and how they actually involved in the flow of code or flow of code exhibition in the control unit will be studied.

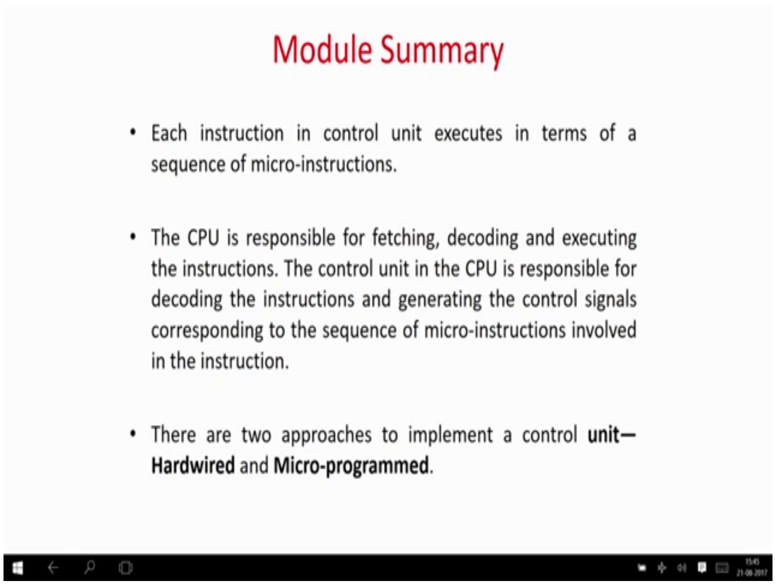
Then we will actually see different functional arithmetic logic units in terms of different control signals. In other words here in this module if you compare to the previous module means look at the same thing, but is the last module we assume that data is fetched, then if there is a control instruction and the flag registers are set accordingly program counter will change, but here we will deal with how the check actually what are the control signals that result in the check and how those control signals are generated?

Like for example, as I told you if there is some flags. So, flags are set and reset they generate some control signals. How those control signals will be utilized and how this how you can design a circuit, which will read the control signals from the ware and modify the program counter.

Now, we will be looking in a much more hardware related issues then a superficial level which will last look in the last model. Like for example, the output of the control units determine the signals which actually (Refer Time: 06:38) to another function module. Like for example, these are peripheral device. So, the peripheral device will interrupt your system.

So, if it is interrupted what type of control signals will be generated? So, all the in an action we look at different type of control signals how they are generated and we look it for different architectures like single bus multiple double bus and three bus architecture.

(Refer Slide Time: 07:01)



## Module Summary

- Each instruction in control unit executes in terms of a sequence of micro-instructions.
- The CPU is responsible for fetching, decoding and executing the instructions. The control unit in the CPU is responsible for decoding the instructions and generating the control signals corresponding to the sequence of micro-instructions involved in the instruction.
- There are two approaches to implement a control **unit—Hardwired** and **Micro-programmed**.

Then basically another very important thing is that whenever how they whenever we will be talking about the control signals, we have to first understand that like a very simple instruction, which we have already seen in the last module. Like ADD a comma B or ADD a comma 30 30 x that is memory location. So, the in fact, this something call a macro instruction, but a macro instruction will also involve some kind of macro structures. Like when we say ADD a comma 30 30 x; that means, first there should be micro instruction which will be involve in to read the data from the memory location 30 30 to the memory to the accumulator or in this case register a or whatever may be instruction like.

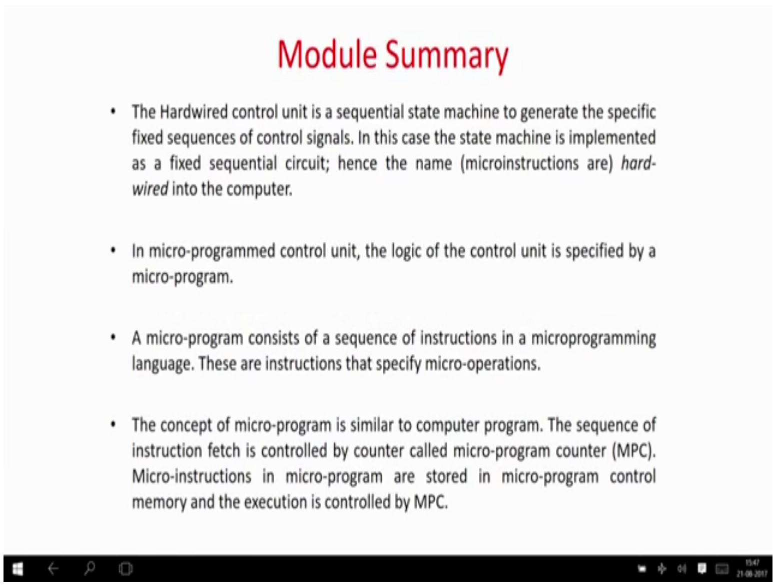
So, in fact, that will involve some kind of micro operation or micro instruction like first we have to give the address of 30 30 to the memory address register, then you have to wait for certain amount of time then the memory will generate a signal the data has been given to the memory buffer register 30 30. So, that is a control signal which has to be rate memory saying that data from 30 30 location has been given to the memory buffer register. Now your instruction register or your accumulator will read the data from the memory buffer register.

So, these small small steps like then; obviously, after instruction is executed you have to increment the value of P C. So, a larger instruction like ADD a comma 30 30 involve several micro instructions. As I told you increment in P C writing the value of the address in the memory address register, reading from the memory buffer register after the signal the memory gives a signal that I have written the data.

So, all this small small instruction we call as micro instructions and we assume that each micro instruction can be operated at one unit of time. So, basically in this module the substantial part will cover that a given a macro instruction or otherwise an instruction; like load, ADD, subtract, multiply, what are the micro instructions involved in it how the how what are the different micro instructions for a macro instruction?

How they are sequenced? How they can be optimized? And how basically we can write for a given code? How can we write it in terms of micro instructions or how basically in fact, you might have observed that I was telling you that for each micro instruction some kind of control signals are generated? Then we will see that basically for a given code what are the micro instructions and then we will start the basically how how we can generate those micro instructions?

(Refer Slide Time: 09:27)



## Module Summary

- The Hardwired control unit is a sequential state machine to generate the specific fixed sequences of control signals. In this case the state machine is implemented as a fixed sequential circuit; hence the name (microinstructions are) *hard-wired* into the computer.
- In micro-programmed control unit, the logic of the control unit is specified by a micro-program.
- A micro-program consists of a sequence of instructions in a microprogramming language. These are instructions that specify micro-operations.
- The concept of micro-program is similar to computer program. The sequence of instruction fetch is controlled by counter called micro-program counter (MPC). Micro-instructions in micro-program are stored in micro-program control memory and the execution is controlled by MPC.

So, micro instructions basically there are two approaches one is called the hardware approach and one is called the micro programmed approach; which basically generate the control signals depending on your micro instructions.

So, in this module we will be looking at depth in both the ways. So, in a hardware control based micro instruction or control signal generation means; we will have a final state machine, which is a hard coded machine, which is implemented in the hardware and based on your micro instructions or the micro instructions corresponded to the main instruction, the your flow will move in this sequential machine, because we know that it is a sequential operation. So, a sequential state machine will move and it is state will generate the control signals. And we called it as a hardware business, because it is fixed to the hardware and you cannot do any changes there.

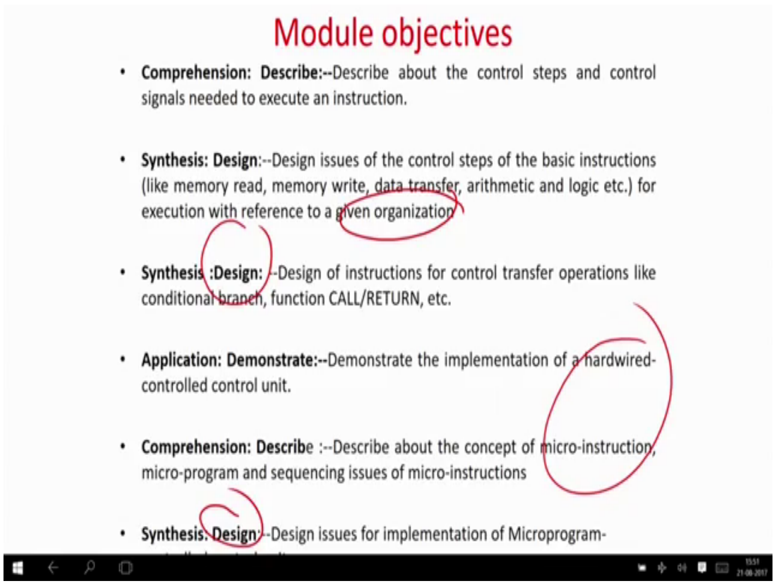
But of course, a more flexible version of this is called micro program environment. So, in micro program environment, you can assume that basically the same thing like a normal code involve, but in this case your micro instructions would not be hardwired like a hardwired control, but you will have a another simplified set of codes for each macro code. In fact, for a given instruction which I am for the timing calling micro instructions, it will involve some kind of micro instruction.

So, micro instructions will be very similar to a normal set of instructions and, but instead of a program counter, which is involve for the macro instructions here there will be something called the MPC. Who is called the micro program control instructions construction unit?

Main memory it will be a separate memory for this micro instructions or in other words in case of micro programmed based unit you can assume, it that for each macro instructions there is a simple code corresponding to the micro instructions and it will also execution sequence like the macro instructions, but instead of large instructions like ADD A comma B here we will have small small instructions like; load memory address register from the instructions register.

Load the value from the memory to the memory buffer register and read the signal; that means, now the instructions are at the micro level, which can be executed at each at each time unit, but you can think in like macro ins macro instruction 1 2 3 4 each macro instruction will lead to some micro instructions. It will also be stored in a micro program memory it is also have moment like 1 2 3 4 and instead of the program counter here we will have the micro program environment. So, it is more flexible. So, more details will come up when will with the absolute exact modules will reach through.

(Refer Slide Time: 11:45)



## Module objectives

- **Comprehension: Describe:**--Describe about the control steps and control signals needed to execute an instruction.
- **Synthesis: Design**:--Design issues of the control steps of the basic instructions (like memory read, memory write, data transfer, arithmetic and logic etc.) for execution with reference to a given organization
- **Synthesis :Design:** :-Design of instructions for control transfer operations like conditional branch, function CALL/RETURN, etc.
- **Application: Demonstrate:**--Demonstrate the implementation of a hardwired-controlled control unit.
- **Comprehension: Describe** :--Describe about the concept of micro-instruction, micro-program and sequencing issues of micro-instructions
- **Synthesis: Design**:-Design issues for implementation of Microprogram-

So, now if you look at it, what are the objectives of this module? So, the objective of the module first it is a comprehensive objective you will be able to describe about the control steps and control signals needed to execute an instruction this is one of the most important part of this module. That will given a instruction sometimes I will call it macro instruction to different is differentiated from the micro instructions. So, if I give you an instruction like I said that note accumulator 30 30. So, you will be able to tell me what are the exact control steps and what is the exact control sequences required to do it; then this is a synthesis objective. So, synthesis objective says that design issues of control steps of the basic instructions like read memory for execution with reference to a give an organization.

That means you will able to design a system or a computer control unit system, with given to a given organization; that means, if it is a single bus or multiple bus, then you will be able to design the control steps require you can also design the micro instructions and the control signals required to be generated you it can be a micro program control it can be hardware control.

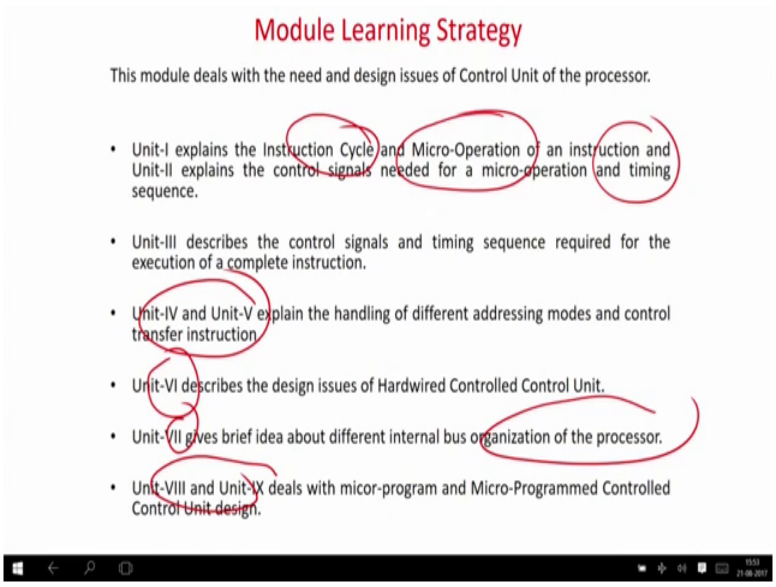
So, you will be able to design such a system based on a given organization. Of course, you will be also able to design that is an synthesis objective design instruction for control operations like branch function called etcetera. So, in fact, I mean the 2 design here we have separated out the 2 in I mean synthesis objectives.

One is for the normal arithmetic data transfer operations and another set for branch instruction interrupt instruction and call return instructions.

Then of course, 2 important things you will be able to design this control signal based on hardware based control unit and micro program based control unit that is the hardware business as well as software business. And the last instruction is again a design instruction. So, you design issues for implementation of the micro program as well as your hardware control unit you will be able to design both, compare among them both and find out which is the more optimized implementation at any point of time.

So, this is actually the module main module objectives.

(Refer Slide Time: 13:49)


## Module Learning Strategy

This module deals with the need and design issues of Control Unit of the processor.

- Unit-I explains the Instruction Cycle and Micro-Operation of an instruction and Unit-II explains the control signals needed for a micro-operation and timing sequence.
- Unit-III describes the control signals and timing sequence required for the execution of a complete instruction.
- Unit-IV and Unit-V explain the handling of different addressing modes and control transfer instruction.
- Unit-VI describes the design issues of Hardwired Controlled Control Unit.
- Unit-VII gives brief idea about different internal bus organization of the processor.
- Unit-VIII and Unit-IX deals with micor-program and Micro-Programmed Controlled Control Unit design.

Now, look at the module learning strategy. So, basically unit 1 and unit 2 will be basics of this module, where we will study instruction cycle and the micro operation of an unit. Basically in first unit we will be learning that what is an instruction cycle that we know fetch decode execute sometimes you have an interrupt and so forth and one of the micro instructions available for them.

And the second one basically will deal with time in sequence that exactly, what is the time cycle and more integrated details of the control signals will be set it over there.

So, first unit will tell what are the signals? The second unit will tell what are the exactly timing sequences in terms of timing diagrams? Unit 3 will describe the control signals I mean timing sequence re required for complete execution of a continuous instruction. In 1 or 2 will be building the basics and in third set unit we will take a large code and will take an organization like a single bus multiple bus and we will see how it goes?

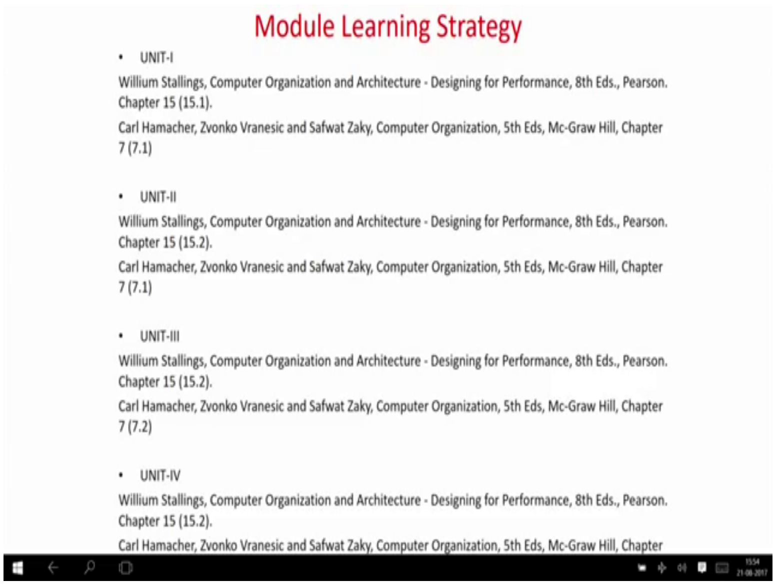
Instruction 4 and 5 will take different addressing mode and we will study the same things. Like we have already seen different type of instruction modes are available oh sorry different type of addressing modes are available like direct indirect base displacement.

So, we will see how the control signal generation changes when are whenever there are different types of addressing modes. Instruction 6 will describe about the hardware control unit. So, now, in the instruction 6 we will deal with, but if you are given a set of instruction, if you want to make a hardware control unit how to design that? Inst unit 7 we will tell you about the different type of bus architecture, basically as I was studying that study with in depth of different type of single bus multiple bus and we will study for all of them how we controlled set instructions? And last 2 units will basically deal on deal with how to design micro program control units for different bus organizations.

Basically that is what the idea is? So, initially we will start with very basic instruction set what are the micro instruction for that, then we will look at the timing sequence then we will see if for different set of instruction or different addressing mode how they change, then we will give you an idea of hardware control based design and then we will also give you an idea for how to design a micro programmed control unit for generating different type of signals. And control signals in a control unit and also we look at different architecture terms of different bus system bus single and multiple.

So, basically I am not going to read out this slide you can look at it we are describing here basically what are the, what are different in different units?

(Refer Slide Time: 16:19)



## Module Learning Strategy

- UNIT-I

William Stallings, Computer Organization and Architecture - Designing for Performance, 8th Eds., Pearson. Chapter 15 (15.1).

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization, 5th Eds, Mc-Graw Hill, Chapter 7 (7.1)

- UNIT-II

William Stallings, Computer Organization and Architecture - Designing for Performance, 8th Eds., Pearson. Chapter 15 (15.2).

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization, 5th Eds, Mc-Graw Hill, Chapter 7 (7.1)

- UNIT-III

William Stallings, Computer Organization and Architecture - Designing for Performance, 8th Eds., Pearson. Chapter 15 (15.2).

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization, 5th Eds, Mc-Graw Hill, Chapter 7 (7.2)

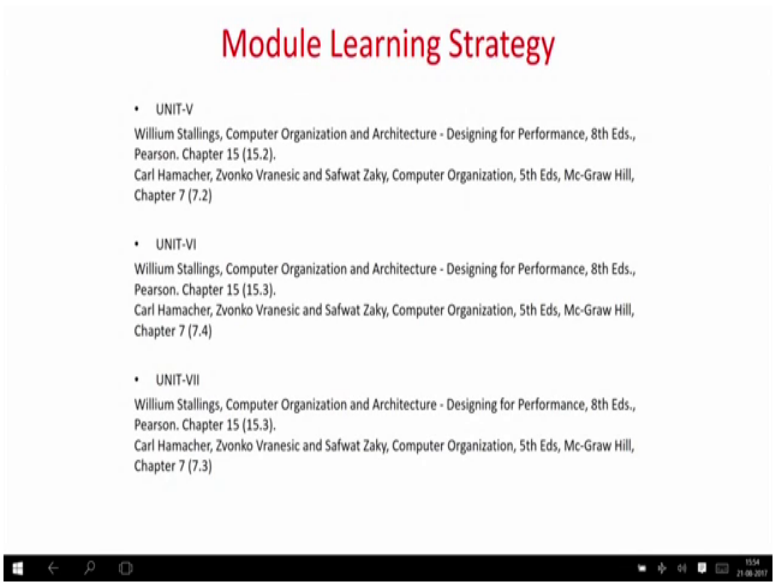
- UNIT-IV

William Stallings, Computer Organization and Architecture - Designing for Performance, 8th Eds., Pearson. Chapter 15 (15.2).

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization, 5th Eds, Mc-Graw Hill, Chapter

And from which book you can study this basically we are referring Willium Stallings, book a Hamachers book. So, exactly which unit which topic and which module you have to read like for example, in unit one which chapter you have to read or all the detail down in this slides basically.

(Refer Slide Time: 16:36)



## Module Learning Strategy

- UNIT-V

Willium Stallings, Computer Organization and Architecture - Designing for Performance, 8th Eds., Pearson. Chapter 15 (15.2).
Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization, 5th Eds, Mc-Graw Hill, Chapter 7 (7.2)

- UNIT-VI

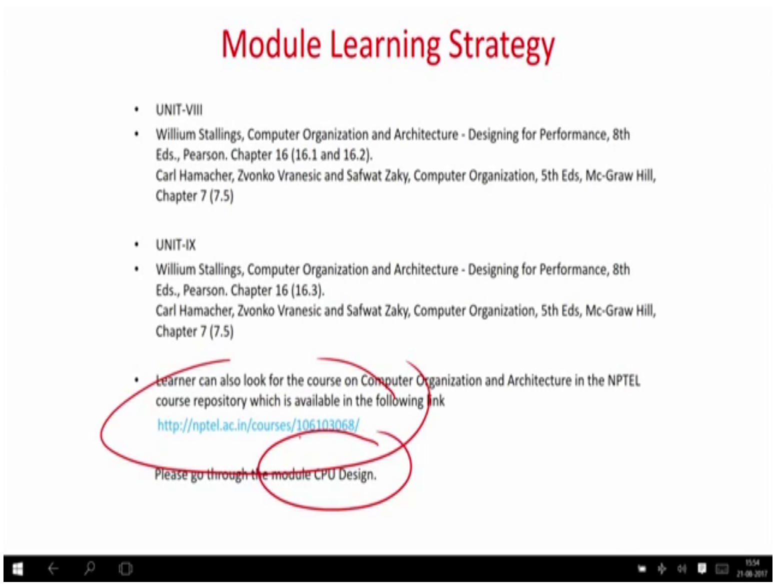
Willium Stallings, Computer Organization and Architecture - Designing for Performance, 8th Eds., Pearson. Chapter 15 (15.3).
Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization, 5th Eds, Mc-Graw Hill, Chapter 7 (7.4)

- UNIT-VII

Willium Stallings, Computer Organization and Architecture - Designing for Performance, 8th Eds., Pearson. Chapter 15 (15.3).
Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization, 5th Eds, Mc-Graw Hill, Chapter 7 (7.3)

So, I am just keeping this slide for a few minutes.

(Refer Slide Time: 16:38)



## Module Learning Strategy

- UNIT-VIII
- Willium Stallings, Computer Organization and Architecture - Designing for Performance, 8th Eds., Pearson. Chapter 16 (16.1 and 16.2).
  Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization, 5th Eds, Mc-Graw Hill, Chapter 7 (7.5)

- UNIT-IX
- Willium Stallings, Computer Organization and Architecture - Designing for Performance, 8th Eds., Pearson. Chapter 16 (16.3).
  Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization, 5th Eds, Mc-Graw Hill, Chapter 7 (7.5)

- Learner can also look for the course on Computer Organization and Architecture in the NPTEL course repository which is available in the following link
  http://nptel.ac.in/courses/106103068/
  Please go through the module CPU Design.

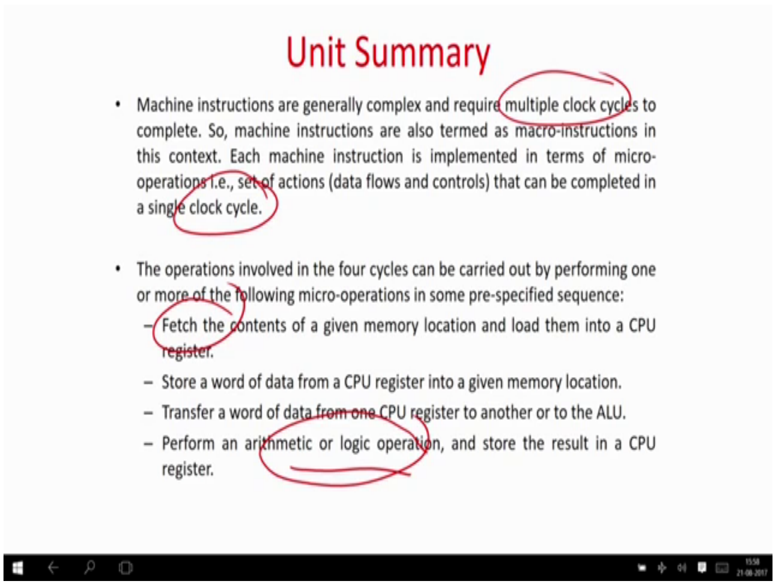
You can look through it also you can go from for this NPTEL based course, which is a web course on a computer organization architecture you can go through this in this way you go to the module on CPU design.

So, basically this gives an overview o on a very pedagogical sense that what is the objective of the module on control unit, what is the basic? Summary of the unit, what an module and unit what we are going to expect out of that and what are the objectives you are going to meet after this module is complete?.

Now we are going to go for the first unit. So, as we have told you that in the first unit is a very basic unit in this case we will just look at different macro instructions. In fact, instructions again I am turning them into macro instructions because I want to differentiate the micro instructions.

So, in this case we will see a instruction cycle, which are already saying fetch, decode, execute store and sometimes there may be an interrupt and what are the micro instructions involve for reach of the instruction that is what is the first unit on. That is we are going to see basically given a macro instruction, how it can be divided into micro instruction and other as already told you micro instructions basically are the atomic instructions, which execute in a single clock unit and which totally comprise the generic instruction. So, this is the first module that is instruction cycle and micro operation.

(Refer Slide Time: 17:59)



## Unit Summary

- Machine instructions are generally complex and require multiple clock cycles to complete. So, machine instructions are also termed as macro-instructions in this context. Each machine instruction is implemented in terms of micro-operations i.e., set of actions (data flows and controls) that can be completed in a single clock cycle.
- The operations involved in the four cycles can be carried out by performing one or more of the following micro-operations in some pre-specified sequence:
  - Fetch the contents of a given memory location and load them into a CPU register.
  - Store a word of data from a CPU register into a given memory location.
  - Transfer a word of data from one CPU register to another or to the ALU.
  - Perform an arithmetic or logic operation, and store the result in a CPU register.

So, in this unit at the pedagogical sense what we are mainly going to look at this unit. So, as I told you machine instructions are generally complex and require multiple clock cycles to complete machine. That is as I told you if you have a indirect machine instruction, that is ADD indirect a 30 30; that means, the location of the variable, which has to be added with treat a is not available at 30 30 at memory location 30 30, you have to we will find another address and you have to go to that address there with they will get the actual value which has to be added with a so, the indirect mode of address.

So, if I have something like ADD a 30 immediate. So, of course, you can understand that the immediate mode of instruction or immediate addressing mode will be extremely fast compared to the indirect. So, it says that all the instructions are of different complexity and they will take multiple clocks to execute. So, basically what happened? So, each instruction basically has to be divided into some kind of atomic instructions or micro instructions then can be implemented in a clock cycle. That is what will be the main emphasis of this unit in, which case we will take different instructions and we will tell you basically what are the micro instructions available for that?

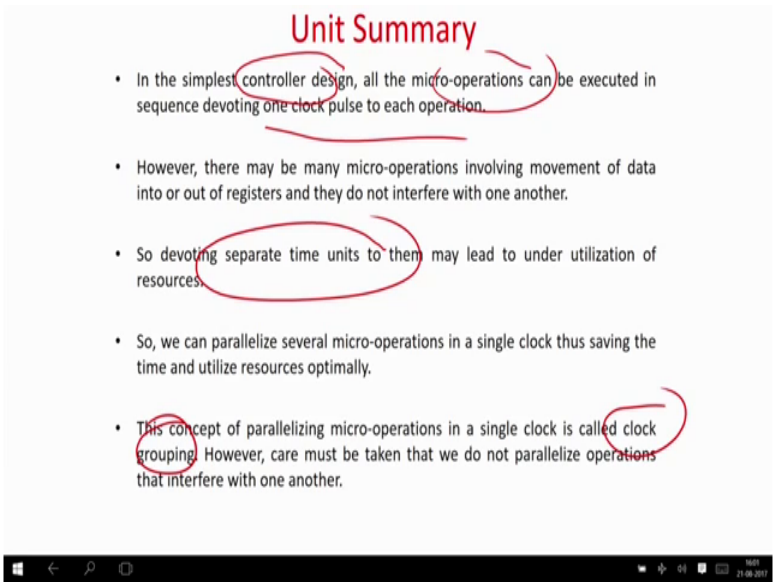
The operation the operations involved in the 4 cycles can be carried out using 1 or 4 micro operations in some predefined frequency. Like basically I mean generally 4 cycles as I already told you fetch decode execute and store. So, basically what happens fetch we already know that. So, first is the fetch the contents of the memory and load them to a CPU register, store the word of a data from a CPU register to a given memory location, that is your load store instruction transfer the data from CPU register to another CPU perform arithmetic operation and store the result in a CPU register. So, you can see that the different types of these are data transfer operation and actually there is one arithmetic operation.

So, if you think about most of the instructions can be pre define in such kind of a thing you pre load from memory location, store to a memory location, transfer data in between different register of the CPU and are you perform the arithmetic and logic operation and we first store it in register and then we can write in this CPU.

So, basically this 4 are in a nutshell in a very broad terms can be classified as different type of data movements in a control unit. So, if I can generate some micro instructions for this given broad flavor of data movement, that will actually give you a very good idea that how a macro instruction can be divided into micro instructions.

Because as I told you memory to memory operation directly, we do not support we has take a data from the memory to a register, then we can again write it back after doing some logical operation. So, in more or less this 4 type give you a basic idea and for this type of four basic ideas; we will try to see what are the different types of micro instructions.

(Refer Slide Time: 20:51)



## Unit Summary

- In the simplest controller design, all the micro-operations can be executed in sequence devoting one clock pulse to each operation.
- However, there may be many micro-operations involving movement of data into or out of registers and they do not interfere with one another.
- So devoting separate time units to them may lead to under utilization of resources.
- So, we can parallelize several micro-operations in a single clock thus saving the time and utilize resources optimally.
- This concept of parallelizing micro-operations in a single clock is called clock grouping. However, care must be taken that we do not parallelize operations that interfere with one another.

So, as I told you in a very very simple Nutshell nowadays process are more complicated you can have different type I mean clock sequence means some of the micro instruction will take 2 clocks some will take 1 clock. So, all this complications are there, but for the time being we are taking a very simple controller design because it is the u g first level course or architecture.

So, we will assume that micro operations are such operation which can be plan in 1 clock wise more or less even in which sophistication architecture also generally one micro instruction means one unit of time, but some variations happen. So, that is what is the idea, we will take a simple control design, which is this one which is simple thing? However, there will be micro operation involving data movement of data in or out of the registers that you are interfere with in another we will see that basically what id idea is there in that case what it says what we will study in this case there is something called optimization.

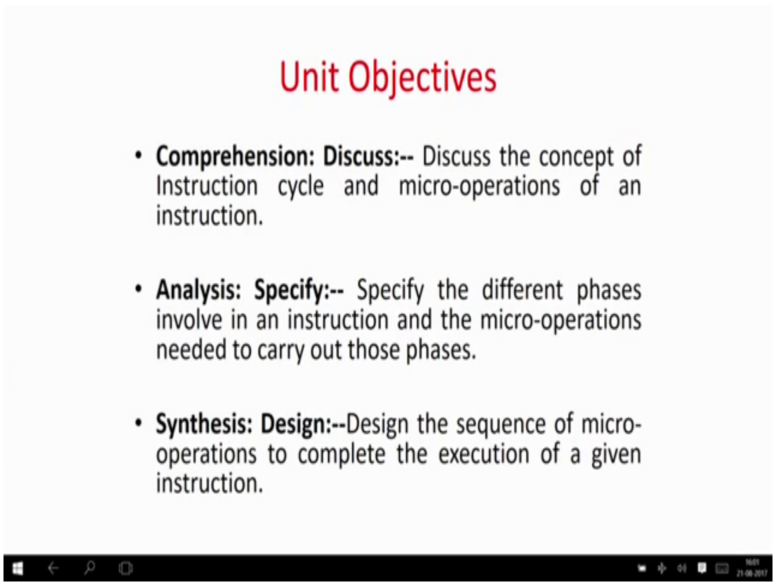
Like for example, one micro instruction you are reading some data from the memory to a register, but as the same time your CPU is bring some computation and writing register to another register to say and you want to at the same time you want to read something from a memory location to register X. So, they are non-conflicting instructions.

So, you can actually mass this 2 instruction at 1 time unit; that means, there all micro instruction they will take one, but you need not sequence them because they are non interfary. So, if you give directly separate time units to all this then you require more number of micro instructions to operate the macro instruction. So, what we can do is that we can optimize and try to put in parallel or in one time unit some of the micro instructions which do not require a, which can which do not require a separation in time there, but if for example, as we will see there is some dependence on the first micro instruction on the other then we cannot put in a single time unit.

So, basically also one important idea is that we assume that all the micro instruction take single unit of time that is fine, but depending on the instructions if they are non-depended instructions we can put them in one time which is actually. So, that which is actually called clock grouping the term is called clock grouping that you put 2 instructions, which are non-depended one another you put in the same time unit. So, there is optimization in time.

So, we actually call it as a crop clock grouping so, groping. So, we will also see that given a macro instructions one of the micro instructions and we assume that is micro instructions take 1 unit of time, then we will see if there is a non-dependent micro instruction, then we will try to put in time unit 1 time unit and we will see what are the minimum number of time units require to execute a macro instruction, which is actually called crop clock grouping and it actually optimizes on the time.

(Refer Slide Time: 23:28)



## Unit Objectives

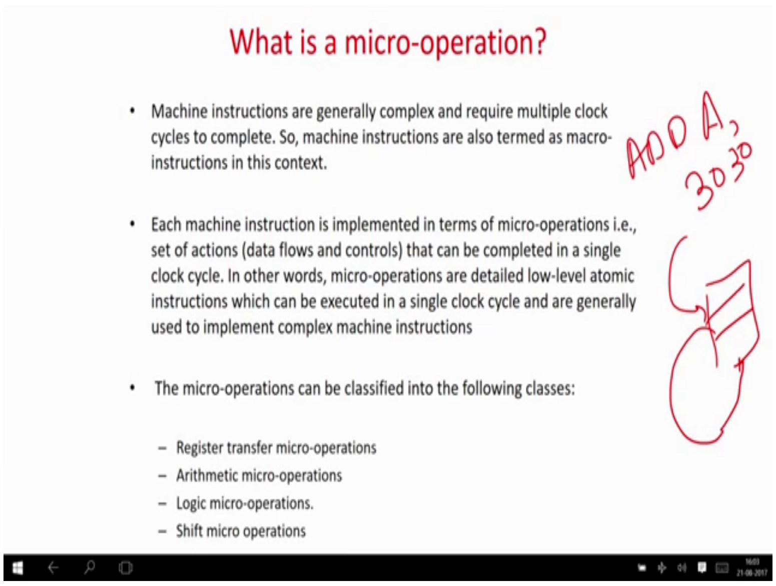
- **Comprehension: Discuss:--** Discuss the concept of Instruction cycle and micro-operations of an instruction.
- **Analysis: Specify:--** Specify the different phases involve in an instruction and the micro-operations needed to carry out those phases.
- **Synthesis: Design:--**Design the sequence of micro-operations to complete the execution of a given instruction.

So, what are the unit objectives in this unit objectives the first objective is a comprehension objective, which you will be able to discuss the concept of instruction cycles, macro operations of an instructions sorry the micro operations involved in a macro operation, that is the first objective of this unit. That given any macro instruction you will be able to tell what are the micro instructions of that.

Also specify the different phases involved in an instruction and the micro operation needed out to carry those stages like, as I told you that if I say that load X or load accumulate 30 30 there are certain state like for example, the instruction has to be taken from the memory to the instruction register by a memory buffer register, then 30 30 will be loaded from the instruction register to the memory address register, then the memory address, then the memory will give the data from the 30 30 memory location to the memory buffer register which will be again read back to the accumulator. So, there are different stages.

So, we will discuss the different stages you will be able to specify the different stages require to execute a macro instruction instead of micro instructions and finally, is a design objective given any instruction set you will be able to design the micro instruction require to opt execute the macro operation.

(Refer Slide Time: 24:37)



## What is a micro-operation?

- Machine instructions are generally complex and require multiple clock cycles to complete. So, machine instructions are also termed as macro-instructions in this context.
- Each machine instruction is implemented in terms of micro-operations i.e., set of actions (data flows and controls) that can be completed in a single clock cycle. In other words, micro-operations are detailed low-level atomic instructions which can be executed in a single clock cycle and are generally used to implement complex machine instructions
- The micro-operations can be classified into the following classes:
  - Register transfer micro-operations
  - Arithmetic micro-operations
  - Logic micro-operations.
  - Shift micro operations

So, again now we are coming to the module. So, what is a micro instruction machine instructions are generally complex and require multiple cycles to complete? So, machine instructions are also termed as macro instructions in this context. So, as I was telling from the very beginning that if you are taking a large instruction like say I was really say ADD accumulator and let me specifically then I called 30 30. So, in this case as I told you is a macro instruction. So, ADD a is a accumulator from 30 30 memory location.

So, you can assume that if it is a direct instruction time take will be certain if 30 30 is not a memory location, but if it a immediate data then it will be extremely fast and if 30 30 ADD this ADD is a indirect instruction; that means, first you have to go the memory location 30 30, there is another instruction over here you have to go to another part of the memory and there you will get the data.

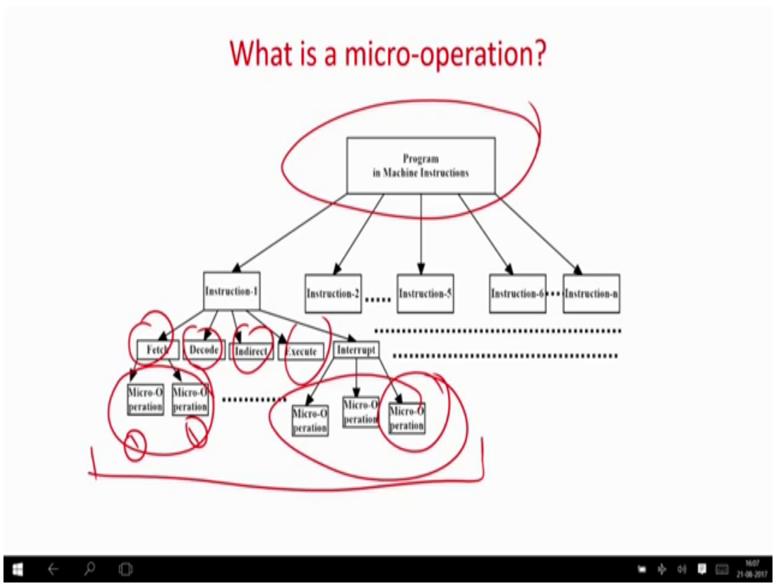
So, memory instructions are very complex depending on the addressing mode and what it operates on etcetera. So, basically we divided into granular level which is called the ma micro instruction and each micro instruction can execute in a single time unit taking multiple micro instructions you go for a macro instruction.

That is what in is told in the second point, that basically it says that each machine instruction has implemented in terms of micro instructions, data flows and controls, that can be executed in a single clock pulse. That is if I want to get a data from a memory to the memory buffer register we generally assume that the clock duration is such, because generally in a simple terms memory itself writes are done in a single clock pulse. So, we mean we actually mean design or clock in such a fashion or we limit our speed in such a fashion that most of the micro instructions execute in a single clock pulse.

That is I mean if you are not able to do it that fast you have to actually relax the clock period. So, in that case the frequency of the person will come down if a very fast memory, your very fast interfaces multiple buses then many of the micro instructions can be executed in less amount of time. If that can be done then you can reduce the time period and you can say there is a poster is a faster ok. And in other words micro operations are detailed lower level atomic instructions, which can be executed in a single clock and are generally used to implement complex machine instructions; that means, you join the micro instruction and make a macro instruction or whenever you are design a macro instruction you have to go for in between you have to go in delay in the leaf level there will be micro instructions.

So, micro instructions I told you different classes register transfer, arithmetic micro operations logic and shift basically that I told you that is a data transfer and basically your logic transfer, where mainly you can mainly look at the broad classes.

(Refer Slide Time: 27:13)



So, what is there you have a this is a very nice figure, which actually shows you gives a description that what is a micro operation is a program, which will have lot of codes like; load, store ,add multiple etcetera. So, it has around n instruction.

So, each instruction as you can see can be divided into fetch decode, indirect and execute. I mean some time indirect may not be there for different addressing modes then fetch; that means, some instruction has to be fetch from the memory to the instruction register.

So, it will involve some micro operations then interrupt it will involve some kind of micro instructions like a fetch. So, what will be the different type of micro instructions, if you can think in such a manner fetch means first the program counter will have the address of the memory from where the data will be loaded.

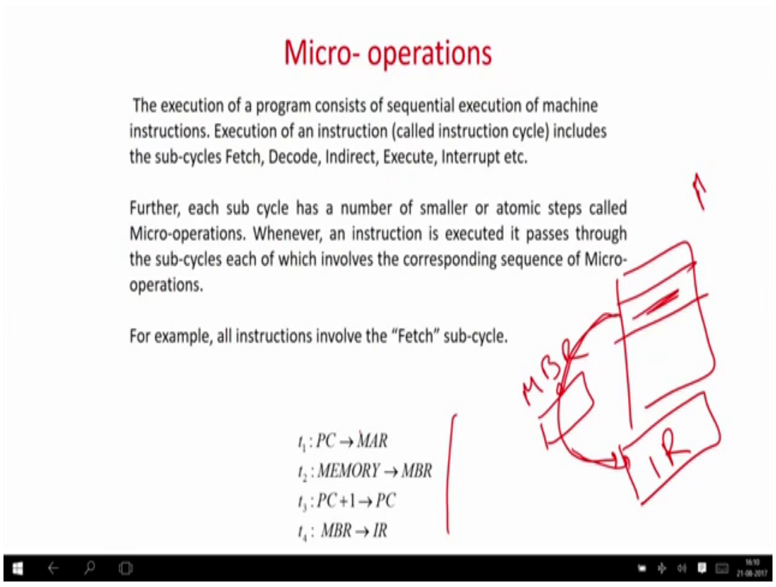
Data will come from the memory to memory buffer register from the memory buffer register it will go the accumulator certain steps are there, and then basically program counter will be implemented. So, these are some of the micro instruction which is required in fetch, decode means what your there will be a hardware, which will actually read this off code from the instruction and generate certain signals in this module will be mainly looking at how signals are generated for instruction. Like for example, if the off code is 0 0 1 may be 0 0 1 stands for load. So, the hardware will generate the signals corresponding to the load after reading this off code of the instruction.

So, that is again will be the decode part. So, again after reading the load part again you have to something some you have to take instruction from the instruction register you have to take the instruction, look at the other part of the instruction that will have the data address like 30 30. So, you have to take that and then that will again go to memory buffer register sorry memory address register again the data will be fetch.

So, lot of small integrated atomic details are required when a instruction is fetch the instruction is executed like instruction has some stage, like fetch, decode, indirect, execute, or interrupt and each stage also has different atomic level. So, there are now actually the micro instructions and this one will take 1 unit of time this will take 1 unit (Refer Time: 29:17) take time and so forth.

So, again the number of micro instructions for a given instruction depends on the instruction time. So, for a simple instruction the number of micro instructions will be less; for a complex instruction the number of micro instruction will be larger, but this micro instruction is taking place at a single clock period of time. So, it is a very nice and a synchronized way of handling the system.

(Refer Slide Time: 29:38)



## Micro- operations

The execution of a program consists of sequential execution of machine instructions. Execution of an instruction (called instruction cycle) includes the sub-cycles Fetch, Decode, Indirect, Execute, Interrupt etc.

Further, each sub cycle has a number of smaller or atomic steps called Micro-operations. Whenever, an instruction is executed it passes through the sub-cycles each of which involves the corresponding sequence of Micro-operations.

For example, all instructions involve the "Fetch" sub-cycle.

$t_1 : PC \rightarrow MAR$

$t_2 : MEMORY \rightarrow MBR$

$t_3 : PC+1 \rightarrow PC$

$t_4 : MBR \rightarrow IR$

Now, we will take different examples of micro operations and see how it repairs. So, for example, we are going to take a "Fetch" sub cycle. So, we are going to take fetch decoding indirect execute interrupt. So, many subs are there we will take one at a time.

So, let us take fetch; that means, memory is there and this is what is the instruction these has to be fetch to the instruction register that is what is the fetch; that means, the program counter basically already has given the value in the memory address register; the memory address register. Now it has the memory address register has to be given the value of the C P.

Now the program counter value is already telling that this is the address of the memory where the instruction is there it will sell it to the instruction, but not directly basically as you already know that there is something called memory buffer register. So, memory will give the data from this in from this location the memory buffer register memory buffer register is in the instruction register and you have to increment the PC.

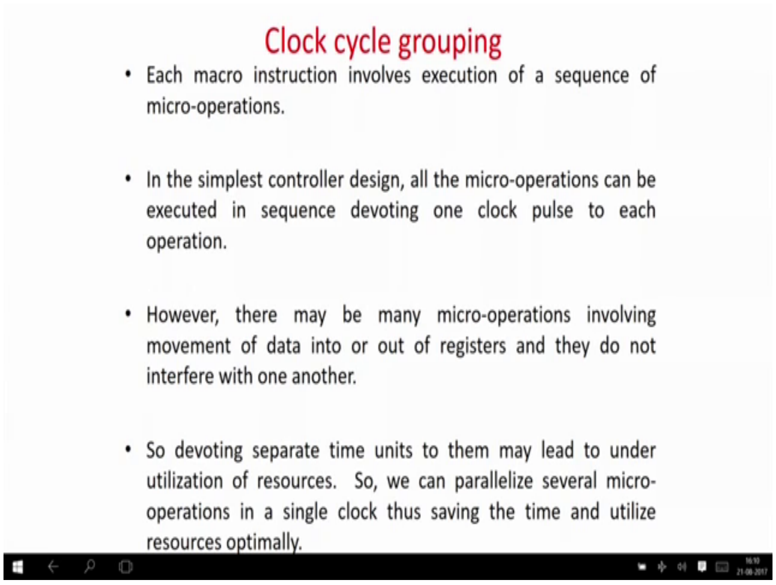
So, these are some of the micro stage of operations which are require to do it. So, what are the first time units what will happen? The program counter will load the memory address register. That is program counter P C value is there it is telling that that is your memory address register. The program counter is telling that this is the location from where I need the instruction. Then this part is done then what we will you do now these memory will put this data to the memory buffer register, that is the second stage second unit of time that the memory will put the data in the memory buffer register.

Now what now you have to increment the value of program counter, because ne next recession has to be fetch in next cycle and then the memory buffer register will write to a instruction register.

So, you can see that I have having 4 basically micro instructions and they involved in the fetch instruction. So, all these program counter value to memory address register, memory buffer; memory to the memory buffer register program counter incrementing, memory buffer register to the incrementing to the instruction register, they all take some single unit of clock and these 4 steps or micro instructions generate correspond to the mac correspond to the fetch sub cycle of the macro instruction, which may be something like ADD A comma B. Because all instructions will have this 4 sub cycles or 4 or 5 cycles; like fetch, decode, execute, in between you have a you have an indirect and towards the memory indirect.

So, if you look at this table. So, many instruction micro instructions will be level for an instruction at present we have look what the micro instruction for the fetch part are.

(Refer Slide Time: 32:10)



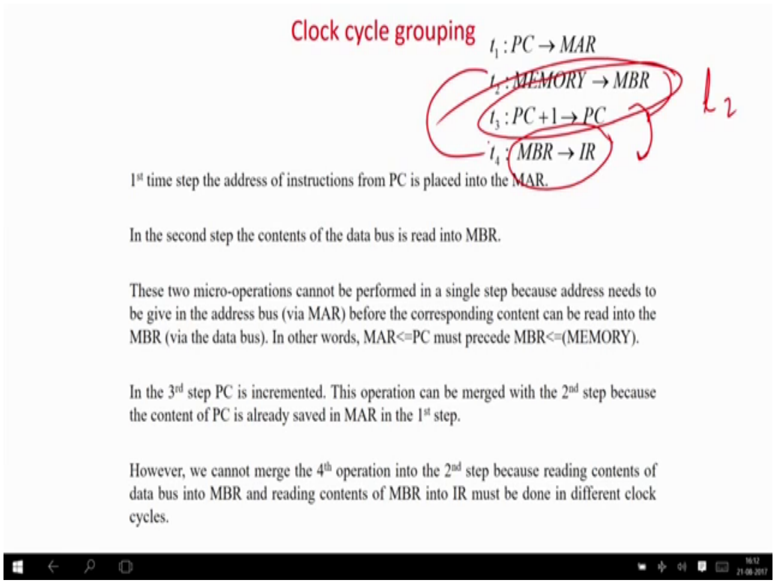
## Clock cycle grouping

- Each macro instruction involves execution of a sequence of micro-operations.
- In the simplest controller design, all the micro-operations can be executed in sequence devoting one clock pulse to each operation.
- However, there may be many micro-operations involving movement of data into or out of registers and they do not interfere with one another.
- So devoting separate time units to them may lead to under utilization of resources. So, we can parallelize several micro-operations in a single clock thus saving the time and utilize resources optimally.

Now, the concept of clock grouping is coming. So, is macro instruction involve the sequence of micro instructions, in if you want to keep it very simple just you take a flat implementation, means whatever in the macro instruction then it is there what are the stages you have fetch decoding indirect execute interrupt divided into the micro instructions and take asthmatic out of time?

But you will find out then we always not require to sequentialized it, because some of the 2 instructions micro process can done at a time, in that case you can save unit of time. So, if you can parallelized that that is actually called clock grouping, like in this.

(Refer Slide Time: 32:45)



## Clock cycle grouping

$t_1 : PC \rightarrow MAR$

$t_2 : MEMORY \rightarrow MBR$

$t_3 : PC+1 \rightarrow PC$

$t_4 : MBR \rightarrow IR$

$t_2$

1st time step the address of instructions from PC is placed into the MAR.

In the second step the contents of the data bus is read into MBR.

These two micro-operations cannot be performed in a single step because address needs to be give in the address bus (via MAR) before the corresponding content can be read into the MBR (via the data bus). In other words, MAR<=PC must precede MBR<=(MEMORY).

In the 3rd step PC is incremented. This operation can be merged with the 2nd step because the content of PC is already saved in MAR in the 1st step.

However, we cannot merge the 4th operation into the 2nd step because reading contents of data bus into MBR and reading contents of MBR into IR must be done in different clock cycles.

So, in this fetch right so, P C is equal to you are keeping the value of the memory address register can I actually merge this 2 in a single time take not possible, because in the first unit the value of the program counter will go to the memory address register, you give some time give one unit of time for that, then the memory address register will be Z, now the memory will know that I have to supply the data which is actually the instruction from the address, which is given in the memory address register.

So, if I merge this 2 together there will be a lot of hotspot, because I you tell me that I have to deliver from this address and deliver it now. How can I do that I require some time to read the address, go to the relevant location, bring the data and give it to you. So, I cannot merge t 1 and t 2 at a single point of time, but if you look at it I am giving some because this job is over, you have told me what is the address?.

Now I go and bring the data from that memory location already you have given me from where I have to face the data. Then the job of the program counter is over, because program counter is telling me the address from where I have to give from the giv from the gat the data from.

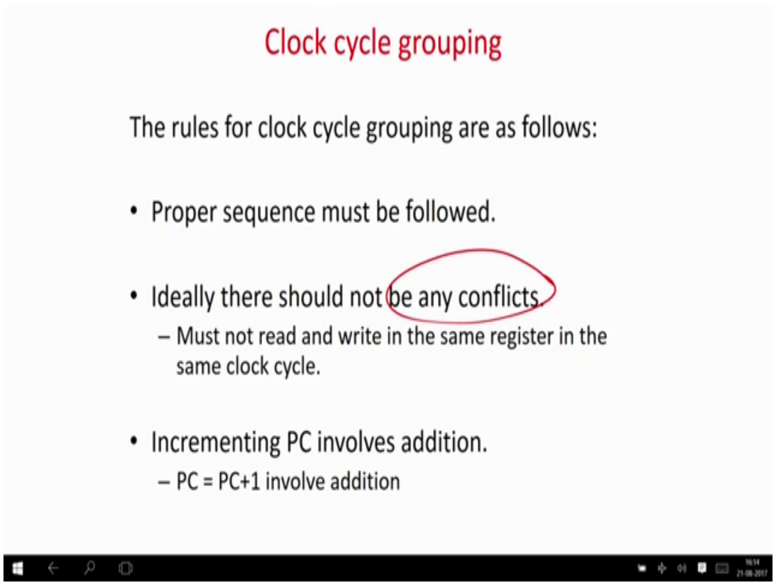
So, you have told me the address and now I have I am going to get the data from the memory. Now the program counter is free. So, better you can implement the program counter. So, it is very officious I can merge step 2 and 3 together, because after reading the value of the program counter to the memory address register my job is done I am free at hand.

So, if you are free at hand then actually I can use the P C that indecent and increment by one. So, memory at taking the data from the memory to the memory buffer register and program counter increment these 2 micro instructions you can do it in time steps 2, because they are 2 non-dependent micro instructions. Of course, this then the last is memory buffer register will be writing it to the instruction register of course, P C you can be merge with t 2 P C increment can also be merge with t 4. So, any way you can merge.

Of course again I cannot merge step 2 and step 4, because step 2 actually tells that I am bring the data from the location. Before the data is bought to the memory buffer register you cannot directly transfer to the I R. So, these 2 are memory in interdependent instructions and 2 and 4 interdependent register instruction. So, there cannot be merge, but this one is a independent instruction even this 2 are dependent. So, you can either merge it with this or you can merge with this.

So, basically 3 time steps or micro instructions are required for 4 micro instructions are require, but in 3 time steps you can complete the fetch stage, that is what is fetch is got a clock grouping that I am grouping. So, either I am grouping basically this 2 guys together or I can must this 2 guys together. So, 4 micro instructions, but I can do it in 3 time steps.

(Refer Slide Time: 35:27)


## Clock cycle grouping

The rules for clock cycle grouping are as follows:

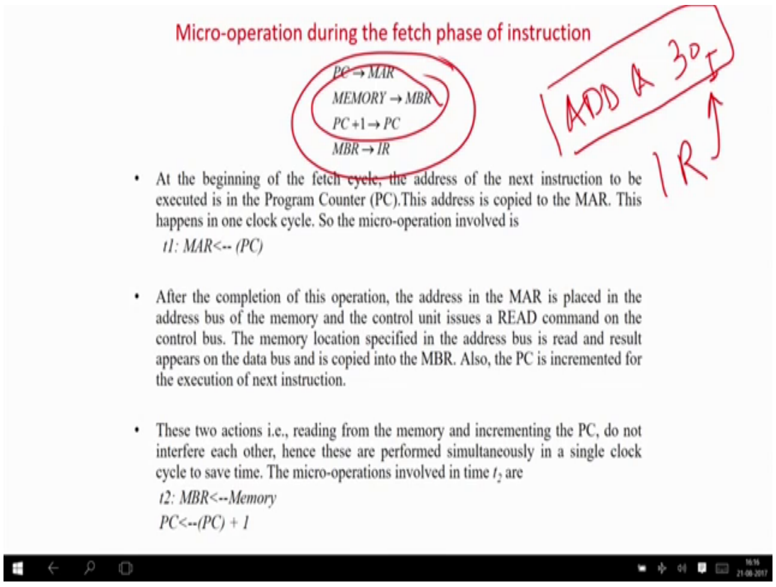
- Proper sequence must be followed.
- Ideally there should not be any conflicts.
  - Must not read and write in the same register in the same clock cycle.
- Incrementing PC involves addition.
  - PC = PC+1 involve addition

So what is clock grouping proper sequence should be maintain you cannot alter the sequence that is I am getting the data; before I put the registers in address that cannot be done. Proper sequence should be follows; ideally there should not be any conflict that is there cannot be any biasness I mean. So, there can be any what we call ways condition kind or a actually conflict of a interest kind of a thing that I want to read from that register at the same time you cannot write to the register. Those instructions are called conflicting instruction, because they are using a common resource either for read and write.

So, you cannot do at the same time and program counter is always involved in an increment. So, you try to feed the program counter at some place after the data is being rate or after the instruction has being rate, then the program counter is free you can try to increment the program counter and try to feed it in some place where there is no conflict.

That is how basically the ideas of clock grouping is very simple sequence you cannot change you cannot have any conflict, like if I want to read from register R 1, you cannot put a micro instruction at that same level in which there is an updating. Similarly we have given an example and, but program counter is bit free because whenever the program counter has given the data to the memory address register to fetch the instruction it is free you can use it and put it in panel.

(Refer Slide Time: 36:42)



## Micro-operation during the fetch phase of instruction

$PC \rightarrow MAR$

$MEMORY \rightarrow MBR$

$PC+1 \rightarrow PC$

$MBR \rightarrow IR$

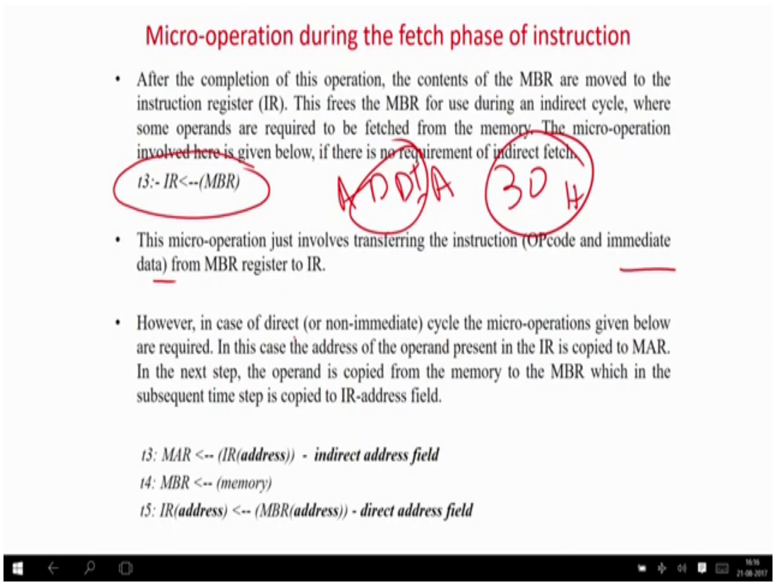
- At the beginning of the fetch cycle, the address of the next instruction to be executed is in the Program Counter (PC).This address is copied to the MAR. This happens in one clock cycle. So the micro-operation involved is
  *t1: MAR<-- (PC)*
- After the completion of this operation, the address in the MAR is placed in the address bus of the memory and the control unit issues a READ command on the control bus. The memory location specified in the address bus is read and result appears on the data bus and is copied into the MBR. Also, the PC is incremented for the execution of next instruction.
- These two actions i.e., reading from the memory and incrementing the PC, do not interfere each other, hence these are performed simultaneously in a single clock cycle to save time. The micro-operations involved in time $t_2$ are
  *t2: MBR<--Memory*
  *PC<--(PC) + 1*

Again we will now see this stage we have already discuss discussing for the fetch.

So, first stage is program counter value the memory address register, memory buffer register will take the value from the memory, you can merge this 2 and the memory buffer register will write to the instruction. This is very simple that is for example, if have an instruction say ADD accumulator, 30 immediate. For such a instruction this is very simple, first you will fetch it then actually may be from this program counter I give the value in the memory address register. So, in that memory address let us say that this instruction is there. So, this will be incremented and then the instruction register will have this part.

Now, you want to actually execute this. Then exactly what happens basically depending on with 30 is an immediate or 30 is a memory location the number of instructions in the decode face will change, that is we are going to look at it.

(Refer Slide Time: 37:40)



## Micro-operation during the fetch phase of instruction

- After the completion of this operation, the contents of the MBR are moved to the instruction register (IR). This frees the MBR for use during an indirect cycle, where some operands are required to be fetched from the memory. The micro-operation involved here is given below, if there is no requirement of indirect fetch.

  *t3:- IR<--(MBR)*

- This micro-operation just involves transferring the instruction (OPcode and immediate data) from MBR register to IR.

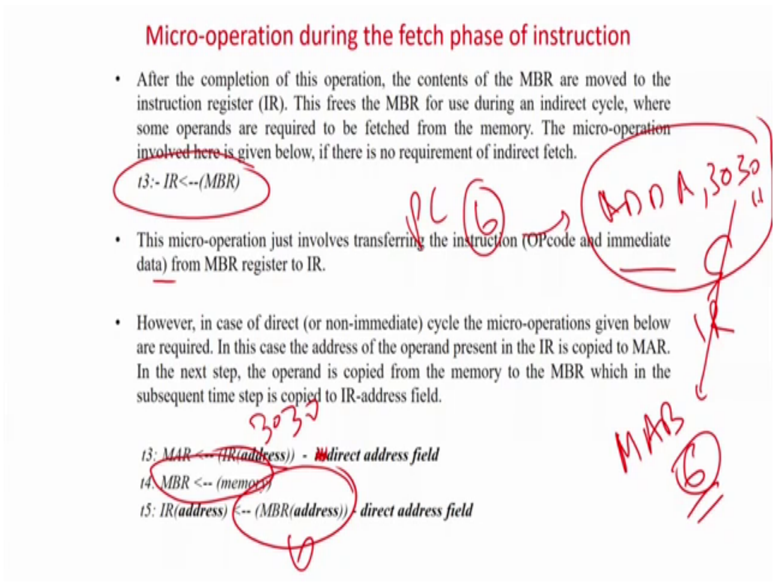
- However, in case of direct (or non-immediate) cycle the micro-operations given below are required. In this case the address of the operand present in the IR is copied to MAR. In the next step, the operand is copied from the memory to the MBR which in the subsequent time step is copied to IR-address field.

  *t3: MAR <-- (IR(**address**)) - **indirect address field***
  *t4: MBR <-- (memory)*
  *t5: IR(**address**) <-- (MBR(**address**)) - **direct address field***

So, in the third stage it says that the memory buffer register will lie to the instruction register. So, if it an immediate data more or less our job is done, because now your instruction register basically has the memory buffer register which is having the data called ADD accumulator 30 H which is a may be immediate.

In case of immediate you did not do anything, because when the memory buffer register has dump the value in an instruction register, you have already face that may be this ADD in ADD immediate off code correspond immediate. So, you can directly take and there is no feather micro instructions require, but assume that in this it is a indirect or it can be a direct, but it is a non-immediate mode of addressing, then is this case what happens?

(Refer Slide Time: 38:28)



## Micro-operation during the fetch phase of instruction

- After the completion of this operation, the contents of the MBR are moved to the instruction register (IR). This frees the MBR for use during an indirect cycle, where some operands are required to be fetched from the memory. The micro-operation ~~involved here is~~ given below, if there is no requirement of indirect fetch.

*t3:- IR<--(MBR)*

- This micro-operation just involves transferring the instruction (OPcode and immediate data) from MBR register to IR.

- However, in case of direct (or non-immediate) cycle the micro-operations given below are required. In this case the address of the operand present in the IR is copied to MAR. In the next step, the operand is copied from the memory to the MBR which in the subsequent time step is copied to IR-address field.

*t3: MAR <-- (IR(address)) - direct address field*
*t4: MBR <-- (memory)*
*t5: IR(address) <-- (MBR(address)) - direct address field*

Say it is let us take ADD accumulator 30 30 H. So, it is a direct addressing mode. So, we are doing that take the value from the memory location 30 30 H, you go to the memory location called 30 30 whatever data is there you ADD it to a accumulator and store it.

So, in this case what happens this one will come to the accumulator. So, this whole thing is now. So, this instruction of the value of 30 30 is now in the instruction register , but now happen you have to now again give this value of 30 30 to again to the memory address register, because this is same some location may called say 6. See program counter value was 6. So, program memory location 6 had this instruction. So, it is now in the instruction register but now again you have to now load the memory address register with this 30 30.

Then again we have to read the value of memory location 30 30 to the memory buffer register and only that can that will again bought to the instruction register and then again you can ADD, it then these are 2 stage process. Immediate means you can directly whatever is in the memory buffer register like we have seen ADD, accumulator 30 you can directly take in the buffer register go to the instruction register ADD it and you are done. There is no feather no step require, but if it an indirect mode or a direct mode and non-immediate. That is data is not available in the instruction itself then you have to go for multiple stages. Like instruction register address you have to again fill in to the memory address register that is this 30 30 again I have to feed it to the memory address register, next memory buffer register memory will right to that.

For example in 30 30 we have the value called 6; that means, we are going to ADD the value 6 2 A as store it back to the accumulator, that is memory location 30 30 let us assume that has the value 6. So, in step 3 basically you are going to read the value of ADD A 30 30 X in the instruction register.

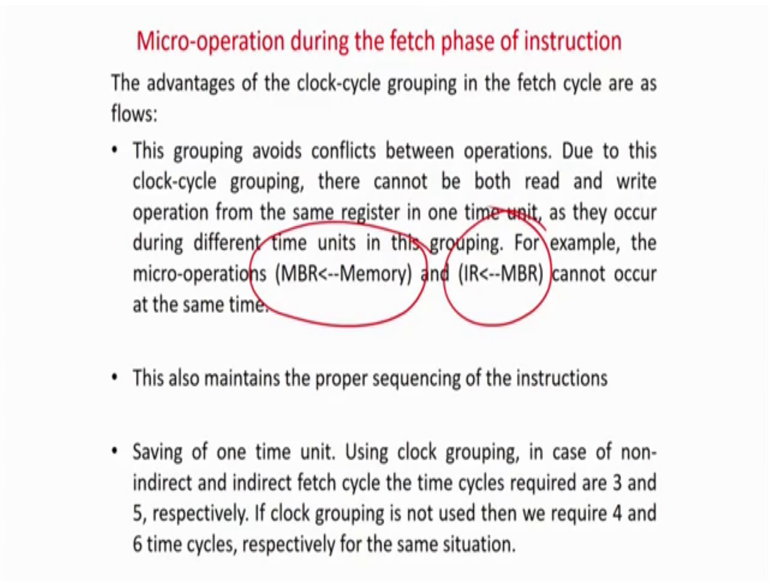
Now instruction register will know that again I have to go for it is a indirect. So, it is a basically direct addressing, but a non-immediate addressing. So, it will again load the value of 30 30 in the buffer register. So, memory buffer register memory address register will have the vale now 30 30 and now the memory buffer sorry the memory address register will take the value of 30 30 from the instruction register, memory ad buffer register will take the value from the memory location 30 30 and put it to the memory buffer register.

And then this memory buffer register, which is having the value of 6 now will be loaded to the instruction register. So, now, the instruction register will have A ADD A 6. So, it will load it to the accumulator after adding. So, in fact, now some more instructions are required like in this case this will be require which you will solve the problem, but again if you see none of the instructions can be put in one time taken. Because in this case you are reading in time t 3 you are reading the instruction resister address that is 30 30 in the memory buffer register. You have to give some time then only the memory address register will be read and you are going to give the value in the memory buffer register.

After sometime the memory buffer register address that is the data from there address that is 6 will be rate to the instruction register. So, in fact, you cannot paralyze because this one is dependent on this and this one is dependent on this. So, you cannot save any time step in that. So, only time which you we save is that in time step 2 we have merge the memory buffer register writing from the memory and we have added program counter.

So, what we have seen. So, if it is a simple immediate mode of addressing then we require 3 steps 1 2 and 3, that is step 1, this can be consider step 2 and this can be consider step 3 and if it is a none immediate mode; that means, may be a direct mode for indirect mode may be several other steps will be require, because more complicate your instruction is more number of micro instructions will be required. So, is that in case of direct we need another 3. So, total number of stages time steps will be 5.

(Refer Slide Time: 42:25)



## Micro-operation during the fetch phase of instruction

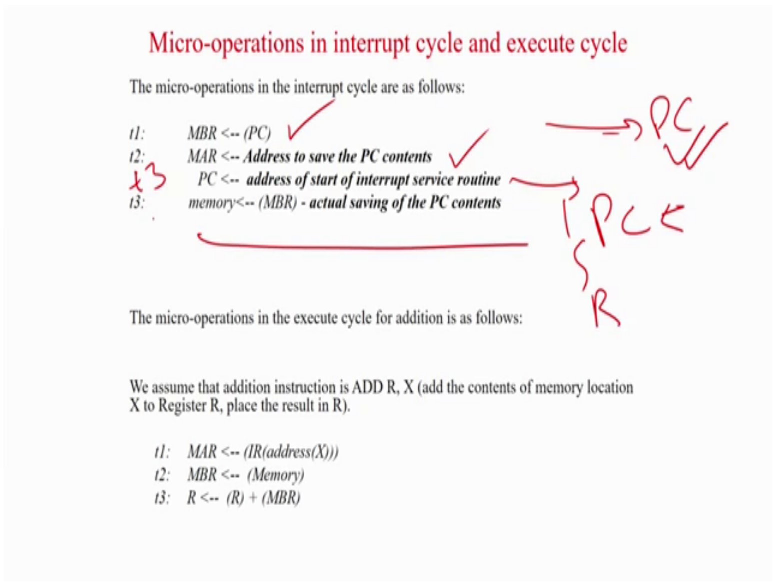
The advantages of the clock-cycle grouping in the fetch cycle are as flows:

- This grouping avoids conflicts between operations. Due to this clock-cycle grouping, there cannot be both read and write operation from the same register in one time unit, as they occur during different time units in this grouping. For example, the micro-operations (MBR<--Memory) and (IR<--MBR) cannot occur at the same time.
- This also maintains the proper sequencing of the instructions
- Saving of one time unit. Using clock grouping, in case of non-indirect and indirect fetch cycle the time cycles required are 3 and 5, respectively. If clock grouping is not used then we require 4 and 6 time cycles, respectively for the same situation.

So, when was telling you is that we are writing in this slide that what are the dependent dependency like for example memory, memory, memory to memory buffer register and instruction register to the memory. We cannot put these 2 micro instructions together, because the memory will memory value will go to the memory buffer register after sometime from memory buffer register it will go to instruction register, you cannot merge this 2. So, you have to have a proper sequence.

So, what we have see if you are not going for any clock grouping when the number of time units require to go for a fetch immediate instruction will be 4. And if you optimize it one will be saved that the increment of the P C will be save then you require 3. Similarly this is a non-immediate mode like a direct mode of instruction. So, these are the 5 or 6, because we are optimizing the program counter increment which can be feet with any other stage.

(Refer Slide Time: 43:17)



## Micro-operations in interrupt cycle and execute cycle

The micro-operations in the interrupt cycle are as follows:

t1: MBR <-- (PC) ✓
t2: MAR <-- **Address to save the PC contents** ✓
t3 PC <-- **address of start of interrupt service routine**
t3: memory<-- (MBR) - **actual saving of the PC contents**

The micro-operations in the execute cycle for addition is as follows:

We assume that addition instruction is ADD R, X (add the contents of memory location X to Register R, place the result in R).

t1: MAR <-- (IR(address(X)))
t2: MBR <-- (Memory)
t3: R <-- (R) + (MBR)

Before we end some other micro operations for different other stages of instruction I am showing our here say for example, this is for the interrupt cycle. So, if there is a interrupt instruction is there what are the micro instructions. So, in the beginning we all know the micro instruction when is interrupt you have to save the value of program counter and whenever the interrupt service routine have been done, you have to again come back and pop up the dairy of program counter and register from when we have left.

So; obviously, we have to store the value of program counter value somewhere. So, you write the value of program counter in the memory buffer register, please note that we are not writing the program counter in the memory address register. Program counter writing in the memory address register means you are fetching some instruction. Here we writing the value of program counter to the memory buffer register, because we are saving the value of P C. So, saving the value of P C means the value of P C I am writing to memory buffer register from memory buffer register it will basically go to a place where it will be seen.

Now, where I have to save it in the memory that will be actually the address to save the P C contain, that is actually a stack address where we save the value of program counter, sometime program register, before the interrupt service would we started. So, you actually in the; what you do? So, you can see that first stage I write the value of the program counter the memory buffer register, second stage I write in the memory address register to save the value of the program counter.

So, now basically already in the first stage we have saved the value of your program counter in the memory buffer register. Now I give the address where you have to save the value of the program counter in the memory, but I thing first stage P C is now free we have already save the value of PC in memory buffer register and after that you are going give the address in the memory address register where the value of M V R has to be save indirect to the P C will be save, but in the first stage only P C S saved.

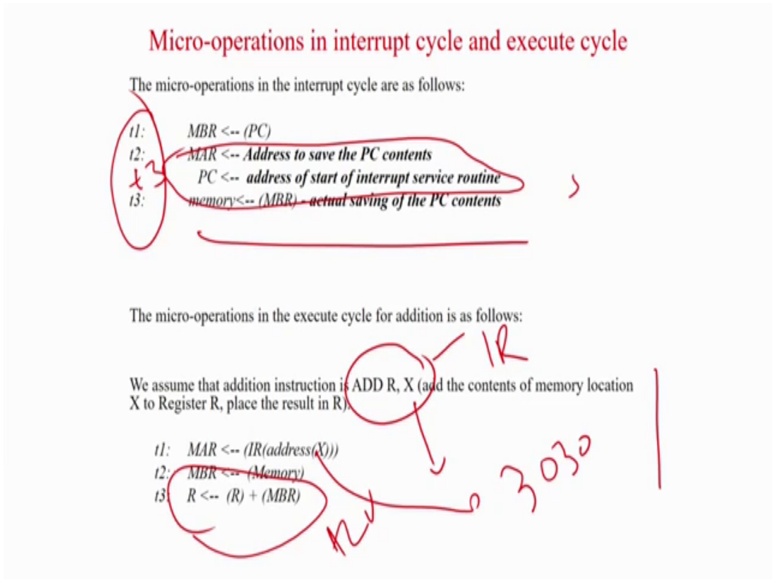
So, in time 2 you can put P C to the starting address of the interrupt service routine you would not require a t 3 over here t 1 you save the P C in memory buffer register step 2 you write the address in the memory buffer register sorry memory address register, where you want to store the value of P C at the same time you need you increment the value of P C, because P C is free.

So, in this case you jump to the interest of service routine and then in the time step 3 in memory buffer register will write the value to the memory, whose address was give in the memory buffer reg memory address register. So, in step 3 actually you are physically saving the value of the program counter. So, instead of 3 you 4 you can save in it 3 time units.

Basically we have merged because of clock grouping which would merge the memory address register write it and the program counter increment, because already we have save the value of program counter. Another simple this was all about basically load, store, fetch and some type of interrupt type of part of the macro routine, but now let us take a very simple instruction like ADD R, X basically that is you want to ADD the value of R some memory location and memory location is X content of X you want to ADD with r and save with it register R.

So, basically what are the micro instruction involve for this fetch already we have seen may be it has been fetch etcetera then what is the case? So, basically this is already in the instruction register, because we assume that it has been fetch.

(Refer Slide Time: 46:30)



## Micro-operations in interrupt cycle and execute cycle

The micro-operations in the interrupt cycle are as follows:

```
t1:    MBR <-- (PC)
t2:    MAR <-- Address to save the PC contents
        PC <--  address of start of interrupt service routine
t3:    memory<-- (MBR) - actual saving of the PC contents
```

The micro-operations in the execute cycle for addition is as follows:

We assume that addition instruction is ADD R, X (add the contents of memory location X to Register R, place the result in R).

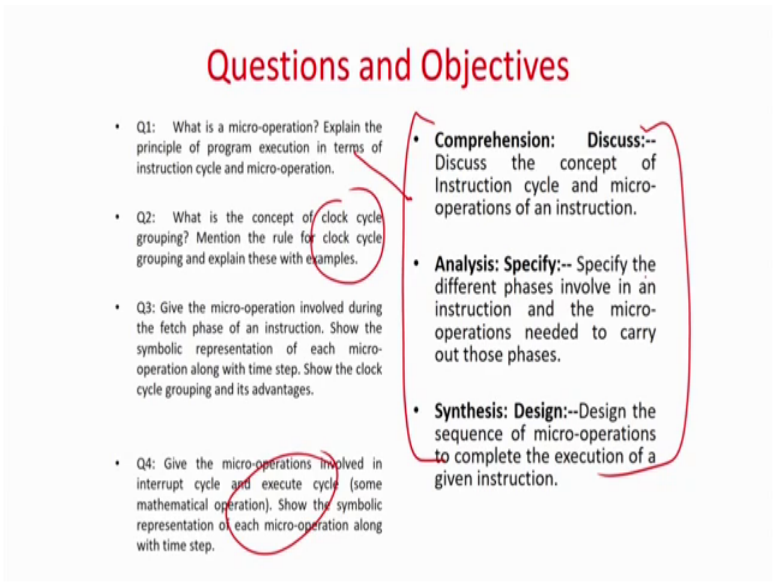
```
t1:   MAR <-- (IR(address(X)))
t2:   MBR <-- (Memory)
t3:   R <--  (R) + (MBR)
```

So, in the from the instruction register you have to get the value of X address X means the value of X that is 30 30 in the example we are looking at it. So, memory address register will be fetch with the value of address of X.

Now, the memory buffer register after some amount of time we load the value that is of 30 30 assuming that is X the value in the memory location X, which we are calling as 30 30 in example that value in that location will load to the memory buffer register in time 2 and with time 3 basically, which is the logic operation the accumulator sorry the arithmetic and logic unit will be involve it will ADD the value of memory buffer register with the register R and it will be save it in R. So, this ADD R X will be done in 3 micro instructions.

And of course, you can find out the lot of interdependence, because neither you can paralyze. Means you cannot put memory this first instruction with second and second with 3. So, so they are all interdependent instruction. So, three instructions are require for that.

(Refer Slide Time: 47:27)



## Questions and Objectives

- Q1: What is a micro-operation? Explain the principle of program execution in terms of instruction cycle and micro-operation.
- Q2: What is the concept of clock cycle grouping? Mention the rule for clock cycle grouping and explain these with examples.
- Q3: Give the micro-operation involved during the fetch phase of an instruction. Show the symbolic representation of each micro-operation along with time step. Show the clock cycle grouping and its advantages.
- Q4: Give the micro-operations involved in interrupt cycle and execute cycle (some mathematical operation). Show the symbolic representation of each micro-operation along with time step.

- **Comprehension: Discuss:--** Discuss the concept of Instruction cycle and micro-operations of an instruction.
- **Analysis: Specify:--** Specify the different phases involve in an instruction and the micro-operations needed to carry out those phases.
- **Synthesis: Design:--**Design the sequence of micro-operations to complete the execution of a given instruction.

So, basically in this unit what we have seen in this unit basically we have got an idea? That there are macro instructions and each macro instructions must be implemented in terms of some lower level instructions, which are we calling as micro instructions. And how micro instructions basically together are responsible for a implementing a macro instruction and each macro instructions are so finite or so, atomic we need not break it down into farther level.

Basically each micro instruction involve kind of some kind of control signals which will actually make the (Refer Time: 47:59), that we are going to see in the future in is to be coming, but macro instructions micro instruction module at the unit level or the atomic level together make the macro instruction and a code executes. For a each micro instruction we will see that there are some kind of signals, which are generated, we will see in the future units how to generate the signals and how basically they involve in real hardware to execute, like if I say ADD, how the corresponding micro instruction? What are the signal it generate to exactly make it ADD the 2 operates by the CPU ok.

So, before we end basically let us look at the some of the questions, like first question is what are the micro instruction explain the principle of program instruction in terms of micro instructions and micro operation. And second instruct if you look at it this and these are the objective, like discuses the concept of instruction cycle and micro instruction operation. So, directly if you are able to answer this question this instruction this object is satisfy. What is the concept of clock grouping like specify the different phases of involve in micro instruction to carry out those phases design the sequence of micro instruction to compete instruction execution.

So, if you are able to specify and design of course, you will be able to optimize it in terms of clock grouping give the micro instructions involve in fetch instructions, give the micro instruction involve in the interrupt cycle and execute cycle. So, this questions basically we have this question for assignment, when will be solving this questions basically you are easily going to satisfy this objectives. Like the first one is basic concept of micro instructions, third and fourth questions basically tells you about different phases of the instructions and basically what are the micro involved in it.

So, basically these questions actually satisfy these objectives. So, after solving I mean after discussing this unit you should be able to solve this questions and hence satisfy the objective. And whenever you are going to try to optimize this sequence then the concept of clock grouping is coming.

So, basically with this we come to end of the first unit, from the second unit onwards basically we will be more specifically looking into the control signals, which are generated by each of the micro instructions, which exactly or accurately result in the flow of the code.

Thank you.