

(Refer Slide Time: 16:36)

## Module Learning Strategy

- UNIT-V  
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-VI  
William 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  
William 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 skipping this slide for a few minutes.

(Refer Slide Time: 16:38)

## Module Learning Strategy

- UNIT-VIII  
William 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  
William 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 our computer organization architecture you can go through this in this way you go through the module on CPU design.

So, basically this gives an overview 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 a module and units

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 as macro instructions because I want to differentiate them from the micro instructions.

So, in this case we will see a instruction cycle, which are always saying fetch, decode, execute, store and sometimes there may be an interrupt and what are the micro instructions involved for each 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 as already told you micro instructions basically are the atomic instructions, which execute in a single clock unit and which totally comprise a generic instruction. So, this is the first module that is instruction cycle and micro operations.

(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. That is as I told you if you have an indirect machine instruction, that is ADD indirect A 3030; that means, the location of the variable, which has to be added with A is not available at 3030, at memory location 3030 you have to we will find another address and you have to go

to that address there we will get the actual value which has to be added with A, so the indirect mode of addressing.

So, and 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 mode. So, it says that all the instructions are of different complexity and they will take multiple clocks to execute. So, basically what happens? 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 operations 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, 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 a data from CPU register to another CPU perform arithmetic operation and store the result in a CPU register. So, you can see that there are 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 predefined in such kind of a thing you pre load from memory location, store to a memory location, transfer data in between different registers of the CPU and how do you perform the arithmetic and logic operation and we first store it in the register and then we can write it in the 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 have to take data from the memory to a register, and 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)

The slide is titled "Unit Summary" in red. It contains five bullet points. The first bullet point is circled in red and underlined. The second bullet point is underlined. The third bullet point is circled in red. The fourth bullet point is underlined. The fifth bullet point is circled in red. The slide is a screenshot of a presentation, with a Windows taskbar visible at the bottom.

### 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 because nowadays processors are more complicated you can have different type I mean clock sequence means some of the micro instructions will take 2 clocks some will take 1 clock. So, all these complications are there, but for the time being we are taking a very simple controller design because it is the UG first level course on architecture.

So, we will assume that micro operations are such operation which can be run in 1 clock cycle. More or less even in very sophisticated architectures 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 will interfere with in another we will see that basically what 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 at the same time your CPU is doing some computation and writing register to another register, register2 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.

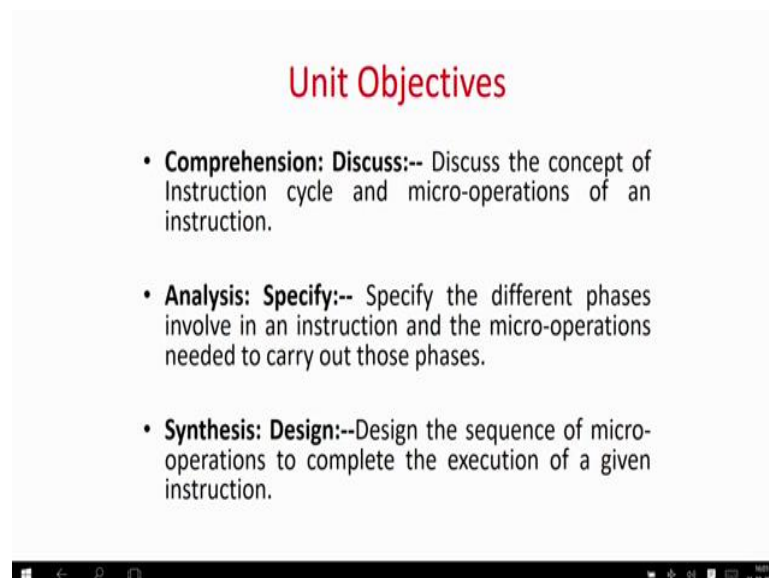
So, you can actually mux these 2 instruction at 1 time unit; that means, they are all micro instruction they will take one, but you need not sequence them because they are non-interfering. So, if you give directly separate time units to all this then you require more number of micro

instructions to operate a 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 instructions take single unit of time that is fine, but depending on the instructions if they are non-dependent instructions we can put them in one time go 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-dependent one another you put in the same time unit. So, there is optimization in time.

So, we actually call it as a 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 required to execute a macro instruction, which is actually called 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 objective 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 phases like, as I told you that if I say that LOAD X or LOAD A, 3030 there are certain steps like for example, the instruction has to be taken from the memory to the instruction register by a memory buffer register, then 3030 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 3030 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 required to execute a macro instruction insteps of micro instructions and finally, it's a design objective given any instruction set you will be able to design the micro instructions require to 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

*Handwritten notes:* ADD A, 3030

*Diagram:* A hand-drawn diagram showing a vertical rectangle with a horizontal line through its middle. A curved arrow starts from the top right, goes around the right side, and points into the bottom half of the rectangle. Another curved arrow starts from the bottom right, goes around the bottom side, and points into the top half of the rectangle, indicating a data flow between the two halves.

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 write the address specifically that I call 3030. So, in this case as I told you it's a macro instruction. So, ADD A is the accumulator from 3030 memory location.

So, you can assume that if it is a direct instruction time take will be certain if 3030 is not a memory location, but if it immediate data then it will be extremely fast and if 3030 ADD this ADD is an indirect instruction; that means, first you have to go the memory location 3030, there is another instruction over here you have to go to another part of the memory and then you will get the data.

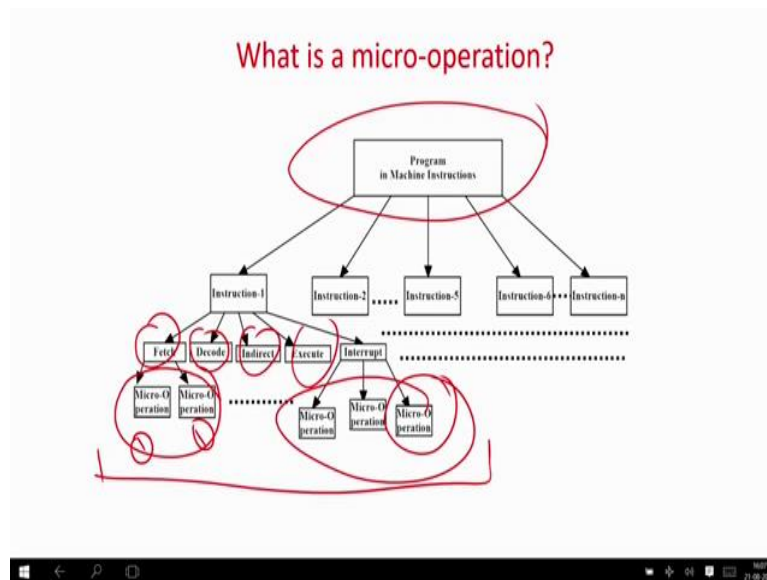
So, memory instructions are very complex depending on the addressing mode and what it operates on etcetera. So, basically we divide it into granular level which is called the micro instruction and each micro instructions can execute in a single time unit taking multiple micro instructions you go for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

That is what in is told in the second point, that basically it says that each machine instruction is 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 reads and writes are done in a single clock pulse. So, we mean we actually mean design our clock in such a fashion or we limit our speed in such a fashion that most of the micro instructions are executed 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 processor will come down. If you have very fast memory, you have very fast interfaces, multiple buses then many of these 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 that the processor is 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 instructions and make a macro instruction or whenever you are designing 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 instruction as 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 this a very nice figure, which actually shows you, gives a description that what is a micro operation. In 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 sometimes 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 are 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, then basically program counter will be incremented. 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 opcode from the instruction and generate certain signals, in this module we will be mainly looking at how signals are generated for an instruction. Like for example, if the opcode is 001 may be 001 stands for load. So, the hardware will generate the signals corresponding to the load after reading this opcode 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 3030. 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 fetched.

So, lot of small integrated atomic details are required when an instruction is fetched 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, they are now actually the micro instructions and this one will take 1 unit of time this one will take 1 unit of time and so forth.

So, again the number of micro instructions for a given instruction depends on the instruction type. 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's 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.

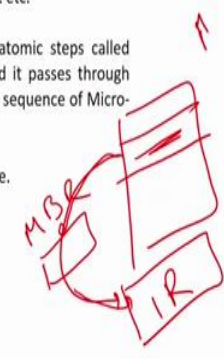
For example, all instructions involve the "Fetch" sub-cycle.

$$t_1: PC \rightarrow MAR$$

$$t_2: MEMORY \rightarrow MBR$$

$$t_3: PC+1 \rightarrow PC$$

$$t_4: MBR \rightarrow IR$$



The diagram illustrates the Fetch sub-cycle with four time steps:  $t_1: PC \rightarrow MAR$ ,  $t_2: MEMORY \rightarrow MBR$ ,  $t_3: PC+1 \rightarrow PC$ , and  $t_4: MBR \rightarrow IR$ . A hand-drawn schematic shows the flow of data: from the Program Counter (PC) to the Memory Address Register (MAR), from Memory to the Memory Buffer Register (MBR), from the MBR to the Instruction Register (IR), and an update to the PC (PC+1).

Now, we will take different examples of micro operations and see how it requires. So, for example, we are going to take a "Fetch" sub cycle. So, we are going to take fetch, decode, 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, this has to be fetched 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 PC.

Now the program counter value is already telling that this is the address of the memory where the instruction is there it will send 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 to memory buffer register, memory buffer register is saved in the instruction register and you have to increment the PC.

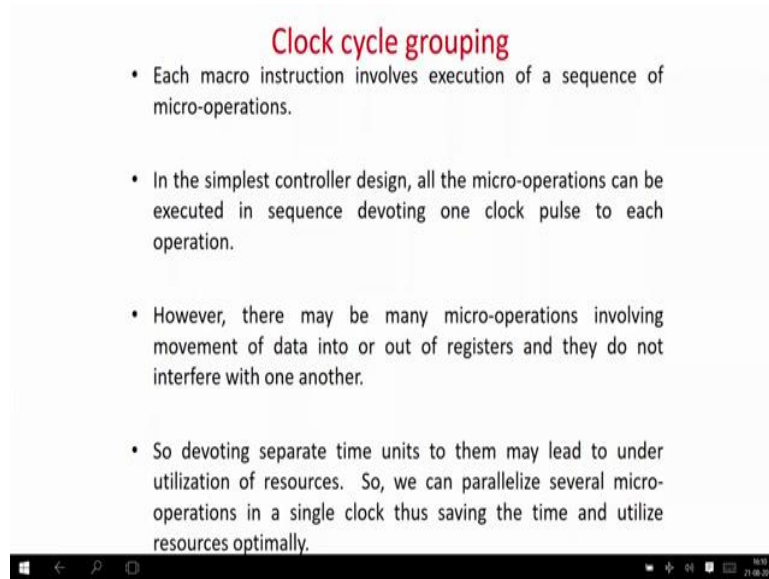
So, these are some of the micro stage of operations which are require to do it. So, what are the first time units what will happen? The program counter will load the memory address register. That is program counter PC value is there it is telling that that is your memory address register. The program counter is telling that this is the location from where I need the instruction. Then this part is done then what we will do now this 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 next instruction has to be fetched in the next cycle and then the memory buffer register will write to the instruction register.

So, you can see that I am having 4 basically micro instructions and they are 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 instruction register, they all take some single unit of clock and these 4 steps or micro instructions generated correspond to the mac correspond to the fetch sub cycle of the macro instruction, which may be something like ADD A, 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 end memory indirect.

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

(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, each macro instruction involves the sequence of micro instructions, if you want to keep it very simple just you take a flat implementation, means whatever is the macro instruction then it is there what are the stages you have fetch, decode, indirect, execute, interrupt divide it into the micro instructions and take that much amount of time?

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