

**Design of Mechatronic Systems**  
**Professor Prasanna S. Gandhi**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology, Mumbai**  
**Lecture 16**  
**Timing and Control Unit: Primitive Microprocessor**

(Refer Slide Time: 02:07)

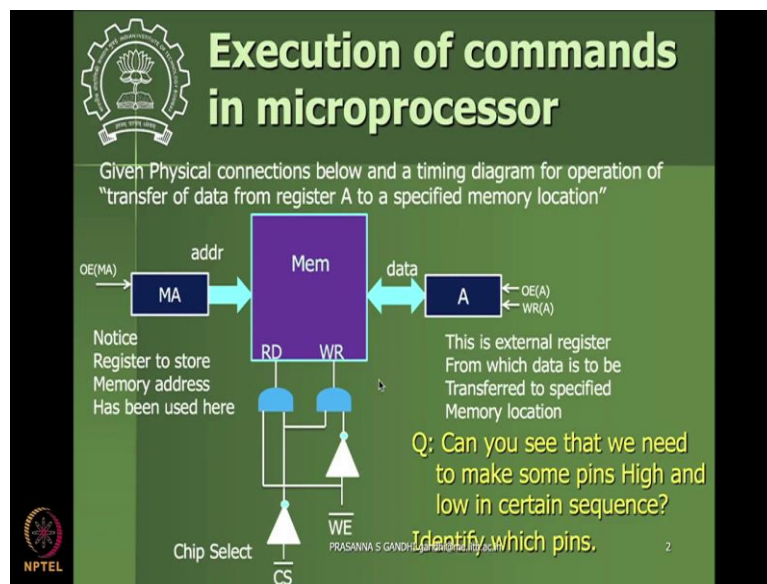


We saw in the last class different, different building blocks of microprocessors. The combination logic circuits, the sequential logic circuits and we saw also memory and its addressing. So, in this part now, we will start putting together everything to construct a microprocessor. We will start off with some small primitive microprocessor construction, and then we will look at how these instructions are executed inside a microprocessor.

So, we have, in the last lecture, we have seen the operation of transfer of data from external register to the memory. What are the sequence of things that have to be happening for different, different pins, that we will carry out that operation. Now, how this operation can be done by using some kind of a timing and control circuit, we will introduce that here, and then proceed for introducing more and more kind of a functionality to this microprocessor.

And get a glimpse of what how things are happening inside. So, here the idea is to kind of introduce you to the basic fundamental understanding here for the timing and control circuits and the basics of microprocessor instruction sets and things like that, architecture instruction sets, everything and then based on this, you will be able to understand the modern microprocessors and microcontroller data sheets in much, much kind of a better fashion. That is how we have introduced these lectures here.

(Refer Slide Time: 02:10)

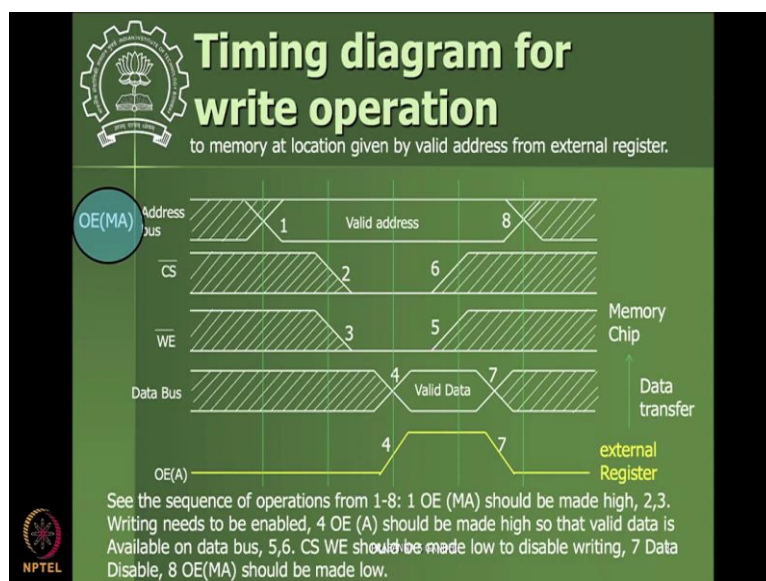


So, we will proceed with, first with the example we saw in the last class. So, you just recall this that, we had this sequence of operation to be performed for getting the data from the external register A now to be transferred to some memory location, where that memory location is indicated by this address, which is on the address bus here. And that address is stored in this register called MA.

So, the sequence of operations would go something like, you need to have this output enable for MA going high, which will take the data from this register and put it on the address, address bus. So, when the data, the address bus has some kind of data, that particular location in the memory will be addressed now.

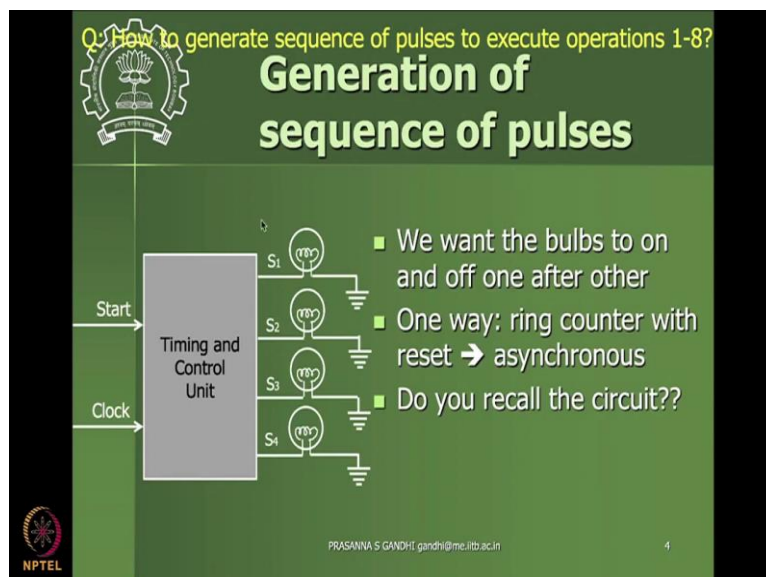
And then you want to make this, enable this chip select and this, so that this this particular write operation is enabled and then we have this valid data to be there on the data bus and then this operation, the writing of this data here will be done. So, how do you get the data on the data bus? To same way like now you have made output enable pin high and then the data from the register A will be now available on the data bus. So, this will be captured in the timing diagram as we have seen in the last class, in this kind of a fashion.

(Refer Slide Time: 03:37)



So, you see at here this is output enable for memory address and then there is output enable for data register. So, so, you know that now these are the like numbers, some kind of a sequence that is there in which the operation needs to happen. How would we get this operation done now? By some kind of a circuit or logic circuit. So, we make use of some sequential logic for this.

(Refer Slide Time: 04:10)



So, let us understand first, suppose, you have some different problem given here. So, how do we generate the sequence of of these pulses which will light this bulb first, then this will light

this bulb and then light this bulb and this bulb like that, it should light in a sequence like S1, S2, S3 and S4.

So, so they need to be on for a while and one, once S1 goes off, S2 should get lit up. So that is what we want to do. So, how do we do this? So, if you just scratch your memory a little bit and see our sequential logic circuits, which we have seen there is one example there which will help you, help us here.

(Refer Slide Time: 05:00)

Recall

## Four Bit Ring Counter

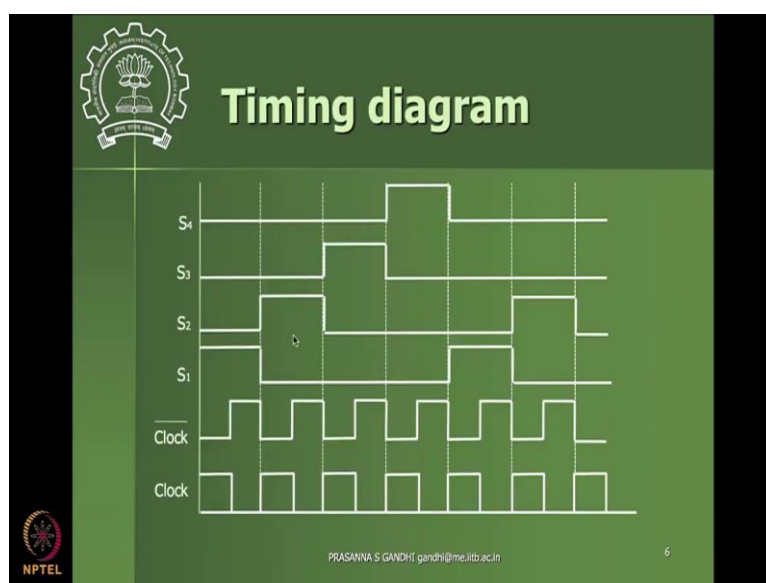
- Simplest counter: timing diagram?

- How many states are getting represented? 1000, 0100, 0010, 0001
- Q: what if we start at 0000??
- Reset to 1000 needed to start
- How to reset to 1000???

NPTEL

So, if you recall, thing and then proceed. So yeah, it is a ring counter. So, so, this in the ring counter you introduce this sequence like 1000, so, the state is 1 here which will enter so all others are 0 and then in the next clock cycle that 1 will come here and then in the next clock cycle it will proceed here and if you have this feedback which is coming like, so, this is continuously this sequence will go on in a loop 1000, 0100, 0010 and 0001 will happen for S1, S2, S3, S4 in a loop and that is what we want for these lights to be lit up in a sequence.

(Refer Slide Time: 05:46)



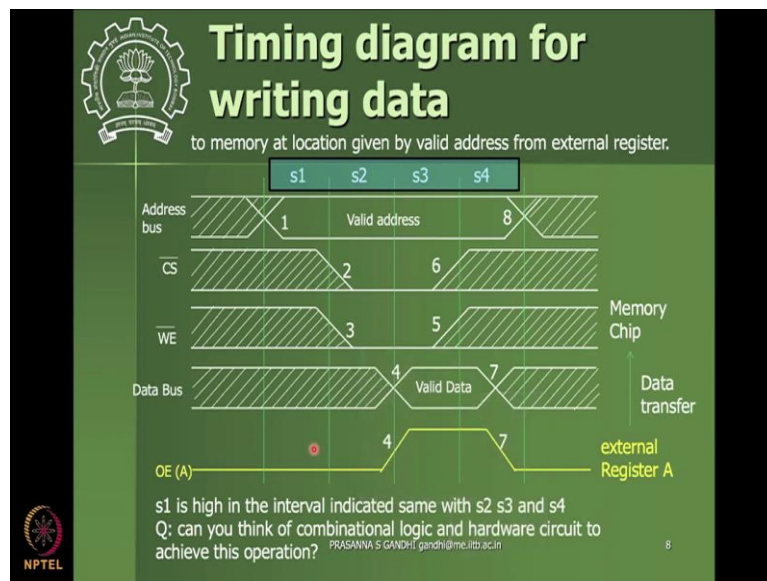
The timing diagram for this such kind of a case will look something of this sort. S1 is lit up first, then S2 is lit up after that, once S2 goes off, then S3 is lit up and S3 goes off S4 is lit up, then again S4 goes off, S1 is lit up again. And then this this continues in the, in the clock.

(Refer Slide Time: 06:10)

The slide, titled "Operation of Data Transfer to Memory", contains a question and a reference to the timing diagram. The question is: "Q: How to use this sequence now to realize the operation of data transfer from external register to memory." The answer is: "See carefully again the timing diagram of operations we want with reference to timing diagram of the sequence we produced". The slide is set against a green background with the NPTEL logo and the text "PRASANNA S GANDHI gandhi@me.iitb.ac.in" and the number "7" at the bottom.

So, now, if we want to you make use of this sequence to realise this data transfer operation for external register to the memory, that is what we are talking about. So, we see the, see the timing diagram again carefully for the operation that we want to do. And then now we put this S1, S2, S3, S4 in some, some kind of a time sequence here.

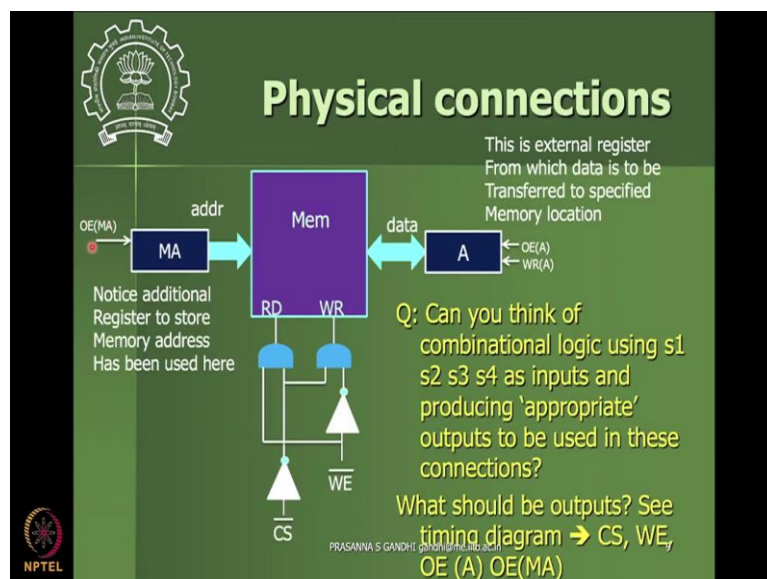
(Refer Slide Time: 06:26)



S1, S2, S3, S4 so they are these four kind of a timing zones which we have mapped here. And in that, like, we know S1 will go high first, then S2 will go high, then S3 will go high, and then S4 will go high. This is what is going to happen in time for us now.

So, now, how do we use, make use of this S1, S2, S3, S4 going high in a sequence to execute like what we want here. So, remember our inputs are these WE, CS and address bus and output enable. So, these are the inputs which we need to provide some kind of proper signals to these inputs, so that our operation happens in a way that is governed by this timing diagram. So, let me get the pointer red.

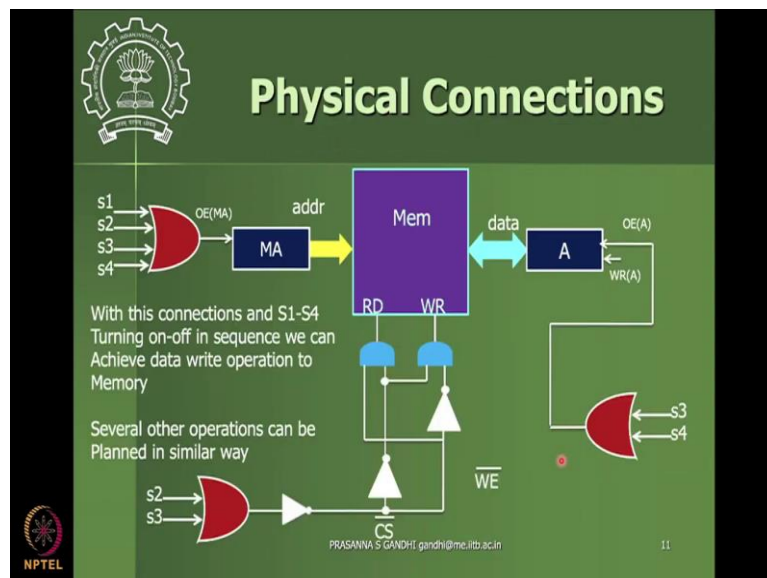
(Refer Slide Time: 07:19)



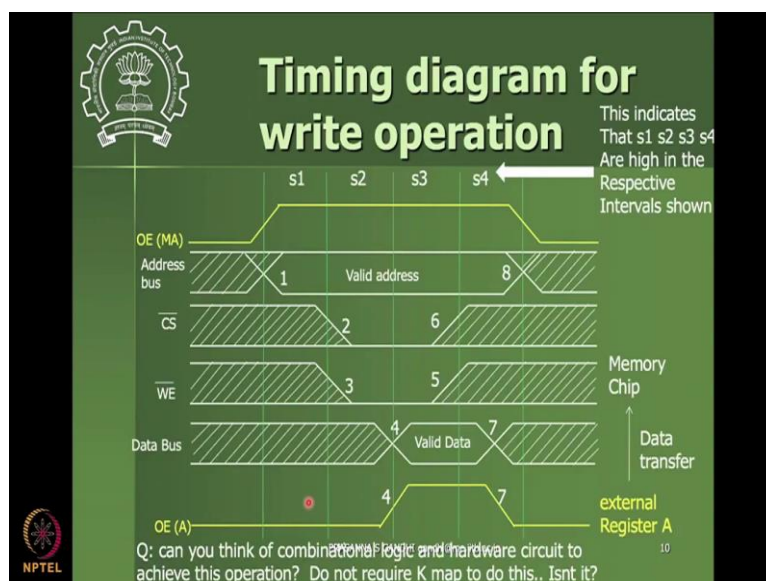
So, now, think about this, what we need to do, so, this combination logic we can use on S1, S2, S3, S4 to produce this appropriate output, which can be connected to this OE pin here or OE pin here, or CS pin and WE pin here, that is what we want to do. So, how do you think about, so, say if you go back and see your S1, S2, S3, S4 all four cycles, this address bus has valid data which is coming from making OE for the memory address to be high.

So, we odd these inputs S1, S2, S3, S4, they are odd with each other to produce some output that output will be connected to this output enable valid address, then all the sequence all S1, S2, S3, S4 durations, your memory address will be holding, the address bus will be holding valid address, that is how when we can think about, like that you think about for other pins what is a combination of S1, S2, S3, S4 to be used and then proceed.

(Refer Slide Time: 08:33)



(Refer Slide Time: 08:48)

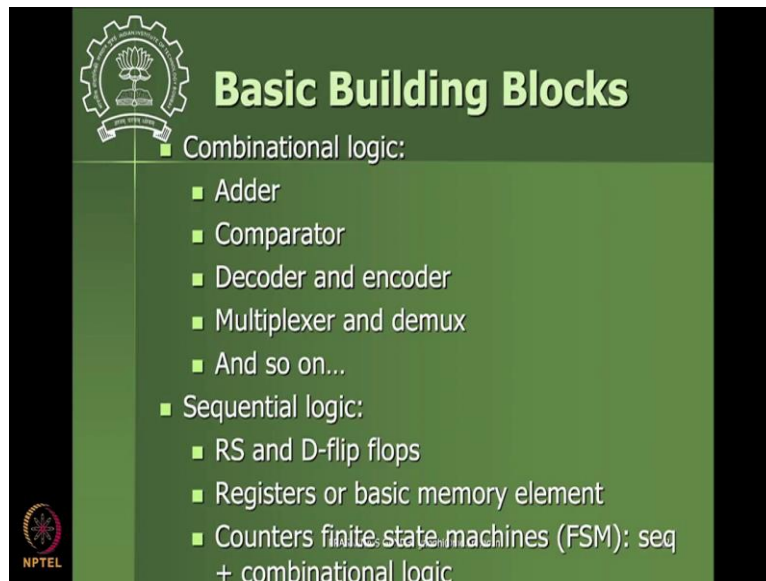


So, you can see now that the circuit is going to look something of this sort. So, you have S2, S3 connected with OR gate here, S3, S4 connected with OR gate here and then you see that in S2, S3 sequence, S2, S3 sequence you will have this, this will be S2 and S3 sequence like this will be happening S2, S3 are here. So, this part will be happening. So, this will be done to the chip and S3, S4 sequence you will have this valid data here.

So, one can see that by using a simple kind of a like that the sequence of pulses going high one after the other and some kind of a combination of these, those pulses in the, in the, in the combinational logic way, we can achieve the task that we wanted to do. So, this is one of the tasks that we had. Now the question is like many such tasks can be performed now, one can see that easily.



(Refer Slide Time: 09:46)



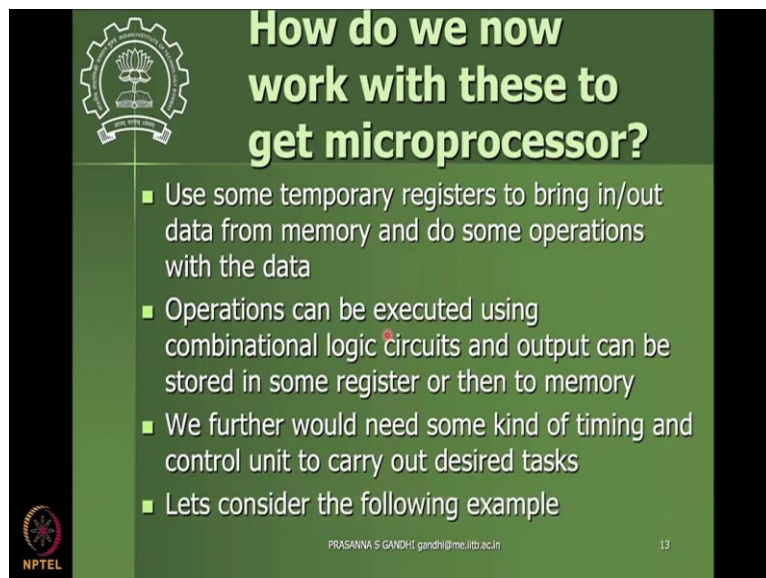
**Basic Building Blocks**

- Combinational logic:
  - Adder
  - Comparator
  - Decoder and encoder
  - Multiplexer and demux
  - And so on...
- Sequential logic:
  - RS and D-flip flops
  - Registers or basic memory element
  - Counters finite state machines (FSM): seq + combinational logic

NPTEL

So, we can put together many different, different kinds of elements, as we whatever we have seen in combinational logic elements, adder, comparator or there is many different kinds of possibilities are there and many sequential elements also can be there and then we can put together all these to build now, our microprocessor unit.

(Refer Slide Time: 10:10)



**How do we now work with these to get microprocessor?**

- Use some temporary registers to bring in/out data from memory and do some operations with the data
- Operations can be executed using combinational logic circuits and output can be stored in some register or then to memory
- We further would need some kind of timing and control unit to carry out desired tasks
- Lets consider the following example

PRASANNA S GANDHI gandhi@mc.itb.ac.in 13

NPTEL

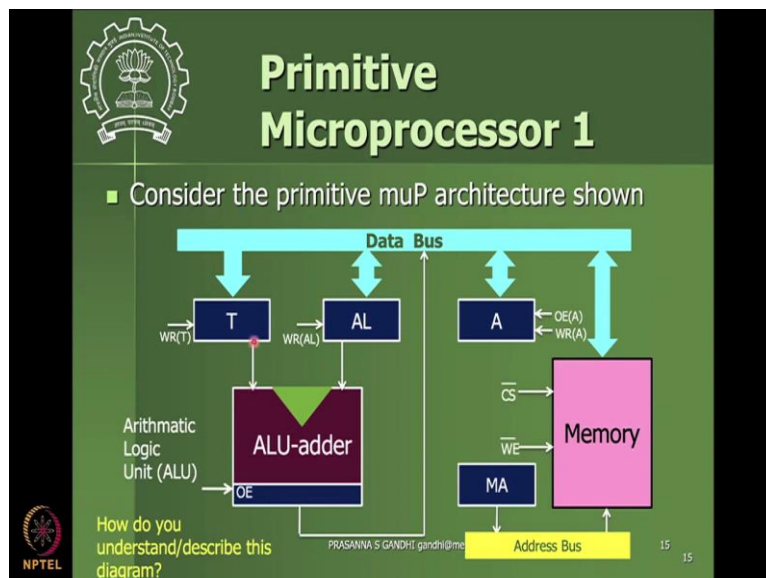
**Primitive Microprocessor 1**

- Now we will put together memory registers and more combinational logic to construct primitive microprocessor (purpose is to understand underlying philosophy)

NPTEL PRASANNA S GANDHI gandhi@me.iitb.ac.in 14

So, how do we do this, this work with like we will see with the example. So, so, we want to have a microprocessor which will consist of some temporary registers and some memory, then you can put in some combinational logic circuits to do some, some additional, addition, subtraction operation logic, output operations, all those kinds of things can be put together and then we will need a timing and control circuitry the way we had these four pulses coming up here, similar kind of a timing circuit to execute this desired task.

(Refer Slide Time: 10:57)



So, we will see one with some example of actually put together this. So, this is a primitive microprocessor architecture that is shown here. So, see this microprocessor architecture or microcontroller architecture where typically given in such kind of a diagrams and these

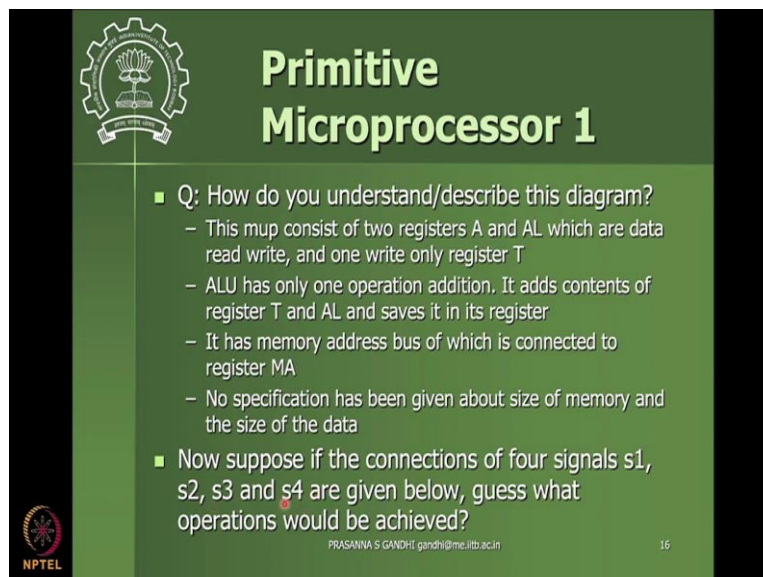
different, different registers there is a data bus then the registers will have some functionality that is given in the datasheet of the microcontroller.

This data bus, this this kind of thing tells you that the data from this register can be written and read also, but in the register T the data is can be only given as an input, one cannot read the T register like that, you can read this kind of a diagram. And you have a memory, small memory available in the microcontroller, this memory address register holds some kind of a address and then one can address this memory, something is there.

Then you have some, this is only adder here. So, this adder is taking a value, value in the register T and value in the register AL and this is adding together and this is values available in this register and when you this, put this output enable pin high for this register, then that added value will be available on the data bus, things like that. So, you can have only this one way kind of direction coming in, the data is coming on this data bus, you cannot write in this register otherwise.

Like that some kind of understanding can be built for by use, by reading such a kind of diagrams. Similarly, you will find these kind of diagrams in many modern kind of a microcontroller data sheets and one you should kind of figure out, have ability to figure out what these diagrams really really mean. So, that that can be done. Now, with this understanding that we have done so far and by reading the datasheet of the, of that microcontroller.

(Refer Slide Time: 12:58)



**Primitive Microprocessor 1**

- Q: How do you understand/describe this diagram?
  - This mup consist of two registers A and AL which are data read write, and one write only register T
  - ALU has only one operation addition. It adds contents of register T and AL and saves it in its register
  - It has memory address bus of which is connected to register MA
  - No specification has been given about size of memory and the size of the data
- Now suppose if the connections of four signals s1, s2, s3 and s4 are given below, guess what operations would be achieved?

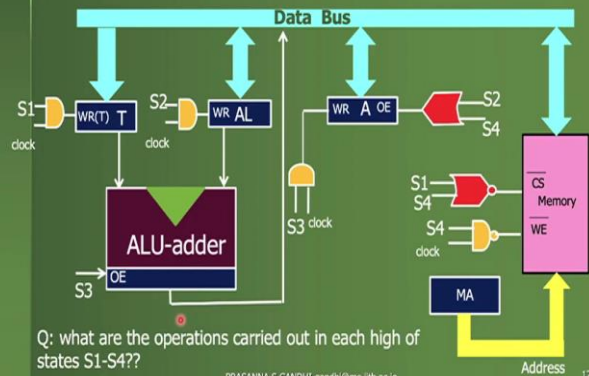
NPTEL

PRASANNA S GANDHI | gandhi@me.iitb.ac.in

16



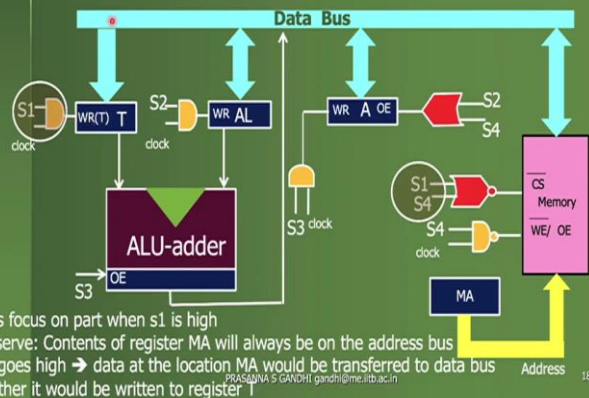
# Use states S1-S4 from previous example



Q: what are the operations carried out in each high of states S1-S4??



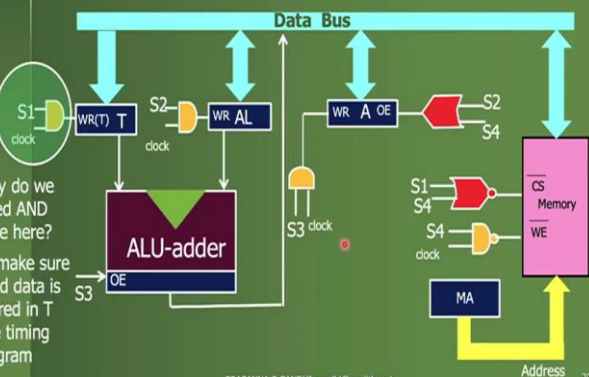
# Use states S1-S4 from previous example



Lets focus on part when s1 is high  
 Observe: Contents of register MA will always be on the address bus  
 S1 goes high → data at the location MA would be transferred to data bus  
 Further it would be written to register T



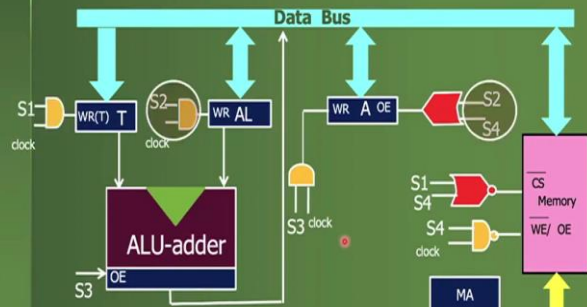
# Use states S1-S4 from previous example



Why do we Need AND Gate here?  
 To make sure Valid data is Stored in T  
 See timing diagram



## Use states S1-S4 from previous example



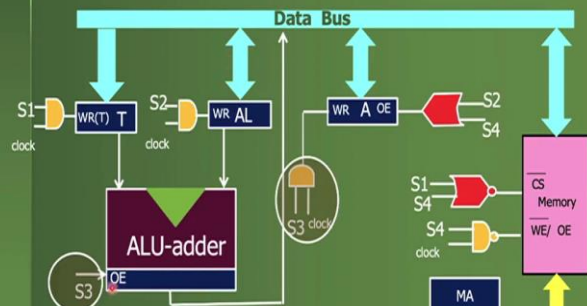
Lets now focus on part when s2 is high  
S2 goes high → data at the location A would be transferred to data bus  
Further it would be written to register AL



21



## Use states S1-S4 from previous example



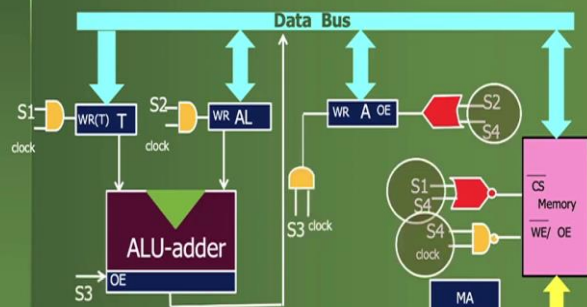
Lets now focus on part when s3 is high  
S3 goes high → sum at the location ALU would be transferred to data bus  
Further it would be written to register A



22



## Use states S1-S4 from previous example



Lets now focus on part when s4 is high  
S4 goes high → sum at the location A would be transferred to data bus  
Further it would be written to register in the memory location pointed by address in MA



23

So, let us proceed with the with some some operations here now. So, these are our understanding of this microprocessor. And suppose we connect now some signals like S1, S2, S3, S4 in this fashion. And I tell you now, can you just guess for such a kind of a diagram, what is the operation that will be happening.

So, read this carefully like how this S1, S2, S3, S4 are connected to these and and they are going in the same sequence high. So, S1 will go high first, then S2, then S3, and then S4. So, you can pause here for a while and figure out like little bit at least to think about what will happen in S1 sequence for example, what will happen in S2 when S2 goes high. So, like that you can think and then you can proceed. We will go one by one in the, in the understanding.

So, let us say S1 goes high. So, when S1 goes high the contents of this register MA, which are always there on the so, see this has no output enable nothing. So, this is always connected to the some memory. So, contents of memory register, this memory address holding register will be available on the address bus, there that is there for always.

Then when S1, S2 goes high, this S1 goes high in this thing then this chip select will happen here and then this memory will be selected. Now, this memory is getting selected and then the data from this will be available on the, so the data from these will be available on the, on the data bus. So, data, which data is available? Data which is at a location which is governed by this address here that will, that data will be available on the, on the data, data bus.

Then what else is happening, that data is this, this is write here. So, when S1 goes high that data is also getting written in the, in the register T. So, data will be fetched from the memory location and it is given to the register T. So, there is a clock that is kept here to make sure that this clock signal is given along with S1.

So, S1 you remember it is half, this S1 it goes high for a full clock cycle and this when it is ANDed with the clock this output will be only half cycle up. This is done to kind of make sure the there is no clash of the data or the the write operation is completed well before like the data gets invalid on the data bus, so that there is no clash with the data. So, some some nitty gritty of that sort are are done, which you do not need to really bother about.

So now let us see what is happening in S2. So, that is what we saw, why do we need AND gate here and then yeah, so, now when S2 goes high, you can see S2 is here and S2 is here, what is happening here is again something is getting written here. What is getting written?

So, see this output enabled for this A register, so that is going high, this OR gate, that is going high, that means data from register A will be available on the data bus now.

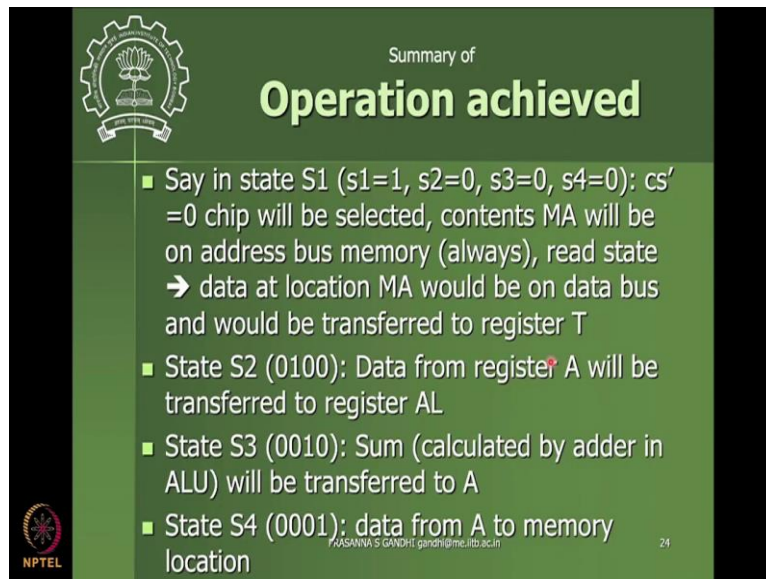
So, you see the data bus, same data bus is sharing in all different registers and memory and so on and so forth. So, so, the data from A will be available on the data bus and it is written in which register now, it is written in the AL register, that is happening in the, in the second part of the sequences 2.

So, you remember the first part, the data from the memory was available here and was transferred here. Now, in the second part like the data from the register A is going to get transferred here. Then what will happen in 3, S3 goes high, you see that these data from this this is now added together, this is the data which is data in the register T plus AL will be added together and that data will be available now on the data bus.

Now, which register that data is written? That data is written in register A again. So, whatever was previous content, it was overwritten and like that addition will be available here. Now, that is what is done here. And then what is happening S4 sequence? You take this data, which is from, from S4 register now, it is put again on the data bus and in S4 it is now write enabled and that data is is written to this memory.

So, so, these are like typically you can you can see that the sequence of operations is going to happen, there are some nitty gritty things that we need to figure out and things like that, but this is, this is giving you some kind of a sense of, oh this is how like something is connected together the same operations can be performed.

(Refer Slide Time: 18:49)



The slide features a green background with a white gear icon in the top left corner. The title 'Summary of Operation achieved' is centered at the top in white. Below the title, there is a list of four states and their corresponding operations, each preceded by a small green square bullet point. The text is white. In the bottom left corner, there is a small circular logo with the text 'NPTEL' below it. In the bottom right corner, there is a small number '24'.

Summary of  
**Operation achieved**

- Say in state S1 ( $s_1=1, s_2=0, s_3=0, s_4=0$ ):  $cs' = 0$  chip will be selected, contents MA will be on address bus memory (always), read state → data at location MA would be on data bus and would be transferred to register T
- State S2 (0100): Data from register A will be transferred to register AL
- State S3 (0010): Sum (calculated by adder in ALU) will be transferred to A
- State S4 (0001): data from A to memory location

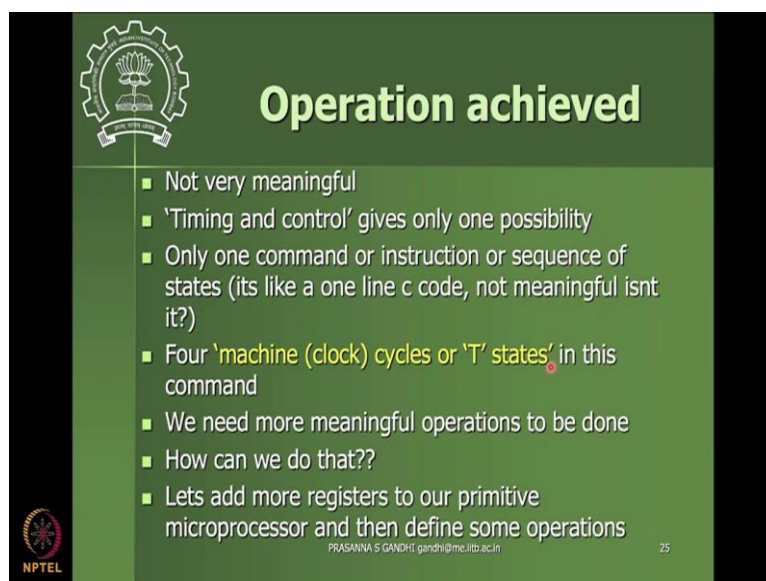
NPTEL 24

And we know that these operations are has not great meaning here. So, one can have the summary of these operations which are achieved in different different state, but you can see that, the some kind of a successive addition is performed here on some data, but it is not a greatly meaningful kind of thing.

And also, suppose we do have this microprocessor, this will be only doing this, there is no other command that is to be, we cannot write any command that saying that okay, oh, this is my, is my programme, and I want to have like something done like that or I have to want to have some instruction set which will, which will control this kind of operation, nothing like that is there right now.



(Refer Slide Time: 19:30)



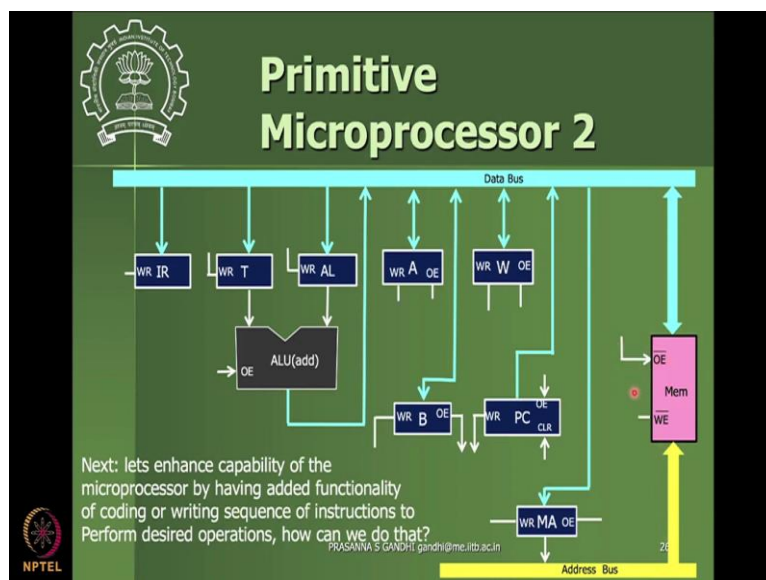
### Operation achieved

- Not very meaningful
- 'Timing and control' gives only one possibility
- Only one command or instruction or sequence of states (its like a one line c code, not meaningful isnt it?)
- Four 'machine (clock) cycles or 'T' states' in this command
- We need more meaningful operations to be done
- How can we do that??
- Lets add more registers to our primitive microprocessor and then define some operations

NPTEL  
PRASANNA S GANDHI, gandhi@mc.iitb.ac.in  
25

So, we want now to have such a kind of a facility, we build some more stuff in the, in the thing. So, so, we add some more registers to our primitive microprocessor and look at some more operations with this microprocessor 2. So, this is a microprocessor which is having now like a lot more number of registers but still that additional, addition ALU is there.

(Refer Slide Time: 19:42)



So, there is, there can be many different logic elements that can be used in ALU, to arithmetic logic unit to kind of perform many many different operations, one can now guess that it is very easy to see that we can have a lot of things put together in in our ALU, you can have some some counter or other kind of things also put as a part of this microprocessor unit.

So, now, what we, we need to introduce here is is we have like notice here we have introduced some some more register called PC register, like a programme counter register. So, this will keep track of which instruction you are, you are running at present. So, you have some sequence of instructions that are typically written as a program, that is what you write when you write a programme, you write some first command line, second command line like that you write this and one after the other they get executed.

So, so, this programme counter register is a special register which will keep keep track of that count, that it will, it will fetch the instruction first whatever it is that you have written that programme, it will fetch that instruction and it will be put that into instruction register is other kind of facility that is there, is a special register. So, these are all, they are all some special registers having some special functionalities, in the microprocessor.

So, typically, you will have these kind of not all registers can be accessed by a user, this is a typical very typical for any microprocessor, you can address for example, this programme counter, so, you cannot write anything into that, you can just read from the programme counter register to see, like that you will have in every microprocessor, some kind of a special registers will be there.

And every registers will have some function, functionality and that functionality will be written in the part of the datasheet, you do not need to guess from just this diagram, the diagram is helps you to kind of guess some of the aspects, but the rest of the thing can be read from the, so, read from the datasheet of the microprocessor.

So, there is, there is this other kind of register which is typically will be there now, it will have this memory address holding register, from that register the address can come on the address bus and that particular like location in the memory will get addressed. So, this is some kind of a description or understanding of this diagram.

(Refer Slide Time: 22:28)

## Primitive Microprocessor 2

- Q: How do you understand/describe this diagram?
  - Description same as Primitive MuP 1 for common registers
  - Notice there are additional registers: IR, W, B, and PC (program counter). W, B are general purpose registers with same read write functionality as A.
  - PC is special purpose register to keep track of sequence of execution of operations. We would like to have this sequence pre-stored and changeable or programmable. (Its similar to writing c code commands one after other).
  - IR is instruction register which allows us to write instructions (similar to each command in C code). Depending on instruction different operations would be carried out.

PRASANNA S GANDHI ghndhi@mc.itb.ac.in 27

## Primitive Microprocessor 2

The diagram shows the internal architecture of the Primitive Microprocessor 2. It features a central Data Bus and an Address Bus. The internal components include:

- Registers: IR (Instruction Register), T, AL, A, W, B, PC (Program Counter), and MA (Memory Address).
- ALU (Arithmetic Logic Unit) with an add operation.
- Memory (Mem) block.

Control signals for each register include Write Enable (WR) and Output Enable (OE). The MA register has a unique feature: its OE signal is connected to the Address Bus, allowing data from the Data Bus to be transferred to the Address Bus when MA is enabled. The ALU, PC, and Mem blocks also have OE signals connected to the Data Bus.

There is special functionality to MA register! Do you see that!?? We want data from data bus transferred to MA when WR is enabled. However the data should get transferred To address bus when OE is enabled !!

Q: How can you achieve this connection?

Q: Why we need such functionality?

PRASANNA S GANDHI ghndhi@mc.itb.ac.in 28

So, you can pause here and read through and understand little better if at all you need to be. So, now, there are some questions that we can raise now. So, these are like some kind of a special registers and how these registers would work is what is get gotten here. So, we do not worry about, so, what we want here is special functionality to this register, we want the data from the data bus transferred to MA.

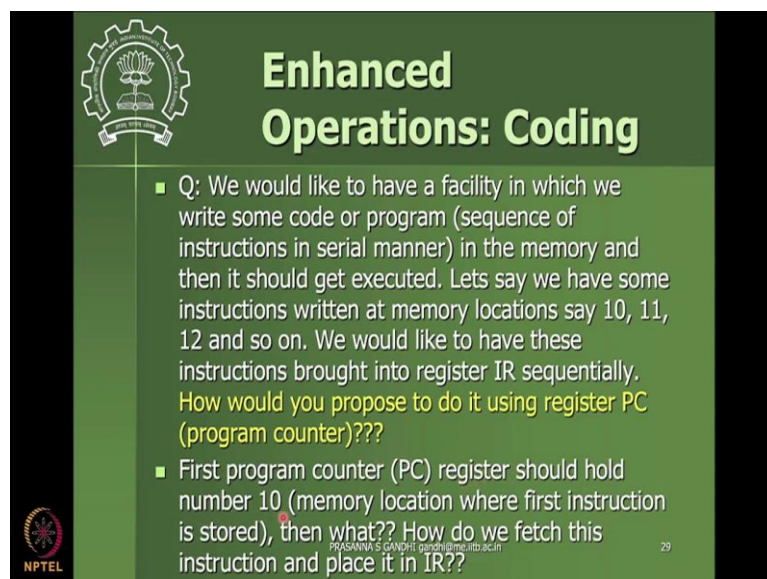
So, some data from the data bus can be transferred to this register, but you cannot read and transfer any data from this register back to the data bus. And that data will be transferred to the address bus when the output enable is is. So, this functionality is needed, can you guess, why this functionality is needed?

See we have written some kind of a programme in some memory locations. So, when we execute, we need to give microprocessor location from where you want to execute, that location when you give that location will be a starting point for this memory address register to hold, and that address will be there on the address bus and then that address can be transferred to the address bus and then your, your address bus is pointing to that particular location where your program is written in the memory.

And now that first command that you have written and for the programme that can be now fetched from, from this data bus, fetched from this memory and gotten onto the data bus and transferred to this instruction register IR, these are the, so this register will typically not see for a user to be accessed.

We are giving this here to just to give you some glimpse of how things are happening inside. So, this register you will not be able to see or many times they will not even kind of put them in the, in the architecture. Because the user is not concerned with those registers too much.

(Refer Slide Time: 24:58)



The slide features a green background with a white gear icon in the top left corner. The title "Enhanced Operations: Coding" is written in a bold, white font. Below the title, there is a list of two items. The first item is a question in white text, and the second item is a proposed solution in white text. The slide also includes the NPTEL logo in the bottom left corner and the number 29 in the bottom right corner.

**Enhanced Operations: Coding**

- Q: We would like to have a facility in which we write some code or program (sequence of instructions in serial manner) in the memory and then it should get executed. Lets say we have some instructions written at memory locations say 10, 11, 12 and so on. We would like to have these instructions brought into register IR sequentially. How would you propose to do it using register PC (program counter)???
- First program counter (PC) register should hold number 10 (memory location where first instruction is stored), then what?? How do we fetch this instruction and place it in IR??

NPTEL 29

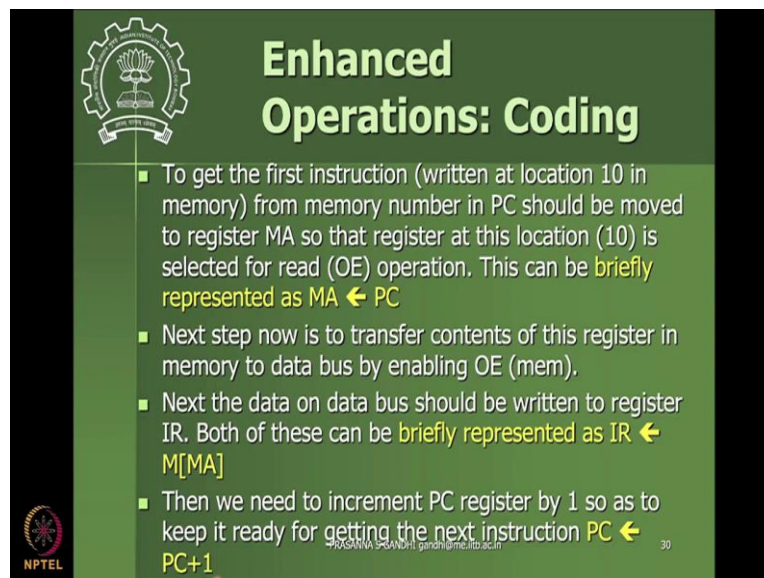
So, now the question is we would like to have this facility to code or programme and that say say some some instructions are written in some memory locations and we would like to have these instructions to be fetched as I said from the memory and given to this register IR and now from IR they will be taken and executed in some way.

So, let us see that how, how things happen little bit in more detail. The say programme counter is holding first memory location and so, this programme counter if you see is, maybe

here you can see, so, programme counter is holding that first memory location. So, that programme counter will be now holding the starting point, that starting point address will be transferred to the data bus from there it should be getting to the memory address register and then it will be available on the address bus.

The moment this is executed then second address, this programme counter will update, increment by one and then second memory address will be there here, then again it will be taken here, from here it will be transferred to this and again it will be executed or this it will be addressed and then the second line of your command or programme will be, second command of your programme will be transferred to the instruction register and then it is executed like that this sequence will continue further.

(Refer Slide Time: 26:46)



**Enhanced Operations: Coding**

- To get the first instruction (written at location 10 in memory) from memory number in PC should be moved to register MA so that register at this location (10) is selected for read (OE) operation. This can be briefly represented as  $MA \leftarrow PC$
- Next step now is to transfer contents of this register in memory to data bus by enabling OE (mem).
- Next the data on data bus should be written to register IR. Both of these can be briefly represented as  $IR \leftarrow M[MA]$
- Then we need to increment PC register by 1 so as to keep it ready for getting the next instruction  $PC \leftarrow PC+1$

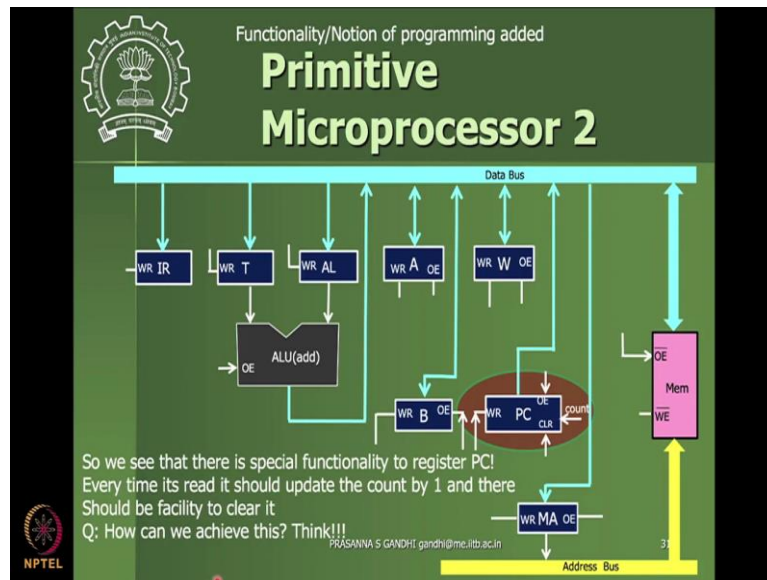
NPTEL

So, so, now, we will now create this kind of brief notion for the, for the instruction. So, PC to MA, what, what it means, contents of this register PC are transferred to MA. So, so for that lot of operations are to be done before, so, so, the way we said okay external location some some data hold in external location to be transferred to the memory, now, this is just to transfer from one memory, one register to other register, there is no memory involved here.

So, so, programme counter transfer some data to memory register. So, we we now see that, this this other instruction that we want to see that memory address will be given by this programme counter register which is now holding the memory address, that is pointing to some memory location and the data from that memory location needs to be fetched and transferred to the IR register.

So, this will indicate briefly as M is stands for memory, memory which is like pointing pointed by this memory address register that particular kind of a location whatever is there is to be transferred to IR register, this is how we represent this small, small commands. And then PC is incremented, so, programme counter needs to get incremented. So, these are like small, small brief, briefly we write these commands.

(Refer Slide Time: 28:30)



## Program Counter Details

- How can we achieve program counter to function in a way described?
  - There can be multiple possibilities again!
    - 1. Use the memory register and adder to achieve the job (this however cannot be achieved in one clock cycle)
    - 2. Use actually the counter we had seen in previous class
  - Lets say we use second option and assume for sake of discussion and understanding fundamentals that its counting only 8 states (3 bits) → we take a counter previously designed and provide additional facility needed!!! **Think How would you achieve this??**

## Program Counter Details

databus ↑ ↓

← WR PC OE COUNT CLR →

Only when the pin COUNT sees a pulse count is updated. How will we develop such circuit?? Think!!!

So what we want is to give control of update (clock input to Dffs of the counter circuit) to be considered as an external input and be connected to COUNT!! Isn't it? See circuit in the next slide

PRASANNA S GANDHI, gandhi@mc.itb.ac.in 33

## Program Counter Details

D2 D1 D0 CLR OE

COUNT

We will use this PC for our discussion further

Can you think of further enhancing PC to initiate it at desired number?? CLR makes it all bit 0 to start with!!!

PRASANNA S GANDHI, gandhi@mc.itb.ac.in

And now, we do not worry about how this this will happen actually. So, let us kind of skip through this part, how the programme counter is function to do some some things that are described. So, let us not worry about this part here, this is more like a how do you digital circuit will design about about that. So, these are more details about how the programme counter is, is made to function the way it is there.

(Refer Slide Time: 29:05)

The slide features a green background with a white gear icon in the top left corner. The title 'Enhanced Operations: Coding' is displayed in large white font. Below the title, the text 'Command example 1' is visible. The slide lists three instructions:  $MA \leftarrow PC$ ,  $IR \leftarrow M[MA]$ , and  $PC \leftarrow PC+1$ . A bulleted list follows, containing three items: 'These operations should happen irrespective of any additional operations for each command', 'Each of these operations can be considered as state', and 'We add additional operation and it becomes one command (sequence of states) Ex move contents of register A to register B'. Below the list, the instruction 'MOV B, A' is shown, followed by the updated instructions:  $MA \leftarrow PC$ ,  $IR \leftarrow M[MA]$ ,  $PC \leftarrow PC+1$ , and  $B \leftarrow A$ . The NPTEL logo is in the bottom left, and the text 'PRASANNA S GANDHI gandhi@me.iitb.ac.in' and the number '35' are in the bottom right.

So, these are the operations that we said should happen. So, programme counter contents should go to memory address register and then whatever is addressed by that register to the memory, that location of the memory, contents of that location in the memory we transfer to IR register and then we update these, these are the, this is the sequence of operation should happen every single command that we execute. So, irrespective of any additional operations that are to be done.

And then we can have these additional operations that move B to A or some some kind of operations that we we can add to this sequence. And now we want this thing to be executed in some kind of a way by using this, some kind of combination of the states or combination of the some of these, these each of these can be designated as a, as a state.



(Refer Slide Time: 30:07)

## Enhanced Operations: Coding

Command example 2

MA ← PC  
IR ← M[MA]  
PC ← PC+1

MA ← PC  
IR ← M[MA]  
PC ← PC+1  
AL ← A  
T ← B  
AL ← AL+T

■ Another command may consist of some different operation (above 3 are going to be common to all commands). Example say add of two numbers stored in registers A and B and store the addition in register A. **By looking at the microprocessor architecture can you write down sequence of operations (states)?**

Funda: To qualify what you chose as an operation or state it should be achievable in one machine (clock) cycle. For example by looking at the microprocessor construction  $A \leftarrow A + B$  cannot qualify as single operation. Think why? You cannot perform this in one clock cycle?

Q: why we need this funda at all? It is to finally develop Sequential circuitary to achieve the command result?

So, we will see how this operation can be executed in a, so, this is example of another kind of a command. So, this is so, these three are common and then now contents of A are to be transferred to the AL register, contents of B are to be transferred to the D register and A should again hold the addition of AL and T which is given by ALU, arithmetic logic unit.

This is another command that we have built up. So, we are we are kind of building these commands now, this is command one where this just moving of contents of A from register A to B happens and then like this is a second command. So, so, we want to see like how these commands can be finally executed.

(Refer Slide Time: 30:57)

No	T States Sequence	Microinstructions executed in each S	Mnemonic	Code	Result
1		<p>S1 (MA) ← (PC) S2 (IR) ← M[MA]; (PC) ← (PC)+1 S3 (B) ← (A)</p>	MOV B,A	00	Contents of A register is moved to be B register
2		<p>S1 (MA) ← (PC) S2 (IR) ← M[MA]; (PC) ← (PC)+1 S4 (AL) ← (A) S5 (T) ← (B) S6 (A) ← (T)+(AL)</p>	ADD B	01	The Contents of B is added to the contents of A and placed result in A
3		<p>S1 (MA) ← (PC) S2 (IR) ← M[MA]; (PC) ← (PC)+1 S7 (MA) ← (PC) S8 (W) ← M[MA]; (PC) ← (PC)+1 S9 (MA) ← (W) S10 (A) ← M[MA]</p>	LDA addr	10	The Contents of the location whose address is given by the second byte after the opcode is moved to the register A
4		<p>S1 (MA) ← (PC) S2 (IR) ← M[MA]; (PC) ← (PC)+1 S7 (MA) ← (PC) S8 (W) ← M[MA]; (PC) ← (PC)+1 S9 (MA) ← (W) S11 M[MA] ← (A)</p>	STA addr	11	Contents of register A are moved to the location in memory whose address is given by the second byte after the opcode

Summary of Other commands

Operation to be carried out

Notice different Feature of instruction

3. Instruction is 2 part instruction. First part is actual command and the 2<sup>nd</sup> part holds more information needed to execute: address of memory location from where data is to be fetched to A

Command with 2 parts: 2 memory location to store

So, we create this kind of a table in which like we have this commands which are now called mnemonic. So, these are like assembly language kind of commands. So, typically this is a way this is syntax that will be used in the assembly language for for moving some data from one register to other register, add something to that register like that, you will have these different different kinds of commands that will be possible.

So, say for this first command we want this, this is micro instructions that are called. So, whatever these we say PC to MA, this is a micro instruction, so these micro instructions are typically which can be completed in one kind of a clock cycle, and you have like second instruction, micro instruction, third micro instruction, like that three micro instruction are the required for executing this one single command.

So, this is your command, and then there will be some code associated with that command. So, we do not need to worry about this code here, this code will be used by this IR register to actually execute this sequence of things and how that happens, we will get some glimpse of it, but we will, we will not get into too many details about it. So, so, these are the three kind of operations to be happening one after the other for first command like that, you can have the second command operations.

And the, what it does here, result and it is code, so, typically, these are the things involved in every of your instruction set. So, this is assembly language is one of the fundamental kind of a language for the instruction, here each microcontroller will have its own instruction set, and it will produce some kind of instructions or some commands it will be mentioned there, and that its commands will get executed by using these kind of a micro instructions typically.

So, you are just getting a small glimpse of that. So, this is a second command that we have generated then some kind of a third command I am generating here. So, we will not get into too many details about how these commands are, how, what are these operations and things like that, what we want to see is that every command that or every instruction that is issued in microcontroller or microprocessor will have this some set of micro instructions to be executed and they are executed by using some kind of timing and control unit.

So, all these micro instructions together, you can see that we want now, only, we will add this S1, S2, S3 we have said here, but S1, S2 are common, because these micro instructions are common, but S4, S5, S6 are new kind of a micro instructions, like that you will identify every command whatever you want to, want the operations to happen for that command, we will

add these micro instructions to this set and all these micro instructions form like distinct kind of operations that are to be performed.

And now these operations can be say up to here when we come there are 11 kind of states that we get distinctly here. So, these 11 things, these are like remember, these are to be executed in one clock cycle kind of a thing, the way we had executed S1, S2, S3, S4 in our previous example, the same way this will happen now but the logic will be a little more complicated and more involved.

(Refer Slide Time: 34:44)

Based on four commands consisting of various operations  
All possible states are collected together

### Summary of state definitions

STATE/ Operation	Data transfer operation
S0	(PC) ← 0
S1	(MA) ← (PC)
S2	(IR) ← M[MA]; (PC) = (PC)+1
S3	(B) ← (A)
S4	(AL) ← (A)
S5	(T) ← (B)
S6	(A) ← (T) + (AL)
S7	(MA) ← (PC)
S8	(W) ← M[MA]; (PC) ← (PC)+1
S9	(MA) ← (W)
S10	(A) ← M[MA]
S11	M[MA] ← A

S0 is added to initiate PC  
Others are from previous

Note: some of the states are common to various commands.  
Q: Are number of Commands and number of states related? Think!  
Q: how to use these definitions in our primitive mup 2 to achieve operations??

Say when signal s1 is high operation (MA) ← (PC) should be carried out.

NPTEL

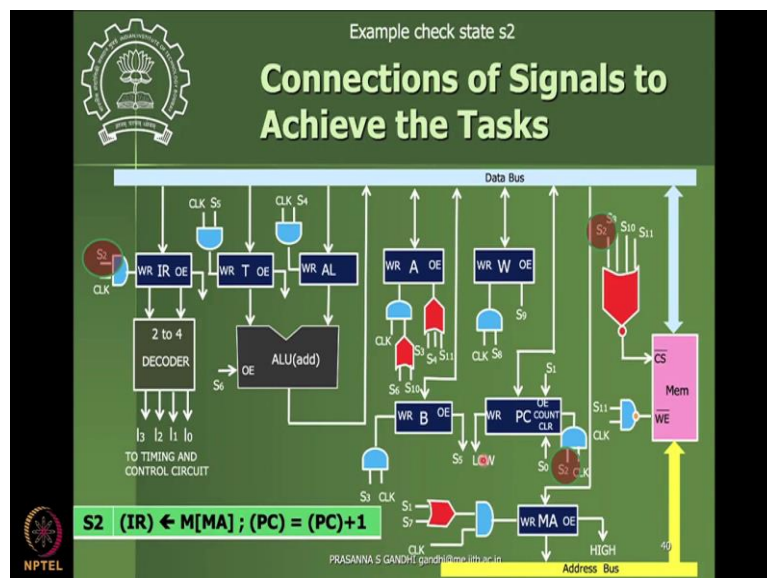
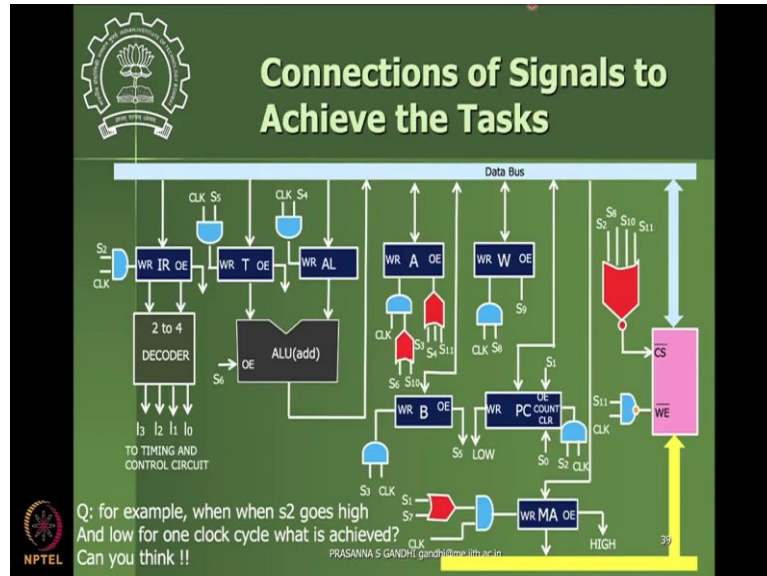
So, we will just get a small glimpse of that. So, these are like 11 kind of distinct operations, which are kind of called the states here. And now we want to have these states executed in a one particular kind of way, when command 1 or Command 2 is issued. So, if we have these four commands that are defined for this microcontroller and we want to execute whatever commands that is written by the, by the programmer.

So, but then they, when they when this command is issued then this operation should happen is what we want to make sure. So, we start generating a state diagram which is now dependent upon the, the instructions that are there, four instruction that we have. So, so, S0, S1, S2 will be instructed, will be executed anyhow, then S3 will be done only when the command issued has some particular format or this is a command which is having code 00.

So, we start drawing this kind of a state diagram from here saying that S1, S2, S3 are same state I will go from S0, to S1 to S2, and then I will go to, go to S3 only when IR is equal to

something, like that, we will start drawing the complete state diagram and then we will start connecting the signals.

(Refer Slide Time: 36:03)



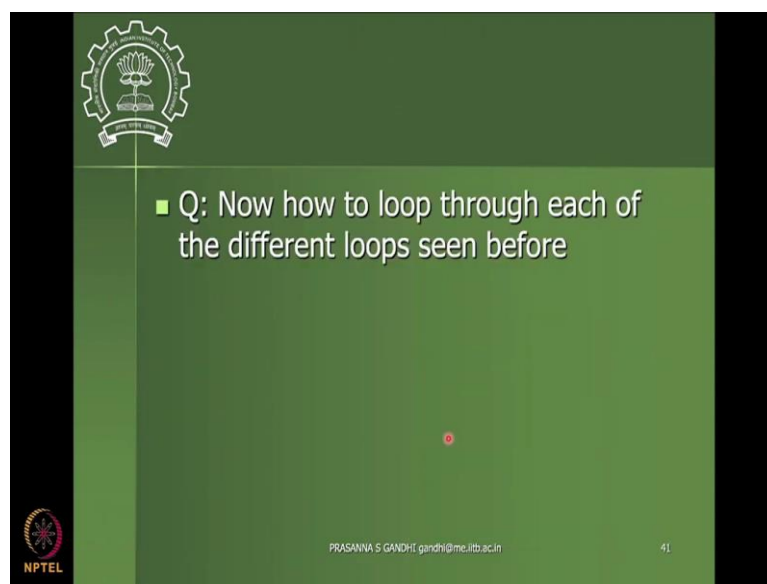
So, the signal connection again can be seen in a way that you want to have these operations to be done. So, this is a kind of very easy as similar to what your previous example. So, we have the signal connections, signal states identified and for each of the states now, we will do these kind of a signals connection to do this task.

So, now, whenever like, you know, this S1, S0 goes high, this part will happen, when S2 goes high for example, this part will happen, that is going to happen, you can check it here. So, so,

one can construct that very easily, and then what we want here is now the loop, the state diagram to be generated for these instructions to happen.

So, say for example, S2 when this happens like S2 is here, S2 is here, S2 is here, we can see that, this, this this will get executed when S2 is going high. So, so, what we have done is so far is identified the states and each of the states when it is to be executed, what is to go high and low in this connection we have done. And now we will see what is, how it is to be executed.

(Refer Slide Time: 37:32)

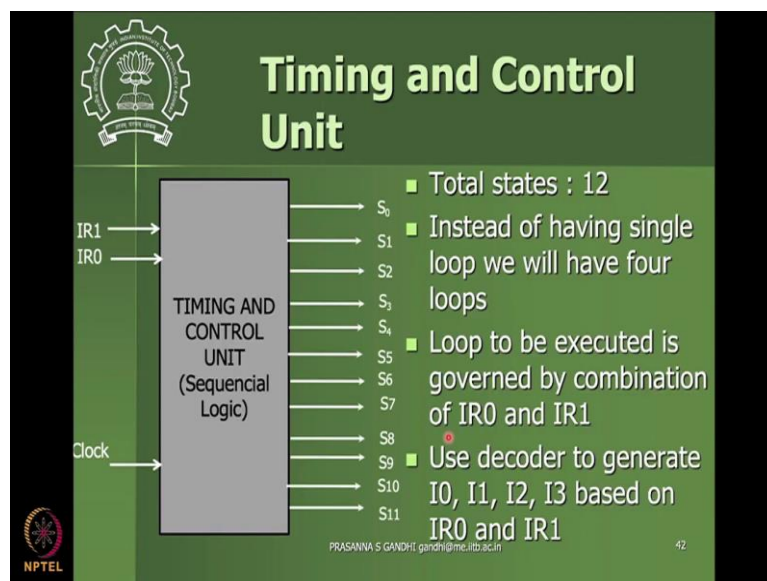


■ Q: Now how to loop through each of the different loops seen before

NPTEL

PRASANNA S GANDHI | gandhi@me.iitb.ac.in

41



### Timing and Control Unit

IR1  
IR0  
Clock

TIMING AND CONTROL UNIT (Sequential Logic)

S<sub>0</sub>  
S<sub>1</sub>  
S<sub>2</sub>  
S<sub>3</sub>  
S<sub>4</sub>  
S<sub>5</sub>  
S<sub>6</sub>  
S<sub>7</sub>  
S<sub>8</sub>  
S<sub>9</sub>  
S<sub>10</sub>  
S<sub>11</sub>

- Total states : 12
- Instead of having single loop we will have four loops
- Loop to be executed is governed by combination of IR0 and IR1
- Use decoder to generate I0, I1, I2, I3 based on IR0 and IR1

NPTEL

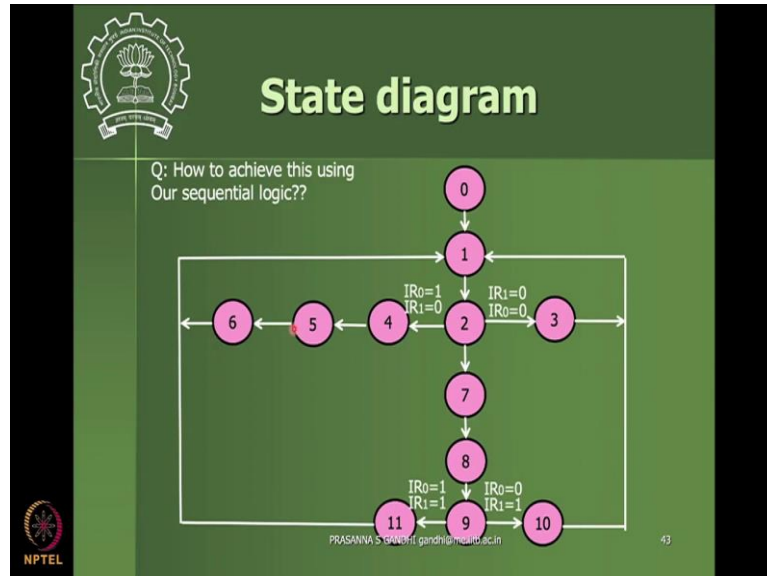
PRASANNA S GANDHI | gandhi@me.iitb.ac.in

42

So, so, for that we saw, we will use this kind of timing and control unit here. So, we had previous unit which has simply to go through S1, S2, S3, S4 that is it, but now here we want

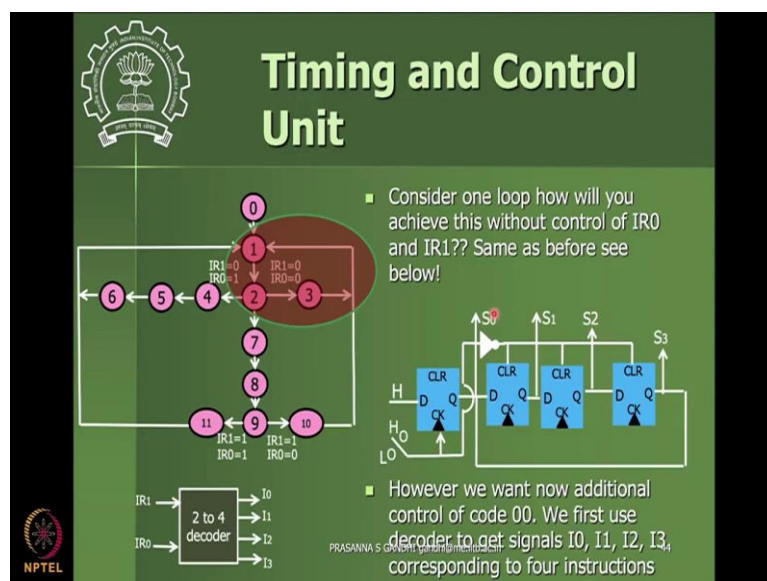
to have this control given by this IR register as I was saying before, that S1, S2, S0, S1, S2 are executed anyhow, but S3 we should go only when IR has something.

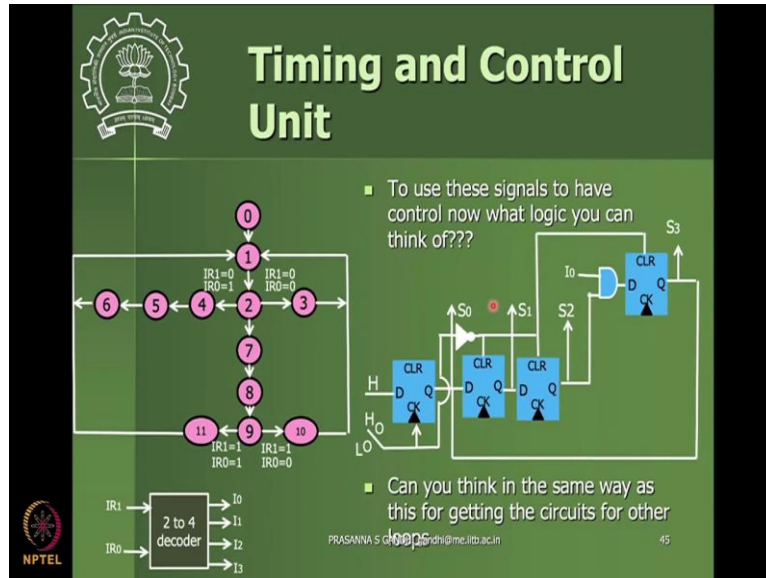
(Refer Slide Time: 38:08)



So, so, so, we start writing it as a state diagram. You remember our sequential logic we developed by using state diagram. So, this is how we develop this diagram. So, when IR is this from 2 you go to 3. And then you go back from, otherwise IR if it is something else you go through these different states, then again you have some, some kind of a way, this is like a you get this sequential logic diagram done.

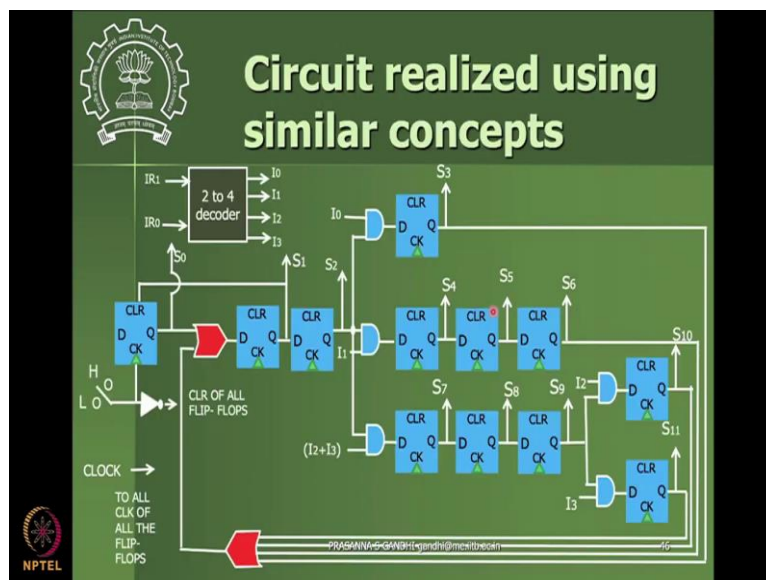
(Refer Slide Time: 38:58)





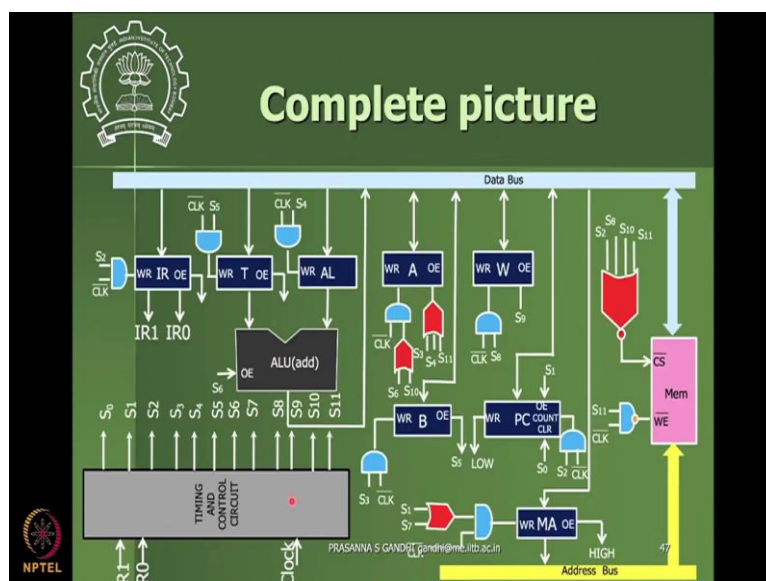
And then you can execute this logic by using your so, each of these states will have one D flip flop selected, and we can start connecting things together and we add on with the D flip flop, some more kind of a logic elements to get this executed. So, I will not get into details of that again, we will just directly see the, see the final result here. So, you can pause and go through some of these things in more detail if you want to later.

(Refer Slide Time: 39:08)



So, this is how this final circuit is going to look like. So, so, this we generate depend upon instruction we use decode the instruction and get these pins high and when these pins are high, then the sequence of operation is going to run according to the state diagram that we have seen.

(Refer Slide Time: 39:29)




So, this is like a complete picture of this stuff. This is what typically you will not get to see these in the microprocessor datasheet, what you will see is only the architecture that this has this registers and that registers have this functionality, how it happens, you do not have to bother about. So, this is like a complete picture of, of a typical microcontroller execution operation execution that happens in the microcontroller.

And of course, there will be a lot of other facilities, you can have additional registers put together for getting these for loops, while loops and then storing something for temporary that stack and other kind of registers can be introduced. So, once you have this base understanding of how things are happening then all the other things can be imagined very nicely how they would have been happening there.



(Refer Slide Time: 40:22)




## Summary

- S1 and S2 are common to all paths: read memory location pointed by PC and place its contents in IR register and increment PC
- Four different paths can be executed by placing appropriate "code" in the IR register
- The sequence of codes to be executed is saved in memory (repeat possible) and fetched from there in every "path" by S1 and S2 commands
- We have designed a microcontroller having 4 instructions (codes 00, 01, 02, 03) executing

NPTEL

PRASANNA S GANDHI, gandhi@me.iitb.ac.in

48



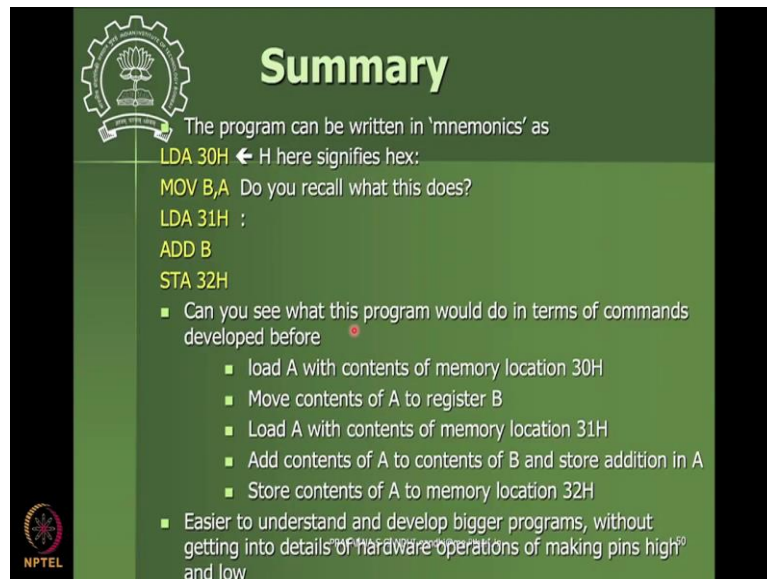
## Summary

- Each of these group of instructions (executed for each Sx in one clock cycle) is called micro-instruction
- An instruction say **MOV B,A** consists of several micro-instructions (3 in this case)
- When muc is put on it first initializes PC to 0 (state S0) then goes through instructions S1 and S2. Thus the execution of instructions stored in memory locations from 0<sup>th</sup> address begins one by one.
- Thus desired program: "in machine language" can be developed

NPTEL

PRASANNA S GANDHI, gandhi@me.iitb.ac.in

49



**Summary**

The program can be written in 'mnemonics' as

```
LDA 30H ← H here signifies hex:
MOV B,A Do you recall what this does?
LDA 31H :
ADD B
STA 32H
```

- Can you see what this program would do in terms of commands developed before
  - load A with contents of memory location 30H
  - Move contents of A to register B
  - Load A with contents of memory location 31H
  - Add contents of A to contents of B and store addition in A
  - Store contents of A to memory location 32H
- Easier to understand and develop bigger programs, without getting into details of hardware operations of making pins high and low

NPTEL

So, so, this is just a summary of this whole thing all process of this four lectures on microprocessors that we have done to get all the basics very very nicely understood here and this understanding will help you read through the data sheets of microprocessor very easily. So, so, we have this these instructions.

These are called typically assembly language instructions and each of these instructions will consist of several micro instructions and each micro instructions will will tell you that much time will be taken for, so, if you have three micro instructions to run these then three clock cycles will be needed to execute this. So, you can get also some sense of how much time will be taken to execute my my command.

And this machine language or assembly language is what is called this instruction set is termed as, and and this is how you write a programme in C or any other language, it is getting when it compiles, what that compiler does is basically converts all your hard language stuff into this assembly language instructions and and then executes that same language instruction the way we saw, they get executed by using the timing and control unit. So, this that is how like, things happen typically in the microprocessor. So, we will stop here for now.