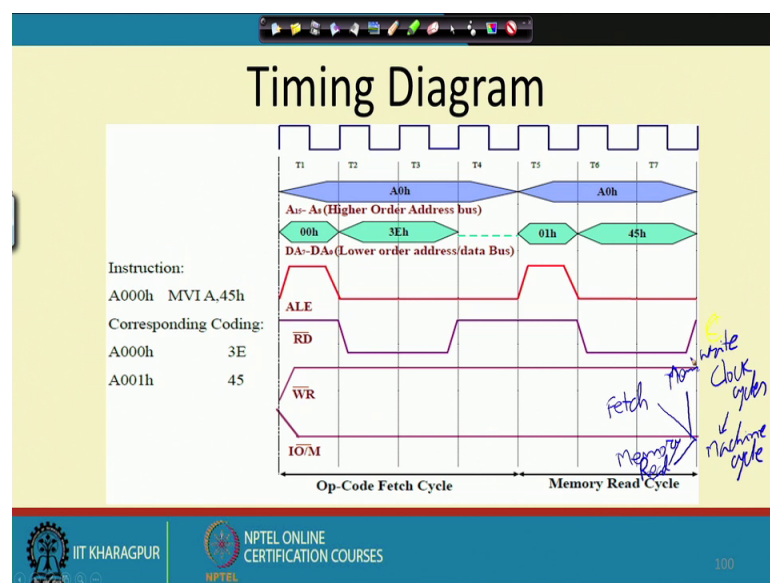


Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 14
8085 Microprocessors (Contd.)

After the MOV instruction, next will look into the timing diagram corresponding to the MVI instruction.

(Refer Slide Time: 00:21)



So, different from the previous instruction is that a MOV instruction was a single byte, instruction size of the instruction was 1 byte. So, when you do the opcode fetch. So, you get the full instruction, but here that will not happen, because now this will be it will try to it has to get 2 bytes because it will look into the code of MVI. So, MVI say for example, this instruction MVI A 45 hex. So, the if the location memory location is a A 0 0 0, then these the code of this is 3E, the MVI A code is 3E and the value the next value 45 also has to be there. So, this is a 2-byte instruction where the first byte is 3E second byte is 45.

So, in the memory the location A 0 0 0 will have the value 3 and the location A 0 0 1 will have the value 45; so for fetching the instruction. So, our first in the first cycle I will get the value 3E, and after getting the 3E in the decode phase will understand that it is it is the instruction to it is the instruction to move some immediate value to the accumulator.

So, the next the processor will understand that now I have to get the immediate value from the memory. So, this is another memory access that will be required and in that memory access. So, the once the value comes to the processor the 45 comes to the processor, then it will be put into the accumulator. So, that is an internal operation. So, that is that can be done by work as far as the outside world is concerned. So, you can see the sequence of control signal activations like this.

So, for the first part of the instructions it is same as the previous 1 that is opcode fetch. So, only thing is that the same similar with the higher order address bus at 8 0 lower order address bus as 0 0 the ALE signal is given at T 1. And so, the value get latched these 0 0 gets latched onto the address latch, and then this read bar signal is given. So, the memory will put the value 3E on to the data bus.

So, 3E comes to the instruction register. So, it is decoded at this time. So, at the time T 4 the instruction is decoded. So, after decoding the processor understands that I need to fetch the immediate operand. And as soon as the memory the processor has done a fetch this program counter values is automatically updated.

So, at T 4 the program counter value is also updated. So, the program counter now contains A 0 0 1. So now, it goes to another cycle which is known as memory read cycle previous cycle was opcode fetch cycle. This cycle is called memory read cycle. The operations are more or less similar. So, the higher order address bus contains the higher order address byte that is A 0 lower order address bus contains the lower order address byte that is 0 1 ALE signal is given, and this read bar signal is given here.

So, has a result the processor will the sorry, the memory will respond with the content of memory location A 0 0 1, and that value is 45 so, that is available on to the data was. So, this value will reach the again the internal inside the processor. And once it is inside the processor, the decoder the instruction decode the decoding and control logic it will understand that these value should go to the accumulator. And that is done sometime here though it is not documented. So, sometime here that operation will be done, because 45 is available inside the in the internal data bus of the processor. So, that will be copied on to the accumulator register. So, this way it is done and this does not require any further decoding, that is why we do not have another clock cycle here. So, 3 T states are sufficient.

So, for opcode fetch we require 4 T states for memory read operation we require 3 T states; so this way depending upon the complexity of the instructions. So, you may require more number of clock cycle, more number of T states. Now the so, this part so, this T 1 T 2 T 3 T 4. So, this is called opcode fetch cycle, it is also called a machine cycle. Similarly, this memory read cycle. So, this is also called a machine cycle.

So, we have got these clock cycles, we have got this clock cycles, which are sorry; we have got this clock cycles which combines into something called machine cycle. This is something called machine cycle, and these machine cycle there can be several type of machine cycle like this fetch is a type of machine cycle, then memory read is a type of machine cycle then memory write is another type of machine cycle, this memory write is another type. Then we can have different types of machine cycle, then when we look into this I O operation, then we will see that I O cycle there are different type of machine cycles like that.

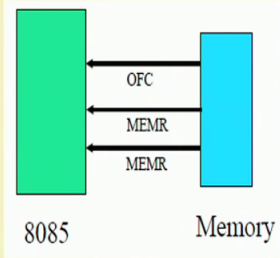
So, this way we can have different types of machines cycle. So, we will look into some more examples, like say the next instruction that will look into.


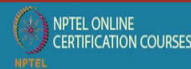
(Refer Slide Time: 06:13)

Timing Diagram

- Instruction:
- A000h LXI,A,F045h
- Corresponding Coding:

A000h	21
A001h	45
A002h	F0



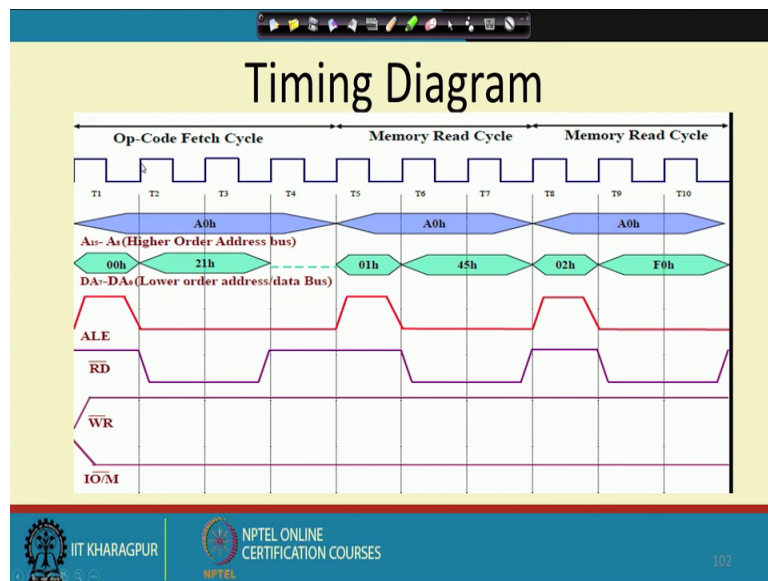


101

Next, instruction that we will look into the LXI instruction; so this is more complex where this LXI A F 0 45 hex; so it is load extended immediate. So, assuming the same thing that this instruction is located at memory location A 0 0 0 hex. So, and this code of LXI A you can find from the manual that is the value is 21 and this F 0 45 is the 16-bit

value that which gives the address from where the a registered will be loaded. So, that was the so that was the meaning. So, it is load extended in direct actually. So, that is giving us this is the address from where it will be loaded.

So, how this will be executed? So, this will be executed by some up-code fetch and read cycle and write cycle like that.

(Refer Slide Time: 07:13)

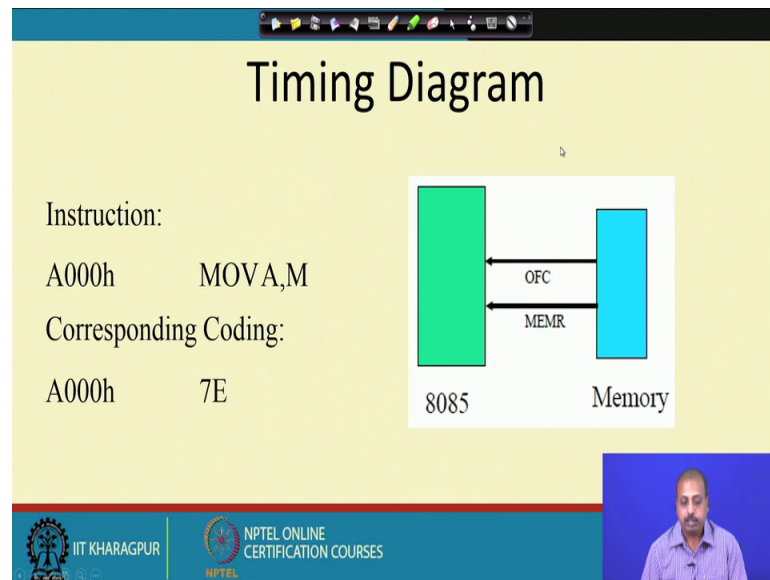


So, what is done? First this first this opcode fetch cycle so, this is giving us this value, this A 0. So, that is on the higher order address bus, and 0 0 is on the lower order address bus. So, this is ALE signal is given, read bar line is given. So, that is getting the value 21 hex onto the address bus.

Now so, this is the 21 hex that we have got sorry at the 21 hex will be in on the data bus. So, this 21 hex comes to the instruction register. So, the instruction decoding is done in the cycle T 4, and once it is done, then it understands that I need to get I need to do to more memory cycles to get the full instruction. So, this 45 hex that is so, this so, this is the so, again this address was 0 0. So, that is incremented. So, it becomes 0 1 higher order address bits remain un altered A 0. So, this higher order address bus it has got A 0 lower order address bus has got 0 1 and again the read bar signal is given. So, so sorry ALE signal is given, and the read bar signal is given here.

So, has a result the memory puts the value 45 hex on to this data bus. So, this 45 hex comes inside the processor. And then I have got the other byte has to be loaded. So, this is this 0 2. So, that is this PC value has been updated. Like every time it do a the PC it the processor does an instruction fetch. So, this PC value gets updated. So, PC value is updated to 0 2. So, this is getting the PC value there. So, so it is getting hex 0 2 here, and A 0 here and the ALE signal is given memory read bar signal is given, and it gets the memory will put F 0 on to the data bus. And for this entire operation, this right bar signal is kept high and I O M bar line is kept low. So, it is a memory operation. So, so this is 3 machine cycle that are needed. So, opcode fetch followed my memory read followed by another memory read.

(Refer Slide Time: 09:32)

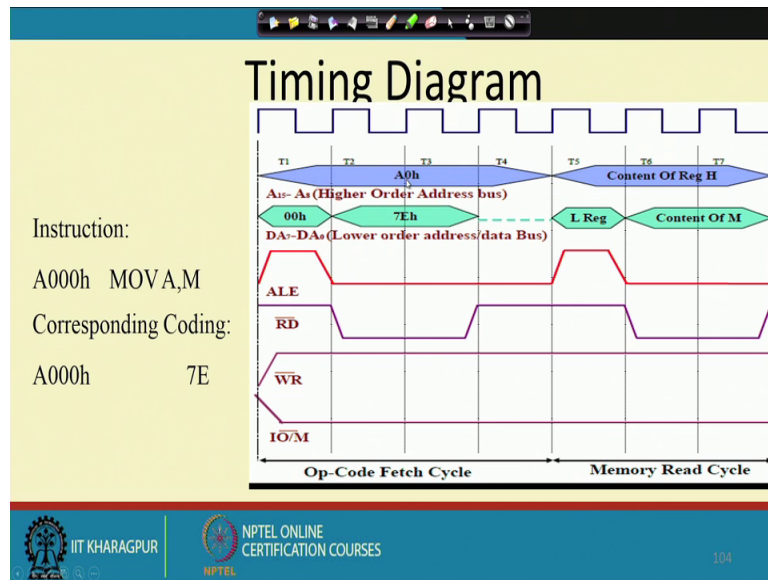


So, next we look into this MOV A comma M instruction. So, this is basically somehow, I need to access the this memory, and the address of the memory is available in the HL register pair. So, this HL register pair acts as this this HL register pair acts as the pointer, like in the previous instruction. So, this LXI LXI H not LXI A is be LXI H. So, after this values are available 45 hex and F 0 hex. So, these 2 values will be loaded into the HL register pair.

Now, this instruction MOV A comma M; so this here the HL register Acts as the memory pointer, and that value is loaded on to register A. So, how this will be done? First this instruction has to be fetched. So, that is an opcode fetch operation and after these inter

the code of this instruction is 7E when they it come to the instruction decoder. So, decoder you will understand that I have to access memory once again to which location, the location whose address is in HL pair. So, that is exactly what is done in the instruction.

(Refer Slide Time: 10:54)

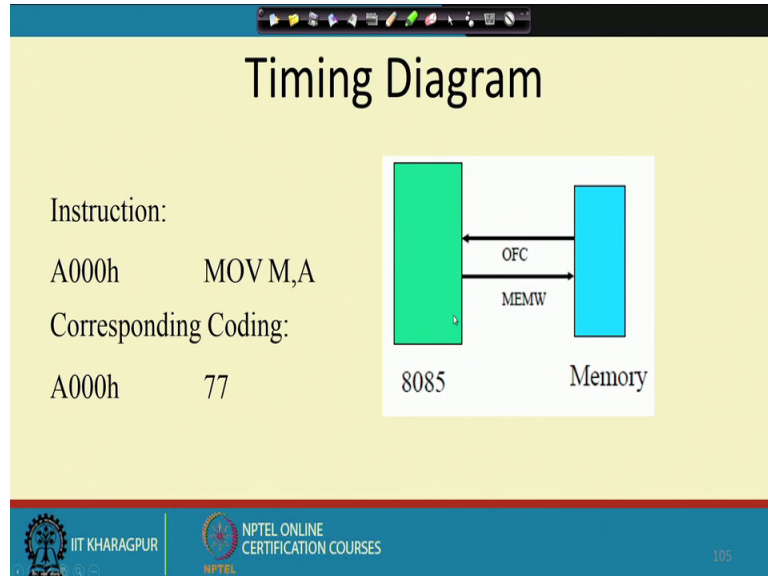


So, let see how it how is it working. So, you see the first part of the instruction execution is same as the opcode fetch cycle. So, it is it is higher order address bus as A 0, lower order bus as got 0 0, ALE is given read bar is given. So, the 7E the opcode of this instruction is available on to the data bus. So, that goes to the instruction decoder in T 4 the instruction is decoded, and now the processor understands that the HL register pair content should be used as the address from the of the memory location from where the content has to come to the accumulator. So, this H register value is put on to the higher order address bus. L register value is put on to the lower order address bus, and the ALE signal is given.

As a result, the address bus of the memory will have the HL register value into it. And this now when the read bar signal is activated. So, memory will give the content of that memory location which is pointed to by the HL pair, and that content of the memory location will be available here and that after it has come. So, it will be immediately loaded into the a register. So, gain the same thing that opcode fetch will take 4 cycles and

this memory read operation will take 3 cycles. That way this MOV A comma M instruction will be executed.

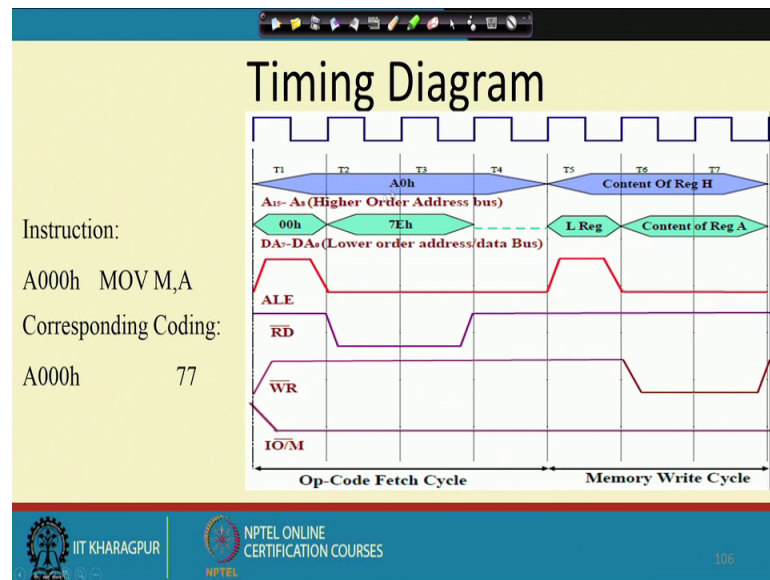
(Refer Slide Time: 12:22)



Otherwise other ways; so if I have considered the instruction MOV M comma A; that is, the content of these accumulators should go to the memory location whose address is pointed to by the HL pair.

So now we have gotten opcode fetch operation. So, opcode fetch cycle where the content from the memory will come to the processor. And this will be followed by a memory write operation where the content of the accumulator register will be written into the memory location, whose pointed to by the HL pair. So, the coding for this MOV M comma A instruction is 77.

(Refer Slide Time: 13:01)



So, the timing diagram will be looking something like this. So, this is the A 0. So, A 0 is the higher order address bus content A 0. Lower order content is 0 0. So, this value should not be 7E this should be 77. So, there some mistake here. So, this value should be 77 because the data bus sorry memory will put the content of location A 0 0 0 on to the data bus and that is 77.

So, once the 77 comes to the processor. So, processor will understand; that it is MOV M comma A instruction. So, in thus in the time cycle T 4 flock cycle T 4. So, it will understand that it is a MOV M comma A instruction. Now it has to put the content of accumulator onto the memory location pointed to by HL. So, how to do that? Again, the same thing that this H register content is put on to the higher order address bus, L register content is put on to the lower order bus. And this content of a l signal is activated. So, the address gets latch onto the address latch, and after that the processor will put the content of a register on to the data bus, and give the right signal. So, this right bar signal is low here you see. So, it is a memory write operation. So, the counter memory will get this address part data part and the right signal.

So, as a result the value will be written on to the memory. So, this is called a memory write cycle. So, you have seen memory read cycle. So, this is a memory right cycle the difference is that instead of the read bar signal going low write bar signal will go low. So,

that is the timing diagram for MOV M comma A. So, this way if you look into the manual of the processor for every instruction so, it depicts the timing diagram.

(Refer Slide Time: 14:48)

The Stack

- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO structure.
 - Last In First Out.
- The stack normally grows backwards into memory.
 - In other words, the programmer defines the bottom of the stack and the stack grows up into reducing address range.

So, that tells like how individual instructions are executed by the processor. So, exact control signals that are going to be activated that are visible from the outside, how were activated and what are the values. So, that can be seen from the timing diagram of the instructions.

Next, we will introduce another very important concept that is used in as in assembly language programs, or for that matter it is a very important part of any program implementation may be high level maybe assembly level. But this is very important. Why is it so important? So, will see it is slowly. So, this is the stack. So, stack is an area of memory identified by the program of for storage of temporary information. So, this is a last in first out structure. So, what happens is that you can say many times what happens is that when I am writing a program. So, if this is the body of the program, then we in any language.

So, maybe it is in assembly level maybe it is in some high-level language or whatever it is, we right some procedures separately or sub routine sometimes they are called subroutine sometimes functions are. So, maybe at this point I call this procedure. And meaning that this part of the procedure will do the calculation, and once the calculation is

over the control will go back to the next instruction in to the program on to which this was from where it was called.

So, this is a very important feature, but how do you implement this thing. So, when the computer or the processor was executing. So, the code here. So, it was doing it in sequence. So, the PC value was just getting implemented sequentially, but when this has to jump to this then after this. So, it has. So, this point in easy like I can load the program counter with the value of this location like if this is the full memory and in this part of the memory. So, the main program this one has been loaded and say in this part. So, this procedure has been loaded.

So, going from here to hear is not a problem, because I can just load the program counter with the value of this address. So, that it will it will next execute the instruction from here, but for going back how do I go back from here to the point from where it was called. So, for that purpose I need to remember the point at which this program was called. So, that is done by means of this stack. So, this sub program implementation or procedural implementation. So, this is going to be an important is going to be an important operation in for any programming language, and then the stack will be required.

So, the stack normally grows backwards into memory. So, it is what it means is that, if this is the memory. So, we define a portion of it define a portion of it as stack.

(Refer Slide Time: 18:19)

The Stack

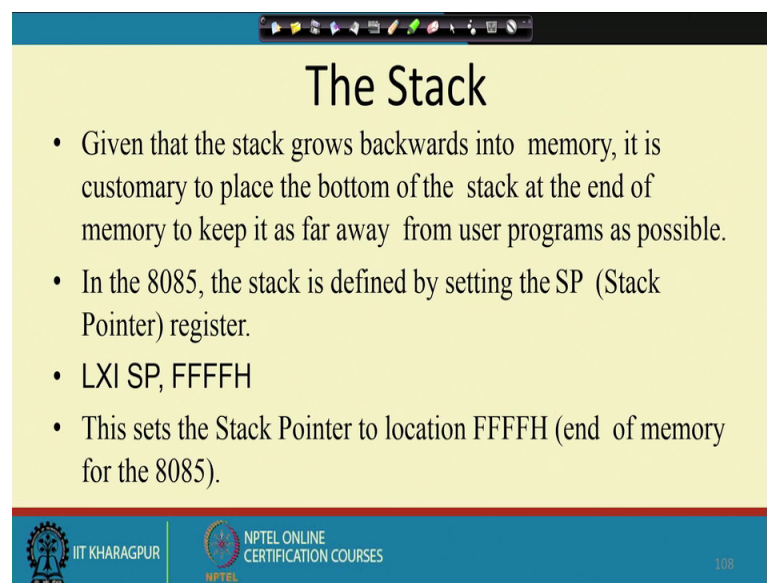
- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO structure.
 - Last In First Out.
- The stack normally grows backwards into memory.
 - In other words, the programmer defines the bottom of the stack and the stack grows up into reducing address range.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if this is the memory so, we say that so, this this is the bottom of the stack at this point the stack starts at this point. So, these addresses maybe say 2000 hex. This may be 2000 hex now as you are putting more and more element into the stack. So, this stack will grow in a upward in the upward direction. So, 2000 next will be a say 1FFF. So, that way it will go on decrementing.

So, the size of the stack will so, the stack will grow backwards towards the lower address in general. So, that is so, that is normally that happened it may or it is not that it cannot grow the other way. So, for most of the processors it is done in this discussion, but that is not mandatory. So, programmer defines the bottom of the stack and stack grows up reducing the in the into reducing address range. So, this address values will decrement as the more and more element are put into it.

(Refer Slide Time: 19:31)



The Stack

- Given that the stack grows backwards into memory, it is customary to place the bottom of the stack at the end of memory to keep it as far away from user programs as possible.
- In the 8085, the stack is defined by setting the SP (Stack Pointer) register.
- LXI SP, FFFFH
- This sets the Stack Pointer to location FFFFH (end of memory for the 8085).

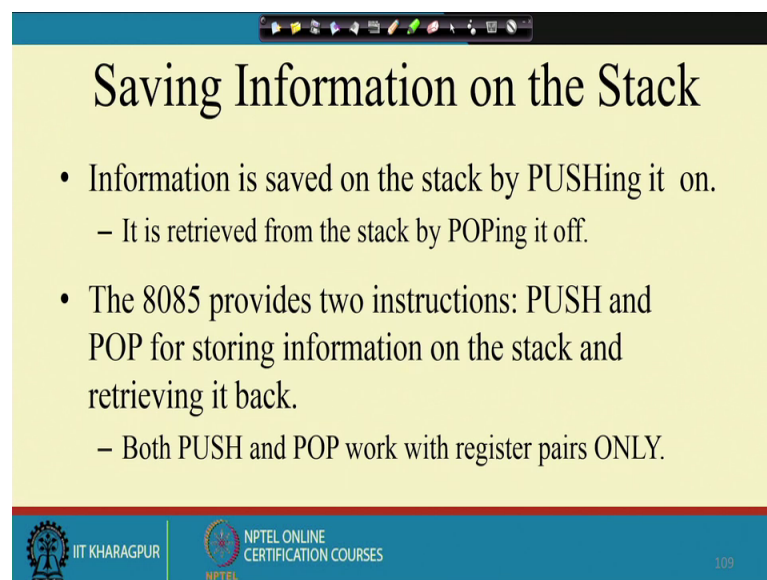
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 108

Now, the next will look into how this how this stack can be useful. So, given that the stack grows backwards into the memory. So, it is customary to place the bottom of the stack at the end of memory to keep it far away from other users program. So, what happens is that say in 8085, we know that the total address space is address range is 0 0 0 to FFFF hex. Now if I assume that this we have got entire memory space occupied by physical memory chips, that is it is not that the actual memory system has got less than 64 k. We assume that the memory system that has that has been integrated with the processor, it has got 64 k space fully available.

So, all the places have are valid. So, there is no gap in the address space, then in that case. So, what will do we should do will initialize this stack pointer with the last the highest address in the address range. So, if everything is there then it will be FFFF hex. If the if the last memory chip is not going up to FFFF hex, if it goes say for example, to will say 6000, in that case the stack pointer is generally initialized to 6000. So, it will go it will be done like that.

So, there is a special register call stack pointer which is used for this stack operation. So, if the instruction LXI is SP FFFF hex this will load the instruction load the stack pointer if the value FFFF hex.

(Refer Slide Time: 21:13)



The slide is titled "Saving Information on the Stack" and contains the following text:

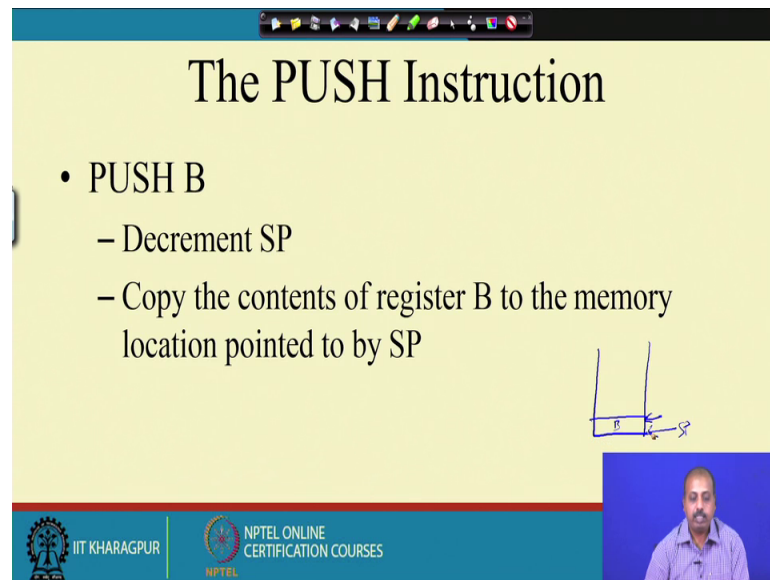
- Information is saved on the stack by PUSHing it on.
 - It is retrieved from the stack by POPing it off.
- The 8085 provides two instructions: PUSH and POP for storing information on the stack and retrieving it back.
 - Both PUSH and POP work with register pairs ONLY.

The slide footer includes the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the number "109".

And then from that point onward so, we can use some instruction there are push and POP there are 2 instruction in 8085 by which you can save some values on to the stack or you can retrieve some values from the stack. So, the push means we are storing the value on to the stack. And POP means we are trying to we are trying to get the value out of the stack.

So, they work with register pair only. So, it is so, you cannot push POP a single register like you have if you do a push operation. So, you cannot say push only the B registered. So, you have to push the B C pair, you have to push the D E pair, HL pair, and for the accumulator the accumulator and the status register pair. So, this push so, how this push B instruction is executed.

(Refer Slide Time: 21:58)



The slide is titled "The PUSH Instruction" and contains the following content:

- PUSH B
 - Decrement SP
 - Copy the contents of register B to the memory location pointed to by SP

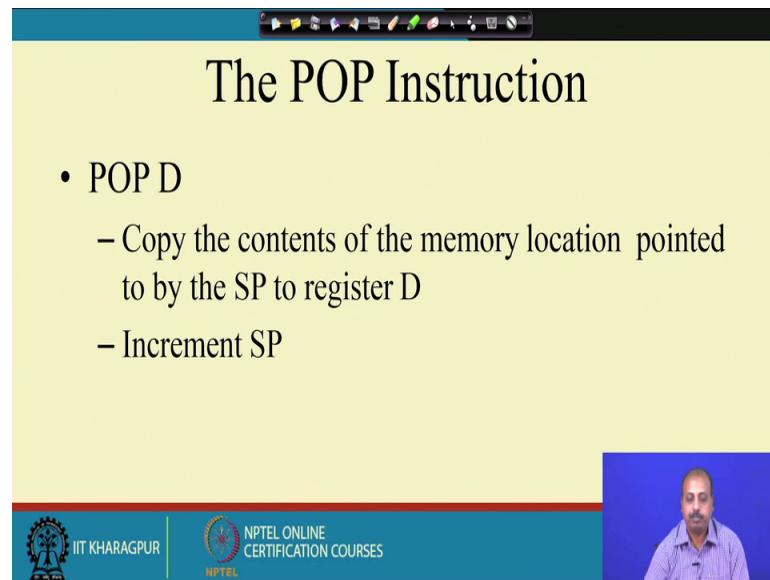
A diagram on the right side of the slide shows a vertical stack of memory cells. An arrow labeled 'SP' points to the second cell from the bottom. A box labeled 'B' is shown below the stack, with an arrow pointing to the second cell from the bottom, indicating that the contents of register B are being pushed onto the stack.

The slide footer includes the IIT Kharagpur logo and the NPTEL Online Certification Courses logo. A small video inset of the presenter is visible in the bottom right corner.

So, first the stack pointer will be decremented, and the content of the register B will be put into the memory location pointed to by SP. And then then this then this this this B register will be will be that value will be becoming will be saved in to the stack.

So, what is it like say if this if the currently the stack current stack is like this. So, this was the bottom of the stack, stack pointer was pointing here. And if I have got this operation this push B instruction, then this B register content will be coming here. And this step pointer will be decremented. So, stack pointer will now point to the previous location. Similarly, we have got this we have got this POP D instruction; so this POP Does or so, POP D instructions.

(Refer Slide Time: 23:11)



The POP Instruction

- POP D
 - Copy the contents of the memory location pointed to by the SP to register D
 - Increment SP

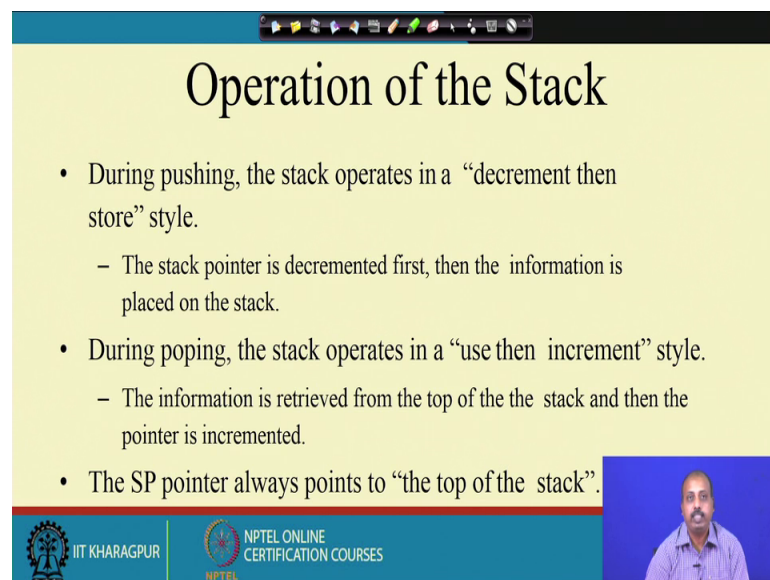
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video inset in the bottom right corner shows a man with a beard and glasses, wearing a light-colored shirt, speaking against a blue background.

So, it will copy the content of the memory location pointed to by a stack pointer on to the register D. So, that that will be that will be there.

Now, other than that, we can have so other than that so, we have to understand the style in which these operations are done onto the stack.

(Refer Slide Time: 23:31)



Operation of the Stack

- During pushing, the stack operates in a “decrement then store” style.
 - The stack pointer is decremented first, then the information is placed on the stack.
- During popping, the stack operates in a “use then increment” style.
 - The information is retrieved from the top of the the stack and then the pointer is incremented.
- The SP pointer always points to “the top of the stack”.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video inset in the bottom right corner shows a man with a beard and glasses, wearing a light-colored shirt, speaking against a blue background.

So, during pushing the stack operates in a decrement and then store style. First the stack pointer value will be decremented, and then the value will be stored on to the stack. So, so at the beginning there is nothing in the stack. So, the stack pointer is FFFF hex. So,

that the value will be decremented, it will become FFFFE and then the value will be stored.

Similarly, on the POP operation the stack of first it will use the value use than increment style; that is, the information will be retrieve from the top of the stack, and then the pointer will be incremented. So, that it goes to the next location. And the stack pointer is always pointing to the top of the stack. So, that way the push POP both the operation. So, they are from the top of the step.

(Refer Slide Time: 24:28)

LIFO

- The order of PUSHs and POPs must be opposite of each other in order to retrieve information back into its original location.

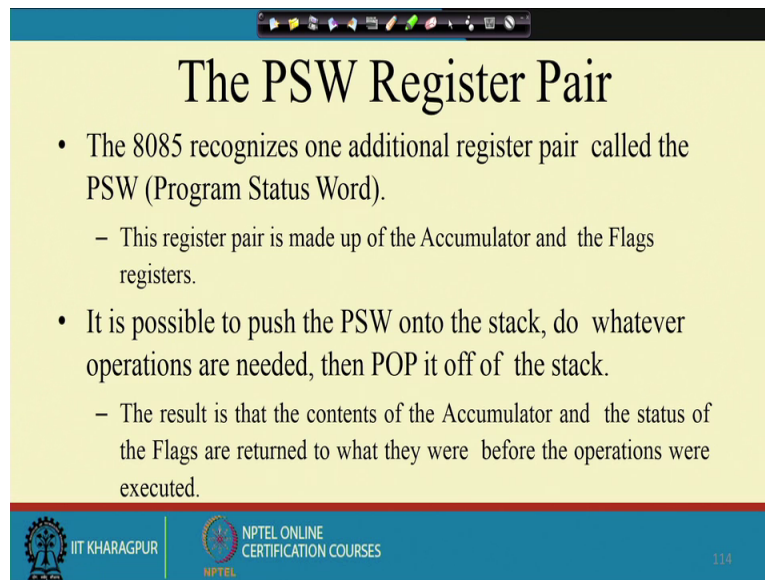
PUSH B
PUSH D
...
POP D
POP B

IIT KHARAGPURNPTEL ONLINE CERTIFICATION COURSES

So, it uses a last in first out structure. So, the order in which the push and POP are executed they must be opposite of each other. So, otherwise the values retrieve will not be the same. Like here, if you have done push B push D like that then at the end. So, if you want that this D value whatever was there at this point should be getting back the we after executing this code. So, D value might have been modified. So, if you want to get back the original D value that we have pushed here, then I should do a POP D.

So, this value of value from the top of the stack will go to the D register and then we can have this POP B. So, this value will be POP 10 stored in the B register; and if you change the orders. So, if you do a POP B here, then what will happen content of D register will come to the B register, instead of going to the D register. So, that way this order is very important. So, you push in in one order, and while retrieving you retrieve in the other order to get the original information back.

(Refer Slide Time: 25:40)



The slide is titled "The PSW Register Pair" and contains the following text:

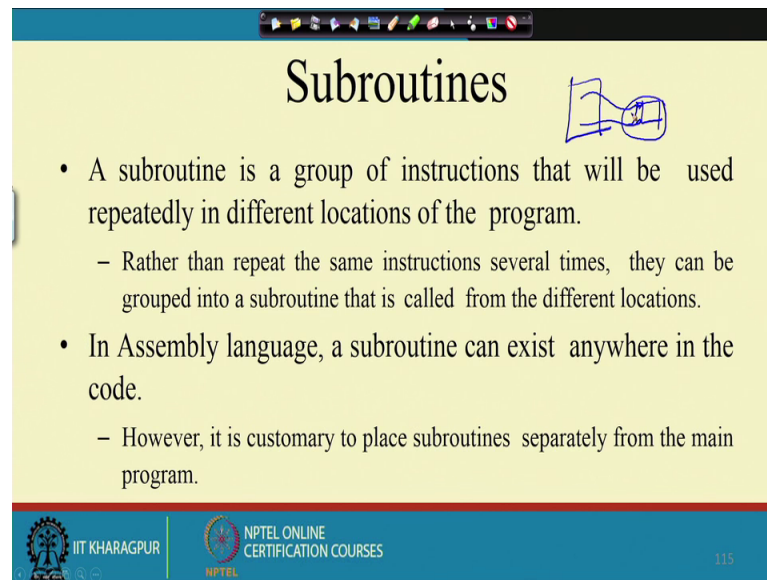
- The 8085 recognizes one additional register pair called the PSW (Program Status Word).
 - This register pair is made up of the Accumulator and the Flags registers.
- It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack.
 - The result is that the contents of the Accumulator and the status of the Flags are returned to what they were before the operations were executed.

The slide footer includes the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the number "114".

Now, in addition there is another register pair which is called PSW. So, this is the register pair made of the accumulator and the flag register. So, the 2 registers accumulator and flag register they make up this PSW register. And this PSW so, you can push this PSW on to the stack and do whatever operation and then POP it out just like any other register pair, this PSW will act as another register pair.


So, accumulator and status register status of the flag. So, they will be saved. So, if you are doing a push PSW, and if you want to do a POP, POP PSW then those values will be taken from the top of the stack and loaded into the a and PSW register status register.



(Refer Slide Time: 26:30)



Subroutines

- A subroutine is a group of instructions that will be used repeatedly in different locations of the program.
 - Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.
- In Assembly language, a subroutine can exist anywhere in the code.
 - However, it is customary to place subroutines separately from the main program.



 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

115

Now, this is stacks are useful when we are going to implement the subroutine. So, subroutine as we know that this is a part or the procedures; so in high level languages there often known as procedures or function. So, it is say group of instructions that will be repeatedly that will be repeated in different locations of the program; so instead of writing it several times. So, we write it once, and then it is called from different places. So, like say this one.

So I have, I am a writing a program and in that program. So, we have got say in this program, at several places I want to use this small routine. For example, maybe I want to compute the square root of numbers at different places in the program. So, what we do instead of repeating that square root routine in that every place. So, we write a separate square root plus subroutine, and where ever I need to calculate the square root. So, I call this routine similarly say again at this point it is necessary. So, again I call this routine. So, it will be executing them.

So, this way at every point I want to use this routine so, I can just give a call to that. So, that way it will save the space, because I do not I am not writing the code again and again. So, that way it is going to save the address space. They have to the may be programs size will be less now, so in assembly language. So, this subroutine can exist anywhere in the code. However, it is better that we put the subroutine at the end of the main programs. So, first you put the main program then we write the individuals

subroutine. So, that is the standard practice, but ideally there is nothing wrong in putting the subroutine at any place in the program.