

Lecture - 53
8085 Microprocessor
(Contd.)

So, we can also use other register pairs.

(Refer Slide Time: 00:17)

Using the Other Register Pairs

- There is also an instruction for moving data from memory to the accumulator without disturbing the contents of the H and L register.
- **LDAX Rp** (LoaD Accumulator eXtended)
 - Copy the 8-bit contents of the memory location identified by the Rp register pair into the Accumulator.
 - This instruction only uses the **BC** or **DE** pair.
 - It does not accept the **HL** pair.

Handwritten notes:
RP → HL
XP = X9
↓ 1000
↓ 2000
LXI H, 1000H
~~LXI H, 1000H~~
LXI D, 2000H
LDAX D
MOV M, A

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, there is one instruction like LDAX. So, this LDAX instruction; so, it means that load accumulator extended. So, this one; so, this also accept say register pair as the operand, and then this register pair we can somehow this register pair if it is loaded with some value and then that will be acting as the address. So, copy the 8 bit contents of the memory location identified by the register pair R p register pair into the accumulator.

So, this will take a value on to the accumulator. So, it does not put it on to other location, but it will put it on to this accumulator register and then this register pair, it can be B, C or D E. So, I cannot use H L, because H L is not necessary H L is used if I am using H, then we have got this move instruction move a comma M or move M comma B is.

So, like that that H L is much more generic in nature. So, that is already it that facility has already been provided. So, what the designers did is apart from H L. So, if you want

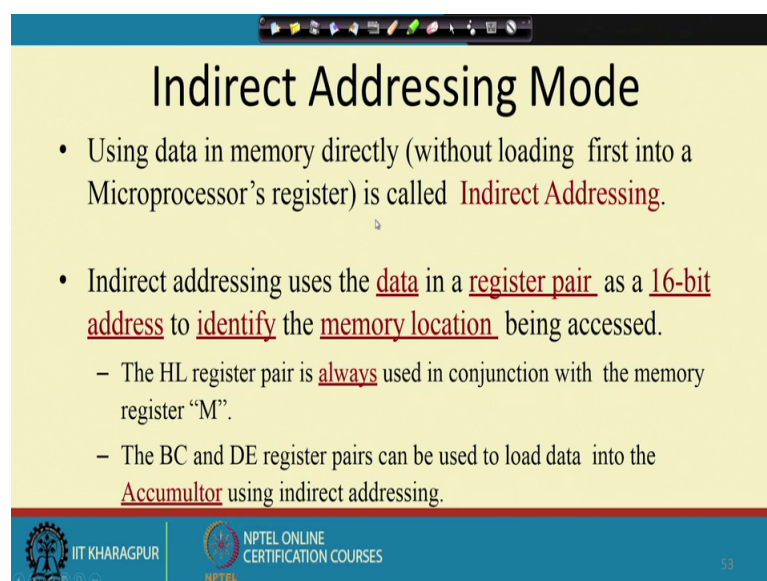
to use some other address also other point like. So, in the previous example; so, we have seen that if I want to have p as a pointer.

So, this p is addresses loaded into the H L pair now it may. So, happened that I want to do star p equal to star q, but p and q are 2 pointers. Now how to do this? So, this somehow this p ps address is loaded into H L suppose, this address is known to be 1000 and this qs address is known to be 2000. So, what you do you first load this H L pair with the value 1000 ok. So, this H L pair contents value of the 1000 and then you this for p, you do for this for this q part, we do LDAX say D comma no, this is not work.

So, first of all this register pair has to be loaded with the value so far that we do say MVI or LXI, LXI D comma 2000. So, LXI H comma 1000 loads the address of p on to H register LXI D comma 2000 loads the address of q on to the D register and then I can do LDAX LDAX d. So, this will give me the content of location 2000 that is star q on to a registered and then we can do move M comma A. So, that the content of that accumulator goes to the memory location.

So, this way, we can do this type of multiple pointers can be accommodated. So, this helps in the realizing the pointer arithmetic this pointer assignments. So, this way we can. So, we can use this register pairs for this memory access, we type of the pointer type of access. So, this LDAX is one instruction for doing it.

(Refer Slide Time: 03:59)



Indirect Addressing Mode

- Using data in memory directly (without loading first into a Microprocessor's register) is called **Indirect Addressing**.
- Indirect addressing uses the data in a register pair as a 16-bit address to identify the memory location being accessed.
 - The HL register pair is always used in conjunction with the memory register "M".
 - The BC and DE register pairs can be used to load data into the Accumulator using indirect addressing.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL | 53

Now, indirect addressing mode; so, using data in memory directly without loading first into microprocessors registered is the indirect addressing.

So, we do not load the address into some register and then do the loading of the load the address or content of that memory location. So, this indirect addressing it uses the data in a register pair as a 16 bit address to identify the memory location being access. So, whatever we were discussing so far that LXI that the move a comma M or that LDAX type of instruction.

So, they are actually using this indirect addressing this H L register pair is always use in conjunction with the memory register M and this BC and DE register pairs are used to load data into the accumulator using indirect addressing. So, in the previous example, we have seen that way that H L pair can be used for the accessing, the memory location accessing memory and in that case, we just tell M and for BC and DE. So, we have to use this LDAX and the accumulator has to be involved in the process.

(Refer Slide Time: 05:12)

Arithmetic Operations

- Addition (ADD, ADI):
 - Any 8-bit number.
 - The contents of a register.
 - The contents of a memory location.
- Can be added to the contents of the accumulator and the **result is stored in the accumulator.**
- Subtraction (SUB, SUI):
 - Any 8-bit number
 - The contents of a register
 - The contents of a memory location
- Can be subtracted **from** the contents of the accumulator. **The result is stored in the accumulator.**

Handwritten notes:
ADD B $\Rightarrow A \leftarrow A+B$
ADI 50H $\Rightarrow A \leftarrow A+50H$
ADD M $\Rightarrow A \leftarrow A+M[HL]$

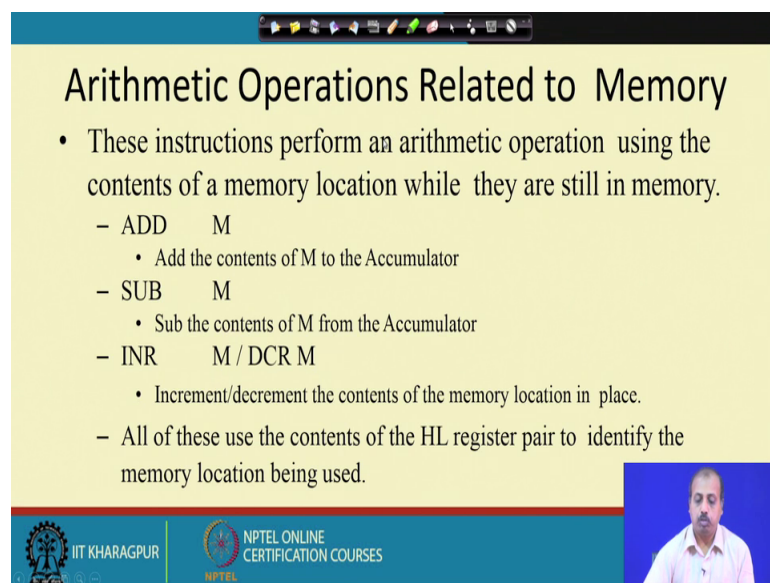
Logos: IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES

Next we look into the arithmetic operations. So, arithmetic operations like say addition subtraction. So, this addition operation we have got two mnemonics available in 8085; one is called add another is called ADI. So, add is any 8 bit number may be added the content of one register and are the contents of a memory location. So, you can use it, you can use an instruction like say add B as I said that this will do A gets A plus B or you can say like an 8 bit numbers. So, we can say like ADI sum 50 hex. So that means, A will get

A plus 50 hex or you can say like add M. So, where a will get the content of a plus memory location pointed 2 by the H L pair ok. So, that way we can have different types of addition operation. So, it can be added to the contents of the accumulator and the result is stored in the accumulator; so, that destination and first operand.

So, their all accumulator similarly, we have got subtract and subtract immediate. So, otherwise it is same. So, only it is instead of addition, it will do the subtraction operation the result and the first operand are the accumulator next.

(Refer Slide Time: 06:49)



Arithmetic Operations Related to Memory

- These instructions perform an arithmetic operation using the contents of a memory location while they are still in memory.
 - ADD M
 - Add the contents of M to the Accumulator
 - SUB M
 - Sub the contents of M from the Accumulator
 - INR M / DCR M
 - Increment/decrement the contents of the memory location in place.
 - All of these use the contents of the HL register pair to identify the memory location being used.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if you are they doing these arithmetic operations with respect to memory. So, we can do like add M. So, at the contents of M to the accumulator and the H L pair will give me the address. So, there is this I have already said. In fact, all these operations are add subtract this increment. So, all these operations can be done directly on memory locations by having the proper address in the H L pair.

(Refer Slide Time: 07:18)

The slide is titled "Arithmetic Operations" and discusses the Increment (INR) and Decrement (DCR) instructions. It lists two key points: that 8-bit contents of memory or registers can be directly incremented or decremented by 1, and that there is no need to disturb the accumulator. Handwritten notes include "DCR B", "INR B" (circled), and a sequence of instructions: "MOV A,B", "ADD 1", and "MOV B,A". The slide footer includes the IIT Kharagpur and NPTEL Online Certification Courses logos, and a small video inset of the presenter.

Then this increment and decrement operations; so, this is another class of arithmetic operation. So, this is the content of 8 bit register or memory locations can be implemented or decremented by 1, it is it does not need to disturb the contents of accumulator. So, I can simply have an operation like say DCR B or DCR B or INRB. So, like that.

So, if you would if you do not have this instruction and if you want to say increment this B by 1, then what you need to do is that move A comma B, then add ADI 1 and then move B comma A. So, this is the sequence that I have to do if I do not have the direct increment facility with the registers other than B other than a. So, this that is why this INR instruction. So, they are taken directly so that they do not involve the accumulator at all. So, there is no need to disturb the content of the accumulator. So, it is directly from the instruction.

(Refer Slide Time: 08:35)

The slide is titled "Manipulating Addresses" and contains the following content:

- Now that we have a 16-bit address in a register pair, how do we manipulate it?
 - It is possible to manipulate a 16-bit address stored in a register pair as one entity using some special instructions.
 - **INX Rp** The register pair is incremented or decremented as one entity. No need to worry about a carry from the lower 8-bits to the upper. It is taken care of automatically.
 - **DCX Rp**

Handwritten notes on the slide include "INX D" and a diagram of a 16-bit register pair. The diagram shows a box divided into two 8-bit sections, labeled (D) on the left and (E) on the right. An upward-pointing arrow is drawn between the two sections, indicating a carry or borrow operation.

The slide footer includes the IIT KHARAGPUR logo and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset of a speaker is visible in the bottom right corner.

So, now how can we manipulate the address ok? So, we have got 16 bit address in some register pair. Now sometimes we need to increment or decremented for example, if I have got an array that array is start address may be loaded array is loaded into say memory location. So, the memory start address is a thousand.

Now, the initially we said the H L pair to the beginning of the array by loading thousand on to it, but after the first item in the array has been processed we would like to take the H L pair to the next address ok. So, for that purpose; so, we have got this INX operation. So, INX; so, it will increment the register pair by 1 by 1 ok. So, that way it is possible to manipulate 16 bit address stored in a register pair as 1 entity and using some special instruction like INX R p the register pair will be incremented as one entity and the carry is already taken care of the good thing about this INX operation is that. So, the there; so, I have got this say if I have got say INX D.

So, this D; so what it will do. So, it will take the DE pair. So, this is D and this is E. So, it will do an increment, but if there is a carry generated. So, it is automatically taken care here. So, we do not need to in a program, we do not need to take care of it separately, but if you are used INRE operation, then the carry that is generated again, you have to take care for doing that thing. So, this INX instruction has been provided for that purpose and similarly DCX instruction.

So, that is for doing the decrement operation ok. So, we have got this INX and DC and DCX operations.



(Refer Slide Time: 10:32)

Logic Operations

- These instructions perform logic operations on the contents of the accumulator.
 - ANA, ANI, ORA, ORI, XRA and XRI
 - Source: Accumulator and
 - An 8-bit number
 - The contents of a register
 - The contents of a memory
 - Destination: Accumulator

ANA	R/M	AND Accumulator With Reg/Mem
ANI	#	AND Accumulator With an 8-bit number
ORA	R/M	OR Accumulator With Reg/Mem
ORI	#	OR Accumulator With an 8-bit number
XRA	R/M	XOR Accumulator With Reg/Mem
XRI	#	XOR Accumulator With an 8-bit number

$ANA\ B \Rightarrow A \leftarrow A\ AND\ B$
 $ANA\ M \Rightarrow A \leftarrow A\ AND\ M[HL]$
 $ANI\ 20H \Rightarrow A \leftarrow A\ AND\ 00100000$

Next we will see some logic operations. So, these instructions perform logic operations on the contents of the accumulator. So, you have got and accumulator then this and immediate, then or accumulator or immediate, then XOR accumulator and XOR immediate. So, these are the different operations that we have. So, and accumulator with register or memory; so, you can say; you can write instructions like say and accumulator B; so a will get A and B.

So, I can have this ANA M. So, in that case, accumulator will get A and memory location pointed to by H L pair. So, this way we can have the AND operation why we say or you can have that and immediate operation; so ANI some 8 bit value. So, it is say the 20 hex. So, what it will do? A will get A and with the 20 hex value that is 0 0 1 0 0 0 0 ok. So, this way we can have different types of operations and operations or operation XOR operation and destination is always the accumulated in the source is also on one of the source is also accumulator.

(Refer Slide Time: 12:13)

The slide is titled "Logic Operations" and lists the "Complement" operation. It states that the complement is the 1's complement of the contents of the accumulator and is performed by the CMA instruction with no operand. The slide includes logos for IIT Kharagpur and NPTEL Online Certification Courses, and a small video inset of the presenter.

Logic Operations

- Complement:
 - 1's complement of the contents of the accumulator.
 - CMA No operand

So, there is a complement operation. So, it gives the once complement of the content of the accumulator. So, all the bits of the accumulator are complemented and the instruction is CMA. So, it is a 0 operand instruction.

(Refer Slide Time: 12:35)

The slide is titled "Additional Logic Operations" and lists the "Rotate" operation. It describes rotating the accumulator one position left or right. It lists four instructions: RLC (Rotate Left), RAL (Rotate Left through Carry), RRC (Rotate Right), and RAR (Rotate Right through Carry). The slide includes logos for IIT Kharagpur and NPTEL Online Certification Courses, and a small video inset of the presenter.

Additional Logic Operations

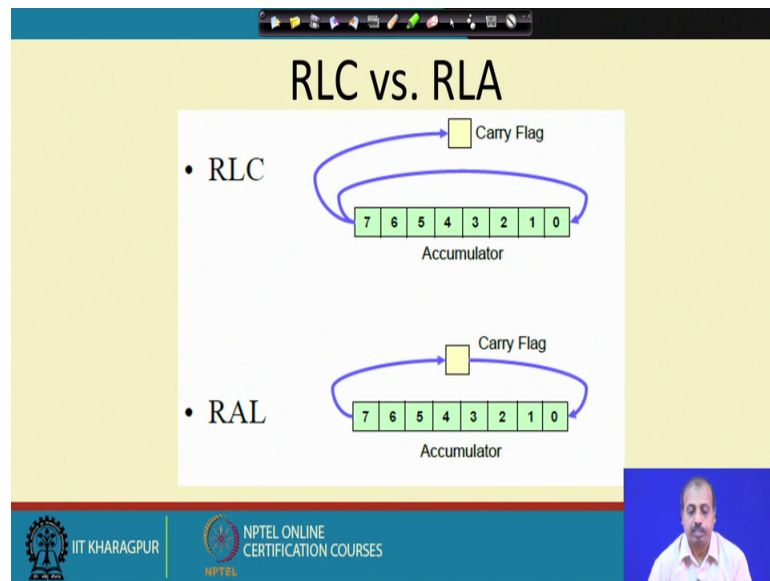
- Rotate
 - Rotate the contents of the accumulator one position to the left or right.
 - RLC Rotate the accumulator left. Bit 7 goes to bit 0 AND the Carry flag.
 - RAL Rotate the accumulator left through the carry. Bit 7 goes to the carry and carry goes to bit 0.
 - RRC Rotate the accumulator right. Bit 0 goes to bit 7 AND the Carry flag.
 - RAR Rotate the accumulator right through the carry. Bit 0 goes to the carry and carry goes to bit 7.

Because it implicitly means the content of the accumulator will be a flipped ok. So, that way it is a 0 operand instruction other logical operations that we have is the rotate is one category of instruction. So, it rotates the contents of accumulator one position to right or

left. So, RLC; so, it will rotate the accumulator left ok. So, bit 7 goes to bit 0 and the carry flag. So, it is rotate left through carry ok.

So, bit 0 bit 7 goes to the carry flag and this have a bits goes to the accumulator bit 7 goes to bit 0 and the carry flag.

(Refer Slide Time: 13:13)



I think there is some example after this RLC. So, this is what happens is that this bit 7, it goes to bit 0 and bit 7 also goes to the carry flag and this RAL instruction. So, this bit 7 goes to the carry flag and the carry flag comes to bit 0. So, this in here this carry flag also becomes a part of the accumulator as, and as if the rotation is done by taking this carry also carry bit also a part of the accumulator on the other hand this RLC.

So, this is not taking this carry bit as a part of the accumulator though bit 7 is transfer to the carry flag, but this carry flag content is lost that way. So, this in the after the RLC instruction the carry flag content will be lost, but in case of RAL nothing is lost. So, the bit 7 goes is now stored in the carry flag and the previous carry bit is available in bit 0. So, all other bits are shifted ok. So, we have got the RLC instruction that way. So, this similarly we have got RRC and RAR. So, that is the rotate right. So, it will be rotated from the right end that bit 0 goes to bit 7 and carry flag and in case of RAR bit 0 goes to the carry flag and the carry goes to bit 7. So, they are there.

(Refer Slide Time: 14:32)

The slide is titled "Logical Operations" and features a list of instructions and their descriptions. Handwritten notes in black ink are present on the right side of the slide. A small video inset of a man is visible in the bottom right corner.

Logical Operations

- Compare
 - Compare the contents of a register or memory location with the contents of the accumulator.
 - CMP R/M Compare the contents of the register or memory location to the contents of the accumulator.
 - CPI # Compare the 8-bit number to the contents of the accumulator.
 - The compare instruction sets the flags (Z, Cy, and S).
 - The compare is done using an internal subtraction that does not change the contents of the accumulator.

$A - (R / M / \#)$

Handwritten notes on the right side of the slide:

- $A - B$
- CMP B
- $A \leftrightarrow B$
- CMP M
- $A \leftrightarrow M[HL]$
- CPI 20H
- $A \leftrightarrow 20H$

Logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES are visible at the bottom of the slide.

So, we have got the logical operation. So, in case of logical operations we compare the contents of a register or memory location with the contents of the accumulator. So, CMP is the instruction. So, we can talk about CMP R register or memory. So, we can have instructions like say compare B. So, so, A and B register. So, they will be compared or I can say like compare M. So, the A and this memory location HL; so, they will be compared A and memory location H L they will be compared. So, this way we can have this compare instruction then we can have this compare immediate, so CPI so that compare immediate. So, we can have an instruction like CPI say twenty hex. So, here a register will be compared with the immediate value 20 hex ok.

So, it compares the 8 bit value with the accumulator. So, the compare instruction it sets the flags like 0 flag carry flag and sign flag. So, they are they are set they are this compare instruction will set them. So, if the 2 contents are same, then this 0 flag will be set ok. So, if 1 is more than the other than the carry flag will be set or the sign flag will be set. So, comparison is done using an internal subtraction that does not change the contents of the accumulator.

So, internally will be doing a whether processor does A minus B. So, if it is compare B. So, it will be doing A minus B and depending upon the result. So, if A and B are same then the result will be 0 in that case, the 0 flag will be set if A is greater than B, if says if A is greater than B, then this carry flag will be set and if A is less than B, then the sign

flag will become negative and the sign the sign operation result is negative and the sign flag will be set.

So, after this if we check this 0 carry and sign flag. So, we can understand; what was the outcome of the comparison. So, accordingly we can say it is greater than less than or equal to ok. So, after doing this comparison we can put that type of condition checks and that is done by looking into the status flags.

(Refer Slide Time: 17:02)

Branch Operations

- Two types:
 - Unconditional branch.
 - Go to a new location no matter what.
 - Conditional branch.
 - Go to a new location if the condition is true.

Handwritten notes on slide:
CMP B
JE 2000H
2000
1000
JMP 2000
2000
PC ← 2000

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next will be looking into another category of instructions which are known as branch instructions; so, there are this branch categorization. So, you can broadly divided two categories the unconditional branch and conditional branch. So, in case of unconditional branch; so, it will tell the when this instruction is executed. So, the processor is told that you go to the new address irrespective of what was the previous value like if it is a. So, suppose at present the processor was executing instructions in memory.

So, this is the this is the memory and say it is executing the instruction at location thousand and here the instruction is jump to 2000 in that case whatever be the content of the next instruction. So, if the processor will simply ignore that and it will start executing the instruction from location 2000 it will go to the location 2000 and it will start executing from this point onwards ok.

So, that is why it is called an unconditional branch. So, there is no condition involved in the process ok. So, what the processor internally does is that after getting this instruction and after decoding this instruction the simply the program counter gets loaded with the value 2000. So, that is why the next instruction executed is from location 2000. So, there is a unconditional branch and there is a conditional branch also where you some condition code is true, then only it will be going to that address for example, if I have got an instruction like compare B and then if I say the jump on equal jump on equal to 2000.

So, if this after in this comparison if A and B register values were same then it will be jump on equal condition will be true the 0 flag will be set. So, equality condition is true. So, in that case only; so, it will be jumping to the location 2000 otherwise, it will continue from here if the condition is valid, then it will start executing from this point if the if A and B are not same, in that case, it will execute the very next instruction and proceed like that. So, that is the conditional branch and previously what we saw that is an unconditional branch.

(Refer Slide Time: 19:39)

The slide is titled "Unconditional Branch" and contains the following text:

- JMP Address
 - Jump to the address specified (Go to).
- CALL Address
 - Jump to the address specified but treat it as a subroutine.
- RET
 - Return from a subroutine.

At the bottom, it states: "The addresses supplied to all branch operations must be 16-bits."

The diagram on the right shows a vertical stack with addresses 500, 1000, and 1001. The instruction at 1000 is "CALL 2000". An arrow labeled "PC" points to the instruction at 1001. Below the stack, there are handwritten notes: "PC → STACK", "PC ← 2000", and "RET: PC ← STACK".

The slide footer includes the IIT KHARAGPUR logo and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset of a man is visible in the bottom right corner.

So, next will be so, this is an unconditional branch type of instruction like jump followed by address. So, this is the address is a 16 bit address. So, it jump to the address specified to something like A go to that we have in some high level programming language. So, there is another unconditional branch which is known as the call instruction. So, call

instruction is similar to the procedure call that we have in high level languages. So, here call address call followed by a 16 bit address.

So, this is this new 16 bit address is a subroutine. So, so it is like this. So, suppose this is the main program that it was executing and then my this is this is the loaded from say memory location five hundred on words. So, when it came to the memory location thousand ok. So, it is calling a subroutine. So, this is a subroutine call 2000. So, the subroutine is a small routine. So, which is loaded from location 2000; so, it starts at location 2000. So, what the processor will do when it gets this called 2000. So, it will jump to the location 2000 and it will start executing from this point onwards.

So, as it is executing when this subroutine is over. So, at the end of the subroutine there will be a return instruction and when this return instruction is encountered then the control will be going back to this point that is 1001. So, control will be going back to the location 1001 ok. So, this is the way the call and return instructions are executed. So, return from subroutine and going into subroutine now this is exactly the point where these stack come into picture like when it was going to this subroutine. So, in the stack it had set this return value 1001 and when it came to this return statement. So, it popped out this value from the stack 1001 and loaded it into the PC.

So, while going from while coming from this call 2000 to this; so, it set the content of the program counter which was equal to 1001 into the plus stack. So, the PC value was this PC value was copied on to the stack and then PC is loaded with the value 2000 and after sometime when the return instruction is encountered then at that time the PC value is loaded from the stack.

So, that it goes back to this 1001 this is how this return statement and this call statement they are executed. So, they are also some sort of branching. So, they are all unconditional branching that we have now other instructions that are possible.

(Refer Slide Time: 22:36)

Conditional Branch

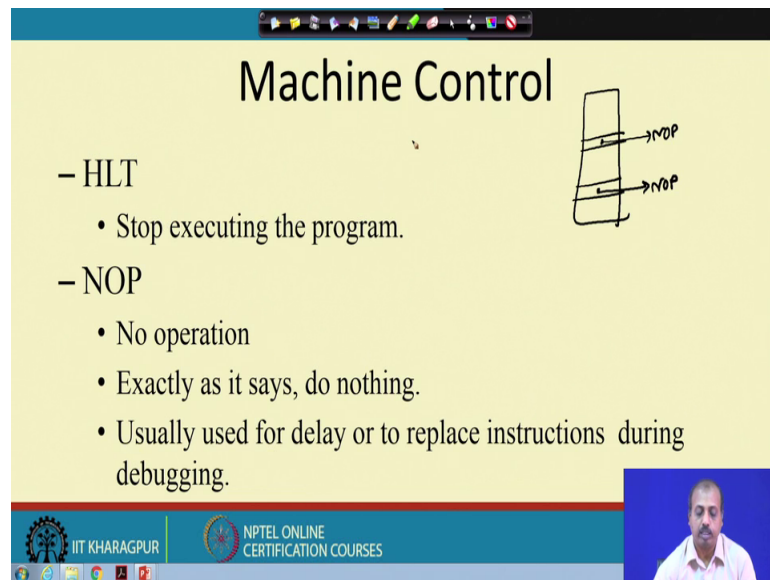
- Go to new location if a specified condition is met.
 - JZ Address (Jump on Zero)
 - Go to address specified if the **Zero flag is set.**
 - JNZ Address (Jump on NOT Zero)
 - Go to address specified if the **Zero flag is not set.**
 - JC Address (Jump on Carry)
 - Go to the address specified if the **Carry flag is set.**
 - JNC Address (Jump on No Carry)
 - Go to the address specified if the **Carry flag is not set.**
 - JP Address (Jump on Plus)
 - Go to the address specified if the **Sign flag is not set**
 - JM Address (Jump on Minus)
 - Go to the address specified if the **Sign flag is set.**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, these are these are the conditional branch. So, go to new location if some specified condition is made. So, JZ jump on 0. So, go to address specified if the 0 flag is set similarly. So, 0 flag will be set as we know that if the value the previous comparison was equal with similarly, JNZ jump on not 0. So, this jump on not 0, it will be true if the address is if the condition if the values were not same. So, that is non zero jump on carry. So, this is the value of comparison first operand was larger.

So, that is the carry flag set. So, jump on carry. So, similarly jump on no carry jump on plus ok. So, it the sign flag is not set and jump on minus if the sign flag is set. So, these are the various jump condition conditional jumps that we have in case of 8085 ok; so, using the carry 0 and sign flags.

(Refer Slide Time: 23:38)



The slide is titled "Machine Control" and contains the following text:

- HLT
 - Stop executing the program.
- NOP
 - No operation
 - Exactly as it says, do nothing.
 - Usually used for delay or to replace instructions during debugging.

To the right of the text is a hand-drawn diagram of a stack, represented as a vertical rectangle with horizontal lines. Two arrows point from the stack to the label "NOP".

At the bottom of the slide, there is a blue banner with the IIT KHARAGPUR logo on the left and the NPTEL ONLINE CERTIFICATION COURSES logo on the right. A small video inset of a man is visible in the bottom right corner.

Some machine control instructions like halt. So, it will stop the execution of the program. So, this I have told some earlier. So, halt will halt the execution of this statements by the processor and it will be waiting till the processor is the waken up by means of some interrupt or some external procedure.

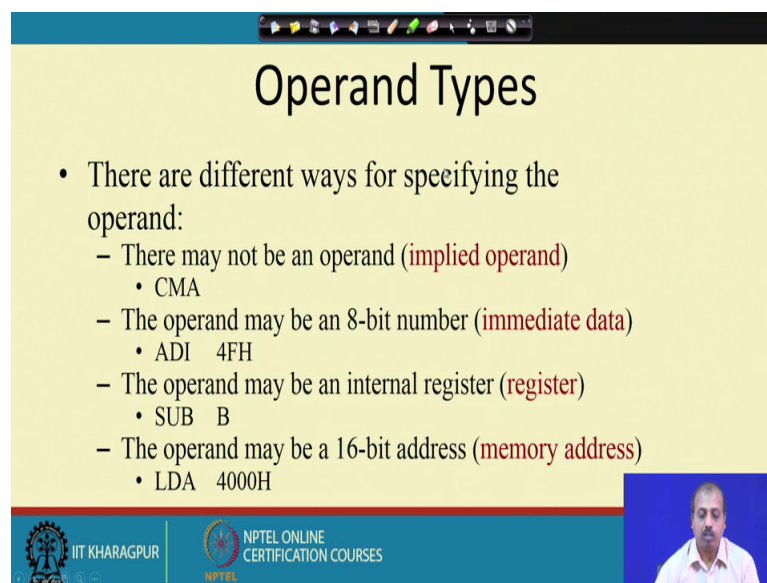
Then another instruction which is there is known as the no operation or NOP instruction. So, it as the name suggests; so NOP is do nothing. So, it does nothing. So, apparently since why do in need this thing. So, as it is not doing anything so, but this instruction is sometimes useful because you want to delay you want to introduce some delay into a sequence of operation for example, it may so happen that you want they the next instruction to be executed after a small delay of some fractions. So, you can put if you NOP statement. So, that some small delay gets introduced into the system.

So, that is one utility another utility of this NOP instruction is for debugging. So, so, when you are developing a program. So, it may have some bugs. So, what we do we put some debugging statements in between in the in the develop program. So, that we can halt we can we can stop the execution of the program in between do some checking of this register contents and all and then resume the instruction. So, what happens is that if this is the program then in between we put some special instruction that help us in the debugging operation. Now after the debugging is over now what is done is this instructions which the basically some sort of interrupt.

So, they are modified to NOP. So, that in the final version there is no debugging instruction included. So, they are all NOP instruction. So, this so, we can very easily modified this debugging instructions by NOP instruction. So, without hampering the code which is otherwise generated; so, all the addresses and all they remain unchanged and only this part they are modified to NOP.



So, this way we can have some machine control instructions with the halt NOP etcetera, so that are going to be helpful.

(Refer Slide Time: 25:59)



Operand Types

- There are different ways for specifying the operand:
 - There may not be an operand (**implied operand**)
 - CMA
 - The operand may be an 8-bit number (**immediate data**)
 - ADI 4FH
 - The operand may be an internal register (**register**)
 - SUB B
 - The operand may be a 16-bit address (**memory address**)
 - LDA 4000H

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

Now, depending upon the operand so that there can be different types of operands therefore, example with the implied operand like CMA. So, this instruction is 0 operand instruction the operand the implied operand is the memory we have got this 8 bit number the immediate data. So, like ADI; so, ADI this is ADI 4FH. So, this is basically this addition operation with the immediate operand then we can have the some internal register or register operand. So, subtract B. So, the B is the register or it may maybe a memory address. So, this is LDA 4000 hex content of memory may be a four thousand hex will be loaded. So, so this way, I can have different types of operand implied operand immediate operand register operand and memory address.

(Refer Slide Time: 26:50)

Instruction Size

- Depending on the operand type, the instruction may have different sizes. It will occupy a different number of memory bytes.
 - Typically, all instructions occupy **one byte** only.
 - The exception is any instruction that contains **immediate data** or a **memory address**.
 - Instructions that include immediate data use **two bytes**.
 - One for the opcode and the other for the 8-bit data.
 - Instructions that include a memory address occupy **three bytes**.
 - One for the opcode, and the other two for the 16-bit address.

The diagram shows three instructions with their bit lengths: CMA is 8 bits; MVI A 40H is 8 bits for the opcode and 8 bits for the data; LXI H is 8 bits for the opcode and 16 bits for the address.

Now, depending upon the operand type; or, the instruction size will vary some of the instructions are only one byte long somewhere more number of bytes. So, for example, the instruction CMA that we have; so, CMA will there is an 8 bit code for that as we know that all opcodes are 8 bit. So, it is only one byte long on the other hand. So, if I take this instruction like M B I A comma 40 hex, then this for this M B I A part.

So, I need to hold there is a code for these parts. So, that this is an 8 bit code and then this 40 hex is another 8 bit number. So, total I will required 2 bytes. So, for holding this M B I A instruction or if I have this LXI H comma A 16 bit value, then this LXI h parts. So, this will constitute the first byte and the next this 16 bit part.

So, they will be taking two bytes ok. So, total instruction is three byte long. So, in case of 8085 some of the instructions are one byte long some of them are 2 bytes long some of them are 3 bytes long so, the instructions which are which are not having any operand. So, they are 0 operand instruction, they occupy only 1 byte some instructions, they are some immediate data or a memory address. So, they will be having for immediate data that is the 2 byte instruction and if you have got a memory of a memory address so that is a 3 byte instruction.