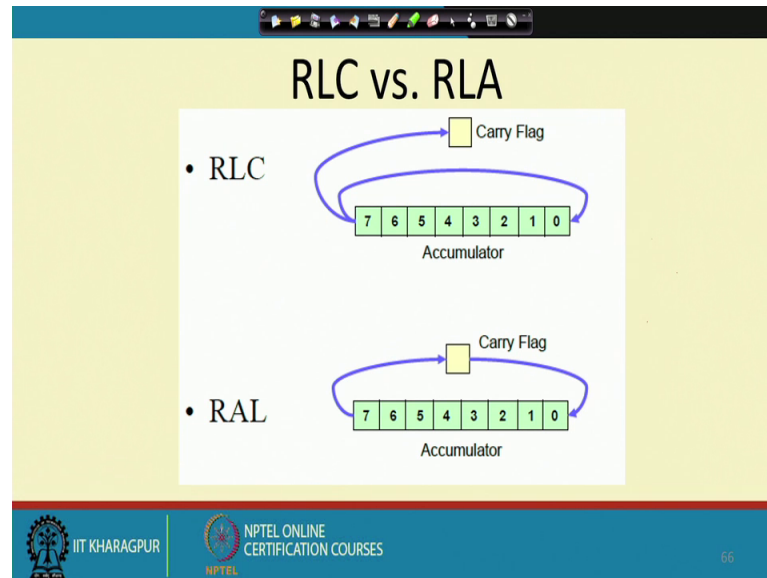


Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 12
8085 Microprocessors (Contd.)

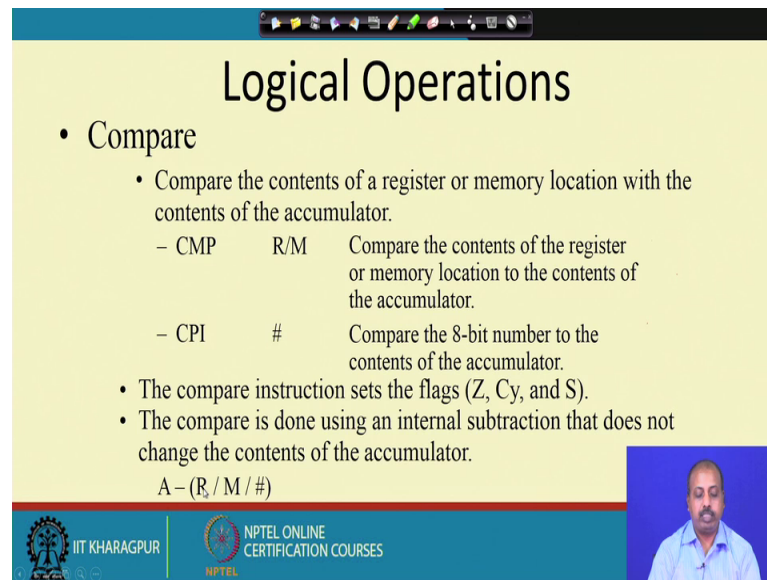
(Refer Slide Time: 00:18)



So, to compare between RLC and RLA so, this diagram will tell that. So, what I was telling that accumulator. So, this is that 8 bit. So, when you are rotating left. So, this is the rotation to the left rotation. Now bit number 7 it will go to bit number 0, as well as it will go to the carry flag of the status register. So, both of them will get modified and all this bit 0 will come to bit 1, bit 1 will come to bit 2. So, entire thing is rotated left like this.

On the other hand, this RAL so, this is also rotate left, but now this bit 7 will go to the carry flag and the carry will come to bit 0. So, this is a rotation through the carry. So, that is that is why it is called rotate through carry. So, carry is also rotated as part of it.

(Refer Slide Time: 01:04)



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.

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, other than that we have got some or more logic operations like compare. So, you can compare content of 2 registers or memory locations with the content of one register or memory location with that of accumulator. So, like look at the general format is compare CMP a register or memory R or M. So, the contents of the register or memory location will be compared with the content of the accumulator. And also, there is an immediate version like CPI so, compare immediate.

So, here you need to specify one 8-bit number, and the content will be compared the content of the accumulator will be compared with this 8-bit number. So, this is compare instruction it will set the flags the z carry and sign. So, the 0-flag carry flag and the sign flag will be affected. So, if the numbers are same, then this 0 flag will be set; if the comparison is done using this internal subtraction and does not change the content of the accumulator. So, it will do this subtraction a minus this is what is the second operand R by register or memory or immediate value whatever you are specified. So, this based on that this operation will be this operation will be carried out and accordingly these 3 flags will be set.

So, if the result is 0, then this 0 flag will be set that is if these 2 values are same. If the if a carry is generated; that means, this if this R by m is larger than this carrying will be generated. So, in that case this carry flag will be set, and the sign flag will be positive or negative depending on the result whether is positive or negative.

(Refer Slide Time: 02:48)

The slide is titled "Branch Operations" and lists two types of branches:

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

Hand-drawn diagram on the right shows a sequence of instructions. The first instruction is at address 1000. A red arrow points from the instruction at 1000 to another instruction at address 2000, illustrating a branch. The instruction at 1001 is shown with a red 'X' over it, indicating it is not executed.

The slide footer includes the IIT Kharagpur logo and the NPTEL Online Certification Courses logo. A small video inset in the bottom right corner shows a man speaking.

So, it will do that. Then we have got another class of operation which is this branch operation. And branch operation so, they are actually for transferring control from one memory location to some other memory location. That is the at present the program is executing at some point in my in the system. And so, suppose these are the program lines.

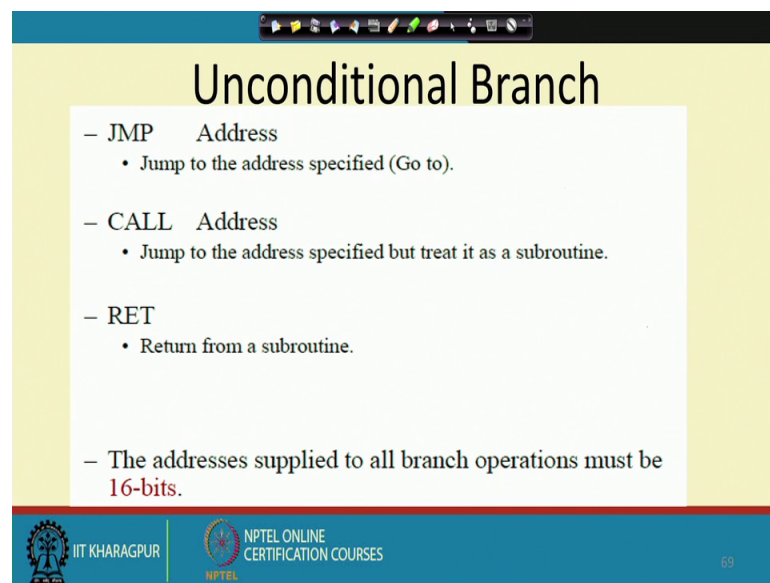
Now, at present the system is executing this particular instruction then it is it is expected, that the next instruction will be this one. But suppose this is the location say 1000 and this is the location 1001. So, after 1000 instructional 1000 is executed the instruction 1001 will be executed. But due to some reason so, if I if I think that the next instruction to be executed is location is at location 2000s. Then from here this instruction should transfer the control from if instead of 1001 it should transfer the control to 2000. Or sometime later so, if you are trying to implement a loop then it may so happen that this is the body of the loop at the end of the body of the loop. So, you need to do the loop again. So, you need to jump back to this position.

So, this way we need these branch instructions for doing these for implementing many of the control transfers. So, these branch instructions are used and there are 2 types of branch instructions one is called this one is called this unconditional branch, where the branching is independent of the outcome of previous operation. And there is a

conditional branch it is this branch will be taken if the previous operation it evaluated to some condition being true.

So, unconditional branch is go to a new location no matter what like whatever was the outcome of the previous instruction, it will branch to the new address, and things like that.

(Refer Slide Time: 04:55)



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.
- The addresses supplied to all branch operations must be 16-bits.

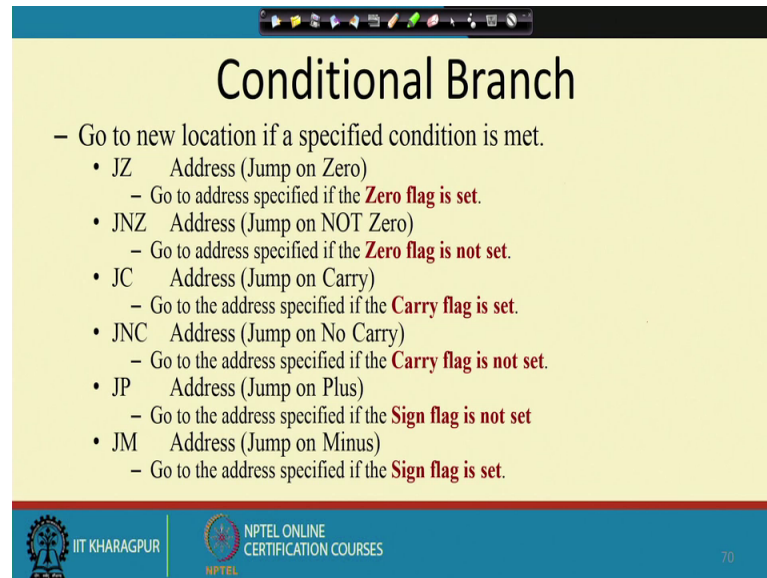
The slide footer includes the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the number "69".

So, this unconditional branch instruction; so this is of the format the mnemonic is JMP, jump and this address. So, address is a 16-bit address, because 8085 supports 16-bit address. So, jump to a 16-bit address. So, when this instruction will be executed, I hope you can guess what is going to happen, the program counter will be loaded with the address that is specified here so that the next instruction that will be executed is the from the address loaded into the program counter.

So, here you see from the instruction register, the value should go to the program counter. So, that the program counter gets the jump address. Another instruction is call instruction. So, that is for subroutine. That is, you want to execute a sub program. So, that is for call address. And there is a rate the return instructions to return from a subroutine. So, all these addresses supplied the address supplied to all branch operations must be 16 bit. So, they are all 16-bit addresses.

So, at the end of the subroutine I should have a return instruction. So, that will be returning from the subroutine. And this call instruction is for starting a subroutine.

(Refer Slide Time: 06:21)



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.**

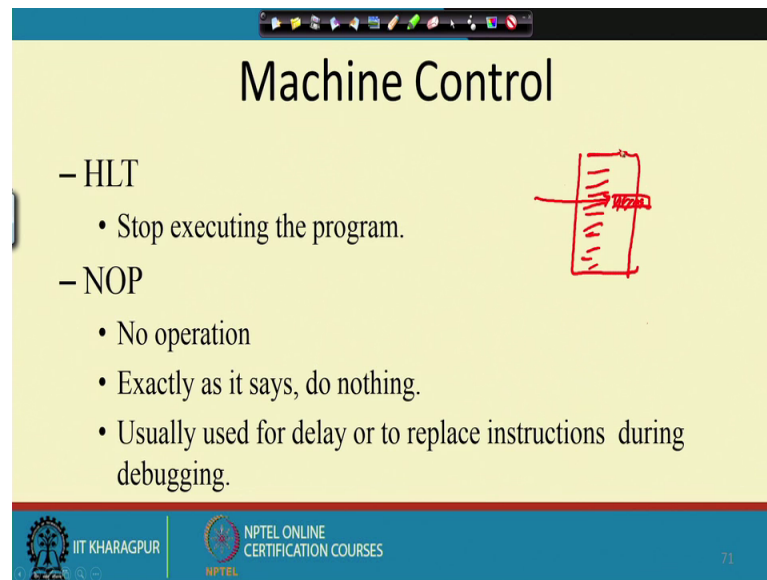
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 70

About the conditional branch so, we have got a number of conditional branch instructions like JZ. So, jump on 0 so, if due to the previous arithmetic logic operation, the 0 flag has been set, then it will jump to the next address that is mentioned. So, JZ address so, it will jump to the specified address if the 0 flag is set.

Similarly, we have got JNZ. So, it will jump if the 0 flag is not set. The 0 flag is 0, then it will be not of it will be jumping at this. Then jump on carry. So, if the previous operation did not previous operation had generated a carry then this JC address. So, this will be jumping to the new address specified by this specified in this instruction, JNC is the jump on no carry then JP is jump on positive. So, if the sign flag is not set then the result was positive. So, we have got the JP instruction for jumping on the result being positive and JM is jump on minus if the previous operation resulted in sign flag being safe the result was negative, then this JM address will jump to the new address.


So, this way we can have these conditional branch instructions in 8085.



(Refer Slide Time: 07:42)



Machine Control

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



71

Machine control instructions; so we have got halt instruction to stop executing the program. So, over this this is a special instruction when this halt is executed then the whole operation of the system will come to a stall. So, this is the system will be. So, no more no more operation will be done by the processor to get the next instruction and all, only thing is that you can say that interrupts to the processor now to take it out of this halted situation. And there is a no operation another very interesting instruction that we have is the NOP or NOB. So, that stands for no operation.

So, what does it do it does it does not do anything. So, it is used mainly for in delay. So, if you want to put some small delay. So, you can put this NOP instruction. And many times, for the program debugging this NOP becomes useful. So, why suppose I have I have written some program. So, this is sorry suppose I have written a program in which we have got. So, this is the program I have written and in this program, I have this program is somehow not working correctly I feel that the program is not working correctly.

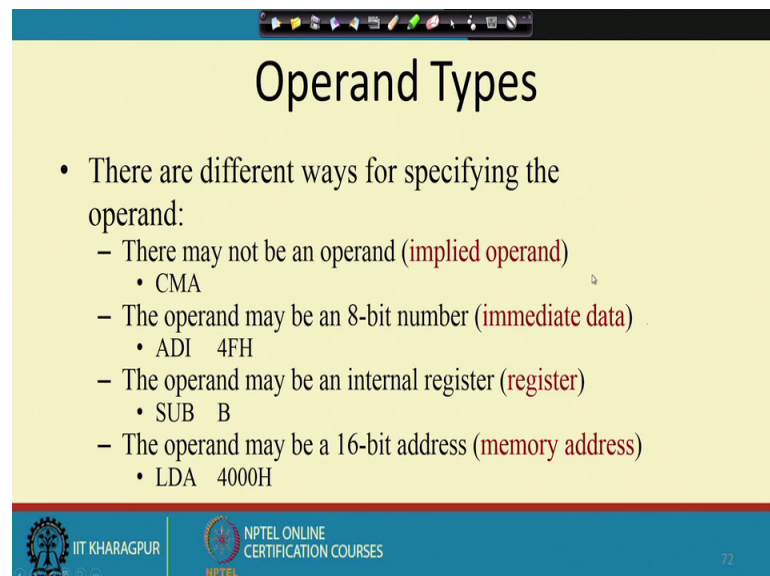
So, what I need to be do is that suspend the operation of the program and somewhere in between. So, what we do we introduce some sort of special instructions which are useful for debugging fine. So, and after that if you execute the program now what will the what the system will do is that when it reaches this point, the debugger will get invoked and

accordingly you can modify the you can check the content of CPU registers or modify them and all those things.

Now, while doing that so, you have incorporated these instructions into the program and this is use this is done at the time of compiling the program itself, or assembling the program itself. Now later on when the program is working properly, then what to do with these instructions. So, if you if you want to remove this instruction means you need to reassemble the program by removing all those places.

So, a better option is later on you may also need this debugging. So, you just replace these instructions by NOP instruction, and NOP instruction code is 0 0. So, you can replace all these locations by 0 0, telling that it is a NOP instruction. So, that is useful that is so forth. So, if you have got some instructions introduced during debugging so, you can later on remove them using this NOP type of instructions. And many of the debuggers have got these facilities. So, though we are in a basic course on microprocessor, so we may not be using that very much, but later on while working is some on some detail larger programs. So, you may need to use this debugger and there this facility may be useful.

(Refer Slide Time: 10:54)



The slide is titled "Operand Types" and lists four ways to specify an operand:

- 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

The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and the number 72.

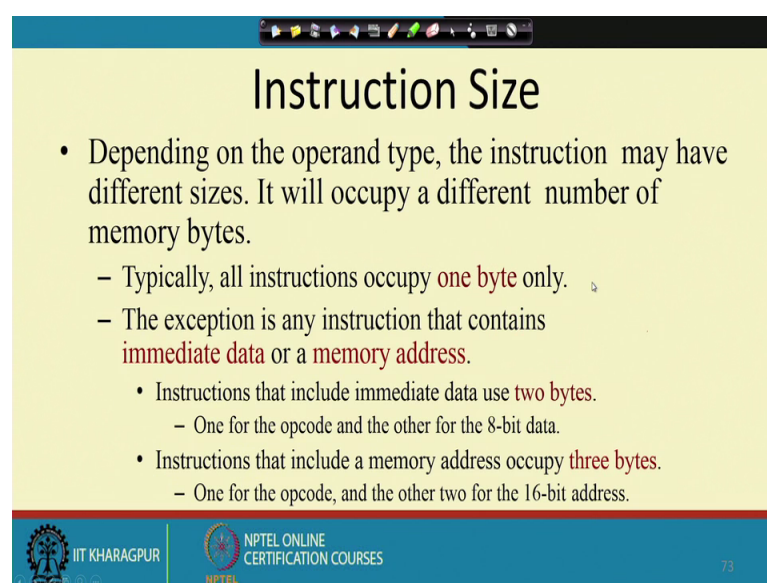
Another way of looking into the program looking into the instruction is the type of operands that are supported.

Now, some of the instructions they are they do not have any operand. So, a operand is implied like this CMA the complement accumulator instruction. So, here the operand is the accumulator, but we do not need to specify it separately. So, that CMA is a 0-operand instruction or implied operand instruction then there some of the operands are immediate in nature like immediate data like say ADI add immediate 4F hex. So, we are telling that the accumulator should be added to the accumulator should be added with the number 4F hex and the result should be stored in the accumulator.

So, here I am talking about some immediate operands that is called immediate data. Or we can have some internal register like subtract B sub b. So, we are subtracting the register B from register A, and that is another internal register so, the register is an operand. So, we have got implied operand, we have got immediate operand, we have got register operand, and also, we have got memory address as operand like LDA 4000 hex or the accumulator will be loaded with the content of memory location 4000. So, here this 4000 hex is a memory address that we need to load.

So, this way we can think about we can talk about the type of operands that this 8085 instruction set has, and for that matter any processor design that we look into. So, we have to look in we can we can characterize or can categorize these instructions based on the operand types.

(Refer Slide Time: 12:44)



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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL | 73

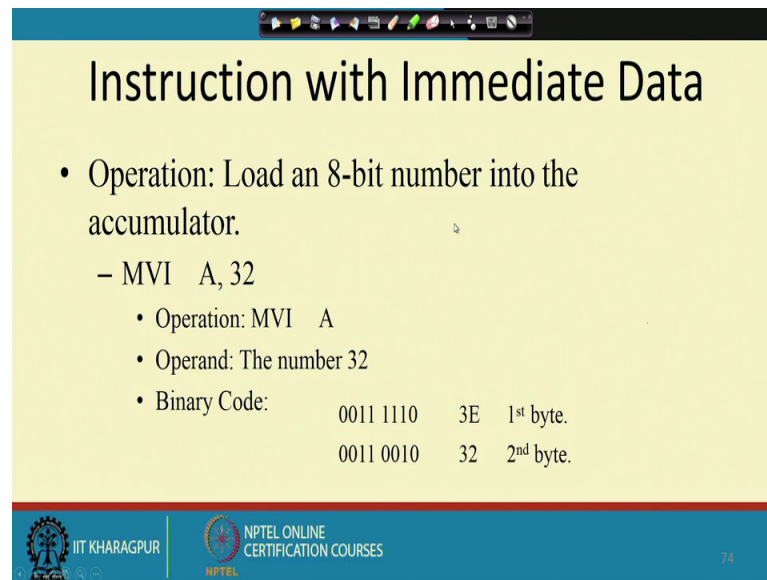
Size of the instruction; like so far, we have we have assumed that all instructions they are of same size, but does not it look a bit odd. For example, if I just go back, and say this instruction is CMA, I do not need to specify anything after this and this instruction is ADI 4F hex.

So, I have to somehow tell the number 4F. So, if I say that certain numbers of bits are dedicated for storing this opcode part. So, that is true for CMA that is true for ADI that is true for sub that is for LDA. So, those bits are already taken up. Now this 4F has to be stored separately this B has to be stored separately this 4000 has to be stored separately. So, if I give say this this instruction 8 bit for coding, then this CMA has got an 8-bit code, then ADI 4F hex it cannot be an 8-bit instruction. Because this ADI part itself will take 8 bit and this 4F will take another 8 bit. So, this is expected to be 2-byte instruction or 16-bit instruction.

Similarly, these LDA so, apart from this code of this LDA; so I need to specify a 16-bit number 4000. So, it is all likelihood. So, this will be a 3-byte instruction the size of the instructions are not same. Depending upon the type of the operand the size of the instruction may be different. So, that is what we are going to look into next. So, depending upon the operand type the instruction may have different sizes and as a result it will occupy different number of memory bytes. Typically, all instructions occupy 1 byte only. Because that is a opcode part has to be coded and the opcode should be 8 bit. And where so, the exception is that with the instruction that contains immediate data or a memory address. For immediate data we have to have another 8-bit value.

So, that is why it will be the instruction will be 2-byte instruction, or the second byte will be holding the immediate value. And in case of 3-byte instruction the memory address. So, the instruction has to be 3 byte wide, because the next 2 bytes will be holding the 16-bit address. So, that way it has to be a 3 byte instruction. So, instruction size will depend on the number of operands that we have in the instruction.

(Refer Slide Time: 15:12)



Instruction with Immediate Data

- Operation: Load an 8-bit number into the accumulator.
 - MVI A, 32
 - Operation: MVI A
 - Operand: The number 32
 - Binary Code:

0011 1110	3E	1 st byte.
0011 0010	32	2 nd byte.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 74

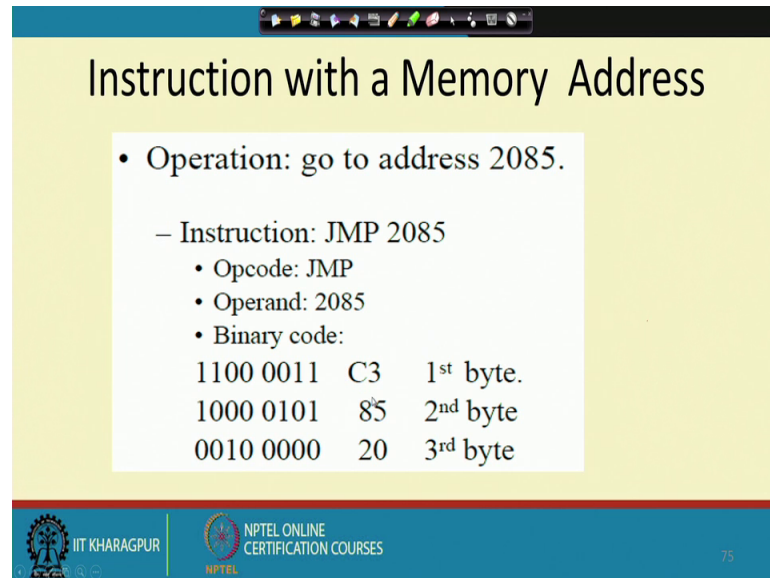
So, let us look into some example. So, load an 8-bit number into the accumulator. So, the instruction is MVI A comma 32. So, what is the operation? Operation is MVI A. So, MVI A since this is the operation, I can allocate one of those 246 codes that 8085 supports 2 MVI A; so processor designers they have given this some code to this MVI A, and that code is 3E. So, 3E is the code for this MVI A. So, the next operand is the number 32. So, this 32 is the next operand, and it is taken as hexadecimal number. So, this is the this is the hexadecimal number 32 that is the second byte.

So, these are so, these MVI A comma 32 will be represented by a 2-byte instruction, at the first byte will be 3E second byte will be 32. So, how the processor will execute this instruction? So, when the pc will put the memory address onto the address bus and give the ale signal. So, this 3E value will come to the data bus and it will reach the instruction register. So, that will go to the instruction decoder by looking into this 3E, the processor the instruction decoder will understand that this is an MVI instruction. So, another byte has to be fetched from the memory.

So now it will again put a read memory read request onto the onto the memory to the memory, and with the address already incremented by one whenever this location has been access the program counter has already been incremented by 1. Now program counter points the next memory location which contains the value 32. So, memory read is done and the memory will put the value 32 on the data bus. So, this will again come to

the instruction register and from there this 32 will go finally, to the accumulator during execution cycle. So, this is how this operation will take place this MVI A comma 32 operation.

(Refer Slide Time: 17:23)



The slide is titled "Instruction with a Memory Address". It contains the following text:

- Operation: go to address 2085.
- Instruction: JMP 2085
 - Opcode: JMP
 - Operand: 2085
 - Binary code:

1100 0011	C3	1 st byte.
1000 0101	85	2 nd byte
0010 0000	20	3 rd byte

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and the number 75.

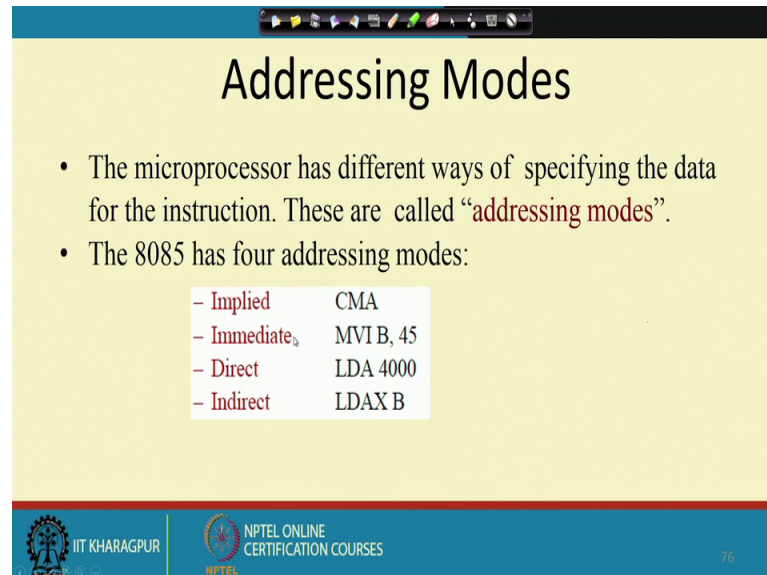
Another instruction example you say go to address 2085, and the instruction is jump 2085 JMP 2085, opcode is JMP operand is 2085 and the code for JMP is C3. So, the first byte will be C3 second byte will be holding 85, and a third byte will hold 20. Why is it like this? Because Intel follows the convention that for whenever we have got memory words whenever you have got words of more than 1-byte width, then the lower order byte should go to the lower order address higher order byte should go to the higher order address.

Now, if this is the first location; say this is the location 1000, then the next location is 1001 next location is 1002. Now when I am storing this 2085; so 85 is the lower order byte that should go to the lower order address that is 1000 to 1, and this higher order byte 20, it should go to the next higher order address that is 1002. So, thousand memory location 1000 will get C3 1000 to 1 will get 85 a 85 and 1002 will get 20. This is how this coding is done.

So, if you are doing trying out some hand assembly, or if you are looking into the assembled file generated by any assembler, you will see these things that the higher order byte has been put in the higher order address space for this thing. But it is not uniform

like say some other processor may do it in other way, but 8085 does it in this fashion the higher order byte goes to higher order address.

(Refer Slide Time: 19:11)



The slide is titled "Addressing Modes" and contains the following text:

- The microprocessor has different ways of specifying the data for the instruction. These are called “addressing modes”.
- The 8085 has four addressing modes:

- Implied	CMA
- Immediate	MVI B, 45
- Direct	LDA 4000
- Indirect	LDAX B

The slide footer includes the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES, and the number 76.

Now, there are different ways by which you can give the data of an instruction. Like these are called addressing modes. So, addressing modes means the how many ways I can tell the operand for an instruction. So, 8085 has got 4 different addressing modes. They are called implied that is we do not have any other operand separately. So, the instruction itself tells what is the mode, then immediate. So, immediate the some 8-bit immediate value is given, direct access. So, that the address is mentioned directly or there is indirect memory access which is all indirect addressing mode which is giving some say pointer sort of thing.

So, the one is direct that the address value is specified in the instruction itself, and in the indirect address the address part is not directly mentioned in the instruction, but whatever operand is told in the instruction that will that will contain the address like LDAX B. So, the BC pair will hold the 16-bit address that can that has to be used for getting the content of memory location to be loaded into the accumulator.

So, this is how these addressing modes are supported in 8085. So, if you look into more complex processors you will find that these there are many other types of addressing modes that are supported, but 8085 being a very simple processor. So, it does not have more complex modes.

(Refer Slide Time: 20:48)

The slide is titled "Data Formats" and contains the following text:

- In an 8-bit microprocessor, data can be represented in one of four formats:
 - ASCII
 - BCD
 - Signed Integer
 - Unsigned Integer.
- It is important to recognize that the microprocessor deals with 0's and 1's.
 - It deals with values as strings of bits.
 - It is the job of the user to add a meaning to these strings.

Handwritten notes in red ink on the slide include the number "25" at the top right, the word "off" with a horizontal line through it, and the binary representation "00010001" with a vertical line separating the first four bits from the last four bits. Below the binary string, there are two "1" characters, one under the first four bits and one under the last four bits, representing their decimal values.

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a man speaking.

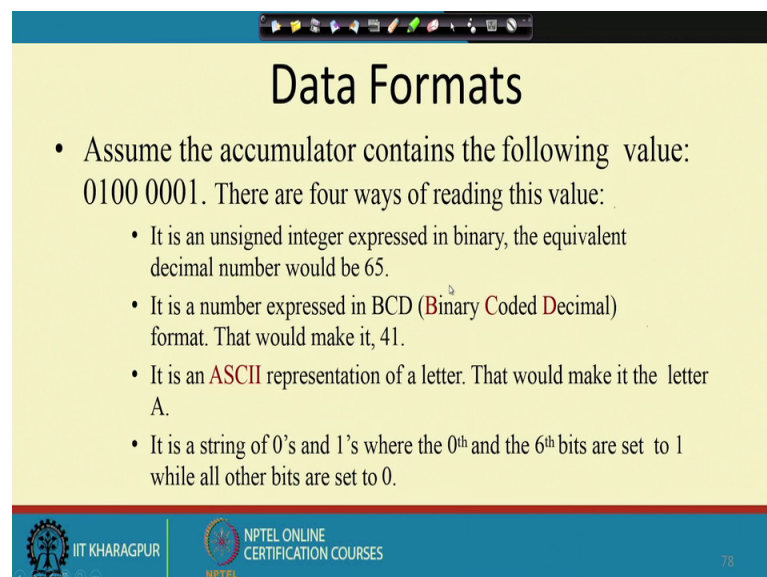
Data formats; so, if you in 8-bit microprocessor. So, we can represent data in different formats like we can be represent the ASCII. In the ASCII format for say I can all the or anything that we are want that we are talking about is nothing but a character. So, if I say like that then each character can be coded into an ASCII code. So, there is a standard ASCII codes are available and ASCII codes are 8 bit wide or 7 bit wide. So, that can be used for this storing the data representing the data or I can have a binary coded decimal.

So, for example, binary coded decimal means. So, for storing the number say 25 sorry, for storing the number say 25. So, I can have in a in a binary number system. So, it will be this is the value 25; now so in so now, if I am talking about in 8-bit representation sorry. So, this is 0 1 0 0 0 1. And before that I should have 2 more 0s. So, these 4 bits this is 4 bit, this is 4 bit. So, that makes it binary this this makes it binary 25 in an 8-bit notation.

Now, this by BCD code will tell that I will consider this part as a separate digit these 4 bits as a separate digit, and these 4 bits as a separate digit. So, this becomes. So, this part is actually one in decimal and this part is also one in decimal. So, 25 if you are representing in it in BCD form. So, it will be 1 1. So, that way also we can represent the number. So, for this particularly useful when you are trying to display the numbers. So, that becomes helpful in many cases.

So, we will come to this BCD format, later when we look into some programming examples. So, we can have signed integer or we can have unsigned integer. So, signed integer means integer with some positive or negative, an unsigned integer means they can be only positive values; so whatever microprocessor will understand. So, it will be consisting of 0s and ones only. And it deals with the values as strings of bits only and as I use that so, for the user the bit string has got different meanings. Like whether it is signed integer unsigned integer BCD ASCII whatever. So, this is for human interpretation as far as the processor is concerned. So, this is nothing but a string of 0s and 1s.

(Refer Slide Time: 23:40)



The slide is titled "Data Formats" and contains a bulleted list of interpretations for the binary value 0100 0001. The list includes: an unsigned integer (65), a BCD number (41), an ASCII character (A), and a bit string with specific bit patterns.

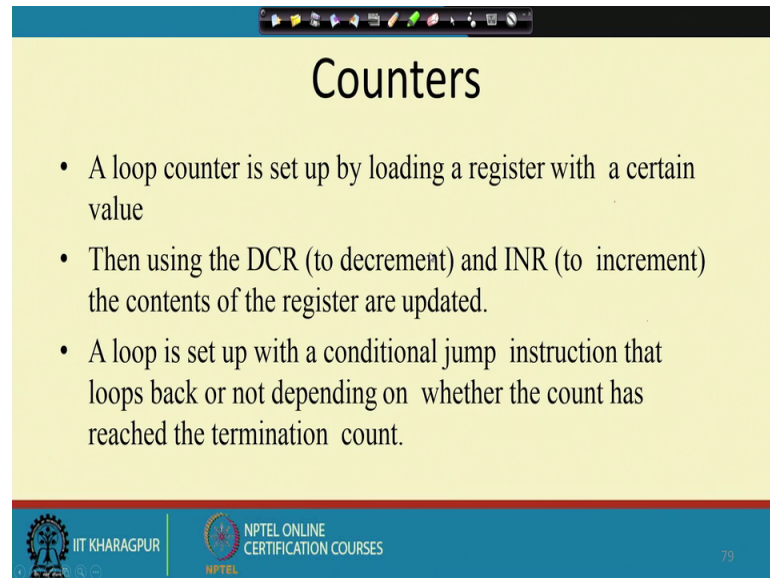
- Assume the accumulator contains the following value: 0100 0001. There are four ways of reading this value:
 - It is an unsigned integer expressed in binary, the equivalent decimal number would be 65.
 - It is a number expressed in BCD (Binary Coded Decimal) format. That would make it, 41.
 - It is an ASCII representation of a letter. That would make it the letter A.
 - It is a string of 0's and 1's where the 0th and the 6th bits are set to 1 while all other bits are set to 0.

So, for example, suppose the accumulator has got the value; 0 1 0 0 0 0 0 1; now if it is to be if it is an unsigned integer expressed in binary. So, this is the equivalent to 65. Now if you are interpreting it as a BCD number, binary coded decimal number, then as I said that the each 4 bit is constituting one digit. So, that is first 4 digit is 4 it this first 4 digit here; so 0 1 0 0. So, this will make it 4 and this will make it 1. So, this 41 is the BCD number for the same bit string. And if you are looking in the ASCII code, now this 65 is the ASCII code of A.

So, you can also say that this accumulate this accumulator is storing the character A. So, if it is a string of 0s and ones where the 0th and 6th bits are set to 1 and while the all other bits are set to 0. So, if you just take it as a bit string then I can say this is a bit string

where these 2 bits are only one and rest of the bits are 0. So, this a interpretation varies from the human understanding point of view, but as far as the system under the microprocessors understanding is concerned. So, it is nothing but a bit string only.

(Refer Slide Time: 24:59)



The slide is titled "Counters" and contains three bullet points. The background is light yellow. At the bottom, there is a blue footer with logos for IIT Kharagpur and NPTEL Online Certification Courses, and the number 79.

- A loop counter is set up by loading a register with a certain value
- Then using the DCR (to decrement) and INR (to increment) the contents of the register are updated.
- A loop is set up with a conditional jump instruction that loops back or not depending on whether the count has reached the termination count.

Next will go towards the implementation of counters, because many a times what happens is that we need to have some actions done repetitively. And we like to implement some counters in the programs. So, a loop counter so, you can implement these counters very easily in microprocessors or say 8085 by loading a register with some value. And go on decrementing the register till it becomes the content becomes 0, or you go on implementing the register till the content overflows and comes back to 0.

So, a loop can be set up with a conditional jump instruction to loop back, and it is when the count becomes a particular iteration count or termination count then the loop will terminate.

(Refer Slide Time: 25:52)

The slide is titled "Counters" and contains a bullet point: "The operation of a loop counter can be described using the following flowchart." Below the text is a flowchart with the following steps: "Initialize" (rectangle), "Body of loop" (rectangle), "Update the count" (rectangle), and a decision diamond "Is this Final Count?". An arrow from the "No" side of the diamond loops back to the "Body of loop" step. An arrow from the "Yes" side of the diamond points downwards, indicating the end of the process. The slide footer includes the IIT Kharagpur and NPTEL logos.

So, this way you can represent some counters. So, the typical counter structure body will be looking something like this. So, it is first there will be some initialization, then body of the loop then update the counter. So, how it is the final count, if it is yes it will come out and if it is no it will go back to the body of the loop. So, this way this loop counter is implemented loop counter is represented in the form of a flow chart.

(Refer Slide Time: 26:21)

The slide is titled "Sample ALP for implementing a loop Using DCR instruction". It displays the following assembly code in a white box:

```
MVI C, 15H
LOOP    DCR C
        JNZ LOOP
```

The slide footer includes the IIT Kharagpur and NPTEL logos.

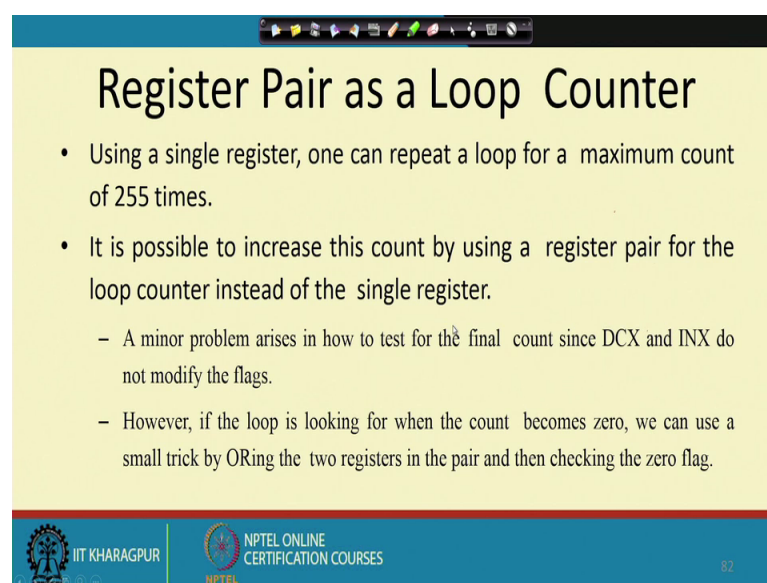
So, will see how these are implemented in programs. So, this is a very simple assembly language program for implementing loops with the DCR decrement register instruction.

So, here what is done is first this MVI C 15 hex; that means, this C register will get the value 15 hex, then this the decrement c. So, decrement C instruction. So, this will decrement the value of the C register. So, from 15 hex it will become 14 hex, 13 hex like that first it is become 14 hex, then we check this. So, because of this decrement operation, this status register the flag register will get affected, and since the value is still positive it is not yet 0.

So, jump on not 0. So, this will be true, and this will go to this say the jump on not 0 to address loop. So, it will come back to this line and it will again do a decrement. So, this way it will go on decrementing and checking whether the value has become 0 or not. And when the value becomes 0, at that time this 0 flag will be set. So, next time we take checks this JNZ. So, this 0 flag is set. So, it will not be going to this instruction, but it will it will be coming back to the next one. So, this way you can implement a very simple loop by using this by using this decrement, and jump instructions. And this is helpful for implementing some small delay.

So, for example, if you have outputted some pattern on to this led, and led some led display and human eyes will not be able to follow if it is changed very fast. So, you want to put some delay for eyes to understand the display. So, we can put some delay by putting this type of delay routine. And then after that we can change the pattern change it to the next pattern. So, that it can so that it will be able to show this.

(Refer Slide Time: 28:19)



Register Pair as a Loop Counter

- Using a single register, one can repeat a loop for a maximum count of 255 times.
- It is possible to increase this count by using a register pair for the loop counter instead of the single register.
 - A minor problem arises in how to test for the final count since DCX and INX do not modify the flags.
 - However, if the loop is looking for when the count becomes zero, we can use a small trick by ORing the two registers in the pair and then checking the zero flag.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

82

So, the next pattern so that display becomes stable; we can also use register pair as loop counter. So, in the previous example that we have taken; so here it is only the maximum delay maximum looping count is to 55 only. So, you if you want to change this value, then you have to use a register pair for this loop; so because if you use a register pair then I have got a wider range because with a 16-bit value. So, it the maximum value is 165535.

So, I can have a larger range of this looping. But there is a small problem because now we have to decrement this when you do the decrement operation by DCX instruction or increment operation during the INX operation. So, DCX and INX they do not modify the flag. So, this is a special thing that the DCX and INX instructions they do not modify the flags; however, there can be some other tricks by which we can correct this problem. And we can correct it by taking or of 2 registers this like D C. So, the DCX if it is if you are use a D C pair, then we can or the D and C registers after doing this decrement operation. And if both of them have become 0 then or of them will become 0 and this or instruction will affect the 0 flag as a result, we can detect the situation that the counter has under flown.