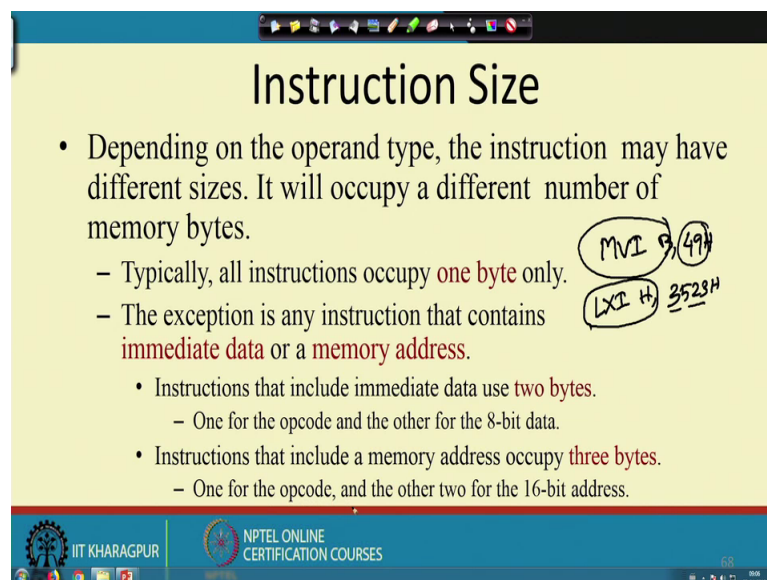


Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 54
8085 Microprocessor
(Contd.)

The other important factor when we are considering a program stored in the memory in case of a Microprocessor Based System is the instructions size. That is how many bytes and instruction occupy when it is, are occupies when it is put into the memory. So, that will determine the overall size of the program.

(Refer Slide Time: 00:35)



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.

Handwritten examples on the slide:
MVI 9, 49H
LXI H, 3528H

NPTEL ONLINE CERTIFICATION COURSES
IIT KHARAGPUR

So, depending upon the operand type, the instruction may have different sizes. Now it will occupy different number of memory bytes. So, typically all instructions occupy one byte only. So, particularly that is for the opcode parts, so this every instruction has got an operator operation code or opcode. And that opcode is generally coded as an 8-bit data as you know that in a microprocessor.

So, we have got 2 for 256 the possible opcodes that can be there of course, microprocessor 8085 is much less than that. But typically, so the opcode part will occupy 1 byte the exception is because of this immediate data or memory address. Like you

some instruction they have got an immediate data like MVI MVI moving immediate that has got an immediate data or if you, if you are mentioning memory address.

So, basically these two type of instructions like this MVI B comma say 49 hex. So, this MVI B parts, so that occupies 1 byte and the 49 hex this number occupy the another byte. So, that way it becomes a 2 byte instruction similarly this LXI instruction for example, LXI H comma say 35 23 hex. So, this will again this will occupy three byte because its LXI H part. So, this will take 1 byte and this 3523 so this is one 4 digit hexadecimal number. So, 2 hexadecimal digits can be put into 1 byte, so total 2 bytes are needed for the hexadecimal address part itself.

So, that way we can have three address 3 bytes. So, instructions that include immediate data use 2 bytes one for opcode one for 8 bit data. And the instructions that include a memory address occupy 3 bytes one for opcode and other 2 for the 16 bit address that you want to put.

(Refer Slide Time: 02:28)

The slide is titled "Instruction with Immediate Data". It contains the following text:

- 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. |

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a "Quick Heal Total Security" notification that says "Updated successfully".

So, like this as I have already said that MVI A comma 32. So, that way, so this is the code of that is instruction like a the manual of 8085 if you concert you will see that this is the code for the for the MVI instruction that is hex decimal number. So, this the number 32 in decimal.

So, this is the sorry this the opcode part, so 3E. So, 3E is the hexadecimal number corresponding to the opcode MVI a and then the next number should be 32. So, 32 this is the corresponding binary coding. So, this first byte is 3E in hexadecimal second byte is 32 in hexadecimal, so that way it is a 2 byte instruction.

(Refer Slide Time: 03:11)

Instruction with a Memory Address

- 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 |

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, then similarly this jump instruction is an example where we have got a memory address and the instruction turns out to be a three byte instruction, first part is the jump. So, that opcode is C3 and then 2085, so that is a to byte that will be required for the 16 bit address that you want to mention. And you notice here that this lower order byte that comes first and then the higher order byte. So, while coding it. So, it could C3 followed by 8 5 followed by 2 0. So, in first memory location will have C3, second location it will have 85, third location will have 20.

So, when it is doing the instruction fetch. So, in the first phase it will get C3 and in the successive phases it will get 85 and 20. And then it will then the processor will make a jump to that particular address.

(Refer Slide Time: 04:02)

Addressing Modes

- 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

Handwritten diagrams:
A ← m[4000] (with arrow from LDA 4000)
A ← m[BC] (with arrow from LDAX B)

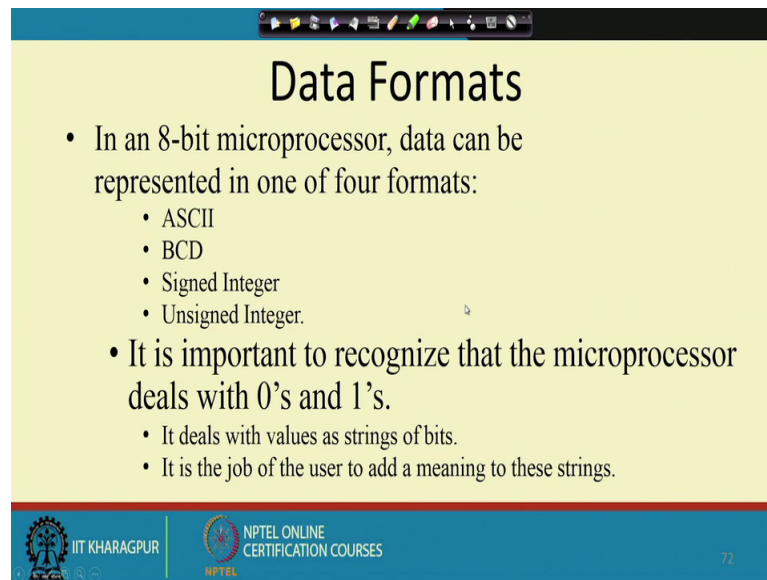
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

As far as addressing modes are concerned, so addressing modes actually tell like, how many ways you can specify the data that that an instruction will process. So, there are 4 addressing modes like first one is the implied addressing mode, where the operand or the data is implied like CMA complement accumulator. So, the instruction itself tells that the operation will be performed on the accumulator. So, that is the CMA implied type of operand, we have got immediate operand immediate data like MVI B comma 45 hex.

So, this here in the instruction itself I am telling what is the immediate value that you want to move to B. So, this 45 is the immediate value then we have got direct addressing. So, in direct addressing, so we directly specify some memory address like the instruction LDA 4000. So, what this instruction does is that this A register the accumulator will get the content of memory location 4 1000.

So, this is LDA instruction by which you can load the accumulator or it can be an indirect of a indirect data like LDAXB. So, this says this means that the A registered will get the content of memory location whose address is given by the pair BC. So, in the first case the LDA instruction, so we are directly specifying the address in the second case LDA instructions. So, we are specifying the as specifying some sort of a point. So, BC registering pair is acting as a memory pointer in this case.

(Refer Slide Time: 05:46)



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.

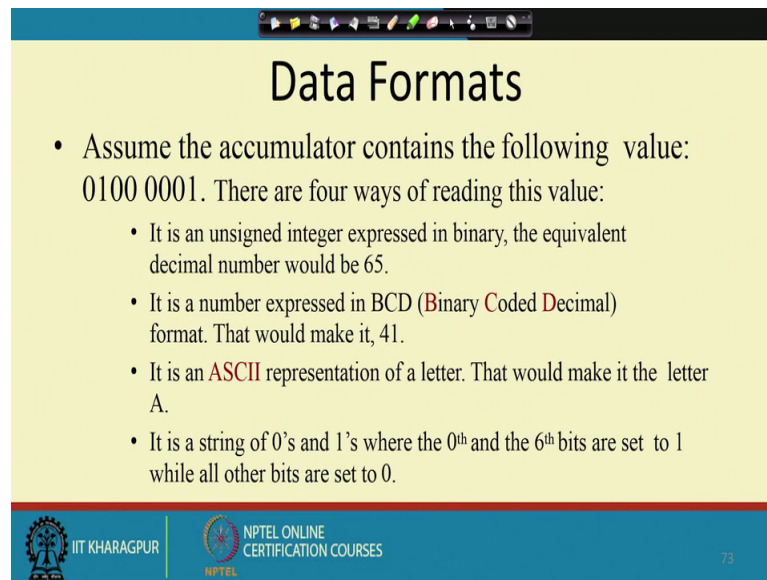
The slide footer includes the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". The number "72" is in the bottom right corner.

So, as per the data formats are concerned. So, in this is an 8 bit microprocessor data can be represented in one of this 4 format which can be ASCII. So, where you are just storing the characters you can BCD Binary Coded Decimal, you can be signed integer. So, integer with sign and unsigned integer so we does not considered say sign. So, since if the data is 8 bit. So, a signed integer we can go for from minus 127 to minus 128 plus 127. Then you are unsigned integer can go from 0 to 255 and your this ASCII BCD, so they are ASCII is 7 bit, but it is extended to 8 bit and BCD is binary coded decimal.

So, since we need 4 bits for coding one decimal number decimal digit between 0 and 9 so, in one in 8 bits, so we can code to decimal digits. So, it is often represented as binary coded decimal format, it is important to recognize that the microprocessor deals with zeros and ones and it deals with values as string of bits. So, it always handles the data as string of bits. And it is up to the user to understand the meaning of what this string will correspond to like when you are putting it one a set of LED.

So, it may be representing a pattern of glowing of those LED 's if you are putting it on a some display, then maybe it is a character that comes out of that that may correspond to the ASCII number ASCII character that you want to display. So, this way this data formats will vary.

(Refer Slide Time: 07:26)



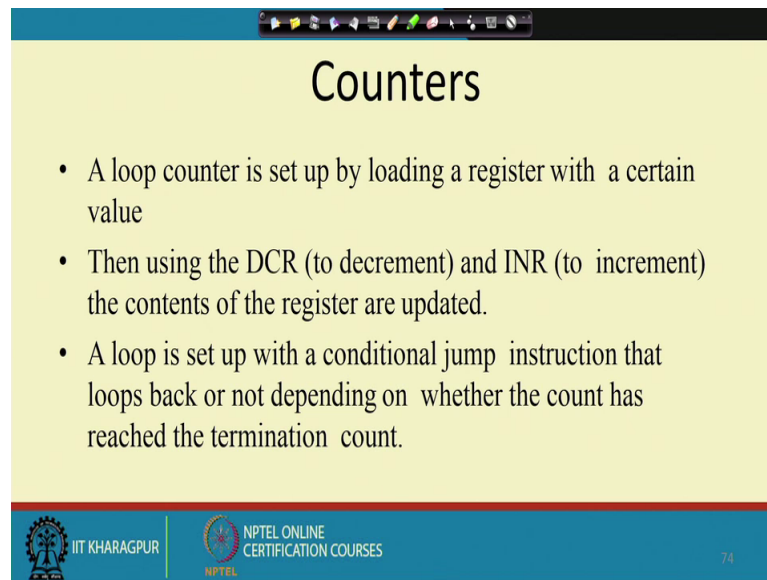
The slide is titled "Data Formats" and contains a bulleted list of four ways to interpret the binary value 0100 0001. The slide also features logos for IIT Kharagpur and NPTEL Online Certification Courses at the bottom.

- 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.

Now, for example, this accumulator suppose the accumulator has gone content 0 1 0 0 0 0 0 1. So, there are 4 ways by which you can interpret this result. So, if you are looking for an unsigned integer expressed in binary. So, this is the corresponding integer number 65. If you are looking it as BCD an a binary coded decimal format. In the first 4 bits will correspond to 1 digit and the next 4 bits will correspond to the next digits. So, it is 41. If you are looking for an ASCII character then this the 65 or this 41 hex you know that this correspond to the ASCII letter a uppercase characters.

So, let us see that is letter character a if you are looking it is a string of 0's and 1's where 0th and the 6th bits are set to one and rest are all 0. So, this may be the interpretation when you are looking it as a bit string. So, this way we can have different interpretations of this of this bit strings that we have in the stored in the microprocessor.

(Refer Slide Time: 08:34)



The slide is titled "Counters" and contains three bullet points. The first bullet point states that a loop counter is set up by loading a register with a certain value. The second bullet point explains that the contents of the register are updated using the DCR (to decrement) and INR (to increment) instructions. The third bullet point describes a loop set up with a conditional jump instruction that loops back or not depending on whether the count has reached the termination count. The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and the number 74.

- 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 be looking into something like a counter like many a times what happens is that you need to set some counter way of all loading A register is some value. So, and then we can decrement that counter and put some generates some sort of repetitive action out of that. So, we put we initialize some register to some value. And then we use the decrement and implement instructions to decrement or implement the register ok, that that has got the value.

So, DCR and INR, so these are the two instruction that we have we have in that purpose. A loop is set up with conditional jump a instructions that loops back or not depending on whether the count as this termination count or not. For example, if you start with set all 0 and say that I will count up to 100. So, when they after every increment the value increases by 1.

So, when it reaches 100, so you can do termination. So, you can do a compare whether you have reach the value 100 or not or other way maybe start with 100 and check for the condition that it becomes 0. So, by means of decrement instruction, so we decrement the value and then when it comes to 0 we stop the counter.

(Refer Slide Time: 09:50)

Counters

- The operation of a loop counter can be described using the following flowchart.

```
graph TD; A[Initialize] --> B[Body of loop]; B --> C[Update the count]; C --> D{Is this Final Count?}; D -- No --> B; D -- Yes --> E[ ];
```

The flowchart illustrates the process of a loop counter. It starts with an 'Initialize' step, followed by the 'Body of loop', then 'Update the count'. A decision diamond asks 'Is this Final Count?'. If the answer is 'No', the flow returns to the 'Body of loop'. If the answer is 'Yes', the flow exits the loop.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

75

So, the overall operation is like this first you have to initialize the counter, and then we have to have this body of the loop will have the body of the loop, whatever with action that you want to do. Then update the count, and then we check whether this is final count or not if it is so then it comes out if it is not it goes back executes the loop body once more.

(Refer Slide Time: 10:16)

Sample ALP for implementing a loop Using DCR instruction

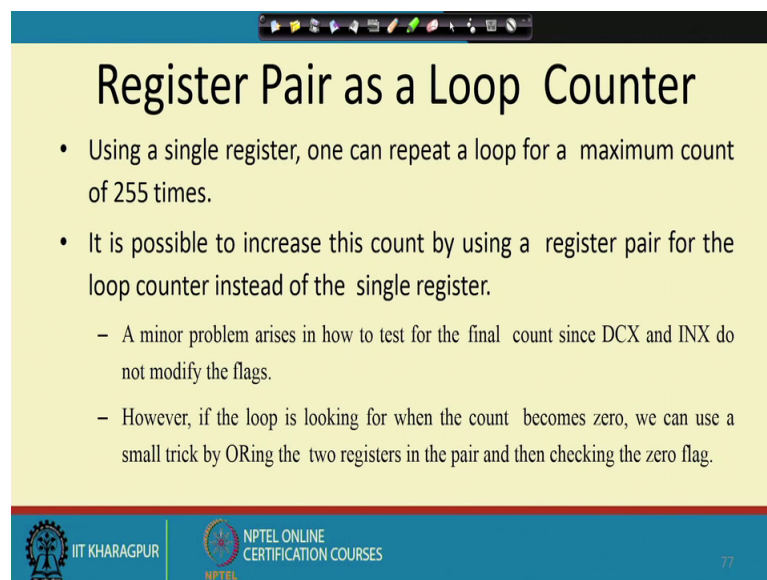
```
MVI C, 15H
LOOP    DCR C
        JNZ LOOP
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A sample of assembly language program for implementing this loop using decrement instruction DCR instruction is like this, we move immediate C comma 15 hex. So, if you have C register gets the value 15 hex then we decrement C and jump on not 0 to loop.

So, we are decrementing the value of C, so then it will become 14 hex, 13 hex, 12 hex like that. So, till the values are not 0 the control will be coming back to this decrement instruction and when this value becomes 0 then it comes out of the loop. So, this way this program as such this assembly language program fragment is doing nothing more than waiting for some time ok. So, one way to wait for some time is by means of this n o p e or nope instruction and other way can we by means of putting this type of loops which are also known as delay loops. So, you put some delay into the operation of the system.

(Refer Slide Time: 11:15)



The slide is titled "Register Pair as a Loop Counter" and contains the following text:

- 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.

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and the number 77.

So, sometimes what happens is that this delay that we produce is not sufficient because this, if you are using an 8 bit register for to hold the count then you can get a maximum delay of 255 units ok. So, you can calculate out of the 255 units the delay that is produced may not be sufficient. So, when you are when you are looking for some periodic task to be done. So, add some periodic intervals will do the operation.

So, that small delay may not be sufficient, so for that purpose what we do we use some register pair. So, it is you use the register pair to increase the count value that is possible ok.

(Refer Slide Time: 12:04)

Sample ALP for implementing a loop Using DCR instruction

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, but there is a one small problem like in the previous your instruction we have seen that after this decrement C we could do jump on not 0 because whenever this DCR instruction it affects the 0 flags. So, if the content becomes 0 then this the 0 flag will be equal to 0 equal to 1.

(Refer Slide Time: 12:17)

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

But in case of this loop pair, so what we have to do is that we have to use this DCX and INX instruction for register pair decrement and increment, and this register pair increment decrement operations they do not affect the status flag.

So, this is by the design of the 8085 processor. So, we cannot answer why is it happening like this, but this by the design of the processor. So, if you are using this 8085 processor and you are using 16 bit count value decrementing or incrementing by DCX INX instructions, then we have to be a bit careful. So, if the loop is looking for when the count becomes 0, so we can use a trick like what you do you or the content of 2 registers in the pair and then check the 0 flag like this.

(Refer Slide Time: 13:09)

Register Pair as a Loop Counter

- The following is an example of a loop set up with a register pair as the loop counter.
- LXI B, 1000H
- LOOP DCX B →
- MOV A, C →
- ORA B
- JNZ LOOP

Handwritten notes on the slide:

BC = 1000H
 B = 10
 C = 00
 A = FF
 B → 0FFFC
 FF
 0F
 ———
 FF

So, suppose we want to have a delay of this 1000 hex ok. So, this 1000 hex we move it to this B register B register pair. So, as a result the BC pair gets the value the BC pair gets value 1000 hex or rather this B register gets the value 10 and C register gets the value 00. So, that is the situation now this DCX B instructions.

So, this will directly operate on this 1000 hex instruction value and after doing this decrement by one the value become 0 FFF ok, then after. So, then I need to check whether the content is 0 or not, but the problem is when this DCX instruction does this. So, it does not affect the status flags ok. So, I need somehow some technique by which I can get the status flags affected and check whether the content has become 0 or not.

So, for that purpose the trick that is taken is MOV A comma C. So, now, A register gets the value of this C register that is FF. So, this is the C register now and this is the B register now, content of B register and C register. So, C register A register gets the value

FF and then we do or accumulator with B registers, so with this FF and 0F their or FF and 0F are or.

So, when you do this or operations, so you get the result FF and since the, but this or instructions, so this will affect the status flags; and you know that this FF it is not affecting the it is not equal to 0, so the 0 flag is not set. So, this jump or not 0 condition will be true and then it will be jumping back to this DCX instruction. So, this way we can have register pair as a as the counter value, so that we can get larger delays.

(Refer Slide Time: 15:19)

Delays

- Knowing how many T-States an instruction requires, we can calculate the time using the following formula:
 - Delay = No. of T-States / Frequency
- For example a “MVI” instruction uses 7 T-States. Therefore, if the Microprocessor is running at 2 MHz, the instruction would require 3.5 μSeconds to complete.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

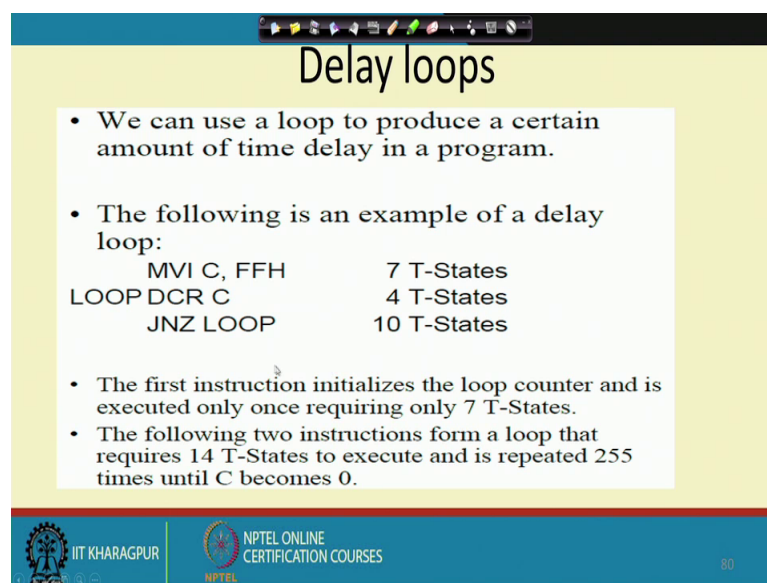
So, but, so far I have talked about delays, but I did not explicitly mention the time part of it I just we just had a notion that there is a delay introduced. That is coming in terms of this counter value that we have put into the loop now how to calculate the delay part? So, if for every instruction I can know how many clock cycles it takes. So, you know that 8085 it operates that some clock frequency for example, we can take it a two operate 2 megahertz ok.

So; that means, will it will for every clock cycle what is the duration, so 1 upon 2 megahertz. So, that is the duration for every instruction. So, that much is the for every clock cycle. So, every clock cycle is like that now if you for every if you look into the manual of 8085. So, you will find that for every instruction its size and its number of clock cycles are mentioned.

So, from there you can calculate how many clock cycles are required and from there you can come to the you come to the time needed for executing particular instruction. So, if for a particular program, so if you calculate the total number of clock cycles that is needed and you divide it by the frequency. So, you can get the delay that will be involved in executing the program fragment. So, if you are looking at a single instruction level for example, this MVI instruction it has 7 T state. So, T state means each clock periods, so how much time each clock period is 1 T state.

So, that is 7 T states and if you assume that this microprocessor is running at 2 megahertz, then this instruction will required 3.5 micro second. So, this is 7 divided by 2 megahertz. So, that is 3.5 micro second to complete. So, MVI MVI instruction requires about requires 3.5 micro second and that is exact ok. So, it is not dependent on any other factor, so that way it is exact calculation.

(Refer Slide Time: 17:24)



The slide is titled "Delay loops" and contains the following text:

- We can use a loop to produce a certain amount of time delay in a program.
- The following is an example of a delay loop:

| | |
|------------|-------------|
| MVI C, FFH | 7 T-States |
| LOOP DCR C | 4 T-States |
| JNZ LOOP | 10 T-States |
- The first instruction initializes the loop counter and is executed only once requiring only 7 T-States.
- The following two instructions form a loop that requires 14 T-States to execute and is repeated 255 times until C becomes 0.

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and the number 80.

Now, if we are writing a loop like this MVI C comma FFH, DRC and JNZ loop. So, much what is the delay that is produced so, first this loop initiation, so that is MVI instruction that required 7 clock cycles. Then we have got a decrement C again you refer to the manual of 8085 and you can find that decrement operation it requires 4 clock cycles. So, that is 7 plus 4 and this JNZ loop, so this requires 10 clock 10 clock cycles ok. So, the, so you see it is quite obvious because this MVI instruction.

So, it offers the in the first byte it will get the opcode for MVI in the second cycle, so it will get the value F F X. So, then that is 4 plus 3 fetch always takes 4 clock cycle or 4 T states. So, 4 plus 3 7 T states the similarly this decrement C of instructions it has got only the fetch part it does not have any other ones it has got the instruction you will decrement it.

So, that way number of clock cycles needed is 4 and this JNZ loop. So, loop is 16 bit value, so I need to 4 clock cycles to get the JNZ part and then 3 plus 3 clock cycles for the getting the 16 bit address parts. So, total 10 T states will be needed for this JNZ instruction. So, the first instruction initializes the loop counter and is executed only ones requiring 7 T states, the following two instruction the decrement and JNZ. So, they form a loop that requires 14 T states taken this these two things together and they are executed 255 times until C becomes equal to 0.

(Refer Slide Time: 19:14)

Delay Loops (Contd.)

- We need to keep in mind though that in the last iteration of the loop, the JNZ instruction will fail and require only 7 T-States rather than the 10.
- Therefore, we must deduct 3 T-States from the total delay to get an accurate delay calculation.
- To calculate the delay, we use the following formula:
 - $T_{\text{delay}} = \text{total delay}$ $T_{\text{delay}} = T_0 + T_L$
 - $T_0 = \text{delay outside the loop}$
 - $T_L = \text{delay of the loop}$
- T_0 is the sum of all delays outside the loop.

JNZ (4-bit)
 ↓
 Succ → 10
 Fail → 7

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, from this you can calculate the time needed. So, so we need to keep in mind that in the last iteration of the loop the JNZ instruction will fail and you required 7 T states rather than 10 T states, because actually the way it executes is first it gets that JNZ part. So, JNZ and the 16 bit value, so this JNZ. So, this way for the when the it if you look into the manual it will find that there are two time of two types or types of timing requirement, one is the loop is successful loop is successful another case the loop is fail it does, so JNZ condition is false.

So, if the loop is successful in that case it requires 10 clock cycles if the loop is a failure then it requires 7 clock cycles or 7 T states. So, the way we should calculate is that the total delay is that, we should subtract 3 T states on the total delay to get an accurate delay calculation and to calculate the delay.

So, we compute the total delay is T_O plus T_L where T delay is the total delay, T_O is the delay outside the loop that is initialization of the counter and all and T_L is the delay of the loop. So, this is the delay of the loop and T_O is the sum of all these loops. So, once you do that, so you can get the actual delay part.

(Refer Slide Time: 20:44)

Delay Loops (Contd.)

- Using these formulas, we can calculate the time delay for the previous example:
- $T_O = 7$ T-States
 - Delay of the MVI instruction
- $T_L = (14 \times 255) - 3 = 3567$ T-States
 - 14 T-States for the 2 instructions repeated 255 times ($FF_{16} = 255_{10}$) reduced by the 3 T-States for the final JNZ.

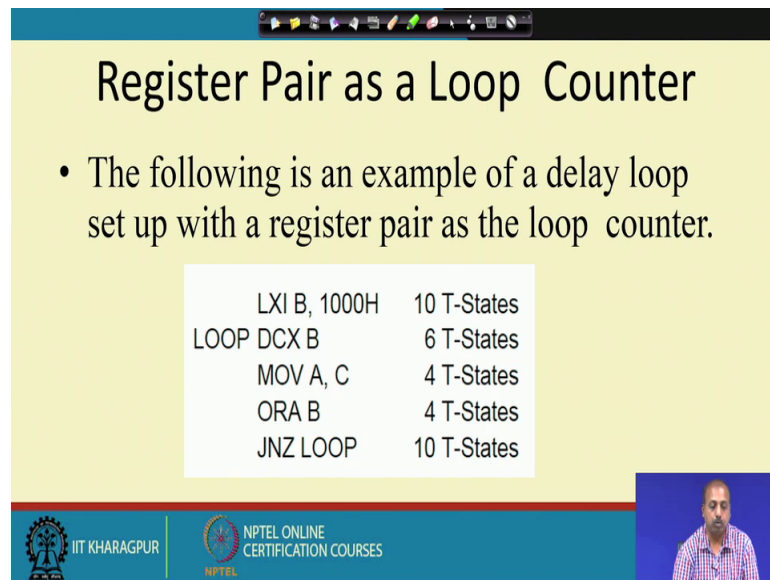
$3567 + 7 = 3574$
 $\frac{3574}{2 \text{ MHz}} = 1787 \text{ ns}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, what we have, so this using this formulas. So, we can calculate the time delay of the previous example T_O that is the MVI instructions. So, that took 7 T states that is the delay of the MVI and the T_L parts. So, that is 14 T states where necessary and that is repeated 255 times. So, 14 into 255 minus 3 because for the last part, so this JNZ when it is failing. So, it takes only 7 clock cycles not 10 clock cycle or 3 clock cycles they are less.

So, that way this becomes 3567 T states, so 14 T states for the two instruction repeated 255 times reduced by 3 T states for the final JNZ it. So, that gives us the total time needed and the total time needed is this is T loop parts.

(Refer Slide Time: 21:33)



The slide features a title 'Register Pair as a Loop Counter' and a bullet point stating: 'The following is an example of a delay loop set up with a register pair as the loop counter.' Below this, a table lists instructions and their T-State counts:

| | |
|--------------|-------------|
| LXI B, 1000H | 10 T-States |
| LOOP DCX B | 6 T-States |
| MOV A, C | 4 T-States |
| ORA B | 4 T-States |
| JNZ LOOP | 10 T-States |

The slide also includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a presenter in the bottom right corner.

So, this, if you say that total delay. So, T delay, so T delay is going to be equal to this 3567 plus T that is 3547 T states and then you know a depending upon the frequency of operation. So, you can calculate what is the overall time needed. So, this is I have got here 3567 plus 7 equal to 3574 clock cycles. So, this 3574 divided by 2 megahertz, so if I say that I will be operating at 2 megahertz. So, this is your equal to 1787 microsecond.

So, that will be the total delay produced by the previous program. So, if for the, so if the program is complex then it will take more time. So, if you are using register pair as a loop, loop counter then what will happen? Suppose, this is the program LXI B then DCX B, MOV A comma C, ORA B or accumulator B and JNZ loop. So, this LXI takes 10 T states I have this part of the, it is part of information as I have already said is available from the manual. Then this DCX B instruction takes 6 clock cycles MOV A comma C takes 4 clock cycles or accumulator B it takes 4 clock cycle JNZ it takes 10 clock cycles.

(Refer Slide Time: 23:14)

Register Pair as a Loop Counter

- Using the same formula from before, we can calculate:
- $T_0 = 10$ T-States
– The delay for the LXI instruction
- $T_L = (24 \times 4096) - 3 = 98301$ T-States
– 24 T-States for the 4 instructions in the loop repeated 4096 times ($1000_{16} = 4096_{10}$) reduced by the 3 T-States for the JNZ in the last iteration.

Handwritten calculation: $\frac{98311}{2 \text{ MHz}} \approx 49155.5 \mu\text{s}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, based on this we can calculate what is the total delay that is required. So, T outside is equal to 10 T states for the LXI instruction T loop body. So, that is 24 because here the total size becomes 6 plus 4 plus 4 plus 10, so total is 24. So, 24 into 4096 that is the, that is the value that I have initialized, so 1000 hex that is 4096 in decimal minus three that is for the last clock cycle. So, I do not do this JNZ instruction requires only 7 clock cycles or 3 clock cycles and less. So, total is 98301 T states. So, 24 T states of a four instructions in the loop repeated 4096 times reduced by 3 T states for the JNZ in the last iteration.



So, total time is 98301 plus 10, so 98311 so based on that. So, you can find out what is the total delay. So, this is your 98 98 9830311 divided by 2 megahertz. So, so that way it is 49 then 1 55.5 microsecond. So, this is the delay that will be produced by the previous instruction previous example, now if you look into if you want to produce more delay ok.

(Refer Slide Time: 24:51)

Nested Loops

- Nested loops can be easily setup in Assembly language by using two registers for the two loop counters and updating the right register in the right loop.
 - –In the figure, the body of loop2 can be before or after loop1.

```
graph TD; Start([Start]) --> Init2[Initialize loop 2]; Init2 --> Body2[Body of loop 2]; Body2 --> Init1[Initialize loop 1]; Init1 --> Body1[Body of loop 1]; Body1 --> Update1[Update the count1]; Update1 --> Dec1{Is this Final Count?}; Dec1 -- No --> Body1; Dec1 -- Yes --> Update2[Update the count 2]; Update2 --> Dec2{Is this Final Count?}; Dec2 -- No --> Body1; Dec2 -- Yes --> End([End]);
```

85

Sometimes this is not sufficient, so you want to put some nested loop and get the delay value increased. So, how do you do this we initialize? So, you used to nested loop. So, there is a loop 1 which is this part, so initialize loop 1 body of loop 1 updated count one. And if it is not this is the final count then you go back and again execute the body of loop 1.

So, this is the inner loop and then we have got another loop over and above the inner loops. So, that updates the counter 2 and then whether it is this the final count or not if it is not it will again do the a body of loop to it will go like this, so this way we can have two nested loops. So, nested loops can be easily set an assembly language by using 2 registers for the 2 loop counters and updating the right registered in the right loop.

So, this body of loop 2 can be before or after loop 1, so that can be there. So, that does not matter. So, what I mean is this body of loop 2, so this parts. So, it can be put before loop 1 or it can be after this after this also so it does not matter ok.

(Refer Slide Time: 26:08)

Nested Loops for Delay

- Instead (or in conjunction with) Register Pairs, a nested loop structure can be used to increase the total delay produced.

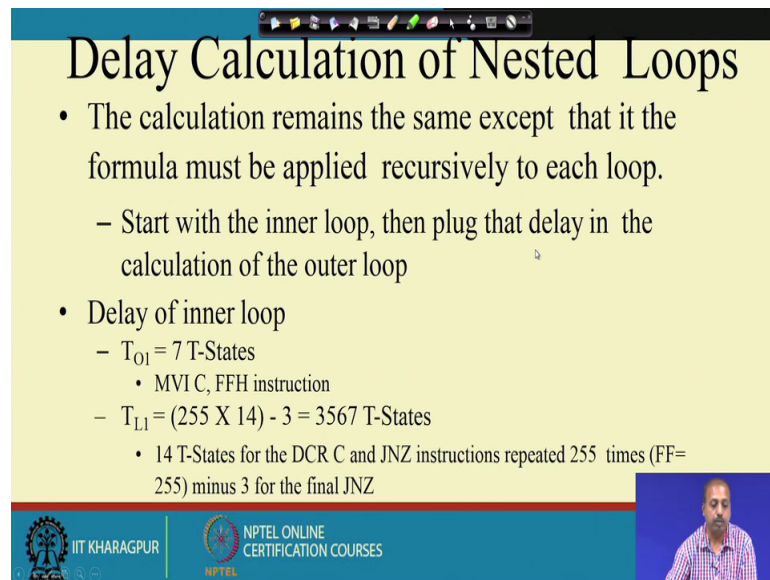
| | | |
|-------|------------|-------------|
| | MVI B, 10H | 7 T-States |
| LOOP2 | MVI C, FFH | 7 T-States |
| LOOP1 | DCR C | 4 T-States |
| | JNZ LOOP1 | 10 T-States |
| | DCR B | 4 T-States |
| | JNZ LOOP2 | 10 T-States |

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 86

So, now we can calculate what is the total delay that is required that is generated by the process so, you see that we have got say this particular program. So, MVI B comma 10 hex then MVI C comma, so these are the 2 counters that we have B and C are the 2 register pairs are the register pair now the inner loop is consisting of the C register.

So, this decrement C JNZ loop 1, so this will be doing this inner loop and then you have got the outer loop in the outer loop we decrement the D and JNZ loop 2. So, that way it is doing this outer loop now if we to concert the manual and determine the number of T states needed for different instructions. So, this is like this, so 7 7 4 10 10 4 10 so like that. So, these are the delays now how to calculate the total delay that is produced.

(Refer Slide Time: 27:03)



Delay Calculation of Nested Loops

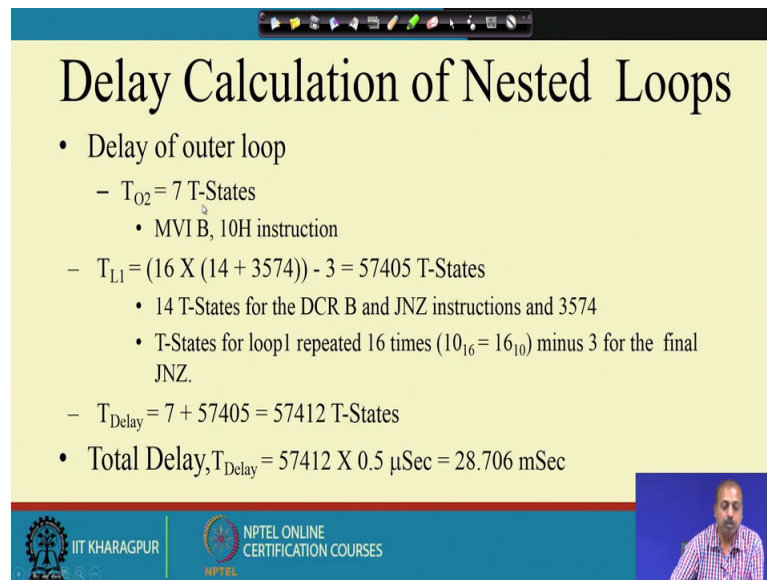
- The calculation remains the same except that the formula must be applied recursively to each loop.
 - Start with the inner loop, then plug that delay in the calculation of the outer loop
- Delay of inner loop
 - $T_{O1} = 7$ T-States
 - MVI C, FFH instruction
 - $T_{L1} = (255 \times 14) - 3 = 3567$ T-States
 - 14 T-States for the DCR C and JNZ instructions repeated 255 times (FF=255) minus 3 for the final JNZ

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we can do it like this, so calculation remains same except that the formula must be apply recursively to each loop. So, start with the inner loop and then plugged the delaying the calculation of the outer loop. So, that is how because the inner loop becomes a part of the body of the outer loop you can say. So, we start with the inner loop calculate the time needed there and then put it into the body of the outer loop. So, for this first initialization that MVI C comma FFX that instruction, so MVI C comma FFX, so that required 7 T states so, these are delay for the inner loop.



So, this is the outer of one and then this is then you have got T L 1. So, the loop of this inner loop body of this the inner loop this two. So, this 10 plus 14 and then for the last loop this JNZ will be 7 T state. So, that way it is done to 55 into 14 minus 3 that is 3567 T states 14 T states for the decrement C instruction and JNZ instructions that is repeated for 255 times and then minus 3 for the final JNZ instruction.

(Refer Slide Time: 28:19)



Delay Calculation of Nested Loops

- Delay of outer loop
 - $T_{O2} = 7$ T-States
 - MVI B, 10H instruction
 - $T_{L1} = (16 \times (14 + 3574)) - 3 = 57405$ T-States
 - 14 T-States for the DCR B and JNZ instructions and 3574
 - T-States for loop1 repeated 16 times ($10_{16} = 16_{10}$) minus 3 for the final JNZ.
 - $T_{Delay} = 7 + 57405 = 57412$ T-States
- Total Delay, $T_{Delay} = 57412 \times 0.5 \mu\text{Sec} = 28.706 \text{ mSec}$

Then for the outer loop, so you have got this outer loop outside the loop. So, that is the 7 T states because of this MVI B instruction and when you are computing the loop outer loop body then it is 16 into 14 for the 14 for this parts. So, outer loop has got this decrement B and this JNZ. So, this is 14 plus whatever be the delay you needed for this loop 1. So, that will come, so that is put here, so 14 plus 3574.

So, this outer loop, so that was 3567 that was 3567 plus 7 3574. So, that 3574 is added and this whole thing minus 3. So, that makes it 57405 clock cycles, so total delay is this outers are outer delay of 7 plus 57405 T states. So, total 57412 my get T states. So, if you are assume 2 megahertz operation then the total delay will be about 28.706 milliseconds. So, it is significantly high when you considering a single loop at a time. So, this way we can use nested loops for generating delays within the programs for microprocessors.