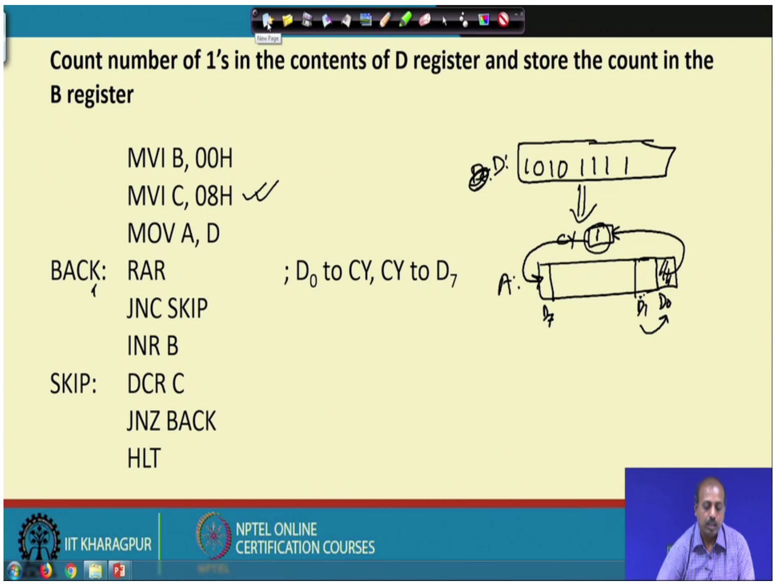
Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology Kharagpur

Lecture – 62
8085 Microprocessor
(Contd.)

(Refer Slide Time: 00:19)



Count number of 1's in the contents of D register and store the count in the B register

```
        MVI B, 00H
        MVI C, 08H
        MOV A, D
BACK:   RAR          ; D0 to CY, CY to D7
        JNC SKIP
        INR B
SKIP:   DCR C
        JNZ BACK
        HLT
```

So, we will look into a few examples of 8085 programming the first example that we will look into it is a program to count number of 1's in the content of D register and we want to store the value in the B register. So, our D register contains some numbers say. So, this is the D register then the content maybe say 10101 111 and we want to count the number of 1, so in this case the answer should be 6. So, how do we do it? So, first this B register since the B register will be holding the final value, so we are clearing the B register. So, B register made equal to 0, then in the C register we take it as a some sort of counter like how many bits we have considered so far ok.

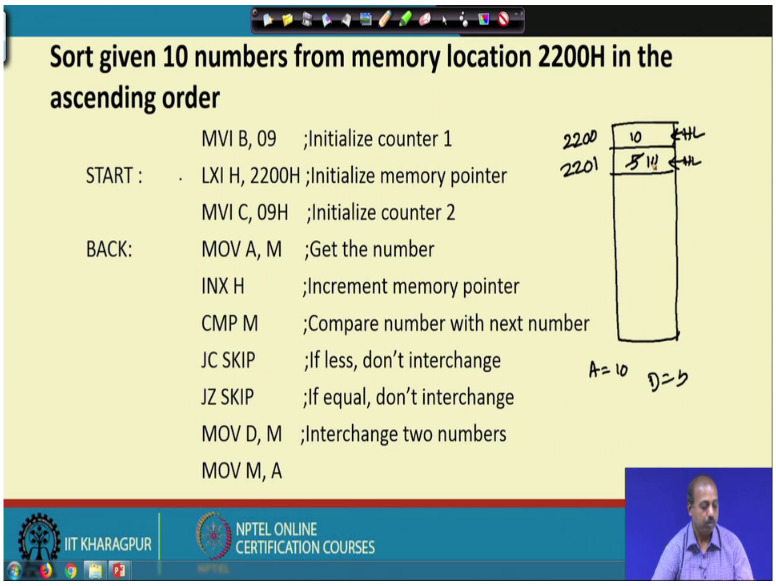
So, basic idea is that so we will move this B register content into A register. So, sorry this D register content into A register and then to we will check what is the value of this least significant bit. And for the technique for checking there can be several ways by which you can check the content of this least significant bit one possible way is like this, so you have that carry flag. So, you do a rotation, so rotation like this and this carry flag comes here to the most significant position. So, this is the bit D0 and this is the bit D7 of the A register.

So, if this if this bit was equal to 1 then after this rotation the carry flag will be equal to 1. So, we can check the status of this carry and after 8 such rotations, so we will have the count we have we have already seen all the 1's in the B register. So, the program is like this that we take the we move the content of D register to A register then rotate accumulator right. So, this rotate accumulator right, so this does exactly the thing that we were looking into the D0 will be coming to the carry bit and the carry bit will come to the D7 register. Then jump on no carry, so if the carry is not saved; that means, the corresponding bit was reset the bit D 0 was reset, so jump on no carry to skip.

So, there we do not increment the value of the count B, but if the carry was set then it will come to this point INR B and then that B register value will be incremented by 1 it and it will be counting that that 1 it has seen that 1 bit that it has seen. And then it will decrement the C register because we have already seen we have already checked the value of 1 bit, so it will be decrementing the C register the C register will becomes 7 and then till C register becomes 0. So, it will be jumping back to it will be jumping jump on not 0 to this back this point and then again we are rotating the next bit.

So, by the by the first rotation what has happened is, so the bit at position D 1 has shifted to D 0. So, in the next rotation this B D 1 will be coming to this carry bit as a result we can again check the carry and then we can take a decision whether bit D 1 was equal to 1 or not. So, this way we can count number of 1's in the D register and store the value in the C register in the B register.

(Refer Slide Time: 03:36)



Sort given 10 numbers from memory location 2200H in the ascending order

```
        MVI B, 09      ;Initialize counter 1
START : LXI H, 2200H   ;Initialize memory pointer
        MVI C, 09H     ;Initialize counter 2
BACK:   MOV A, M       ;Get the number
        INX H          ;Increment memory pointer
        CMP M          ;Compare number with next number
        JC SKIP        ;If less, don't interchange
        JZ SKIP        ;If equal, don't interchange
        MOV D, M       ;Interchange two numbers
        MOV M, A
```

Next we will be looking into another program that will sort a given 10 numbers from memory location 2200H in the ascending order. So, from 2200H we will have 10 numbers and then those values will be sorted. The sorting algorithm that we will use is the basically whenever we find 2 numbers out of order we exchange them and this process we repeat for all the 9 entries and that way it is done.

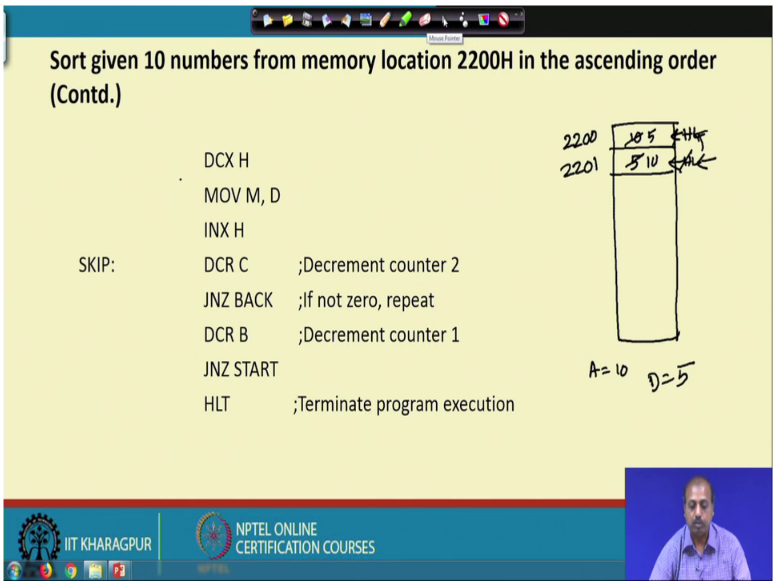
So, how are you doing it? So first this B register is having the value 09, so this B register is holding the counter there we actually need 2 counters. So, this B register any loop comparison, so you know that any sorting algorithm it uses nesting or nesting on 2 loops the bubble sort type of algorithms, so it uses nesting on 2 loops. So, here exactly what we have done here sp for the first loop the counter is in B register for the second loop the counter is in C register then this LXI H 2200H. So, this is the 2200H value is loaded into the HL pair and then we get the first number into the A register. So, from 2200 onwards we have this numbers stored.

So, if this is the location 2200 the very first thing that we need to do is to check the contents of 2200 and 2201 and if the numbers are out of order we need to exchange them. So, my HL pair is now pointing to this location 2200 by means of this instruction and then move A comma M, so whatever be the value suppose the value was 10. So, the A register is having the value 10 now then INX H now the HL pair is moved to this point ok.

So, HL pair points to this the next entry and then compare memory, so this 10 is compared with the next 1 suppose the value here is 5, so 10 is compared with 5. So, if it is less then jump on carry the jump on carry means the number memory the location the value at the next location is larger than the accumulator.

Otherwise the if the number is smaller than the accumulator which is the which is the situation here, then the carry flag is not safe or is not set, so jump on carry and jump on 0, so they will be checking whether it is less or equal; if it is not so then we need to interchange between these 2 memory locations. So, here actually that is that is the situation so for that purpose we move D comma M. So, D register gets the value 5 and then move M comma A, so memory register memory is pointed to by HL and A register value is 10, so at this location we get the value 10.

(Refer Slide Time: 06:43)



Sort given 10 numbers from memory location 2200H in the ascending order (Contd.)

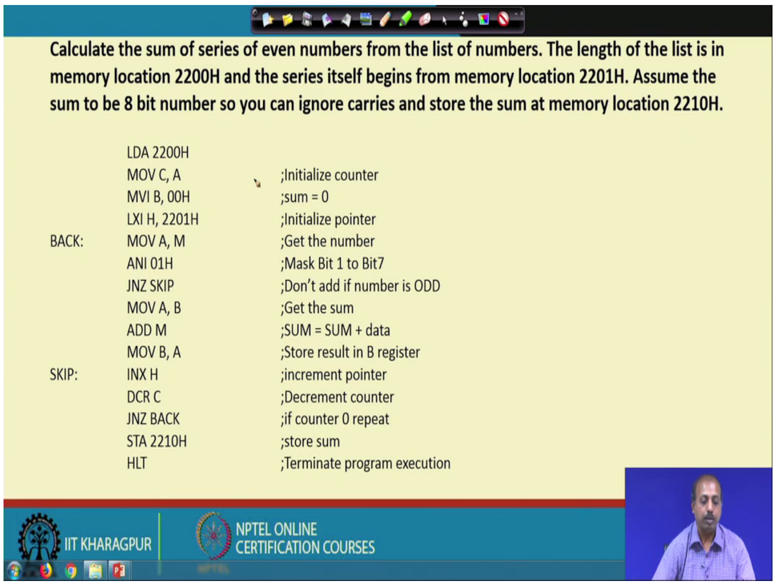
```
            DCX H
            MOV M, D
            INX H
SKIP:       DCR C       ;Decrement counter 2
            JNZ BACK    ;If not zero, repeat
            DCR B       ;Decrement counter 1
            JNZ START
            HLT         ;Terminate program execution
```

Then in the next sorry then in the next point what we are doing we are decrementing H, so that this HL pair this HL pair was pointing to yeah. So, this HL pair was pointing to yeah, so HL pair was pointing to this location and then this DCX H, so DCX H will take the HL back to this point the previous point and then we are doing move M comma D, so move M comma D will put this 5 value into this location.

So, as a result so these 2 values are interchanged. So, then we are doing INX H. So, H so the so H HL pair the HL pair points to this and then we are decrementing C, so countered second counter value is decremented and we are checking whether the first loop should end or not. So, JNZ jump on not 0 back. So, it will be going back to the location it will going back it will be going back to the location back here if the count value is non 0.

So, this C register value has become non 0, so that way this will be continuing the loop, so now it will compare. So, the next location so it will compare the next location with the with the next entry, so as a result these loop these 2 loops so in they are repeated 10 times in that is 9 into 9 that is 10 times, the each loop will be executed then all the numbers will get sorted. So, this is basically the bubble sort type of algorithm that we can implement in 8085 program.

(Refer Slide Time: 08:36)


Calculate the sum of series of even numbers from the list of numbers. The length of the list is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 8 bit number so you can ignore carries and store the sum at memory location 2210H.

```
        LDA 2200H
        MOV C, A        ;Initialize counter
        MVI B, 00H      ;sum = 0
        LXI H, 2201H    ;Initialize pointer
BACK:   MOV A, M        ;Get the number
        ANI 01H         ;Mask Bit 1 to Bit7
        JNZ SKIP        ;Don't add if number is ODD
        MOV A, B        ;Get the sum
        ADD M           ;SUM = SUM + data
        MOV B, A        ;Store result in B register
SKIP:   INX H           ;increment pointer
        DCR C           ;Decrement counter
        JNZ BACK        ;if counter 0 repeat
        STA 2210H       ;store sum
        HLT             ;Terminate program execution
```

Next we will be looking into another program that will calculate the sum of series of even numbers from the list of numbers. So, we have got a list of numbers and then we are interested to sum only the even numbers from there, so odd numbers will not be added. So, the check for odd number is that the least significant bit is equal to 1 and for even number the least significant bit is equal to 0. So, the number of numbers is a stored in the location 2200 and then the actual numbers begin from the location 2201.

So, for the program we just we first load this accumulator with the content of the location to 2200, so the count value is available in the A register that is moved to C register. So, the counter is initialized to C and then B register so that is initialized to 0. So, B will contain the full sum so that is initialized to 0. So, the HL pair they should point to the successive numbers, so in the numbers start from 2201.

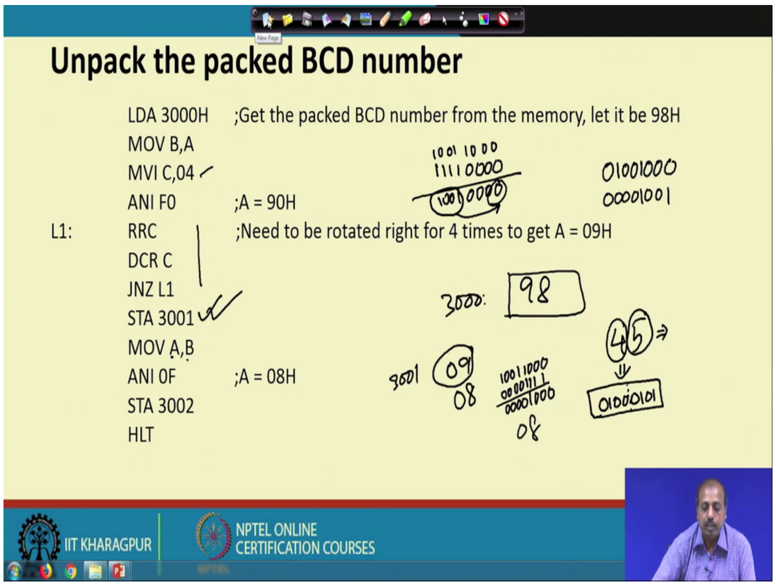
So, HL pair is initialized to 2201 move A comma M, so this will get the first number into the accumulator and then we and immediate with 01 H. So, that way the bits 1 through 7 so they will be masked out only bit 0 will remain because, they will be added with this 0 1, then if that bit is if that bit was also 0 then this jump on not 0 will fail, that means the number was even. But if the number is odd then this JNZ will be successful and in that case it will be jump it will not increment the sum value and it will come to this point.

But if the number is an even number then this check will be fail will be a failure, in that case this you will get the number into we will get the B register content into A register then we will be add M. So, this is basically with the previous sum we are stored we are adding the current even number and then storing back the result to the B register move a B comma A, so this will store the result of this addition into the B register. Then we increment H decrement the C register and then JNZ back, so if the C register value has not become 0 that is all the numbers are not checked, so it will be jumping back to this position and finally the number whether added values. So, they will be stored at the location 2201 H.

So, this way this program is doing the addition of only the even numbers from a series of numbers. So, we can find out we can write this type of programs in 8085, so whatever you have got whatever program you can think about in high level language. So, they can be written in assembly level or we can take help of some compiler and assembler to convert the high level program into assembly language program.

The advantage with the assembly language programming will be you know the exact size of the program, you also know the exact amount of time that will be taken for executing this program. So, if you consult the 8085 manual, then you can find out the individual number of bytes needed for individual instructions to compute the size of the program and you can also find out the total number of clock cycles needed by individual instructions. So, you can trace through the algorithm trace through the program and compute what will be the exact time needed for executing this program using 8085. So, that is how that is why we can take help of this assembly language programming.

(Refer Slide Time: 12:14)



Unpack the packed BCD number

```
        LDA 3000H    ;Get the packed BCD number from the memory, let it be 98H
        MOV B,A
        MVI C,04
        ANI F0       ;A = 90H
L1:     RRC          ;Need to be rotated right for 4 times to get A = 09H
        DCR C
        JNZ L1
        STA 3001
        MOV A,B
        ANI 0F       ;A = 08H
        STA 3002
        HLT
```

Next we will be looking into another program which will be converting a packed BCD number into an unpacked BCD number. So, packed BCD number is like this say suppose we want to we want to so we have the number say 98 ok. So, this 98 is stored in the in the location 3000 in the memory location 3000 this number 98 is stored. Now this is stored in a packed BCD format, so BCD stands for binary coded decimal. So, if you if you take any decimal number say 45 x then it consists of 2 decimal digits 4 and 5 ok

So, 1 possibility of storing this numbers is that you code this 4 into binary numbers, so 0100 and code this 5 into binary number so that is 0 1 0 1. So, this number so this is a BCD number BCD coded form of the decimal number 45. So, we can instead of converting this into binary number, so we can we can store them in the BCD format binary coded decimal format and this has got several utility while you are particularly displaying some decimal digits on to some display so this format often useful.

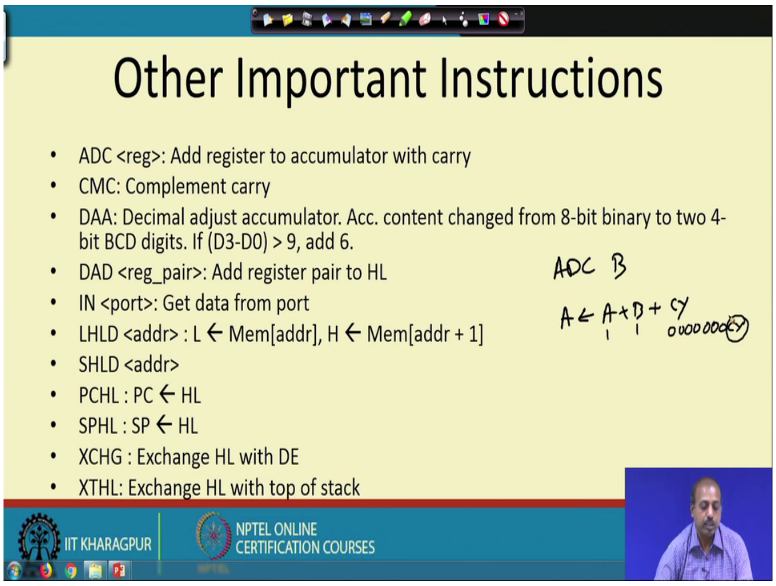
So, many a times what happens is that we store 2 BCD digits per memory location. So, if you memory locations that generally byte organized so, per memory location we store 2 BCD digits. But, in the application so it may be required that we access them individually each BCD digit we accept individually, so that is known as the unpacking of packed BCD number. So, when they are stored 2 BCD number 2 BCD digit per 8 bit pattern so that is called a packed BCD notation and when you are giving entire 8 bit for 1 number, that is when we are converting say for example this 98 when you are converting into 0 9 in 1 first byte and 0 8 in the second byte, so that is the unpacked BCD numbers. So, how do we do this so you see first in the assuming that the number is 98 H, so we first move this and it is stored at the location 3000. So, IDA will get the number into A register, so A register is now holding 98 now that is moved to B register and this in the C register we move the value 0 4 ok.

Now, we do an and immediate F 0. So, if you do and immediate F0, then F is 1 1 1 1 and 0 is this is 0000. So, this is the anded with 98. So, so 9 is 1 0 0 1 and 8 is 1 0 0 0 so if you do an and immediate so you will get 1 0 0 1 0 0 0 0. So, you get the value, so you get the first digit of the BCD number. Now we have to , but we have to get this we want to get 0 9 from here for getting 09 so I have to rotate this right by 4 positions. So, this is exactly done in this part of the program, so RRC will shift this thing by 1 bit position so this 0 will come.

So, this 0 will come to the most significant position then it will become 1 0 0 1 and then 0 0 0. So, it after 1 RRC so we will get the content like this , then we decrement C and jump on not 0 to l 1 so that means, if the C value was been initialized to 4. So, it will do 4 such rotations after 4 such rotations I will get the content as 0000 1001 so that is 0 9. Then we store the value at memory location 3001, at 3001 we store the value 09; so this is stored at location 3001 and then we have to do the similar thing now we have to get the value 8 ok. So, B register was already having the value 98, so we are again getting the value of moved to A register and now and immediate is 0F, so this 1 0 0 1 1 0 0 0.

So, if you and immediate with 0 F so you will get you will get so this 4 bits will becomes 0 and this will become 1 0 0 0 so you will get 0 eight. So, now we do not have to rotate anything because, this is 8 is already in the least significant position so we just store the number in the location 3002. So, this way I can unpack a packed BCD number into individual digits so this is another example. So, this way there can be innumerable number of programs that you can try out in 8085 assembly language.

(Refer Slide Time: 17:08)



Other Important Instructions

- ADC <reg>: Add register to accumulator with carry
- CMC: Complement carry
- DAA: Decimal adjust accumulator. Acc. content changed from 8-bit binary to two 4-bit BCD digits. If (D3-D0) > 9, add 6.
- DAD <reg_pair>: Add register pair to HL
- IN <port>: Get data from port
- LHLD <addr> : L ← Mem[addr], H ← Mem[addr + 1]
- SHLD <addr>
- PCHL : PC ← HL
- SPHL : SP ← HL
- XCHG : Exchange HL with DE
- XTHL: Exchange HL with top of stack

ADC B

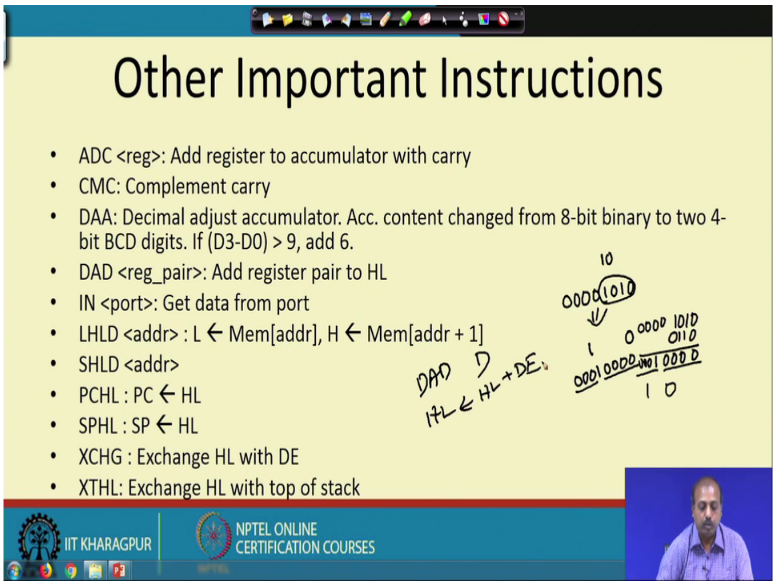
$A \leftarrow \underset{1}{A} + \underset{1}{B} + \underset{0000000\boxed{CY}}{CY}$

So, there are some more important some other important instructions that we might not have covered in our course, but you can just have a have a look at them in any manual for 8085, some of the interesting instructions that we have is 1 is ADC. So, add register to accumulator with carry so because, we have got the add instruction which does not take at the carry into the number, but when you have got multi byte addition so then in that case this add with carry maybe essential ok.

So, this add with carry so this also adds the carry bit like if you say ADC B, so what we what we mean is a we will get A plus B plus the carry flag and this carry flag will be will be the least significant bit. So, this A and B so they are 8 bit values and this carry will be converted into an 8 bit value like this and this is the carry. So, whatever be the carry so that will come at the least significant position and they that will form another 8 bit, so those 3 bit 3 will be added.

Then there is another important instruction which is CMC or complement carry. So, complement carry so this it just complements the carry, so if the carry was 0 it becomes 1 if the carry was 1 it becomes 0. Then there is another instruction which is known as DAA which is decimal adjust accumulator. So, it is for converting 1 8 bit binary number into 2 4 bit BCD digits. So, it is for you can say that converting 1 binary number into the BCD numbers, so DAA is an instruction for doing it. So, in our digital circuit combinational circuit classes, so we have seen that if for converting 1 number to BCD so we need to add 6 if the number is greater than 9, because if the number is say for example if the number is say 10 ok, so in binary notation if I use an 8 bit notation so this will be stored as 0 0 0 0 1 0 1 0.

(Refer Slide Time: 19:28)



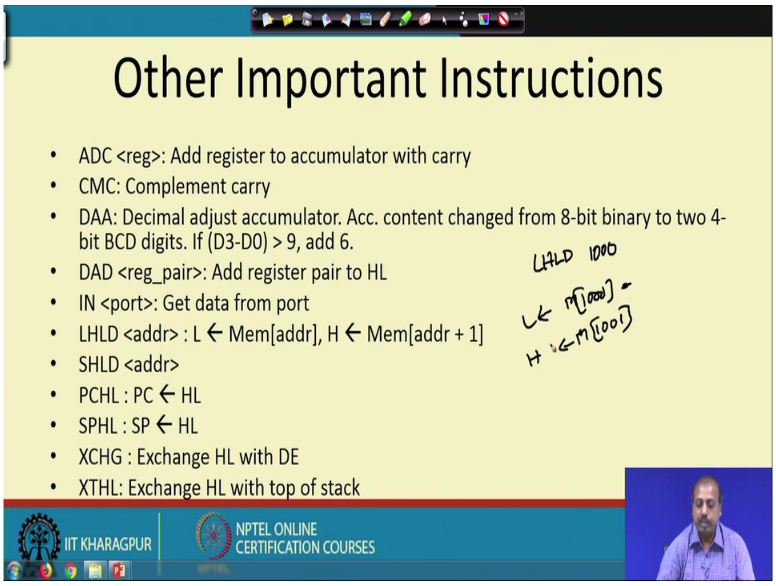
Other Important Instructions

- ADC <reg>: Add register to accumulator with carry
- CMC: Complement carry
- DAA: Decimal adjust accumulator. Acc. content changed from 8-bit binary to two 4-bit BCD digits. If (D3-D0) > 9, add 6.
- DAD <reg_pair>: Add register pair to HL
- IN <port>: Get data from port
- LHLD <addr> : L ← Mem[addr], H ← Mem[addr + 1]
- SHLD <addr>
- PCHL : PC ← HL
- SPHL : SP ← HL
- XCHG : Exchange HL with DE
- XTHL: Exchange HL with top of stack

Now you see that when I am converting in into it into BCD notation, so what is required is that this number will be converted to 1 and 0 that is 0 0 0 1 in the first 4 bits and 0 0 0 0 in the next 4 bit. So, that is what we want and then this can be done this can be done by doing the logic is that when this part of the number is becoming greater than 9, so we have to add 6 to it so this number is 10. So, 1 0 1 0 so with that with that if we add 6. So, that is 0 1 1 0 then the number will become 0 0 0 0 and then 1 and these 3 bits remain 0, so you get this 1 0. So, this way we can get the number as 10 ok, so when the number is greater than when this least significant 4 bits are greater than 9, so we can just add 6 and get the number.

So, this way this decimal adjust accumulator, so this is useful when you are converting a binary number into BCD number, then there is a double addition or DAD instruction, so that is for adding registered pair to HL so this register pair value. So, you can say like DAD D, so this HL pair we will get HL plus DE ok. So, this is the DAD instructions the in instruction so far with this we have already discussed for getting some data from some input port, so you can use this in instruction. Then there is LHLD instruction, so LHLD with some address, so in that case the L register the gets the content of the memory location which is pointed to which is given in this address.

(Refer Slide Time: 21:31)



Other Important Instructions

- ADC <reg>: Add register to accumulator with carry
- CMC: Complement carry
- DAA: Decimal adjust accumulator. Acc. content changed from 8-bit binary to two 4-bit BCD digits. If (D3-D0) > 9, add 6.
- DAD <reg_pair>: Add register pair to HL
- IN <port>: Get data from port
- LHLD <addr> : L ← Mem[addr], H ← Mem[addr + 1]
- SHLD <addr>
- PCHL : PC ← HL
- SPHL : SP ← HL
- XCHG : Exchange HL with DE
- XTHL: Exchange HL with top of stack

LHLD 1000

$L \leftarrow M[1000]$

$H \leftarrow M[1001]$

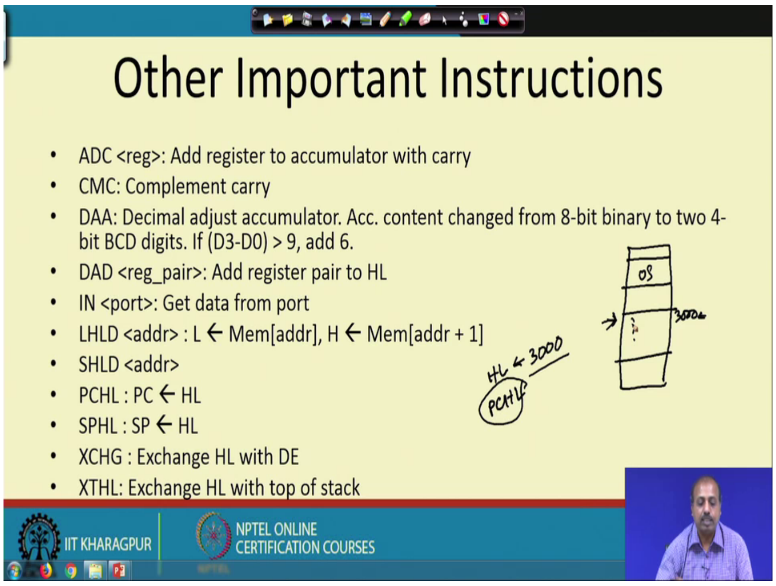
IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

For example if you say like say LHLD LHLD 1000, then content of memory location 1000 will come to the this content will come to the L register and the content of memory location 1001 will come to the H register, so this way we can get this LHLD. So, this will load the content of 2 memory locations into HL pair, then we have got SHLD instruction. So, which is just the opposite of LHLD, so instead of loading the HL pair, so it will be storing the value of HL pair into the corresponding addresses.

Then there is another very useful instruction which is known as PCHL, so this PCHL instruction the program counter is loaded with the value of HL register pair. And this is very much useful when you are trying to execute some other program, particularly from the operating systems point of view this PCHL is very very much useful.

(Refer Slide Time: 22:33)



Other Important Instructions

- ADC <reg>: Add register to accumulator with carry
- CMC: Complement carry
- DAA: Decimal adjust accumulator. Acc. content changed from 8-bit binary to two 4-bit BCD digits. If (D3-D0) > 9, add 6.
- DAD <reg_pair>: Add register pair to HL
- IN <port>: Get data from port
- LHLD <addr> : L ← Mem[addr], H ← Mem[addr + 1]
- SHLD <addr>
- PCHL : PC ← HL
- SPHL : SP ← HL
- XCHG : Exchange HL with DE
- XTHL: Exchange HL with top of stack

For example if I have got say in my memory my program to the user program starts from location 3000 and the OS part is loaded up to the location this part is got OS now. How does OS transfer the control to the user program? So, this can be done by means of this PCHL instruction because for executing this users program, so somehow the value 3000 has to go to the pc register now how will it go. So, it is there is no direct instruction to load the PC register, so what the program will do it will what the operating system does is that it once it knows that the start address is 1000, somehow in the HL pair it loads the value 3000 and then it executes the PCHL instruction. So, that PC now gets the value of this HL pair so we get this PC value. So, this now once the PC has got the value 3000, so it automatically starts executing from this point.

Similarly, we have got this SPHL instruction in which the HL pair is saved into the stack pointer, so this stack pointer is loaded into the HL pair. So, this is another this PC register and sp register they cannot be loaded directly so we can load it through the HL register pair. Then there is an exchange instruction so that exchanges the HL value with the de value. So, this is also useful in some cases so this is exchanged, so this there is no the operand. So, it just exchanges the value of HL with the DE and we have got XTHL instruction. So, it exchanges the HL register with top of the stack. So, top of the stack content is exchanged with the HL pair.

So, this way these are some of the other instructions that we have for the 8085 programming. So, if you are so when you are you are some this instructions as we have seen. So, there more relevant when you are considering system programming into consideration; for example, when you are this instruction like PCHL SPHL then XTHL, so these instructions are useful when you are considering um system programs.

On the other hand this ADC this ACMC this type of instructions are useful when I doing some computation and if you are having some decimal interface with your system then this DAA instruction is useful, that is for converting any number into it is BCD format and for unpacking we have already seen another code which can unpack a BCD number packed BCD number into unpacked BCD number. So, we can refer to any book on 8085 that has got this whole discussion on this programming and all.