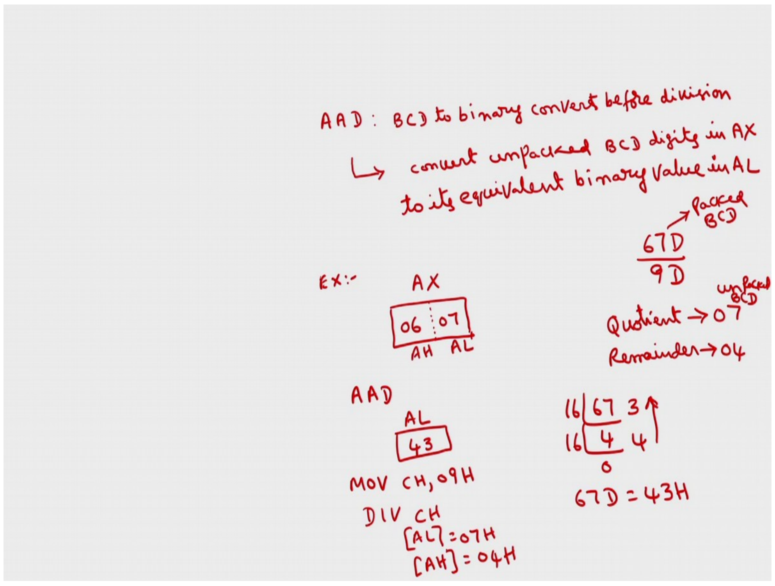
Microprocessors and Interfacing

Prof. Shaik Rafi Ahmed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Lecture – 09
8086 Logical Instructions

So, in the last class we are discussing about the arithmetic group of instructions, so in that we have discussed about the addition, subtraction, multiplication, division, and then we have discussed about this decimal adjust accumulator, decimal adjust after subtraction. Then we are discussing about some ASCII instructions, AAA, AAS, AAM and the last ASCII instruction is AAD.

(Refer Slide Time: 00:53)



Even though here A stands for ASCII, the operation that we will perform here in AAD is BCD to binary conversion; BCD to binary convert before division. So, the remaining instructions, there will be a user after this arithmetic operations such as addition, subtraction, multiplication, whereas this is before division, ok.

So, here what happens is in this AAD, so this will convert unpacked BCD digits. I have already discussed what is unpacked, means only one BCD digit will be present in each byte. Unpacked BCD digits in AX, this implied register is AX to its equivalent binary value in AL, this is also implied. So, this instruction basically converts unpacked BCD digits in AX into its equivalent binary value in AL.

If I give one example, this will be clear. Suppose if you want to divide 67 decimal divided by 9 decimal say. So, what is expected quotient and remainder? So, quotient will be 07, if I take unpacked BCD, this is unpacked BCD and remainder should be 04, this is packed BCD.

Now, if I want to perform this and if I want to get these results, we will use this AAD instruction and I will explain how this AAD instruction will be used to get the BCD quotient and remainder of BCD division, ok.

So, first I will take this 6 7 into AX register. So, we can use the MOV instructions, AX will be loaded with unpacked BCD. So, AX will be having AH and AL, this is AH and this is AL. So, AH we are going to take 06 and AL we are going to take 07. This is called unpacked BCD digit. So, the 67 packed BCD I am going to take in two 8 bit registers in unpacked form, 06, 07. Then, if I write after taking this into AX register if I write AAD, then what happens in AL?

We will get the binary equivalent of this. So, what is the binary equivalent? So, the 67 decimal, hexadecimal, if I convert into binary value 67 if you divide with 16, 16 4s are 64, 3 is the reminder 16 0's are 4; 4 3 H. So, 67 decimal is equal to 43 H. So, in AL 43 H will be present.

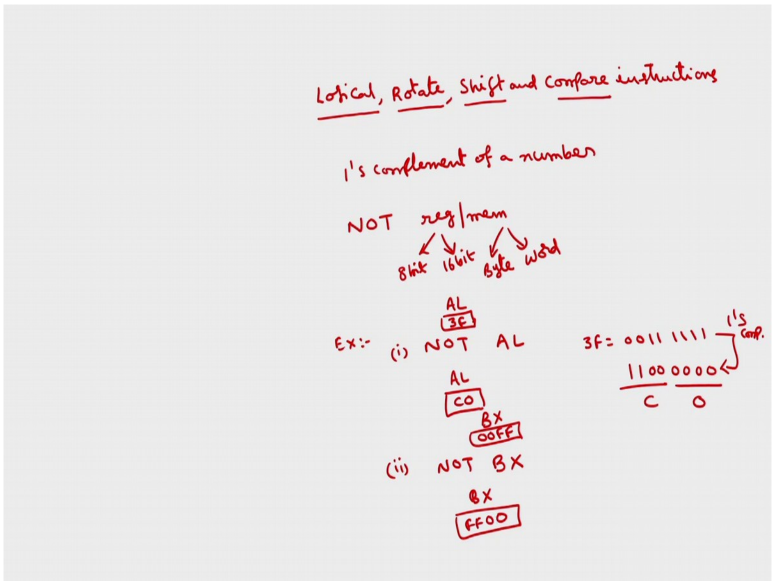
This is about this AAD instruction. So, this AAD instruction basically takes two unpacked BCD digits into AX register if I write AAD. In AL, this I mean binary equivalent means hexadecimal, hexadecimal equivalent of this one will be stored in AL.

Suppose, if want to perform this 67 by 9 BCD, so, we have to load MOV any register with CH with say 09 D, it will take a hexadecimal. Then if I write DIV, as I have told this is before division. So, here what happens is what is the quotient and what is the remainder? So, in AL you will get, contents of AL becomes the quotient which is 07 H and contents of AH becomes 04 H. So, this is about the AAD instruction.

I will explain again. So, this AAD instruction basically converts unpacked BCD digit in AX into its equivalent binary value in AL. So, binary in the sense, here this hexadecimal value. So, if I take the example of 67 decimal by 9, if I want to perform this BCD division then this packed BCD you have take into unpacked 2 8 bit registers AH and AL which is called AX. Then, 9 you can take into any other register then you can perform the division, but the thing is this division instruction, the microprocessor instructions assumes the data that is given is in hexadecimal form, but this is actually decimal. So, in order to convert this decimal into equivalent hexadecimal 43 so, we will use the instruction AAD. If I write the instruction AAD the contents of AL becomes the hexadecimal or binary equivalent of 67 H is 43.

Now, this will perform 43 by 9. So, because this 43 is the hexadecimal equivalent of this decimal, so, this will give quotient and reminder as 7 and 9. And we know that in division DIV CH, we have to write DIV CH. So, in DIV CH what happens is internal operation of this we have already explained in the last class. So, AX contents will be divided with CH contents, quotient will be stored in AL and reminder will be stored in AH. So, this is about this AAD instructions. So, these are the different, I mean arithmetic instructions in 8086. The next group is logical group.

(Refer Slide Time: 07:21)



Logical, Rotate, Shift and Compare instructions

1's complement of a number

NOT reg|mem

8bit 16bit Byte word

AL 3F

Ex:- (i) NOT AL

AL CO

3F = 0011 1111

1's comp.

1100 0000

C 0

BX 00FF

(ii) NOT BX

BX FF00

Next group of instruction is logical, rotate, shift and compare instructions. So, in this group we will discuss about various logical expressions rotate, shift and compare instructions, coming for the logical instructions. So, what are the various logical operations we have? So, one is we can take the 1's compliment of a number or we can take the 2's complement of a number. We can take the OR operation between the two numbers, AND operation, exclusive OR operation. So, these are the various operations correspond to each operation we have the 8086 instructions.

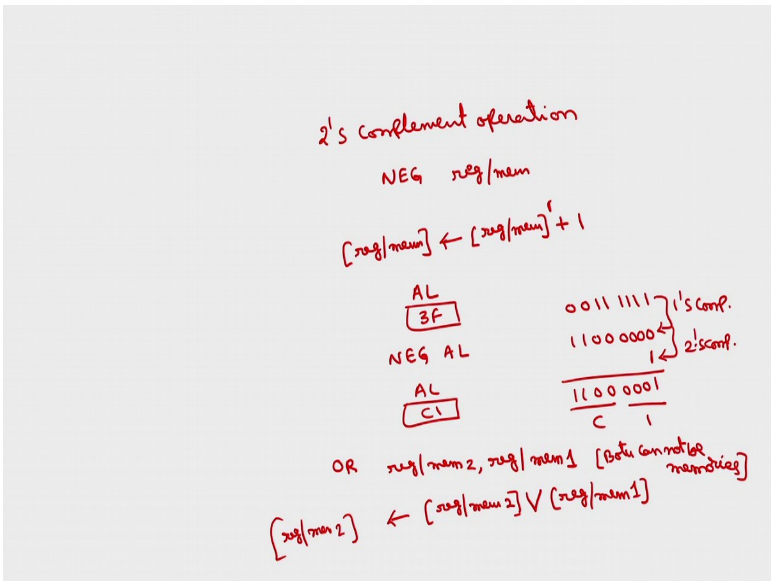
First, if you take the 1’s complement if you want to find out the 1’s complement of a number, we have direct instruction which is called NOT, if you write NOT we have to specify register slash memory. So, this register can be either 8 bit or 16 bit, memory can be byte or word. Byte means 8 bit, word means 16 bit.

So, what about the register or memory that we are going to specify in the instruction? The 1’s complement of that will be evaluated and it will be stored back into that particular register only, ok.

So, if I take the example, NOT if I write AL before the execution of this instruction if AL contents is some 3 F H. So, after this what happens is 3 F H will be complemented. What is 3 F H is 0011 1111. If you take the 1’s complement of this 1’s will be changed to 0, 0’s will be changed to 1; 0000 0011. So, this will be C, this will be 0. So, after this the contents of AL becomes C 0. This simply determines the 1’s complement of the register or memory location that is specified here.

Again wherever this throughout this 8086 instruction wherever memory means, the memory location that can be accessed by any of the addressing modes that we have discussed in the earlier classes. Similarly, you can take 16 bit register. So, if I take NOT BX, if BX contents is 00FF, the complement of 1’s complement of 0 is F, 1’s complement of F become 0. So, after these contents of BX becomes FF00. This is about the 1’s complement.

(Refer Slide Time: 10:39)



2's complement operation

NEG reg/mem

$[\text{reg/mem}] \leftarrow [\text{reg/mem}]' + 1$

AL $\boxed{3F}$

NEG AL

AL $\boxed{C1}$

```
0011 1111   1's comp.
1100 0000
        1   2's comp.
---------
1100 0001
  C    1
```

OR reg/mem 2, reg/mem 1 [Both cannot be memories]

$[\text{reg/mem 2}] \leftarrow [\text{reg/mem 2}] \vee [\text{reg/mem 1}]$

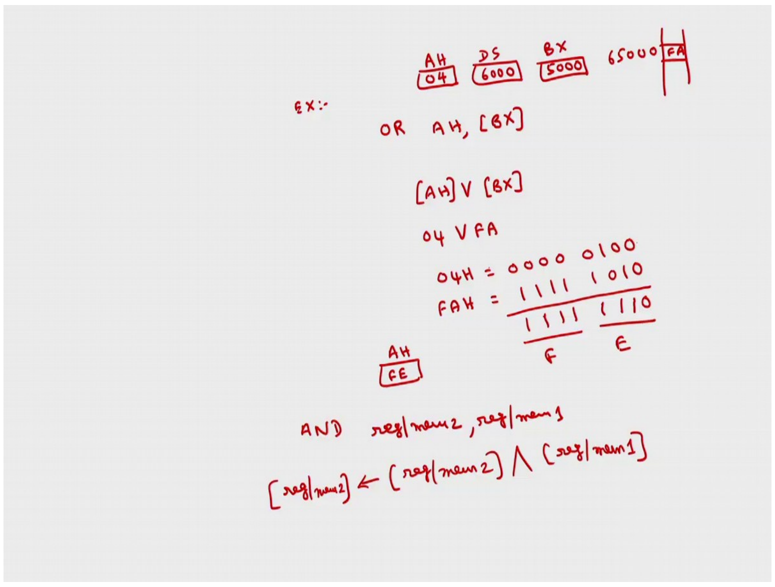
Similarly, we have instructions for the 2’s complement. This is 1’s complement plus 1, so the instruction for this one is negate, NEG. NOT is for the 1’s complement and negate is for 2’s complement. Same format will be same as that you have to specify either register slash memory. Register can be 8 bit register or 16 bit register, memory can be a byte or word. This simply take the 2’s complement of the register or memory, register or memory, 1’s complement is dash plus 1 will be stored back into register slash memory.

Here all the flags will be affected including overflow flag also. So, there are some control flags which will not be affected by the instructions. So, all the conditional flags will be affected by this particular instruction. So, we can explain the same in a similar manner. If AL contents is 3 F H, if I write negate AL, then 2’s complement of 3 F will be 0011 1111, 1’s compliment will be 1100 0000, 2’s complement will be we have to add 1. So, this becomes 1100 0001, this is C 1. So, the contents of AL becomes C 1.

Similarly, we have the other logical operation such as OR, AND. So, the next operation is OR operation. OR, the general format of OR instruction is OR, you have to specify two operands, so each operand can be taken in either register or memory. So, we have register slash memory 2 comma register slash memory 1, in all these instructions both cannot be memory simultaneously, ok. So, here we can have one of this one should be registered, say that register comma memory or memory comma register or register comma register, memory to memory is not allowed. So, in this instruction we have 3 possibilities. This can be registered source can be a destination can be register source can be memory or vice versa or both can be registers both cannot be memories.

So, it is basically this will perform register slash memory. 2 contents will be logically bit by bit odd with the contents of this register slash memory 1. Result is stored back into the destination which is memories, register slash memory 2. Both cannot be memories simultaneously. Again, the memory means the memory location which can be accessed by using any of the addressing modes.

(Refer Slide Time: 14:01)



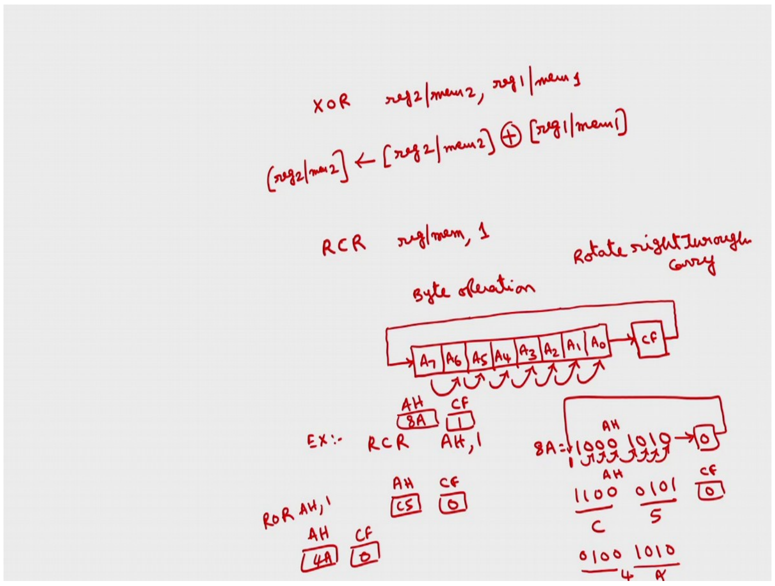
If I give an example of this OR operation, if I write OR AH comma contents of say BX So, now this is destination is register source is memory. So, before these contents of AH is something like 04 H and contents of DS is 6000 H and contents of BX is 5000 H, then the physical address will be DS into 10 plus this one, 65000. The contents of 65000 memory location is some FA H.

So, here this operation is going to logically OR the contents of AH with the contents of the memory location which is pointed by BX register. So, in this we have a physical address of 65000, in that 65000 we have FA. So, this operation will be AH contents will be logically ORed with the contents of BX. So, AH contents will be 04 logically ORed with FAH.

So, if I take logical operation in terms of the bits 04 H is 0000 0100 and FAH will be 1111 1010, 10. So, bit by bit you have to OR. If both the bits are 0's then only output is 0 in for OR gate, if anyone input is 1 output is 1. So, 0 0 means output is 0, this is 1, this is 1, 1 and anyhow this 4 are 1's. This will be F this will be E. So, after the execution of this one the contents of AH Result is stored back into this destination, register in this case this can be memory also FA H. This is about the OR operation. We can OR the contents of any two registers. We can OR the contents of register with any memory. And you can store the result back into either memory or register.

Similarly, we have AND instruction, exactly similar operation. Here instead of a OR, AND operation will be performed. AND register slash memory 2 comma register slash memory 1. So, basically the operation is register slash memory 2 is logically AND with register slash memory 1 and result is stored back to register slash memory 2. No need to give the example here, this is exactly similar to this OR operation except that here instead of OR operation AND operation will be performed.

(Refer Slide Time: 17:25)



Similarly, we have exclusive OR operation also, XOR. Symbol for this one is XOR, same thing register 2 slash memory 2 comma register 1 slash memory 1. So, the same operation. The contents of register 2, both cannot be memories here also in all the instructions both cannot be memory simultaneously.

Memory 2 will be exclusive OR with this is symbol for exclusive OR the contents of register 1 slash memory 1 and the result is stored back to register 2 slash memory 2. So, this is all about 5 logical instructions, 1's complement, 2's complement, OR operation, AND operation, exclusive OR operations. There are some rotate instructions. The next group is here rotate instructions.

So, we can rotate the contents of any register or any memory location either to the right or left. So, we know that in digital if I if I shift these contents towards the right by 1 bit position, so it will becomes division. If I shift the contents towards left by 1 bit position that will becomes multiplication. This is the beauty of this shift operation. So, we can shift the contents of any register or in any memory towards right or towards left by any number of the positions.

First we will start with a simple 1 bit rotate RCR, rotate the contents of any register or memory through right along with the carry. So, this is general format of this rotate instruction is RCR, after that you have to specify register slash memory comma 1. If I want to rotate by 1 bit position you have to specify here this one. And which location which contents has to be rotated by 1 bit position? The contents of whether register or memory. Again, this register can be 8 bit register or 16 bit register, memory can be a byte or word, ok.

So, if I take the example of this actually operation here will be, if I take byte operation. So, there will be total 8 bits, this is A naught, A 1, A 2, A 3, this can be contents of any register or memory 8 bits, A 4, A 5, A 6, and A 7. So, there are totally 8 bits. And we will be having carry flag also, suppose here the carry flag is stored.

So, this is going to shift the contents of this one towards right by 1 bit position, so in that the contents of A 7 will be shifted to the A 6 right position, A 6 to A 5, A 5 to A 4, A 4 to A 3, A 3 to A 2, A 2 to A 1, A 1 to A naught, A naught will goes to the carry flag and the carry flag contents will goes to A 7. This operation is called through carry. This is called rotate operation through carry, rotate right through carry.

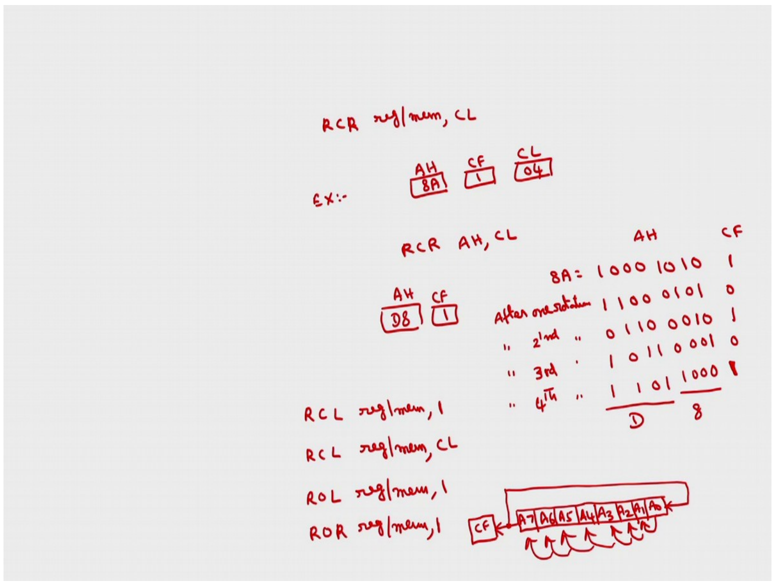
So, for example, if I take an example of this, RCR, some AH comma 1. Before this if the contents of AH is some 8 AH, 8AH. Now, what will be the contents of AH after RCR AH comma 1? Result will be stored back to this register slash memory. So, 8 A is nothing, but 1000 1010 and if I assume that the carry flag here will be initially set. Then what happens is this will move to the carry flag, carry flag becomes 0, the previous carry flag contents will be 1 that will comes here, that carry flag contents will come to here which is 1, so this will move here, this will move here, this will move here, this will move here, so like this.

So, what will be contents now after this this becomes 1100 0101, this 1's last 0 has been entered into this carry flag. This will be contents of AH that the contents of the before the rotate instruction and this is after rotate instruction. And what happens to carry flag? Will be reset. So, this will be C 5. So, contents of AH will be C 5 and carry flag will be reset. This is how this rotate instruction will be executed.

So, the contents of 16 bit register the similar operation will take place, the contents of A 15 will become the contents of carry and A 0 will be in any case will transfer to the carry flag, ok. Similarly, we can have this as a byte from the memory location or word from the memory location. This is you found I mean shift by 1 bit position, if I want to shift by 4 bit position instead of writing this 4 times we can directly mention that 4 into a register called CL which is implied register.

So, for multiple bit rotations, for single bit rotation we have to mention 1, for multiple bit rotation you have to write RCR register slash memory comma CL. So, whatever the you have to store in CL that many number of times the contents of this register or memory will be rotated to the right, rotated to the right RCR.

(Refer Slide Time: 23:51)



So, exactly same operation, but here this CL if I take the example of this one if you take the same example say AH is having some 8 A and carry flag is set. And if CL is having say some 4, then this will rotate 4 times. If I write RCR AH comma CL then after this operation will be 8 A is equal to 1000 1010 and carry flag is initially set this is AH contents, this is carry flag contents, this was initially set.

So, after one rotation, what will be the contents of this one? If I say have explained in this previous class the contents of carry will come here and each bit will move towards right way on 1 bit position. So, this becomes 1100 0101, carry becomes 0, this is after one rotation. After second rotation, again this carry if you have you will comes here 0110 and this all bits will be shifted to this one 0110, so 0010, this carry becomes 1, this 1 will be shifted here.

After third rotation. So, all these bits will be shifted here. This will become 011 0001 and this 1 will becomes carry, this 1 becomes this and after fourth rotation, so this previous carry flag 1 will come here, so 1101 1000 on this 1 becomes this. So, after this instruction the contents of accumulator AH will be this is D and this is 8 and carry flag will be set. So, AH will be having D 8 and carry flag will be set. This is how we can rotate the contents of any register or memory by multiple number of the bits. The number of bits time that has to be shifted has to be taken into CL and this is default register, you cannot take into any other register. So, CL contains the number of rotations to be done and we have to specify here a register or memory location this is about RCR register slash memory CL.

Similarly, we have this is right operation we have left operation also we have two instructions for left. RCL register slash memory we can shift by 1 bit position, the contents of the either this register or memory will be shifted to the left by 1 bit position through carry this is through carry, and we can have RCL register slash memory comma CL. If I want to rotate through carry the contents of register or memory to the left by CL number of times. So, there are about 4 rotate instructions through carry, so two instructions will be for single byte rotation and the other two instructions will be for multiple byte rotations. But all the 4 instructions will be through carry.

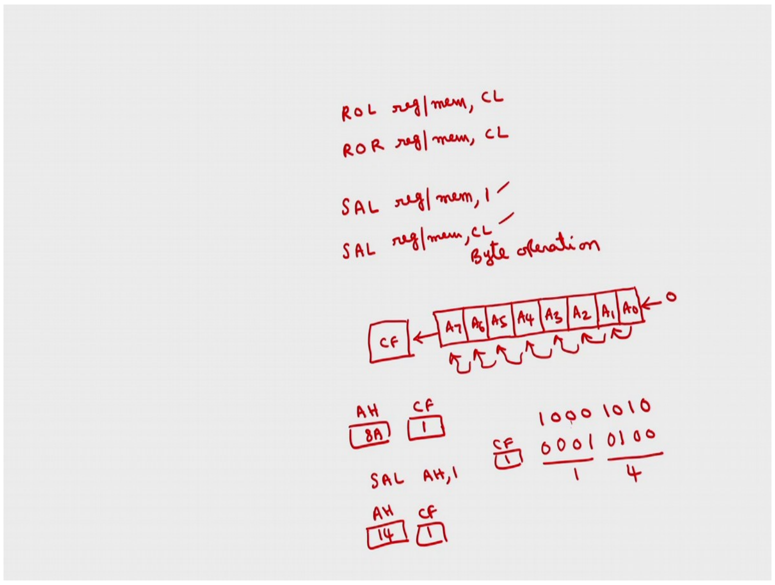
And if you want to perform this rotate operation without carry, there are 4 more instructions which will be ROL. They correspond to RCL, there is an instruction called ROL. The same format register slash memory comma 1. The difference between these RCL and ROL is in case of ROL, this is A naught A 1, A 2, so on. So, this will be rotated to the left. So, A naught here somewhere carry flag will be there. So, this A naught will goes to A 1, A 1 to A 2, A 2 to A 3, A 3 to A 4 and so on.

In case of RCL, the contents of A 7 will go to carry, the contents of carry will go to A naught whereas, in case of ROL the contents of AC will go to see carry flag as well as A naught, ok. This is only difference. In case of RCL, this A 7 will goes to carry the carry contents will goes to A naught, whereas, in case of ROL A 7 will goes to carry as well as A naught, ok.

So, if I take the example of the same 8 A and similarly we have ROR register comma memory comma 1. So, this is exactly same this operation, but the contents will be processed without carry, ok. So, if I take the same previous example and if it is ROR here. So, if I take this instead of RCR if we have ROR what will be the contents of this I mean AH register after this if carry flag is set before this? What will be the contents? The same operation, so the contents will move here. So, this will move into the carry flag as well as this. So, what are these contents, last contents? 0 is moving to here as well as here. So, what happens here? This becomes 0 this is 8 is 1000 0100 1010 and carry flag is anyhow 0.

So, this content becomes 4 A, ok. The contents of AH becomes 4 A and carry flag will be reset in both the cases. If this is ROR AH comma 1. Now, I think you understand the difference between RCR and ROR, ok. So, the contents of the last bit will goes to LSB bit will goes to carry as well as MSB in case of ROR whereas, in case of RCR the contents of LSB bit will go to the carry, carry bit will goes to the MSB position. This is for single bit rotation. We have multiple rotations, similar to this we can have write CL.

(Refer Slide Time: 31:53)



So, the two multiple byte rotation of contents of register or memory, we can write ROL register slash memory comma CL, this is without carry; ROR register slash memory comma CL. So, these are for rotating the multiple number of times. So, these are the different rotate instructions. So, we can rotate through the carry without carry also. We can rotate by only 1 bit position or we can rotate by multiple number of bit positions.

Similarly, we have shift operation. The next one is shift operations. SAL register slash memory comma you can rotate we can shift by 1 bit position or you can shift by multiple bit positions, similar to the rotating instruction. So, this is also exactly similar to the rotate instruction except for the difference that, so here we have register can be 8 bit or 16 bit memory can be byte or word. So, if I take a simple byte shift operation and the shift left, L stands for left. So, this diagram will explain the operation of this SAL this is A naught, A 1, A 2, A 3, A 4, A 5, A 6, A 7.

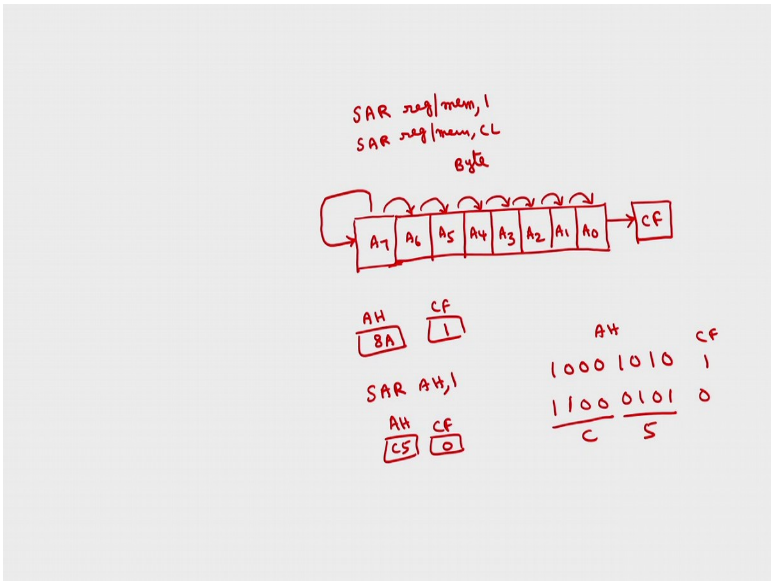
So, in case of SRL, we are going to rotate RLL. So, if it is a instead of shift if it is a rotate operation same the contents of A naught will moves to A naught, A 1 to A 2 A 2 to A 3, A 3 to A 4, A 4 to A 5, A 5 to A 6, A 6 to A 7, ok. Here also same operation. But the only difference here will be the contents of this A 7 will be in case of shift operation. This will move to carry flag. Whereas, in case of rotate instruction in one case this A 7 will goes to A naught, in other case the carry flag contents will goes to A naught, whereas, in shift operation this is open loop configuration. In case of rotate we have closed loop whereas, here we have open loop. Here so many 0's will be added, A naught will be feeded with 0's, if this is 1, ok

So, now to understand this operation. So, if I give an example. Suppose, if say same AH is equal to 8 A if carry flag is set. If I write SAL AH comma 1 then 8 A will be 1000 1010 and if you rotate left by 1 bit position, so this becomes this 0 here and here so many 0's will be added. So, this 0 will move here, this 1 will move here, this 0 will move here, this 1 will move here, this 0 will move here, this 0 here, this 0 here, this 1 will be moved to the carry flag, initially also carry flag set now also it remains set.

Now, here so many 0's will be added, so here this position will be loaded with 0's. So, this becomes 1 4. So, the contents of AH becomes 14 H and carry flag will be anyhow set. This is about shift left operation and how this will be different from the rotate left operation.

So, in rotate left operation what happens is the contents of A 7 will move to C set carry flag and then carry flag to the LSB or A 7 contents to the A naught whereas, here so many 0's will be added at the A naught position. Similarly, if I want to shift by multiple positions, so if you have this SAL register comma memory comma CL. Here also you have take this number of times the shift operation to be performed in CL, at CL has to be mentioned here. So, this is for single bit shift, this is for multiple bit shift. Similarly, we have two shift I mean right operations, one for 1 bit position, another for multiple bit positions.

(Refer Slide Time: 37:01)



Means we have SAR register slash memory comma 1. Here there is some slight difference is there between the shift left operation and shift right operation, if I take say byte only same byte operation. This is A naught, A 1, A 2, A 3, A 4, A 5, A 6, A 7. So, the contents will be shifted to the right position. So, A 7 will goes here, A 6 will goes to A 5, and so on, this will go to the carry flag.

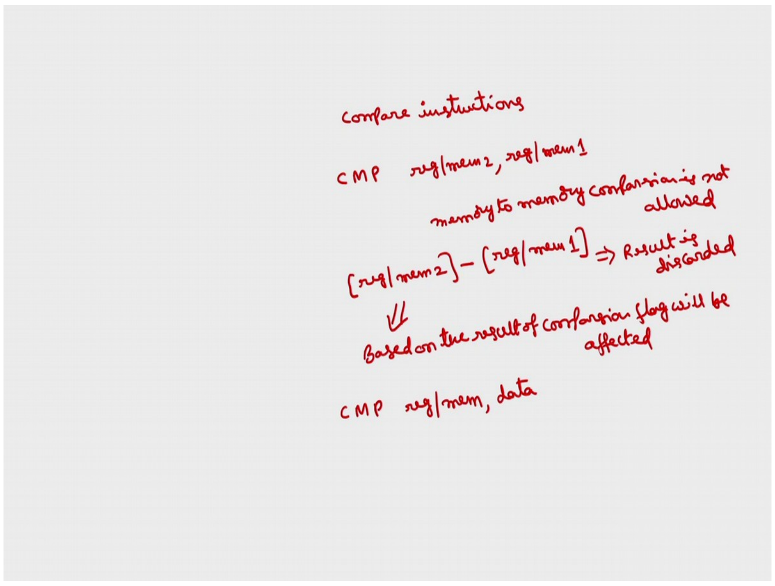
In case of left operation so many 0's will be inserted in A naught whereas, here the contents of A 7 itself will be loaded into A 7 also, MSB bit is simply copied, ok. Instead of inserting 0's, in case of shift left operation here the contents of A 7 the MSB bit will be moved to the MSB bit, so simply MSB bit will be copied, this MSB bit will be this bit as well as this bit, ok.

This is the main difference between shift left operation and shift right operation for single bit position. We have a similarly for multiple bit position you have SAR register slash memory CL, in all these shift operations and rotate instructions CL is by default stores the number of this to be shifted or rotated, ok. So, if I give now example of this SAR 1, if the contents of AH is same say 8 AH, if the carry flag is initially reset say.

If I write SAR AH comma 1 so what will be contents of this one? 8 will be 1000, A will be 1010, carry flag will be, this is AH, carry flag is set, this is before the instruction. After the instruction, what happens? This 1 will transfer to here, 0 to here, 0 to here, this 0 here, this 1 here, 0, 1, this 0 will goes to this carry flag, the same 1 bit will be copied here also, ok. So, this becomes C 5 and carry flag is reset. AH will be C 5 and carry flag will be, initial also reset now also because the last bit is 0; initially this carry flag is set if I take this as set, set then now it will be reset, ok. So, this is about shift right operation. The only difference is instead of inserting the 0's in case of left operation, in shift right operation the same MSB bit will be copied that is the difference between shift left and shift right operations. So, there are 4 shift operations, two shift left, two shift right, so 1 bit position or multiple bit positions.

And the last I mean instructions in this group is compare instructions. It is also one of the important operation.

(Refer Slide Time: 40:55)



compare instructions

CMP reg|mem 2, reg|mem 1

memory to memory comparsion is not allowed

[reg|mem 2] − [reg|mem 1] ⇒ Result is discarded

⇓

Based on the result of comparsion flag will be affected

CMP reg|mem, data

So, in compare instruction, this is general mnemonic is CMP compare. Then we can have register slash memory 2 comma register slash memory 1. We can compare two registers here also memory to memory operation is not in not allowed. Memory to memory comparison is not allowed. So, one must be register. So, we can compare one register contents with one memory or we can compare two register contents, ok.

So, the basic operation here will be, this is exactly similar to the subtraction operation, register slash memory 2 contents will be subtracted from register 1 come slash memory 1. So, the only difference between this and subtraction is in case of subtraction, result will be stored back into this destination register or memory whereas, here the result is discarded. So, this will be performed, but result is discarded.

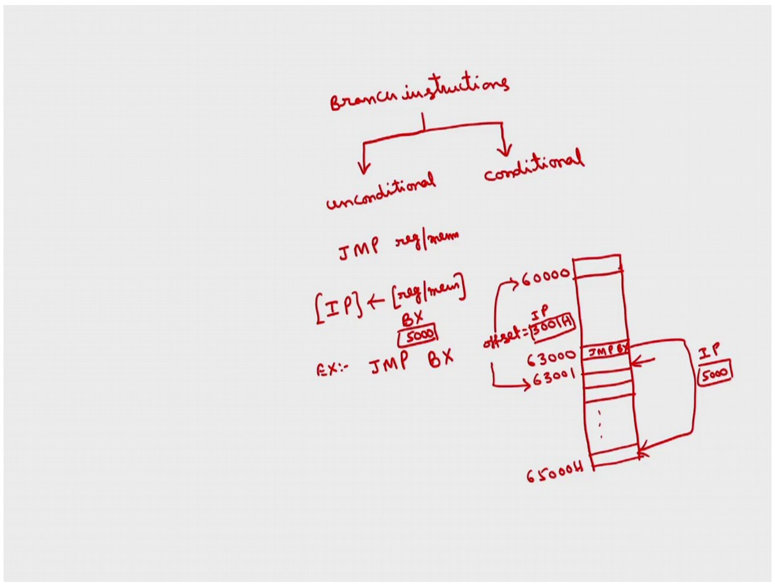
Now, based on this result, so it will set or reset the flags, based on the result of the comparison flags will be affected. So, based on this we can compare two numbers, if carry flag is set means there is a borrow, so this number is lower than this number. If carry flag is not set after this comparison, then what does it mean? The first number is greater than second number. If 0 flag is set after this operation means both the numbers are equal.

So, like that we can compare, which number is larger or whether these two numbers given two numbers are equal or not through this compare instruction. But in case of subtraction result will be stored back into the destination register or memory whereas, in case of compare the result is simply discarded except further there is no difference, ok, only result is discarded here, in case of subtraction result is stored. And we can compare directly with a data also, immediate data. The second compare instruction is register slash memory contents with data.

So, we can give 8 bit register or 16 bit memory can be either byte or word. You have to use correspondingly the data which can be 8 bit or 16 bit. If you give 8 bit register, data also should be 8 bit. If I use 16 bit register, data must be 16 bit. If I read a byte from the memory location data size should be 8 bit, if I read a word from the memory location data size should be word, means which is 16 bit. So, these are about different compare instructions, ok.

So, the next group of instructions is branch instructions.

(Refer Slide Time: 44:37)



If the microprocessor control transfer from one place to another place then you have to write the branch instructions. There are two types of the branch instructions, one is called conditional another is called unconditional. As the name implies unconditional without checking any condition, so this will branch to a particular location. So, the examples of unconditional are simply we can write jump, JMP comma we can specify register slash memory.

So, whenever the microprocessor execute this instruction, simply the instruction pointer will be loaded with the register contents, the instruction pointer will be simply loaded with the contents of register slash memory. So, we know the purpose of this instruction pointer which always holds the 16 bit offset from the starting all that particular segment to the which is to be executed next, ok, is always holds the address of the instruction.

Address in the sense here only the offset, the address of the instruction which is to be executed next, ok. If you want to execute the instruction which is located at this particular address then you have to write simply jump comma that address.

If I take example jump JMP some BX, if BX contents is 5000, so the microprocessor is executing we will come to the conditional instructions later, ok. So, this is code segment microprocessor is executing the instruction one by one. This is starting of code segment say some 60000 H, suppose the microprocessor is executing this instruction. So, this is jump instruction jump BX, if BX contents are 5000.

Suppose, if this is there in something like say 63000 H, so what is the offset here? The next location is 63001, ok. So, the offset from here to here will be, here to here offset will be 3001 H. This will be hold by instruction pointer. The instruction pointer contents becomes 3001 H which have explained in the earlier class also.

So, the instruction pointer always holds the address of the instruction which is to be executed next. So, this instruction needs to be executed next if this is not a jump instruction, so the microprocessor is supposed to execute this instruction. What is the off set of this instruction? It is 3001, that will be hold by instruction pointer.

Suppose, if you want to skip this instruction and if you want to execute the instruction which is located at some other location, which is say some 65000, then how does this microprocessor after this this will jump to this particular instruction? For that the instruction pointer has to be loaded with. So, what is the offset of this address? From the starting is 5000. So, this instruction finds simply if I load these with 5000 H, then this will skip all these instructions in between and it will jump to the instruction which is located at 65000 H, ok. So, this is how this jump instruction will be takes place.

So, whatever the previous contents of the instruction pointer will be now replaced with the address which is specified in the register, so that the microprocessor control will transfer to that particular instruction. Here these can be represent a memory also, ok. This is about unconditional.

Without checking any condition it simply jumps from that particular instruction to the offset which is specified in the instruction itself, to the memory location whose offset is specified in the instruction itself this is about unconditional jump. And there are some conditional jumps. So, it will check for some condition. If some condition is satisfied then only it will jump otherwise it will simply neglect the jump instruction, it will go to the next instruction.

So, we have different conditional instructions that we will discuss in the next lecture.