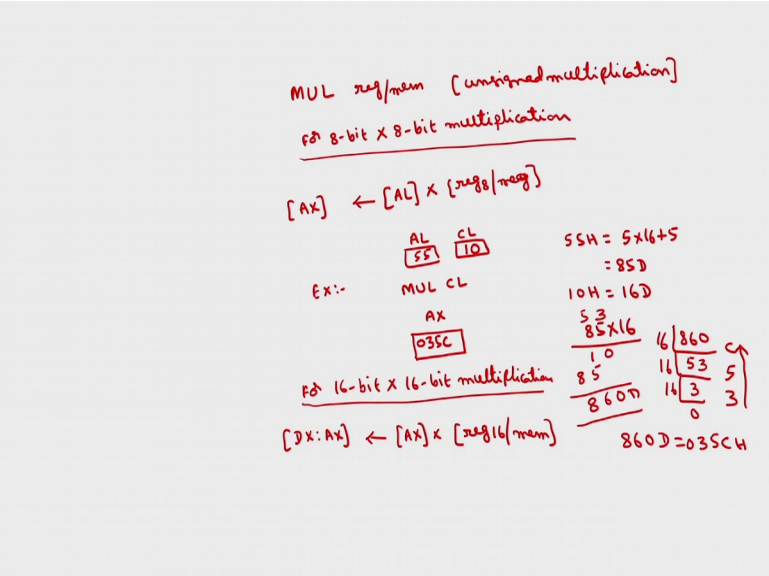
Microprocessors and Interfacing
Prof. Shaik Rafi Ahmed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Lecture – 08
8086 Arithmetic Instructions II

So, in the last class we are discussing about Arithmetic Group of Instructions 8086 instructions. So, in that we have discussed about the addition instruction with and without a carry, subtraction instruction with and without borrow. So, the next arithmetic operation is multiplication.

(Refer Slide Time: 00:51)



So, the multiplication instruction is like MUL so, we can give register slash memory. So, we have to specify register slash memory after the MUL. So, the operation inside this one is so, for 8 bit by 8 bit multiplication; we can have 8 bit by 8 bit multiplication or 16 bit by 16 bit. For 8 bit by 8 bit multiplication, now to specify the register 8 bit register or the memory location 1 byte.

So, this is one operand, the contents of either 8 bit register or from memory 1 byte has to be multiplied with, the other operand will be by default in register AL and the result is also by default stored in AX. So, whatever the register or memory that is specified in the memory instruction that will be multiplied with AL contents and the result is stored back to AX.

For example, if I take so, before this if I give AL as 55 some CL as 10 h, these are hexadecimal numbers. If I write MUL CL, by default other register is AL. So, the multiplication of this one will be; so, result will be there in AX. So, what will be the result? The hexadecimal multiplication of these two numbers ok, we can directly perform the hexadecimal multiplication also or otherwise you convert these numbers into decimal, you perform the multiplication decimal number, convert back the result to hexadecimal.

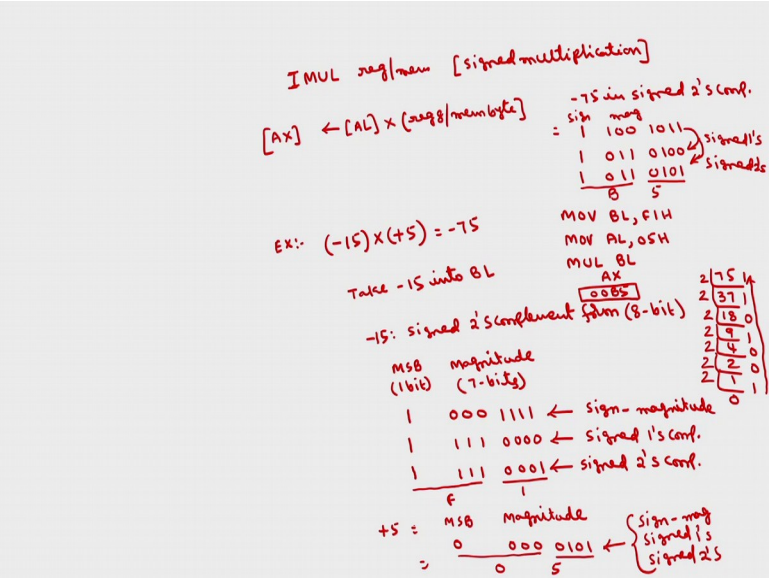
So, 55 hexadecimal is equivalent to what is the decimal, 5 into 16 plus 5 which is equal to 85 decimal, then is 10 H is 16 decimal. So, if I multiply 85 by 16 decimal multiplication 860 is the decimal value. So, if you convert back this 860 to hexadecimal 860 you have to successfully divide with 16, 16 5 is 80, 60 will be again 16 5s are 80 60 will be 4 3 will be 48. So, what is the remainder? 53 16s, the remainder is 80 60 is 48 12 C and 16 3s are 48, 48 remainder will be 5, 16 0s remainder is 3. So, the answer will be 3 5 CH 860 decimal is equivalent to 3 5 C hexadecimal, you can add 0 also here.

So, this the contents of AX will be 0 3 5 C H and this is how the multiplication will be performed. So, we can give here instead of register, we can give the memory also. Which memory again? The memory can be accessed using any one of the addressing modes that we have discussed in the earlier classes. This is for 8 bit by 8 bit multiplication, for 16 bit by 16 bit multiplication; so, we have to specify the register 16 are a word from the memory. This is one operand and the other operand will be by default the contents of AX.

And, the result will be by default stored in two 16 bit register DX comma AX; because if we multiply two 16 bit numbers the maximum result will be 32 bit. So, these 32 bits result will be stored in two 16 bit registers. In that most significant 16 bit will be stored in DX and the least significant 16 bit will be stored in AX.

This is about the 8 bit by 8 bit multiplication, 16 bit by 16 bit multiplication. What this multiplication is called unsigned multiplication, here we will assume the numbers, the data to be unsigned numbers. So, on the other hand we have signed multiplication also, because in 8086 we will use the signed representation sign 2’s complement representation.

(Refer Slide Time: 06:41)



So, the only difference between this MUL and signed multiplication is, the mnemonic for this signed multiplication is I MUL; same thing you have to specify register slash memory. So, again for 8 bit by 8 bit multiplication, we have to use AL contents of register 8 slash memory byte; result will be stored in AX. This is exactly same as that of unsigned multiplication, the only difference is here the data will be given in signed format.

And the results also we will get in the signed 2's compliment format. If I take one example so, I want to multiply minus 15 into plus 5, the expected result is minus 75. So now, how to give this data? You take this minus 15 into one of the registers either AL or any other of the other of the register or any memory location, you take the plus 5 into the other register and you multiply. So, how to take? If I want to take this minus 15 into some BL register say.

So, what has to be loaded into BL? The microprocessor understands only 1's and 0s only, it will not understand the minus sign. So, for that so, minus 15 has to be represented in first signed 2's complement representation. So, in case of signed 2's complement representation because these BL register is 8 bit, we have to I mean represent 8 bit form because, we are going to store this in 8 bit register.

If want to I mean store this minus 15 16 bit register, you have to represent minus 15 in 16 bits 2's complement form ok. Here I am using 8 bit because I want to store this in BL.

So, MSB bit in sign 2's complement representation, the MSB bit is always sign bit followed by magnitude bits. This is 1 bit and this is the remaining 7 bits.

So, for negative numbers MSB bit is 1, for positive numbers MSB bit is 0 and for my 15. So, what is this 7 bit binary form of 16 15? Is 0 0 0 1 1 1 1. So, this is called signed 1's complement representation or signed magnitude representation, sign magnitude representation. There are 3 ways to represent signed numbers, but in microprocessor we will use sign 2's complement representation. This is called signed magnitude representation.

Whereas, in signed 1's MSB bit is 1 because of a negative number, you take the 1's complement of this, is 1 1 1 0 0 0 0. This is called signed 1's complement representation, then signed 2 MSB bit is 1, you take the 2's complement of this means 1's complement plus 1. So, this will be 1 1 1 0 0 0, you have to add 1 to the LSB. This is called signed 2's (Refer Time: 10:19) signed 2's complement representation. Now, what is the hexadecimal equivalent of this minus 15 then? This will be if this will be 1.

So, you have to take BL with F 1, if you want to perform the multiplication of minus 15 into plus 5; so, you have to load these F 1 into BL. And, what about plus 5? Plus 5 will be in signed 2's complement representation MSB bit and magnitude bits. MSB bit will be for positive numbers 0, magnitude bits for 5 will be 7 bit equivalent of this 5 will be 0 0 0 0 1 0 1.

If the number is positive all the 3 representations are seen. So, this is you know all the 3 combinations signed magnitude as well as signed 1's signed 2's, all the 3 representations are same if the number is positive. So, these representations will be different only if the number is negative ok.

So, what will be hexadecimal this now? If I take the 0 0 0 0 as 0 this is 5 0 5 H, then to perform this multiplication the instructions will be; so, I want to take 1 into BL. So, MOV BL comma so, minus 15 in signed 2's complement representation will be F 1 and MOV one of the operand should be in AL for 8 bit by 8 bit multiplication. So, AL with the second number which is plus 5 is 0 5 H itself, then you write MUL BL.

So, inside this AL and BL will be multiplied, result will be stored in AX. Now, what will be contents of AX after this instruction? So, in binary decimal form the result will be

minus 75, here we will get in AX 2's complement representation of minus 75. So, what is the 2's complement representation of minus 75? Minus 75 in 2's complement representation signed 2's complement representation will be MSB bit is 1.

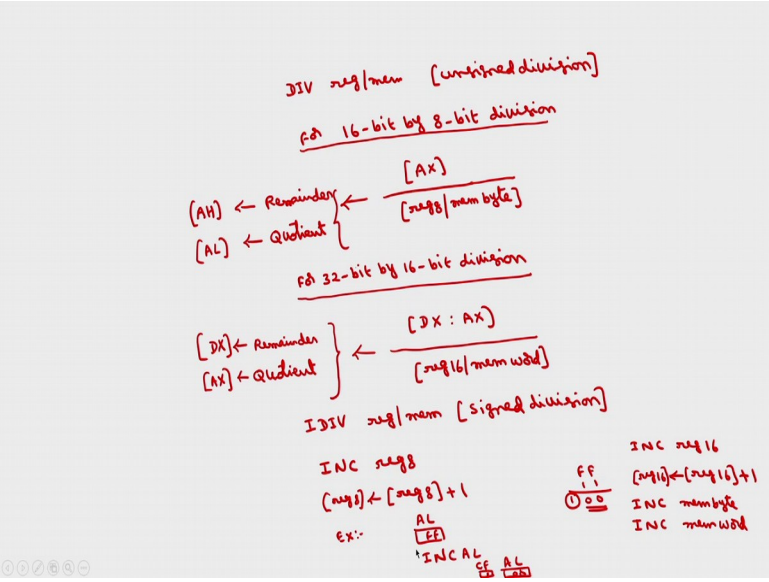
If we take this magnitude signed magnitude representation 75, what is the binary equivalent of 75? 75 you can convert into binary 2 3s are 6 7 is 14. So, 1 remainder 2 1's are 2 2 8 16 1 is the reminder 2 9 0 2 4s are 1 2 2's are 0 2 1's are 0 2 0 1. So, you have to read from bottom to top. So, the magnitude which will be 1 0 0 1 0 0 1, this will be 1 0 0, this is signed bit, this is magnitude 1 0 0 1 0 1 1.

This is in signed magnitude representation, but the result will be in signed 2's complementary because we have given the 2 datas in signed 2's complement representation. So, from these signed 1's will be, you have to take the 1's compliment this remains 1. This will be 0 1 1 0 1 0 0 signed 1's, signed 2's means you take again add 1 to this, signed 2's will be you add 1. If I add 1 this MSB becomes 1 0s 1 1 0 1 0 1.

So, what will be result? The hexadecimal equivalent of 1 0 1 1 is B 1 0 1 0 1 is 5. So, you will get 0 0 B 5, this is about the signed multiplication or any signed arithmetic. So, in signed arithmetic whether it is a addition, subtraction, multiplication or division; so, you have to take both the operands in signed 2's complement representation and you will get the result also in the signed 2's complement representation. So, you see the difference between MUL and I MUL.

So, I MUL is first signed multiplication, this is signed multiplication. Similarly, we have division for unsigned numbers, division for signed numbers DIV.

(Refer Slide Time: 15:29)



The next instruction is unsigned division DIV, again you have to specify register slash memory. Memory can be byte or word, register can be 8 bit or 16 bit. This is called unsigned division. Here also we can have 2 types of the divisions, it can be either 16 bit by 8 bit division or it can be 32 bit by 16 bit division, for 16 bit by 8 bit division; division will be by default in AX.

So, this will perform the operation like AX will be the division always, divided sorry divided by AX. Division will be the 8 bit register slash register 8 bits are memory byte then we will get quotient and remainder. So, this performs and this will store the remainder in quotient in AL.

This is the division operation, division register 8 slash memory. This is 16 bit by 8 bit division by default divided will be an AX and division we have to give whether 8 bit register or memory byte. So, after this division reminder will be stored in higher order 8 bit register of AX which is and lower order 8 bits quotient will be stored.

Similarly, for 32 bit by 8 bit division so, the divided will be 32 bit register which is combination of two 16 bit registers DX and AX, this is by default. And, then divisor will be 16 bit register contains or memory word. Again the result of this one, the reminder will be stored in the higher order register which is DX and quotient will be stored in AX. This is about 16 bit by 8 bit division and 32 bit by 16 bit division.

Similarly, we have for signed division we have I DIV. Now, to specify register slash memory, this is called signed division. So, the operation is exactly same except for that here all the datas has to be represented in signed 2's complement representation. I have given one example for the multiplication; so, here I will not give any examples.

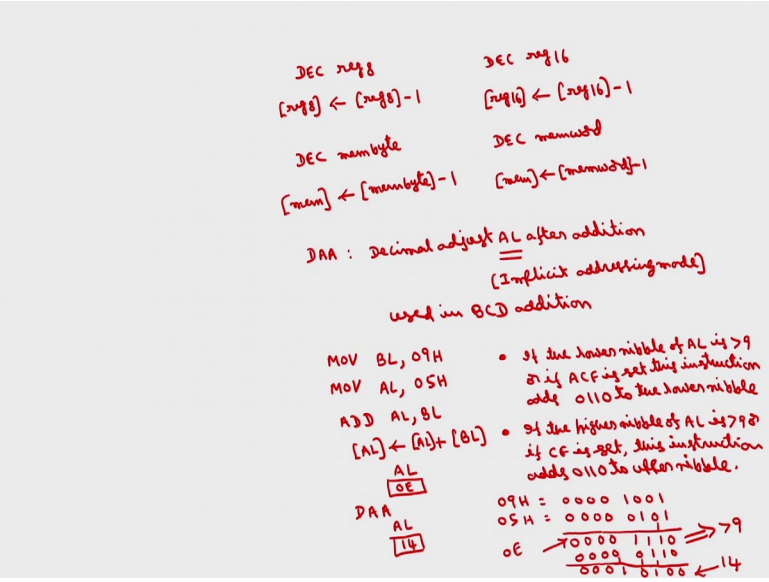
So, in a similar manner we can perform this division also. This is about the division, unsigned division and signed division. So, these are the four main important arithmetic operations, addition with and without carry, subtraction with and without borrow, multiplication for unsigned numbers as well as signed numbers, division for signed numbers as well as unsigned numbers.

So, these are the four important arithmetic operations, the next arithmetic operation is increment and decrement operations. INC is the mnemonic for increment operation. So, we can increment the contents of register 8. So, simply the operation is the contents of register 8 will be incremented by 1 and stored back to same register. If I take the example, if AL is FF H, if I write INC AL. So, what will be contents of the AL after this? It will be 0 0. Why? Because, FF if we add 1, this becomes 1 0 this becomes 1 0. So, the 8 bit result will be 0 0, but carry flag will be set.

Similarly, you can have 16 bit register increment also INC register 16. So, the 16 bit register contents will be incremented by 1 and stored back to same 16 bit register. Similarly, we can increment the contents of a memory location byte from the memory location, INC memory which you can take as byte or INC memory word.

So, you can take a word from the memory location, again the memory location will be specified by one of the previous option that I have given. Out of the 5 options we can specify the anyway domain access that particular memory. These are about 4 I mean increment instructions, similarly we have 4 decrement operations.

(Refer Slide Time: 21:35)



So, we have DEC, mnemonic for decrement is DEC register 8 means register 8 contents will be decremented by 1 and stored back to same register 8. Similarly, we can have DEC register 16, register 16 contents will be decremented by 1 and stored back to same register 16. We can have DEC memory byte.

So, the contents of the memory byte will be decremented by 1 and stored back to same memory location and you can have decrement memory word; as I have told word is 2 bytes. So, the contents of the memory location, memory word will be subtracted by 1 and result is stored back to the memory. So, these are the 4 I mean increment and decrement operations.

So, the next I mean arithmetic instruction is, this is one of the very important instruction which is DAA: decimal adjust AL after addition. This instruction operates only on the register lower order register AL. So, that is why the addressing mode here will be implicit addressing mode. There are some instructions which operates only on a specific register, those all instructions belongs to implicit addressing mode. Here also is DAA operates on AL only.

So, we know that after the addition, after the addition if I have to perform the BCD addition; basically this will be used for user in BCD additions. So, if I take say for example, two BCD numbers MOV BL comma 0 9. Even, though if I give this BCD

number the microprocessor will take only hexadecimal data, this will assume that this is hexadecimal, but the ordinary layman he knows that this is only decimal line.

Then if I take MOV AL comma 0 5 H and if I write add AL comma BL. The operation is continuous of BL will be added with contents of AL and the result is stored back to AL. So, what will be the AL; AL contents, because this the microprocessor will take a hexadecimal numbers, if it perform hexadecimal addition this is 14. So, this AL contents will be 0 EH, but the user the ordinary layman he expects 9 plus 5 should be 14 ok.

So, to get this 14 from this AL that is I mean 0 E, we can write DAA. After DAA the contents of AL becomes 1 4 which is the BCD addition of 9 plus 5. So, how this operation will be takes place. So, what is the internal operation that takes place inside the DAA is. So, this internal operation is, if the lower nibble means lower 4 bits of AL is greater than 9 or if auxiliary carry flag is set.

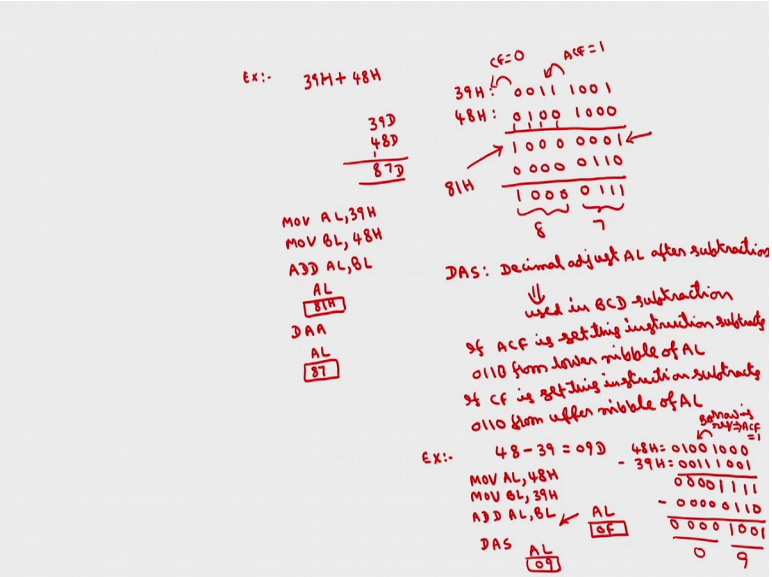
This instruction adds 6, this instruction adds 6 to the lower nibble. And, second one is similarly it will check for higher nibble also. If the higher nibble of AL is greater than 9 or if carry flag is set, this instruction adds 6 2 upon nibble. If I take the same example here. So, if I add these 0 9 plus 0 5 0 9 H will be 0 0 0 0 1 0 0 1 and 0 5 will be 0 0 0 0 0 1 0 1. If I perform the addition 1 plus 1 is 1 0 1 1 1 0 0 0 0, we check the lower nibble result. What is the result of lower nibble?

This is 13, there is no auxiliary carry flag, but if either of this condition satisfies whether auxiliary carry flag is set or if lower nibble of this one is greater than 9. So, because this lower nibble, this value is greater than 9 this is 13. So, this instruction what happens? This will add 6 to the this lower nibble whereas, this one is either carry flag is set this result is greater than 9. So, the microprocessor this instruction DAA instruction will not add anything, only 0 0 0 0 will be added.

So, what will be result now? So, this will add because greater than 6 you add 6. So, what is the result? 0 1 plus 1 is 1 0 1 plus 1 plus 1 is 1 1 1 plus 1 is 1 0 1 0 0 0 this is nothing, but 1 4. So, this was actually this was this 0 E, as I have told here the contents of AL will be 0 E because, the microprocessor assumes that these two numbers are hexadecimal numbers, but what I want is I want BCD result ok. So, if I pass through the DAA the contents of AL becomes 14 which is the addition of BCD addition of 9 and 5. So, this is the operation which takes place inside this instruction DAA.

Since, we will check the lower order 8 bit bits, if this magnitude is greater than 9 this we add 6 or if the auxiliary carry flag is set this will add 6. Similarly, higher order 8 bits if there is carry flag or if higher order 8 bits is greater than 9 this instruction will add 6 otherwise nothing. So, then we will get the final result in BCD. I will take an example where axillary carry flag is set magnitude is less than 9, if I take second example.

(Refer Slide Time: 29:45)



Suppose, if I want to add something like 39 H 48 H. So, what is 39 H? 39 plus 48 so, I want actually a simple addition. If they are decimal numbers so what is the expected result? 9 plus 8 is 17 and this one is 8 87 decimal is expected, but microprocessor will take 39 as hexadecimal number. We will take this as 0 0 1 1 1 0 0 1 then 48 also it will take hexadecimal number.

So, the corresponding binary is 0 1 0 0 1 0 0 0 that will perform the addition. This is 1 0 0 1 0 this magnitude is less than 9 lower order nibble. But, there is auxiliary carry flag because there is a carry from this position to this position lower nibble to upper nibble; so, auxiliary carry flag is set.

So, because of this instruction adds 6 and here this will be 1 plus 0 plus 1 is 1 0 1 0 1 0 1, this is neither of these condition is satisfying. This magnitude is 8 which is less than 9 and then there is no carry flag, no carry flag carry, flag is reset. So, this instruction simply adds 0 0 0 0. So, if I write the instruction like MOV AL comma 39 H MOV BL comma 48 H.

If I write add AL comma BL, the contents of AL will be it will be the hexadecimal result which is nothing, but the hexadecimal result is this. And, this 8 1 H this will be 8 1 H, but what is expected if I write DAA after this AL will be having the corrected decimal result will be 87. So, you can see here that after addition you get 87 1 1 1 0 0 0 0 1. So, this is 8 this is 7. So, this is how we can use this DAA instruction.

So, whenever you want to perform the BCD addition or even BCD subtraction of these other instruction. For BCD addition you want to adjust the result as a BCD result from the hexadecimal, we can use DAA instruction. Similarly, we have next instruction which is called DAS decimal adjust AL, same thing but, here after subtraction. Same thing here this will perform BCD subtraction, this will be used in BCD subtraction.

So, if this auxiliary carry flag is set this instruction subtracts (Refer Time: 33:19) 6. If auxiliary carry flag is set, this instruction subtracts 6 from lower nibble. Similarly, if carry flag is set this instruction subtracts 6 from upper nibble of AL. This also instruction will be only for AL region, this was (Refer Time: 34:14) addressing mode.

Here there is no question of this result is greater than 9 because, this is I mean BCD subtraction we have to give only BCD numbers, we cannot give here. You remember that you cannot give here A B C D E because, this is decimal means BCD. So, this should be 0 to 9, this should be 0 to 9, this also 0 to 9, this also 0 to 9. If I perform the subtraction, the maximum value of digit is 9, if I subtract the subtraction will get only the value which is more than 9 only.

So, if I take one example here also, if I take the same 48 minus 39; I want to perform the decimal subtraction or BCD subtraction 48 minus 39. So, what is expected is 9 0 9 decimal, but if I give this values MOV AL comma 48, it will take as hexadecimal by default MOV B H or BL comma 39, it will take by default as hexadecimal.

Then if I add these two add AL comma BL then after this the contents of AL will be. So, whatever this hexadecimal subtraction value that will come. So, what is 48? 48 H will be 0 1 0 0 1 0 0 0 minus 39 which will be 0 0 1 1 1 0 0 1, as you have discussable discussing about this different flux of microprocessor, auxiliary carry flag will be set during the addition. If there is a carry from lower nibble to upper nibble, during the subtraction if there is a borrow from upper nibble to lower nibble in that case also auxiliary carry flag will be set.

Here if we want to subtract 0 minus 1; so, we have to take the borrow. So, if we take the borrow, this becomes 2 2 minus 1 is 1, again to perform 0 minus 0 no borrow is required, but already this has given on borrow to this one, to pay that it has to take again borrow. So, this becomes 2, out of this 1 has been given to this 1. So, 1 minus 0 becomes 1 and this also again same thing.

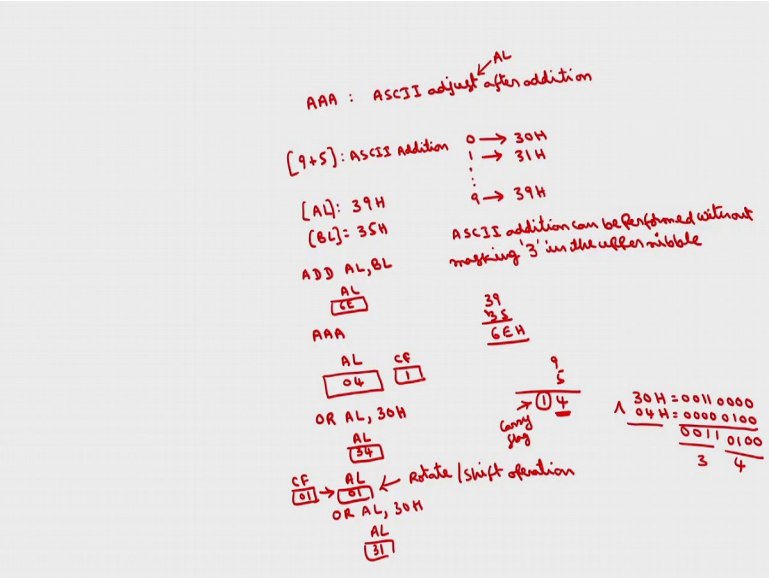
So, 0 means 0 no borrow you required, but it has given 1 to the previous stage. So, to pay that 1 it has to take a borrow. So, that it becomes 2, out of these 2 1 is given 2 here and 1 minus 0 is 1 and this 0 has been given this 1 has been already given. So, this becomes 0 minus 1 to perform 0 minus 1 this has to take borrow from upper nibble, borrow is required from upper nibble. So, borrow is required from upper nibble to lower nibble implies auxiliary carry flag is set.

So, after taking the borrow what will be the result? This will becomes 2 2 minus 1 is 1, again here 0 minus 1, it requires a borrow 2 minus 1 already 1 has been given here. So, 2 minus 1 minus 1 will be 0, this also 0 in a similar manner, this is 1 has 0 minus 0 is 0 minus 0 is 0. So, this will be the result that we are going to get which is nothing, but 0 and F. So, the AL will be 0 F, but what is the decimal real that will expect 0 9.

So, if I write DAS so, what these instruction will do is as I have told so, this will check the lower order 8 bits. So, this is more than 9 and auxiliary carry flag is also set, this instruction simply subtracts the 6. So, instead of adding in case of DAA it will add 6 whereas, here it will subtract 6 0 1 1 0 and this is neither carry flag is set nor this magnitude is greater than 9.

So, it will subtract 0 simply means no change; so, this has to be subtracted. So, if you subtract 1 minus 0 is 1 1 minus 1 is 0 1 minus 1 is 0 1 minus 0 is 1 0 0 0 0. So, what will be this? 0 9 this is the correct result, after these DAS AL will contain the BCD subtraction result which is 0 9. This is how we can perform the BCD addition and BCD subtraction using DA and DAS.

(Refer Slide Time: 39:09)



So, there are four more instructions; so, which are called AAA, these are all for addition, subtraction, multiplication and division, but they are an ASCII operation. So, the previous one is one base BCD, where these instructions are for ASCII, this is ASCII adjust after addition. So, we know the ASCII code, we have discussed in the earlier class; ASCII is an American Standard Code for Information Interchange.

So, correspond to these keys 0 key, if I press the 0 key the corresponding decimal hexadecimal digit will be 3 0 H will be generated 1 means 3 1 H. So, up to 9 will be 3 9 H, similarly correspond to alpha bits we have standard code. So, this ASCII if I want to perform addition ASCII numbers addition without masking with 3 we can perform the ASCII addition.

So, the ASCII addition can be performed without masking these 3, these 3 is extra this is 0 actually BCD, but 30 H is the hexadecimal, this 3 is extra. So, without masking these 3 also we can perform the ASCII addition by use of this AA instruction. ASCII addition can be performed without masking 3 in the upper nibble. So, how this will be performed? For example, if I want to add ASCII characters 9 plus 5, I want to perform 9 plus 5 ASCII addition.

So, we take this one into accumulator AL, this also will adjust AL, this also works on only register AL only always. If I take AL as 3 9 H which is perhaps equal to correspond to 9, this using MOV instruction I am not writing that, then BL say 35 H. This is the

ASCII code correspond to 5 ok, say if I right add AL comma B l. So, what will be result? It should be 39 plus 35 which is equal to E 6 6 EH.

So, after this AL will be having because the microprocessor assumes only hexadecimal data and see how we have discussing in the earlier instruction also. Even if you give these BCD number, it will take a hexadecimal; even here if you give ASCII number it will take hexadecimal. And, it will give the hexadecimal result which is 6 EH, but I want this as on 4.

So, this will give if I write instruction AAA, the contents of AL becomes 0 4. Why 0 4 is 9 plus 5 is in fact 1 4 so, everywhere this is carry flag will be set and the result is only 4 ok, because this AL is 8 bit register, this will store only 8 bits. So, this 16 bits result cannot store, but carry flag will be set here.

So, if we want this to result in ASCII. So, what is the addition all this one it should be? So, 9 plus 5 should be 14. So, for 1 the ASCII code is 31 H, for 4 ASCII code is 34 H ok. To get this 34 H we can do it ORing with, you write something like OR AL comma 30 H. I am going to discuss this OR instruction in the logical group. This is very simple only OR instruction OR logically or the contents of AL with 30 H and store the result in AL.

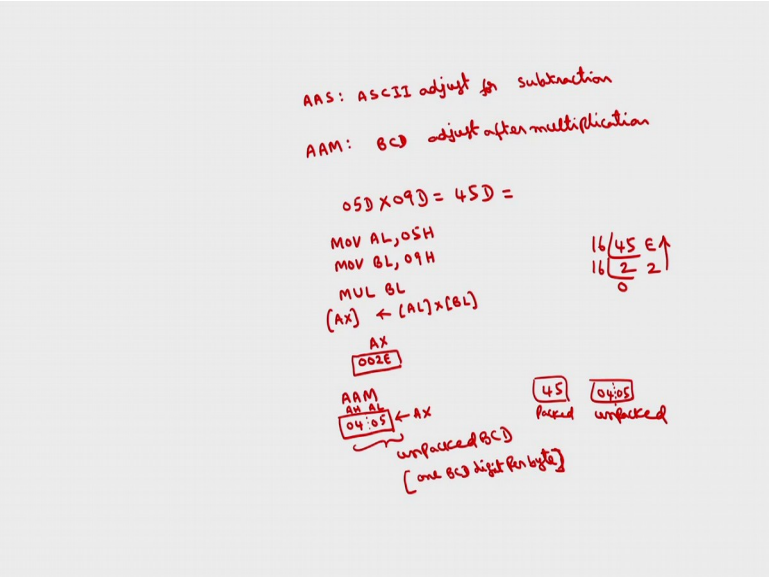
So, after these AL will become now so, 30 H and 4 H if we OR. So, 30 H and 0 4 H we are ORing this. So, this will be 0 0 1 1 0 0 0 0, 4 will be 0 0 0 0 0 1 0 0. What is logical OR between 0 0? Bitwise logical OR operation, 0 0 logical OR is 0 0, if any 1 bit is 1 output is 1, this is 1, this is 0, this is 1, this is 1 0 0, this is 3 and this is 4.

So, AL will content becomes 3 4 H which is the ASCII code correspond to the lower order result 4. Then, again if I want for the higher order also; so, this carry flag contents you take on to the AL register merging the shift register shift a main operation of rotate instructions, that we are going to discuss in the logical group.

So, I can take this 0 1 on to the AL, 0 1 the contents of carry flag, carry flag contents will be shifted to AL for this you can use rotate and shift operations. We will discuss this rotation shift instructions in the next group which is a logical group, then again OR with AL with 30 H. So, that now AL will be having 3 1 H. So, this result 2 ASCII characters.

So, 4 correspond to 3 4 and 1 31 we can store in to AL in this see of course, if you want you can move into some other register because here we are using AL. This is how we can adjust this ASCII addition, after ASCII addition the result will be stored back into ASCII by means of this AA instruction. So, you see the operation how this ASCII will be stored in AL, similarly we can have ASCII at just after subtraction.

(Refer Slide Time: 46:11)



We can write after or for there is no problem, only for in a similar manner; no need to explain this. There is one more instruction for the multiplication which is ASCII adjust for multiplication, ASCII adjust this is after multiplication. Even this is ASCII, this is actually BCD sorry BCD adjust after multiplication. So, I will explain the operation inside this AAM, this also another interesting instruction.

So, if you want to perform the multiplication of 2 BCD digits. So, in DAA and DAS we have discussed about the multiplication of addition and subtraction of 2 BCD digits whereas, for multiplication of the BCD digits to get the BCD result we will use AAM actually. So, if I take say for example, if I want to perform 5 decimal 05 decimal into 09 decimal. What is the expected result? Means 45 decimal ok, this is exactly similar to DAA and DAS.

Now, to perform this multiplication I will take one of this one into AL AL 05, but if I give 05 this will by default I take as hexadecimal, even though have on decimal by default it will take as hexadecimal. Similarly, if I write MOV BL comma 09, it will by

default take hexadecimal if I write MUL BL. So, we know that the entire inside the operation will be AL in to BL and the result is stored in AX.

So, what will be contents of AX after this? The hexadecimal equivalent of these 45 so, this is decimal. So, if we perform the hexadecimal multiplication as I have told one way to obtain the hexadecimal arithmetic is you perform the decimal into convert back to hexadecimal numbers 45 decimal. So, this will be equal to how much hexadecimal? 45 16 16 2's are 32 13 13 is E 16 0s 2 so, 2 E.

So, this will contain 0 0 2 E H. But what is expected? Expected value is 45 D; so, to get that 45 D in unpacked BCD we are going to use AAM. After, this AX will be having 0 4 in after AAM AX contents, this will be content, this will be AL content is 0 5. This overall this will be AX, this is what is called unpacked BCD. So, what is meant by unpacked BCD is only one BCD digit per byte is called unpacked BCD.

Packed means if we want packed means something like 45, if I take 45 this equal to packed BCD whereas, 0 4 and 0 5 is unpacked BCD. This is packed and this is unpacked. So, this is the definition of unpacked BCD, one BCD digit per byte, this is one byte, this is another byte. So, if you have one BCD digit, this called unpacked.

So, this AAM instruction will gives unpackaged BCD result of the given two numbers; multiplication of the given two numbers. So, this will be result of AX after this multiplication. So, this is as you called as BCD adjust after multiplication. So, this is exactly similar to DAA DAS, in DAA to get the decimal addition value we will use DAA, to get the decimals of subtraction value BCD so, we will use DAS. And, AAM for getting the BCD multiplication value, similarly we have AAD division.

I will continue this in the next class.

Thank you.