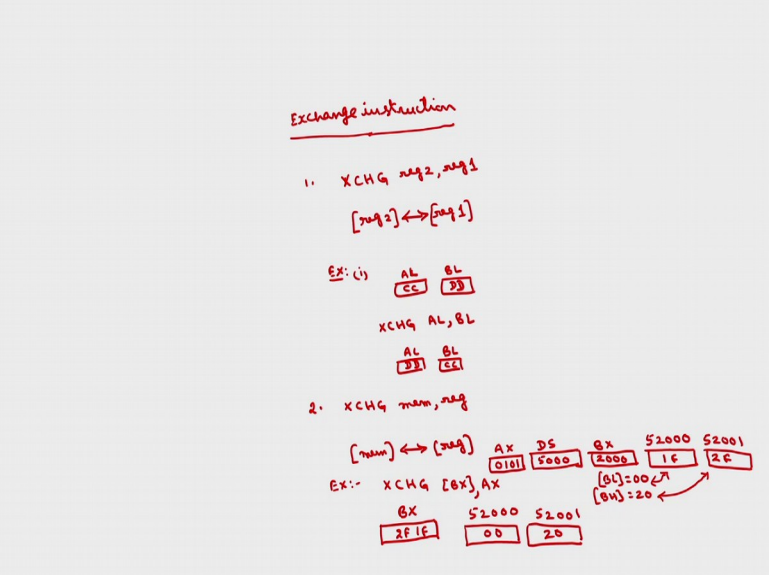
**Microprocessors and Interfacing**
**Prof. Shaik Rafi Ahmed**
**Department of Electronics and Electrical Engineering**
**Indian Institute of Technology, Guwahati**

**Lecture – 07**
**8086 Arithmetic Instructions I**

In the last class we were discussing about the data transfer instructions in which we have discussed MOV instruction, load instruction, ok. So, the next data transfer instruction is exchange instruction.

(Refer Slide Time: 00:41)




We have two; I mean exchange instructions in 8086, 1 is XCHG; XCHG register 2 comma register 1; the contents of register 1 can be exchanged with the contents of register 2. If I take the example so, the sizes (Refer Time: 01:24) of registers must be same both can be either 8 bit or both can be 16 bit. Say if AL contents are CC, BL contents are DD, if I write XCHG AL comma BL, after this instruction AL contains DD and BL contains CC. This is a simple operation; so, these two contents will be exchanged.

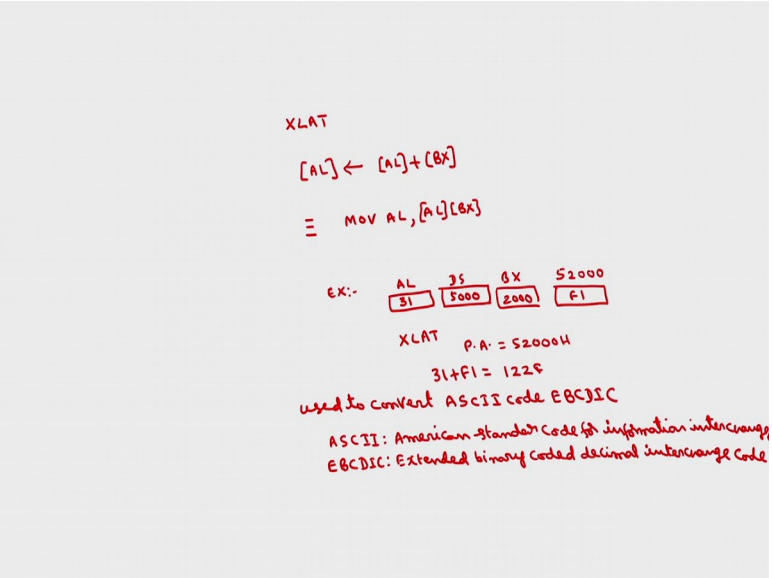
Similarly, you can write both 16 bit registers also, then the second exchange instruction is exchange between memory and register. So, the contents of register are exchanged with the memory contents, which memory location again so, there are five ways to specify the memory.

So, if I take an example, if I write XCHG contents of BX comma AX, if the contents of AX here is 0101 H, if DS is this memory we know that for all these memory calculations by default the segment register is data segment. And, this is 5000 and BX is 2000 then physical addresses we have calculated in the earlier examples also 5 2 0 0 0 H, here if you have some 1 F in the next location 5 2 0 0 1 H if you have 2 F.

So, this particular instruction simply exchanges. So, this between this, this will be this 8 bits will be BL and this will be BH, here BL will be having 0 0, BH will be having 20. So, this 0 0 with the lower order contents 1 F higher order contents will be 2 F. So, after this what happens is the contents of BX becomes BX is 2 0 0 0, now it will becomes 1 F, 2 F and the contents of 5 2 0 0 0 will become 0 0 5 2 0 0 1 becomes 2 0.

So, this BL will be exchanged with this and BH will be exchanged with this; similarly, we can have different memories this can be based addressing mode or relative addressing mode, ok. So, we have different memory representations. So, next instruction in the data transfer group is XLAT.

(Refer Slide Time: 05:03)




So, this instruction basically; so, inside this here the thing is mentioned here no register no memory nothing, but the operation inside this is AL contents. This will be operated on AL only; AL will be loaded with contents of AL plus contents of BX. The operation inside this XLAT operation is, the contents of AL will be loaded with contents of AL plus the contents of the memory location pointed by BX register, ok

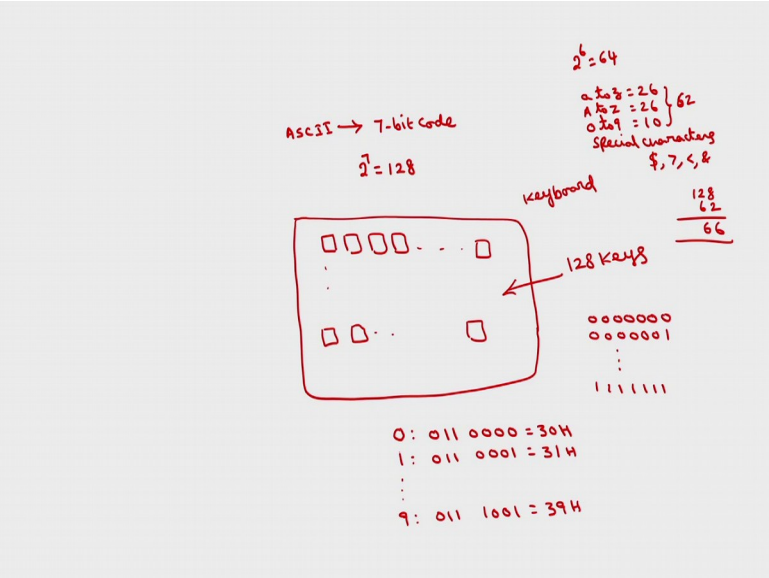
So, this is actually equivalent to this instruction is equivalent to MOV AL comma [AL] [BX]. So, if I take some example, if AL contents is CC, let us take say 0s 0 1 otherwise, basically this operation will be used for converting ASCII data into EBCDIC data. Say, 31 H and is DS is take same 5000, BX is 2000 and 52000 the contents of 52000 location will be say F1.

So, after the execution of this one what happens is, before the execution of this one if I write XLAT. The contents of AL becomes contents of AL 31 plus contents of the memory location pointed by this which physical address will be now 5 2 0 0. So, the physical address of this one we have already discussed several times this one.

So, 52000 H, the contents of this 52000 is F1. So, this instruction basically so, this contents of BX means this is the address where, in that address some data will be there. So, what will be this F plus 3 F plus 1 is 101112 so, 122F. So, basically this particular instruction will be used for converting ASCII code into EBCDIC code. ASCII stands for American Standard Code for Information Interchange; whereas, EBCDIC is Extended Binary Coded Decimal Interchange Code.

So, among the basic various codes so, these two are ASCII and EBCDIC code are the famous codes. So, ASCII is a 7 bit code; so, what is basically this. So, you might have heard about the ASCII keyboard. So, what is meant by ASCII keyboard? So, each key will be coded by ASCII code. So, this ASCII code is a 7 bit code; we have 8 bit code also, this is a 7 bit code.

(Refer Slide Time: 09:25)




So, we know that using 7 bits we can how many combinations is 2 raise to the power of 7 combinations which is equal to 128 means if I use the ASCII code to code the keyboard. So, if you have the keyboard here; so, again you have 128 different keys. So, total number of keys will be in this keyboard we can have 128 keys correspond to each code if you have 7 bit code, if all sevens are 00000 this is correspond to 1 key, this is 0000001 correspond to 1 key; so, each key will be having unique code. So, like that if all bits are 1 is corresponded to the last key.

So, in case of ASCII code 0 will be represented by this is standard means which is equal to 0110000 30 H, this is in hexa symbol even you can insert this 0 here. So, this will be equivalent to 30 H, 1 will be represented by this is 31 H so, on up to 9 will be represented by 39 H. Of course, for this 3 we require 4 bits we can insert 1 0 here so, to express in hexa decimal form.

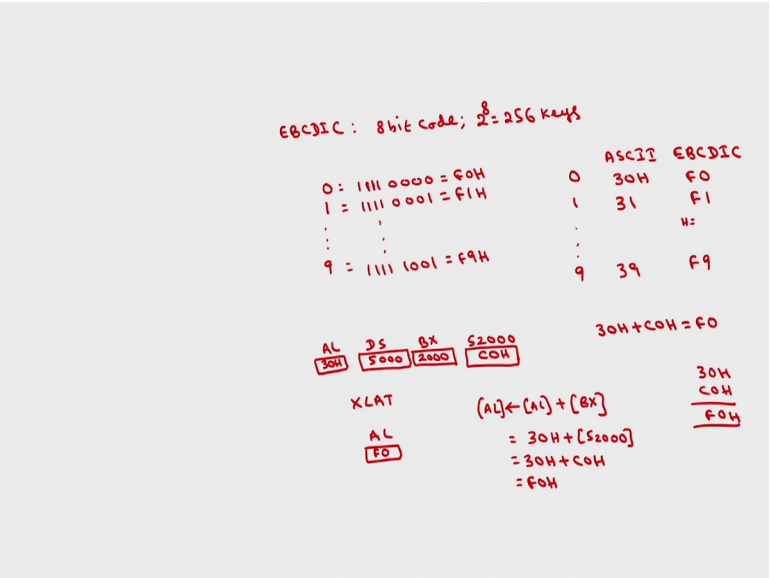
You see about the ASCII code. So, similarly we have extended binary coded decimal. So, why this many code I mean keys are required if I use a 6 bit code, initially there are computers with 6 bit code which is called 6 bit internal code. So, using 6 bit internal code we can have how many different keys 2 raised to the power of 6 which is equal to 64, but in this keyboard.

So, we require all lowercase letters a to z which is 26, uppercase letters a to z 26 then we require digits from 0 to 9 is 10. Then we require some special characters such as dollar

we have greater than, less than different symbols. And so, if I use only a 6 bit internal code so, we can have only 64 different keys; so, these 3 together is 62. So, we can have only 2 special characters, ok. So, to include more special characters we will go for the more number of the bits. So, this ASCII is 7 bit code we can have 128 different characters. So, out of that 62 will be to represent the lowercase and uppercase alphabets and 10 for decimal digits.

So, remaining if you can have up to, 128 minus 26 62 is 6 you can have 66 special characters so, that is why will go for ASCII code. Even if you want more than this special characters more than I mean total 128 bits we will go for extended binary coded decimal EBCDIC code, this is 8 bit code.

(Refer Slide Time: 13:41)




So, using this we can have how many keys 2 raised to the power of 8 which is 256 different keys we can have, these can be incorporated in EBCDIC keyboard, ok. Now, in EBCDIC if it take say 0 1 so on up to 9, this will be equivalent to this is F 0 which is 1 1 1 1 0 0 0 0 which is equal to F0H, 1 is equal to F1H so on up to F9. They are standard which is defined by inventors; similarly, we have for lowercase letters some code, uppercase letters some code. Now, this particular XLAT will be used to convert ASCII code into EBCDIC code.

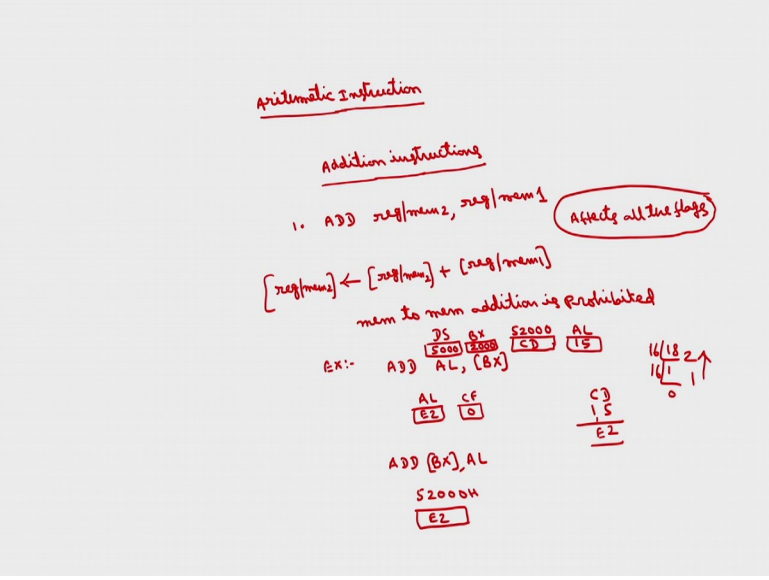
So, corresponding to 0 this is the EBCDIC code, corresponding to 0 ASCII is. So, you have seen in the last slide its equal to 30 H EBCDIC is F0, 1 is 31 F1 so on up to 9 is 39

F9. So, what is the relation to get the EBCDIC code from the ASCII code, so, how much you have to add this is 3 and this is F you have to add 12. So, means F0, if I add 30H plus C0H this will be F0, this is 30 H plus C0H so, C plus 1 is DEF; F0H. Similarly, if I add C0H to so, all these values you take any value and if you add 30H, we will get these values; so, in XLAT how this conversion can be performed.

So, what I will do is, so, I will take in AL first the code corresponding to ASCII which is 30H and if I take say DS is equal to 5000, BX equal 2000 and in 52000 I will store the value which is to be added to get EBCDIC from ASCII, what is that value is C0H. So, if I write XLAT, what will be contents of AL as I explained AL will be contents of AL plus contents of the memory location pointed by BX. So, this will be equal to AL contents are 30H plus this is contents of 52000H contents of 52000. So, this is equal to 30H, the contents of this one is C0 so, you will get F0H.

So, before the execution of this in AL we have 30 H, after the execution we have F0H. So, this is; I mean, 30H is the ASCII code corresponding to 0 and this is EBCDIC code corresponding to 0. Similarly, for 1 for 1 we can store here 31, if I store 32, 33 we will get corresponding this, if I change this to 31 we will get F1, 32 we will get F2, 33 will get F3 like that we can convert ASCII code into EBCDIC code. So, this is the I mean importance of this XLAT instruction.

(Refer Slide Time: 18:21)



So, these are all about the data transfer instructions the first group, the second group is arithmetic group arithmetic instructions. In arithmetic we will discuss about addition, subtraction, multiplication, division, increment, decrement these are the various arithmetic operations. First, I will discuss about the add instructions addition instructions.

So, there are four addition instructions without carry, four with carry. So, the first addition instruction is ADD is the mnemonic, we can register slash memory 2 comma register slash memory 1. So, we can add the contents of this register or memory 1 plus contents of register slash memory 2 and the result will be stored back to the register slash memory 2.

This is the internal operation, both cannot be simultaneously memories here remember that both cannot be so, memory to memory adding is not allowed. So, both cannot be memory, memory to memory addition is not allowed. So, one of this one must be a register. So, if I take the examples add AL comma contents of BX. So, by default the physical address can be computed by using DS and this is same as the previous example 2000. So, in this 52000 if you have CDH say and this AL contains say 15 H.

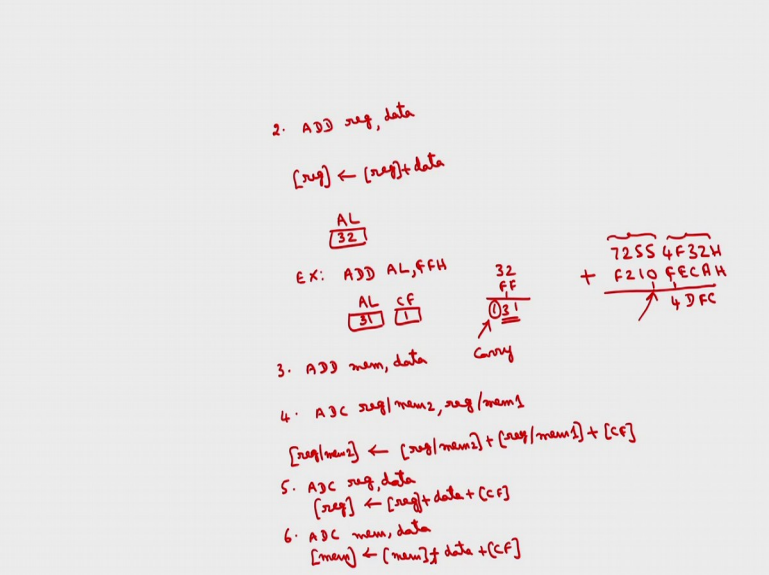
So, after the execution of this instruction what happens, what will be the contents of AL; AL will be sum of this CD and 15 because the physical address of this one using this DS and BX will be 52000. So, CD plus 15 so, D plus 5 will be D plus 1 become E, D plus 2 will be F, D plus 3 will be 101112, this will be 12. Another way to I mean find this one is D is nothing, but 13 5 say 18; 18 if you convert into hexadecimal 18 ones are 18, 2 is the reminder, 18 0s 1 is the reminder so, 12 is the hexa decimal of 18.

So, in decimal this D is 13 plus 5 is 18, in hexadecimal its 12. So, C after that DE; so, this is E2. So, this will be having E2, carry flag will be reset. So, this particular addition instructions affects all the flags; similarly, if you have here 16 bit register it will take from 2 consecutive locations, ok.

So, lower order will be added with the first location, higher order will be added with the second location. Similarly, we can have so, this side memory this side register the same thing, but the result will be stored in memory. Now, the result is stored in AL if you have ADD [BX] comma AL then this result E2 will be stored in this 52000 H. So, one of these

2 I mean operands should be register, both cannot be memory so this is a simple add instruction.

(Refer Slide Time: 23:21)




So, second add instruction is ADD register with data if this is 8 bit register 8 bit data, 16 register 16 bit data. So, directly the data will be add with register and stored in register back. If AL is having 32 if the data is if you give write the example ADD AL comma FFH. So, these two 32 and FF will be added here result will be stored in AL itself, this will be 1 0 1 1, this is 1 0 1 3.

So, this is carry flag is set the result will be 31 because AL is only 8 bit register, this stores only 8 bit. So, this 31 will be stored in AL, but the carry flag will be set, ok; this is about add register comma data. Similarly, we can have add memory comma data, we can add directly some data to the contents of some memory location, again that memory location can be addressed by using either indirect addressing mode, direct addressing mode, based addressing mode, index, based index, ok.

So, these are the 3 ADD instructions and then there are 3 ADD instructions with carry they are without carry. For example, if you want to perform something like if I want to ADD multi-byte numbers 72554F32H plus F210FECAH. See here what happens if I want to add this using the byte or even if I use the 16 bit time because 8086 is 16 bit microprocessor it can perform, this 16 bit addition in a single clock cycle, ok.

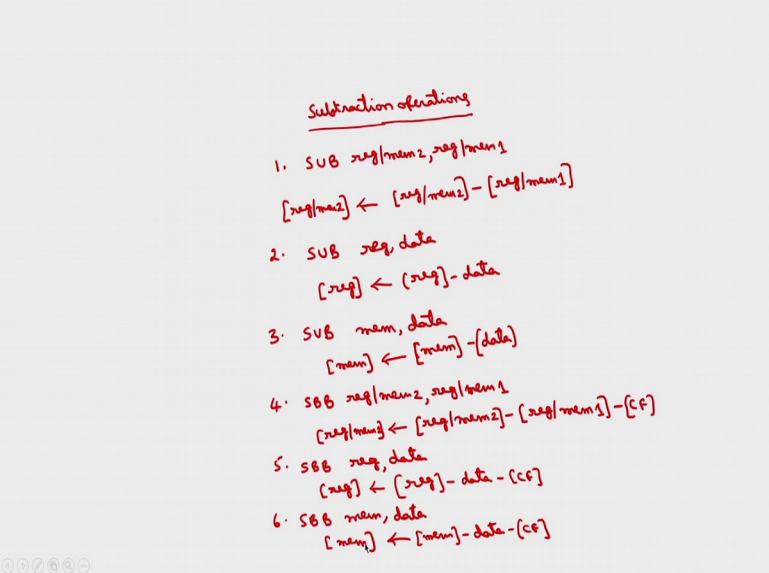
So, if I perform this; so, if I perform this what happens here this A plus 2 becomes C, C plus 3 become F, F plus E becomes 1D and this becomes 14. So, in the first cycle it will add these two numbers. Now, in order to add these two 16 bit numbers in a single clock cycle it has to add these two along with the carry which is generated from previous stage, ok.

So, in order to I mean perform this type of addition, we require ADC instruction. We can have ADC register slash memory 2 same the first one comma register slash memory 1. Both cannot be memories the operation is here register slash memory 2 will be added with register slash memory 1 plus the status of carry flag is also added, and stored in register slash memory 2.

Similarly, you can have corresponding to this ADC register comma data here the operation is register plus data plus the status of carry flag will be added and stored in register. The only difference between this instruction with this instruction is here an extra thing is carry flag, if carry flag is reset 0 then both ADC and ADD are both are same. And another 6th instruction is ADC memory comma data and also same thing contents of the memory pointed by 1 of the addressing modes plus data plus status of the carry flag and the result is stored in memory.

So, there are 3 ADD instruction without memory, three are without carry; 3 ADD instruction with carry, there are total 6 ADD instructions.

(Refer Slide Time: 28:19)



Similarly, we have 6 subtraction instructions 3 without borrow, 3 with borrow. The mnemonic for this one is SUB same register slash memory 2 comma register slash memory 1, both cannot be memories all the flags will be affected here also. So, in place of plus we have to write only minus.

So, this is this minus this, we perform result is stored back into register slash memory 2. Similarly, we can have SUB; I am not giving the example because they are similar to the addition instructions. We can have register comma data, simply the data that is specified in the instruction will be subtracted from the register and the result is stored in register can have sub memory comma data. Here also the contents of the memory location specified by the given addressing mode will be subtracted, data and the result is stored in memory.

Similarly, we have 3 with borrow, in subtraction we require borrow in some cases, SBB register slash memory 2 comma register slash memory 1. Here register slash memory 2 minus register slash memory 1 then carry flag is also subtracted. So, in case of ADC carry flag is added, in case of subtraction with borrow, carry flag will be subtracted because we know that so, for carry flag to be set during the addition, carry has to be generated during the subtraction, borrow is required means if the subtrahend is more than the minuend, then the borrow flag the carry flag will be set there is no separate borrow flag.

So, that is why the carry flag status will be subtracted here, this will be stored in register slash memory 2, similar, to the case with the other two instructions SBB register comma data. So, register contents minus data minus carry flag is stored in register. Sixth one is memory comma data so, data is subtracted from the memory along with carry, result is stored back to the memory. So, these are 6 subtraction instructions, then we have multiplication and division instructions that we will discuss in the next lecture.

Thank you.