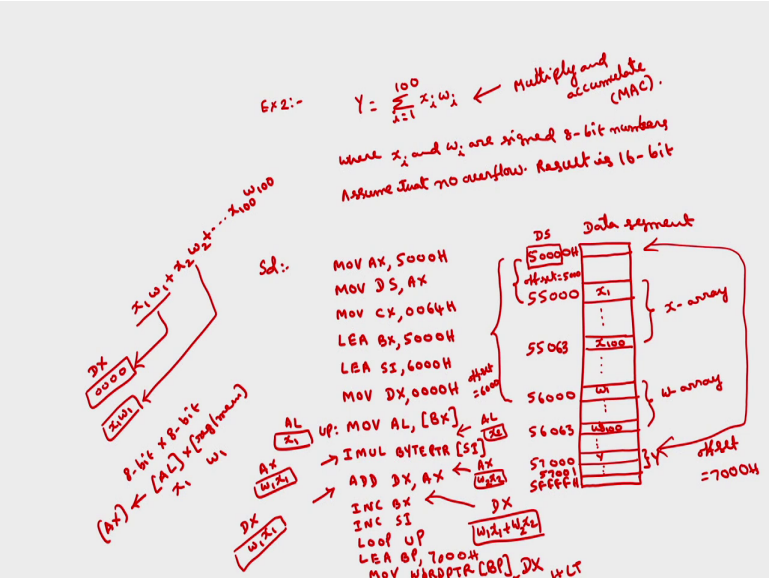
Microprocessors and Interfacing
Prof. Shaik Rafi Ahmed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Lecture - 12
Sum of Products, Multi-byte addition

So, ok in the last class, we have started the Assembly language programming of 8086. So, we have discussed one program.

(Refer Slide Time: 00:41)



So, the next program is to compute say Y is equal to sigma w_i, x_i say, i varies from 1 to 100, where x i and w i are signed numbers 8-bit numbers and I assume that there is no overflow. So, to compute Y is equal to sigma x_i , w_i, so we have 100 numbers of x, 100 numbers of w, you have to multiply and then add. It is nothing but MAC operation, multiple and accumulate. This type of operation will be used in many of the digital signal processing applications called multiply and accumulate.

So, before writing the program as I have told, you have to assume the appropriate memory locations. I am assuming that x_i is stored in some locations in the data segment starting with 50000H, ending address will be 55000H 5FFFFH 64 kilo bytes of the data segment. I am assuming that this is 55000 to if I assume the total 100. So, this is 55063H; here x_1 is stored and here x_{100} is stored.

Similarly, y_1 if you assume that. This is x-array and here I am storing y-array. This is y_1 shown up to y_{100} and let us assume that this is some 56063 H and you store the result somewhere in 57000H y. I am assumed that y is also 8-bit. So, this is the basic assumption sorry this is y this is w.

So, with this assumption so first you have to initiate data segment register with 5000H. As we know that data segment register will contain the upper 16-bits of starting of that particular segment. We cannot directly take the direct data onto this data segment register or any segment register, you take onto some other register say x ,5000 and you move this data to data segment.

And the number of bytes to be added, I will take into CX register. So, that I can use loop instruction 0064 H. Then, you take this effective address. What is the effective address of x array? The starting effective address of the x-array? from here to here the offset is 5000 and from here to the w-array, the offset is 6000. So, I am taking these 5000 offset, where the x-array is present onto some register that can be either BX, BP, SI or DI.

Similarly, you can take this 6000 into some other register. So, I am taking this BX with 5000; so LEA. So, we are loading this SI with the offset of w-array which is 6000H. So, let us assume that result is 16-bit. Because if you multiply two 8-bit numbers, the minimum result is 16-bit. I am assuming that here result is 16-bit. So, I am storing in 5700 and 5701, y value; 57001, I am storing here y value ok.

So, I am starting with this is like you have to multiply and then accumulate. So, for the accumulation, initially I will take that accumulator; I will clear that accumulator and whatever this multiplied value I am going to add to the that accumulated value ok. This is something like if you want to multiply 5 and 6, if you want to add say for example, what I will do is here the x_1, w_1 plus x_2, w_2 plus so on up to x_{100}, w_{100}.

So, I am taking 1 register with 00 initially. You multiply this x_1, w_1 you add to this register and this value will be stored here after the first iteration, x_1, w_1 will be stored. Then, you multiply x_2, w_2 you add to this one, so that the sum of these two will be stored again in this register. This I am taking as some register DX, then again you perform x_3, w_3 and you add to the contents of the DX.

So, it is why I am initiating initially the DX register with 0000; MOV DX, 0000 H. So, all the partial products I am going to add to the DX. Initially, I am clearing the DX. So, after the first iteration, I will add x_1, w_1. So, 0 plus x_1, w_1 will be x_1, w_1 itself. In the second iteration the contents of DX plus x_2, w_2. The previous contents will be x_1, w_1, now x_2, w_2. The sum of these two, this partial product will be stored. After the 100 iterations, this DX contains the total sum that I am going to store in two consecutive locations.

Here, w and x are 8-bit numbers. So, in order to multiply, I am taking one onto some register 8-bit register. MOV AL, contents of BX. So, what is this instruction? This instruction will take x_1 , BX. BX is pointed to offset of 5000. With offset of 5000, what value is there? x_1. So, after the execution of this instruction AL will be loaded with x_1. I have to multiply this with w_1 ok.

So, w_1 is there in location 5600, whose offset from the starting is 6000 which is pointed by SI. So, we know that for 8-bit by 8-bit multiplication and this is we are assuming that they are signed numbers. So, you have use IMUL instruction. So, for 8-bit by 8-bit multiplication, one of the operand will be by default you have taken to AL. So, I have taken this one of the operand to AL that is why. The second operand you can take in to memory or register, any register or memory and where this 16-bit result will be stored? In AX. This is the operation in multiplication ok.

So, here I am going to multiply IMUL BYTE PTR contents of SI. So, what will be operation in this particular instruction? This will multiply AL with this is memory location. Which memory location? Whose offset is present in SI, ok. So, the memory location whose offset is 6000 is w_1. So, this will multiply with this AL is having x_1 and this memory; now here, we have memory which is w_1.

Because I want to take only a single byte, I am assuming that w is 8-bit number. So, that is why I have written BYTE PTR. So, a question may arise that why you have not mentioned here BYTE PTR BX? Because the destination register is 8-bit, by default this will take only byte ok. Whereas here, so one is AL, the other is SI means it can take either byte or word, that is why you have to mention BYTE PTR.

So, now what will happen? This AX will be having w_1 x_1. This I have to add to DX and you store this result into DX. You add DX, AX. So, the initial value of DX is 0000 and AX is w_1 x_1; these two will be added and result is stored back into DX. So, after this

instruction DX contains 0000 plus w_1 x_1. So, this is w_1 x_1 ok. So, this operation, you have to perform 100 times ok.

So, for that in order to point the next x_2 w_2; x_2 is with the offset of 5001, w_2 is with offset of 6001 here. Where we have 5000 and 6000 will be pointed by DX and SI. You increment both the register by 1. INC BX, INC SI, then because CX is pointed to the array of length ok.

And if I use loop instruction by default that CX will be decremented by 1. So, I can use now LOOP UP so that the CX will be decremented by 1. If CX is not equal to 0, it will go to the up. So, in the UP we have to perform x_2 into w_2, we have to add to w_1 x_1. So, where should we use UP, this UP will be now in here.

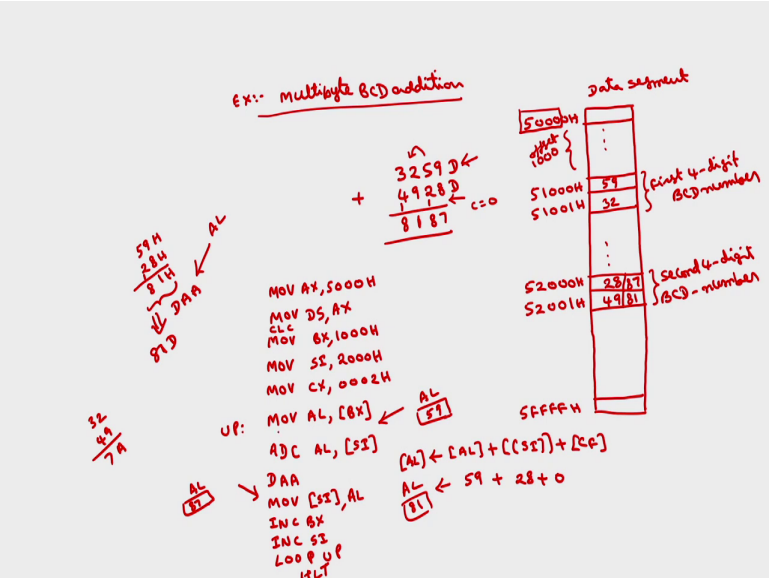
Now, what will be taken into AL, in the second iteration? The contents of the location whose offset is present in BX. So, what is the offset in BX? BX was initially 5000 here, I have incremented by 1. So, that this will be 5001. So, the 5001 contents will be 55000 means 5001 is offset x_2. So, here x_2 will be taken into AL in the second iteration and here IMUL BYTE PTR SI ok. SI was 6000, after one increment 6001, the offset with 6001 will be w_2 value. So, w_2 and w_1 will be multiplied and here in AX, what will be there? w_2 x_2. This I am adding to DX. What was there in DX? w_1 x_1.

So, now after this DX contains your earlier contents are w_1 x_1; now, we are adding w_2 x_2. So, this will be w_1 x_1 plus w_2 x_2. Again, we will increment BX so that it will point to x_3; we will increment SI, it will point to w_3 because it come in 100 multiplications are not over. So, CX will be decremented by 1. So, again it will go to the loop. So, when does this this will come out of the loop after multiplying all the 100 numbers ok. Once it is coming out of the loop, where this result in there in DX ok. You have to store this DX onto the register this.

So, for that you can write LEA BP, you have base pointer with a what is the offset where you have to store the result from here to here, the offset is 7000. So, I am pointing this to 7000. Then store the result where the result is there y 16-bit result is in DX ok. So, MOV WORD PTR BP, DX. So, the result 16-bit result of y was there in DX that I am moving to two locations.

Because WORD PTR; so, BP has pointed to 7000, but the result is 16-bit. So, this will store in 57000 and 57001, then HALT. This is the program which computes y is equal to sigma x_i w_i. So, this is the important operation in DSP, this is multiply and accumulate operation ok. If you take the filter same operation, if you take the transform also we have similar type of operations, this is the second program.

(Refer Slide Time: 17:45)



And I will take third program on multi byte BCD addition. Means if you want to add if I take 2 bytes, 3259 decimal plus 4928 decimal, I want to add this. Let us assume that there is no result is also only 16-bit; there is no overflow. So, what is the expected result? 178118. So, you can extend this to any number of bytes ok. So, I am taking for the sake of simplicity only 2 bytes. Multi byte means you can have any number of the bytes; this is the BCD addition I want to perform ok.

So, for that you assume the appropriate memory locations; same data segment say starting address is same 50000 H ending address will be if it is 64 kilobytes 5FFFFH. So, to store this first number, we require two locations ok. So, I am taking this location with the offset of say 1000. So, what will be the address? 51000 H; 51001 H ok. Here I am storing the first number. This is 4-digit which is 2-byte. So, I require two locations ok. So, even if you store this decimal numbers, the microprocessor will assume that this is a hexadecimal number. So, you have to do some manipulation to get the decimal result ok.

And then, I am storing this second BCD number which is 4-digit BCD number or 2-byte number with the offset of say 5200. Here, I am storing second 4-digit or 2-byte BCD number ok. If I take this example, this can be any data. If I take this example, so this 59 will be here; 32 will be here; 28 will be here and 49 will be here. So, for the sake of simplicity, what I will do is I want to store the result back into this location ok, after the execution of this one I want to store this 87 here and 81 here.

After the execution, I want to store, I want to replace this 28 with 87 and this 49 with 81 for the sake of simplicity ok. Now, what will be the program? We have two-byte addition. So, initially we will MOV that this common instructions will be there. So, data segment will be 5000. MOV AX, 5000H and MOV onto data segment register, then you point this where the first I mean four digit BCD number is there starting of that one that offset into some register this is same as the previous programs ok.

So, I am using again BX and SI and because I have to store, I have to I mean add two bytes, you can store two onto some register. MOV BX, 1000H; MOV SI, 2000H. MOV CX so that I can use the loop instruction. How many byte addition you have to perform? is 2-byte addition. So, we can take 0002 H. Then, first you have to add. Now, the question may arise that why do not you perform? Directly 16-bit addition. Because the microprocessor 8086 is 16-bit microprocessor and we have 16-bit registers, we can add 16-bit together.

The reason for that one is here this is BCD addition. So, in order to get the BCD result back ok. So, this will operate only on register AL only ok. For example, if I take if I add this 59 and 28, the microprocessor will take this one as hexadecimal the actual decimal result will be 87. But this will comes to 9 plus 8 is a 17, 17 is nothing but 15 is F, 10 will be 16; 11 will be 17. This will be 81 H in hexadecimal addition. But we want 87. To get 87 decimal from 81H, 87 decimal and we have discussed a instruction. So, which instruction will be use? There is a instruction called DAA.

So, this DAA instruction operates only on AL, only this is only 8-bit data only that is the reason why I am performing this 2-byte addition. If this is hexa decimal, I can perform 16-bit addition in a single step ok. Because this is decimal BCD, so I am taking this as 2-bytes because I have to manipulate the data in AL only. So, this DAA register will operates with only 8-bit data which is stored in AL, I think the justification is clear.

Then what you have to do in order to perform this addition one of the operand has to be taken into AL? So, MOV AL, BX. So, with this instruction, what happens here? No need to write the BYTE PTR because the destination is 8-bit. So, this AL will be stored with the contents of the memory location whose offset is present in BX. So, BX is pointed to 1000 with the offset of 1059. So, AL will be loaded with 59 value. So, even you have given this one as a decimal, but the microprocessor assume that its hexadecimal number ok.

Now, you have to add to the previous one, A add along with carry; of course, here there is no carry. There is no carry; carry is 0. After this addition, you may get carry from here to here ok. So, for that what you have to do is you write a instruction to clear the carry CLC, we have discussed in the earlier class. This will clear the carry flag. So, even if I add ADC here because the carry flag was reset and this none of this move instructions will affect the carry flag. So, carry flag is 0.

So, this two data, ADC AL , contents of SI. So, what is the operation here? The contents of AL will be added with contents of contents of SI plus carry flag status. These three will be added and result is stored in AL. The operation inside this ADC AL, contents of SI is contents of contents of SI which is contents of SI is 2000 in a 2000 offset 28 is there.

So, this with this instruction, this AL is having 59. This will be 28 and because you have cleared this carry flag, this is 0 ok. So, these three will be added and the result will be stored in AL. So, what will be there in AL? This will be 81 because this will assume hexadecimal, but the correct actual decimal result is 87. So, to get that after this you have to write DAA. So, that in AL, we will get 87 ok.

Then, we have to decrement CX, until this CX is equal to only 2-bytes. If want 4-bytes, you just simply load this with 4 ok. So, this can be extended to any number of bytes. So, the next instruction you have to decrement CX, but if I use the loop instruction inherently CX decrement is present. So, I will write. So, because the second byte is pointed in 1001 and 2001. So, before writing the loop instruction I have to store the result also.

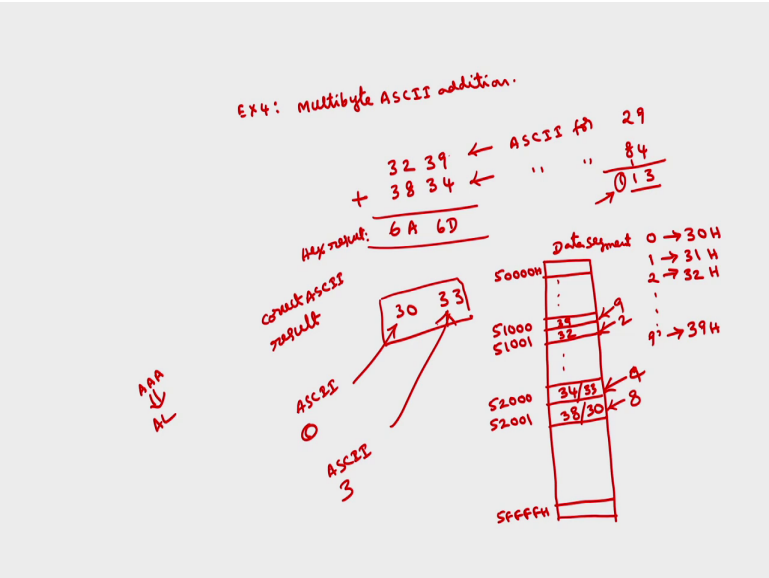
So, where I want to store the result? I want to replace the second number with result; where the result is present now after this in AA, AL? The result 8-bit result 87 is present that I want to store back with the offset of 2000 which is pointed by SI. So, the instruction will be MOV contents of SI, AL. So, that this 28 will be replaced with the lower order result 87 ok.

Now, INC DX, INC SI, LOOP UP. Where should it stop? Now, it should be here. So, that after this INX BX, now AL will be loaded with 32 and contents of SI becomes 49. So, these two will be added, result is stored in AL which will be. So, if I add this 81H, this is 32 plus 49, this will be A7.

So, after the second iteration the contents of AL will be 7A because this will perform hexadecimal addition. If I write DAA, then this 7A will be converted into 81; 81 will be stored and that 81, I want to store back into the destination register, where this second 4-bit BCD number is present. So, I will write MOV SI, AL. So, this 81 will be replaced by 49. Then, INX BX, INX SI because in the first iteration CX was reduced to 1.

Now, after the second addition CX will become 0, this will come out of the loop that is the task and you have to HALT. This is about the multi byte BCD addition. So, this program can be extended to any number of the bytes. If I have 5-bytes, simply load this with 5 ok; if I have 10 bytes, simply load with A ok. So, like that you can extend to any number of bytes, this is about the second the third program which is multi byte BCD addition ok. So, if it is multi byte binary addition, we would have performed without this loop and all we can perform in a single stage ok. The third example is I will do some program on ASCII. Again, say multibyte ASCII addition.

(Refer Slide Time: 31:31)



Here, I will assume that the same two ASCII digits are there. So, I have to add two ASCII digits, then you have to store result in ASCII. For example, if I take this is 32, 39 plus 38,

34. This is ASCII code corresponding to 29; this is ASCII corresponding to 84. As we have already discussed while discussing about this ASCII instructions. So, 0 is nothing but 30 H. If I press the 0 key, if it is ASCII keyboard internally 30 H will be generated, simply you have to add 32, but see 1 means 31; 2 means 32 so up to 9 means 39 ok. So, this is the actual addition will be 29 plus 84.

So, this should be 1311. So, 1 is carry flag. So, the correct expected result will be 13 ASCII ok. If I perform this, if I give this data to this microprocessor, this will assume that these are hexadecimal numbers and what result you will get this 9 plus 4 is 10, 13; 13 is nothing but D 6 A 6, you will get 6 A 6 D, but I want the result as 13 ok.

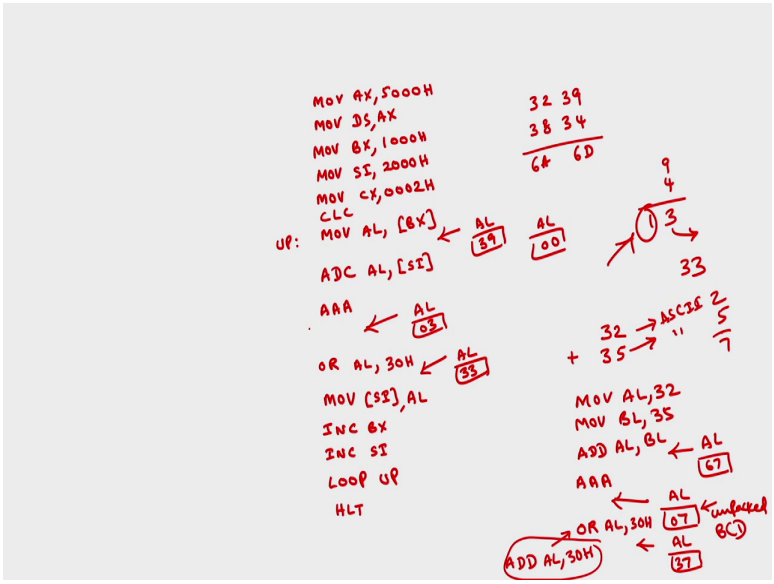
So, for that what is the procedure? Again, you assume that the same memory locations. This in to consecutive locations with some offset, the second ASCII digit in two more consecutive locations and you store back the result in the second number, you just replace the second number with result similar to the previous program.

So, let us assume the same locations; data segment 50000 H with offset of 1000. This is 51000. Here say 39 is stored and 50001, 51001; say 32 is stored and 52000, 34 is stored 52001, 38 is stored. This is ASCII corresponding to 9, ASCII corresponding to 2, ASCII corresponding to 4, ASCII corresponding to 8 and you store the result back onto this ok.

So, what is the ASCII corresponding to this 1? This is decimal result. So, ASCII will be, the result of ASCII will be this this is hexadecimal result. What is the correct ASCII result? This is 9 plus 4; 9 plus 4 is 13. So, this will be 1 is carry; carry flag will be set. So, this will be 3 means 33, this is 8 plus 2 is 10 ok. So, which is 10. Carry flag will be set; 0 means 30.

So, this is the result correct result that I want to store, this is the ASCII corresponding to 0, this is the ASCII corresponding to 3. In this case, we are getting carries ok. So, this result I am going to store back this here. This 33, I want to store here; 30 I want to replace with 38 ok. So, what will be program?

(Refer Slide Time: 36:53)



So, MOV AX, 5000, these are common in all the programs till now I have discussed. MOV DS, AX, then MOV BX, 1000, MOV SI, 2000 and each contains you have to perform again the same logic here also. Now, you have to perform this addition first, then this addition because the instruction I am going to use here will be AAA. ASCII is just after addition ok.

So, here also you have to operate this will operate only on AL. So, I have to use 2-byte additions. So, I have to perform this in two iterations. So, I am going to take this 2 onto CX. So, that I can use loop instruction MOV AL, contents of BX. So, I am taking with, so this 32 39 38 34. So, BX was 39; 39 will come into AL initially, then add along with carry here also, you have to clear the carry flag.

So, that in the first addition, there is no carry; during this, you may get the carry. But you have to add here ADC AL, contents of SI. Then, you have to write AAA. As I have told after this what will be the contents of this AL register? This will be D6. So, if I write this AAA instruction, what is the correct this ASCII 9 plus 4 is? In fact, 13; 1 is carry flag. So, these 3 ASCII character will be 33.

So, I want to I mean I stored 33 into destination register, but after ASCII what happens is this will give unpacked BCD result. So, you can refer to this AAA instruction. So, AAA instruction basically store the result in unpacked BCD form, it is not direct ASCII. So, this

only this 03 will be stored after this, ASCII will I will repeat that again that AAA instruction.

So, in AAA instruction if I add two ASCII numbers say 32 plus I am taking here example without carry so that it will be clear say 35. So, if you want to add these two MOV AL,32 MOV BL, 35, ADD AL, BL so that what will be there in AL here? 67.

If I write instruction AAA, so what is the correct this is ASCII corresponding to 2; this is ASCII corresponding to 5. So, 7, ASCII of this one is 7. So, after this what happens is AL will be containing 07 which is unpacked BCD. This is called unpacked BCD. So, this AAA instruction are just AL contents to unpacked BCD, but it will not give the ASCII.

If you want to get ASCII back, what you have to do is; so, this is 7 ASCII. What is the code? ASCII code corresponding to 7 is 37, but here after this AAA, this will contain only 07. To get 37 back, we can add 30 OR with 30 H. OR AL, 30 H or you can simply ADD also. So, that after this what will be the contents AL will have 37 H; 0 if OR with 3, we will get 3 itself; 0 if you OR with 7, you will get 7 or instead of this you can write ADD AL, 30 H; you can write anyone of this equations this instructions.

Now, here coming back to this main program, after this addition what happens is this 39 plus 34 which is 6D, but actual this one is 9 plus 4 is 13. So, this 3, 03 will be there in AL, but what is ASCII code corresponding to 03 is 33. So, you have either ADD or you have to OR; OR AL, 30 H. So, that AL will be having 33 H which is the ASCII corresponding to 3. This I have to store back into, I have to replace the second number with this result.

So, you have to store means MOV contents of SI, AL. So, that this 33 will be saved back into it will replace the second number 34 with 33. Now, the first byte addition is over you have to go for the second byte addition. Before going for the second byte addition, you have to increment the contents of BX SI and you have to decrement CX that is not required if I use the LOOP instruction.

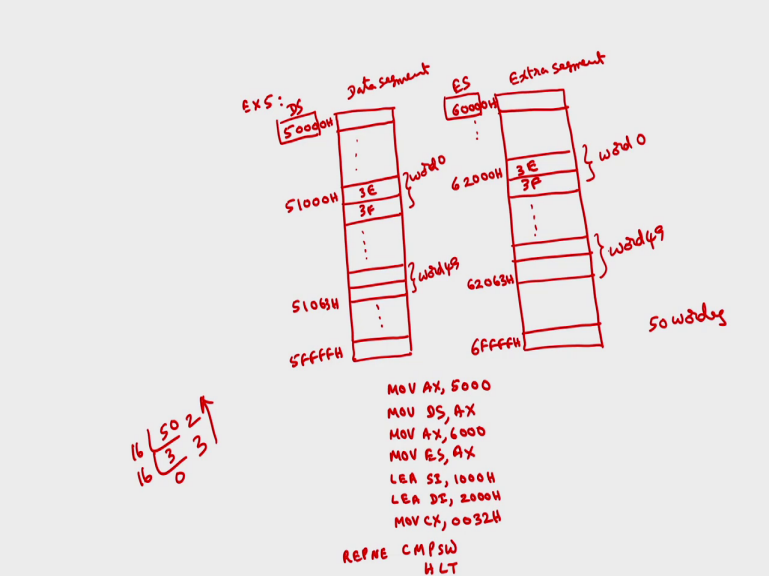
So, INC BX, INC SI, LOOP UP. So, this where should be this UP now? This UP should be here. So, that the second data 32 will be taken in AL; 38 will be pointed by SI because we have incremented SI these two will be added. So, the result will be this 32 replaces the 6A; 6A will be there here in AL and if I use ASCII before ASCII 6A will be there.

So, after ASCII this AL will be containing unpacked BCD. BCD of this one is 00 because 8 plus 2 is 10; 1 is carry. So, result is 10 AL will be having 00. So, to get 30 which is the ASCII code corresponding to 000. So, you have to OR with 30 H, then the result is stored back into the second number, then only halt this the program for ASCII addition.

So, mainly these two programs will demonstrate the functions of DAA and AAA. So, these two are the so two important I mean instructions because a Layman understands only the decimal digits. So, that is why he will expect the result also in decimal. Similarly, both these are actually ASCII, but if I press this key, any key, the corresponding ASCII code will be generated ok.

So, you have to convert back that ASCII code into I mean decimal also, we can stop here without adding the ORING with this 30 H and if I want ASCII only for one further processing of the ASCII digits I can OR with 30 H . This is the fourth program.

(Refer Slide Time: 45:23)



Then, I will discuss another program say compare 50 decimal bytes or words say. So and you stop whenever a match is found or end of the array is reached ok. This is the next program. So, I will explain what is this. Here I will use two different segments, this is data segment; this is extra segment. As I have discussed in the earlier classes, whenever the data is in excess of 64 kilobytes you can store in extra segment.

So, this is starting of this is say some 50000 H, 64 kilobytes 5FFFFH. So, the next address itself is 60000 H is the extra segment and this is 6FFFFH and you take some offset here. This is the offset of some 1000 and you take here offset of some 2000, you compare 50 words nothing but 100 bytes. But I am comparing as a whole word itself. This is word 0, this is the word 49. Totally, I want to compare, this is word 0; this is word 49. Totally, I want to compare 50 words. You take this 16-bit data, you compare with this 16-bit data.

Similarly, you go for the next 16-bit data, you compare with this; like that you goes on comparing this 16-bit words with source and destination. See how to stop, when so whenever the end of the array is reached and this can be this will be see 51063 H and this one will be 62063 H. So, how to end after comparing all the 50 words or we can end in between if there is a match, if this contents are same.

For example, if you have something like 3E3F; this is also 3E3F, after the first comparison itself you have to stop; otherwise if these two are this 16-bit word is different, then you have to go for next comparison ok. So, you have to stop at a point where both the words are matching is one condition and second condition is if the end of the array is reached ok.

Now, here this is slightly somewhat different because I am using two different segments. So, the program will be MOV AX, 5000. This is data segment register and this is extra segment register is 6000. MOV DS, AX, again you will take MOV AX, 6000, MOV ES, AX. So, that I am initiating this data segment and extra segment with 5000, 6000 respectively.

Then, the offset load effective address. So, in string operations we know that this will be source index this is destination index. SI with the offset of this one is 1000 and load data segment with the offset of 2000 and total 50 words you have to transfer. So, MOV CX, 50, what is the hexadecimal of 50? 50 is 16, these are 48, 2; 16x0 are 300, 32 H. 64 will be 100 bytes 32 hexadecimal is equivalent to 50 decimal.

Now, we can directly write compare string word. So, inside this instruction, it will compare the source and destination, but it will compare for the 100 iterations, 50 iterations. If I add a prefix called REPEAT NOT EQUAL, if there is no match, then only it will go here is one condition; without this until the end of this one, once the end of the array is reached, it will come out of this loop.

If I add this prefix also whenever the match is also there, it will come out of the loop. Without this after the end of the array only it will come out of the loop, if I add this prefix whenever match is there also, it will come out of the loop. LOOP UP register is up sorry here there is no loop because this is inside this itself is there; then only simply HALT because this compares SW, this repeat will access loop ok. So, this is the program to transfer to compare 100 bytes and it will stop whenever the end of the array is reached or otherwise if there is a match. So, I will discuss some more programs in the next class.

Thank you.