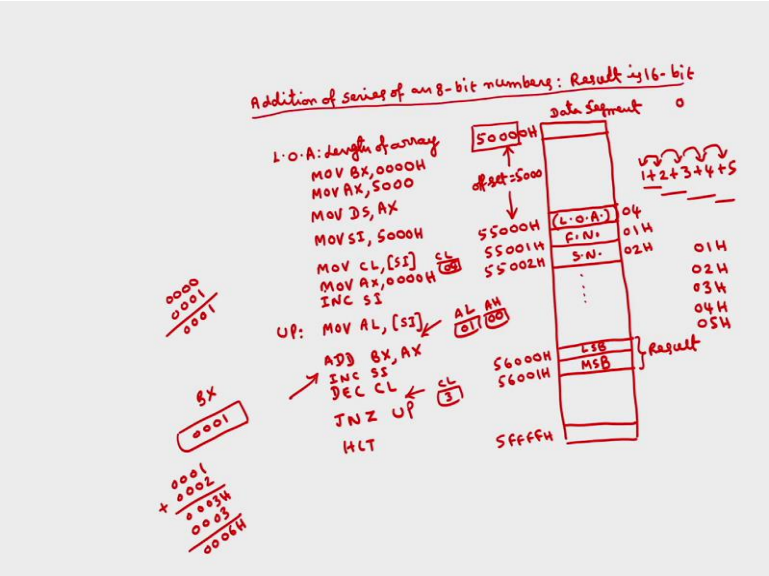
Microprocessors and Interfacing
Prof. Shaik Rafi Ahmed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Lecture – 13
Largest Number, 2's complement Programs

In the last class we are discussing about different 8086 programs. So, the next program is addition of a series of 8 bit numbers and result is 16 bit.

(Refer Slide Time: 01:12)



So, initially we have to assume some appropriate locations before rating the program. So, here I am assuming that, this is the data segment. So, starting address of this one is 50000 H, this we have discussed in the earlier classes also, ending will be 5 FFFFH. So, I am assuming that with the offset of 5000 the length of array is stored say 55000 H the offset from the starting of this address is 5000.

Here I am storing length of array LOA is Length Of Array. And first number is stored in the next location, second number and so on. So, I am going to store the result in two consecutive locations with offsets of say 6000, 6001, here I am going to store LSB and here MSB of the result. Now with this assumption I will write the program. So, the first two instructions are same as the instruction that we have discussed in the earlier programs MOV AX comma, 5000 because the data segment register points to the upper 16 bit address of starting of the segment, then you can take onto data segment register.

Then you point the first number, so total length of array you have to store into some register ok. Here I am assuming that length of array is only 8 bit means you can add at the most 255 numbers. So, MOV SI comma 5000 H; because the offset from here to here is 5000, that offset I am storing in SI register ok. Now I am taking this length of array onto CL. So, that CL will be having length of array, then the first number you have to increment this INCX INCSI so the first number will be pointed by SI. Then you have to add the second number, so for that MOV AL comma contents of SI, so that the first number will be taken into AL because the result can be 16 bit ok.

So, I am going to add 8 bit numbers continuously. So, here I am initiating some other register say MOV DX comma or BX comma 0000. So, I am taking the first number into AL then I am adding with AX also will make 0000 I can make here I am going to write this MOV AX comma 0000 H. So, that both AH and AL will be 00 here I am taking the first number into lower order 8 bit register of AX. Then I am going to add BX comma AX. So, this instruction will add the contents of AX with BX and store the result into BX ok.

I will explain this with some example also that it will be clear. So, after this addition, suppose if I want to add 5 numbers, say 1 plus 2 plus 3 plus 4 plus 5, how many additions you have to perform only 1 2 3 4. So, in general if you want to add n numbers we have to perform only n minus 1 additions that is why here after this MOV CL comma SI in CL we have length of array. So, 1 way is you can directly store the length of array here or you can store length of array minus 1 in 55000 H, because if I want to add n numbers only n minus 1 additions you have to perform.

That is why instead of storing length of array here I will store length of array minus 1 ok. Then decrement DEC CL jump not zero up where should be this up now? So, this should be here, because before going for that one you have to increment INC also INC SI. Because the first number I have added with the second number, then you have to add the third number with the first number. So, like that we have to add this is halt, where should be this up? Up should be here. I will explain you with an example so that this program will be clear ok. Let us take the same numbers I want to add 0 1 H 0 2 03 04 so on up to 05.

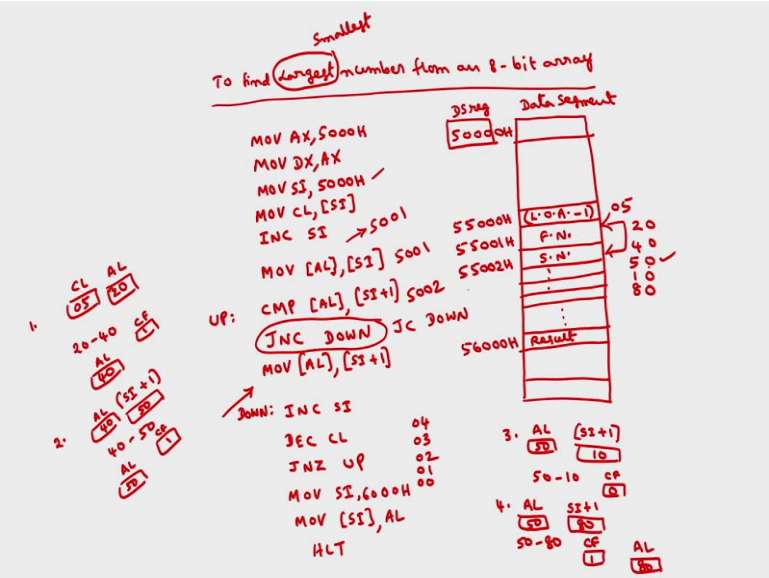
So, I will store here 4; 04 because if I want to add five numbers we have only 4 additions ok. Now in the first iteration after this CL will contain 4 AX will be 000 INC SI, SI means SI will points to 5001 the first number is this 01 here 02 here and so on. So, after this instruction AL contains 01 and what is AH? Because AX is cleared so, AH is 0000 and if I add BX comma AX in BX we have 000, after this BX will contain 0000 plus 0001 this will be 01. So, the addition will be BX was 0000 and AH was is 00 AL is 01, so you will get the 0001 H.

Then we are incrementing SI, so SI will points to the second number the address of the second number then DCS CL. So, CL becomes now 3 after adding the first number DCS becomes 3; it has to be 0. Then jump not zero up because this is not zero it will go to the up and because you have incremented SI the second number will be taken into accumulator. So, in the second iteration, accumulator AL will be having second number which is 0 to H that will be added with AH is 00 only. So now, in the second iteration the addition that will be performed will be BX contains 0001 plus AX contains AH is 00 and AL is 02 this will be 0003 H.

Similarly, CL will be decremented now it will becomes 2, jump not zero it will go up because INC SI is there, so SI will be incremented. So, it will point to third number which is 3. So, in the third iteration what happens, this will be there in BX and AX contains AH is 00 AL contains 03, this will be 0006 H. Then in the next iteration fourth number will be added, next iteration fifth number will be added. So, after adding all these numbers so it will come out of the loop ok. So, because we are adding this first number to 0; we require length of array it should be here, it should be length of array only this will be 05 ok.

So, this is about the addition of series of 8 bit numbers. Actually here if I add two numbers at the same time, 1 and 2, then the sum of this with 3, sum of this with 4, sum of this with 5 then we require 4 additions. But here what I am doing is, I am taking first 0 I am adding first 1 then 2 then 3 then 4 then 5. So, we have total 5 additions ok, so length of array you have to take in 5500 instead of length of array minus 1. So, this is about the addition of series of 8 bit numbers result is 16 bit.

(Refer Slide Time: 12:12)



So, the next program is finding the larger of a series of numbers. We have to find the largest number from an array. To find the largest number from an 8 bit array, here also you have to first assume the appropriate memory location before writing the program. So, I will take the same data segment address 50000. So, data segment register point to 5000 and I am assuming that length of array is with the offset of 5000 H. Then the first number in 55001 second number in the next location and so on.

And I am going to store the result in location because this is only 8 bit we need only 1 location 56000 H this is my specifications of the address. Then the first two instructions will be same MOV AX comma 5000 MOV DX comma AX. Then to take the length of array into CL register the same instructions which we have used in the earlier program MOV SI I am pointing SI to the offset of length of array which is 5000 H, MOV CL comma contents of SI then increment SI. So, that the SI will points to the first number, then you have to compare with first two numbers. So, MOV AL comma contents of SI compare AL comma contents of SI plus 1 ok.

So, with this instructions CL will points to length of array and first number I have taken into AL, second number I am comparing with AL using compare instruction second number will be there in SI plus 1. So, SI was after this increment 5001 SI plus 1 becomes 5002, first two numbers will be compared ok. Now my logic is here I want to keep always the larger of this number into AL ok. If AL is already having the larger number,

then after this comparison operation what happens to carry flag? So, carry flag will be set, if a borrow is needed or if this subtrahend is greater than minuend ok.

So, here if AL is greater than SI plus 1 AL is greater than SI plus 1 then carry flag will be reset. If AL is less than SI plus 1, then carry flag will be set ok. So, if jump no carry down so, when does this will be true no carry, if AL is already larger, in that case we have to go for comparison of the third number. In case if this is false, means if carry flag is set then what you have to do? SI plus 1 that is second number is larger than the first number that I have to take into accumulator. So, for that the instruction will be MOV AL comma contents of SI plus 1. So, at this point in any case AL will be having larger of these two numbers ok. If AL is larger directly here this will skip this one because carry will not be result, so it will come out of the loop. Otherwise the contents of SI plus 1 if it is larger than AL then that will be taken into AL. So, here you have write the down program.

So, comparison of the first two numbers is over, then we have to go for the third number which is stored in 55000 3H ok, for that we have to increment SI. So, that now SI will points to, here SI was 5000 after this 5001 so this is 5001 and this is 5002. So, it compares first and second number to point to 5003. So, we are incrementing this SI, so that SI plus 1 becomes 5003 ok.

Now I am not going to take any number onto AL; DEC CL jump not zero up, this up I will take to direct comparison operation. So, what happens now? So compare AL, so between these first two numbers the larger of this number is present in already AL, because if AL is having this larger number I will just skip this instructions and I will go to the jump. If second number is larger I am going to replace second number onto AL, then I am going to go for the next comparison ok.

So, in any case this AL will be having, after this particular first loop first iteration AL will be having the larger of the first two numbers that I am going to compare with the third number. So, here this SI will points to now this was 5002, I have incremented by 1 so now it will becomes 5003. So, 5003 will be third number and in AL what will be there the larger of the first two numbers. So, I am going to compare the larger of the first two numbers with third number.

Again the same logic if this no carry means AL is the contents of that is the larger of the first two numbers is larger than third number also. Otherwise third number which is there in SI plus 1 I am going to replace into AL same logic. So, at the end of this loop what happens AL will be having the result and that we have to store in this 56000 H for that MOV SI ok. There is BX BP also because SI have used but the purpose of SI is over I can use same use the same SI for some other purpose.

So, 6000 H and where the result is there in AL, I am going to store into SI AL this is halt ok. I will explain this also with an example so that it will be clear whether I have to store the length of array here or length of array minus 1. Like in previous case addition it is length of array only, because we are adding 0 each number to 0. So, that is why the number of array length of array is equal to number of additions. Whereas, here so I am comparing the first two numbers, so here probably this you have to store length of array minus 1.

This will be clear if I take one example you take the same 5 numbers. So, if I take 5 numbers here, so probably this is equal to 4 I am going to prove this ok. So, 5 numbers are in 5000 50001. So, we have first number, second number, third number, fourth number, fifth number; this is some 20, this is some 40, this is some 50, this is from 10, this is from 80 say. So, I explained this with this particular set of data. So, initially I am going to store CL with 5, so I will come to that point. So, whether we have to store 5 or 4, so I am taking CL with 5 ok. So, here CL with the 5 first, then AL I am taking the first number which is 20 in the first iteration.

Then I am comparing 20 with 40, so in this comparison because SI plus one becomes 55002. So, I am comparing 20 with 20 minus 40 this is the operation that is performed in the comparison in the first iteration. So, for 20 minus 40 what happens, carry flag will be set because this requires borrow. So, carry flag is set so jump not carry this will be false. So, what happens this will skip and it will go to this instruction? So, SI plus 1 contents will be 40, so this AL will be loaded with now larger of the first two numbers which is 40.

So, we have compared the first two numbers in the first iteration in the first iteration this one and AL will be loaded with 40. So, after that I am incrementing the SI, so that SI will points to 5002 and CL I am decrementing CL I am decrementing. So, CL becomes 4

because this is not 0 so it will go to up. So, in the second iteration what happens compare AL, AL is having 40 after the end of the first iteration and what is SI plus 1? SI was after this SI increment 5002 SI plus 1 becomes 5003, in 5003 what is there 50. Now here AL contains 40 and SI plus 1 will be 50, so these two will be compared so 40 minus 50.

As we have discussed in these instruction set compare means the operation inside this comparison is subtraction only ok. The difference between the compare and subtraction is in case of compare subtraction is performed and the result is discarded. Whereas, in subtraction the result is also stored, here depending upon this subtraction result it will set or reset the flags. Now what happens to carry flag? Here the carry flag will be set because borrow is there, here also carry flag will be set because borrow is needed ok. So, now what happens? So because this is false so the contents of SI plus one which is 50 will be taken into accumulator. Now CL will be decremented by 1, CL will be decremented by 1 so CL becomes 3.

So, because this is not 0 it will go to the up before that it has incremented the SI by 1. So, that in the third iteration in the third iteration AL will contains the larger of the first two the largest of the first 3 numbers which is 50 and SI plus 1 points to now the fourth number which is 10. Then here again comparison means 50 minus 10 will be performed, because 50 is larger than 10 carry flag will be reset. So, if this is carry flag is jump not carry this is true.

So, this will go to the down it will skip this operation MOV AL comma SI plus 1 operation ok. Now AL will be containing this again increment SI then decrement CL, CL becomes 2 CL becomes 2. So, that in the fourth iteration AL was 50 only, as it is because the 10 number the fourth number was less than 50, so AL remains 50. Now SI plus 1 will points to the last number which is 80, so here 50 minus 80 will be performed. Carry flag will be set because 50 is less than 80.

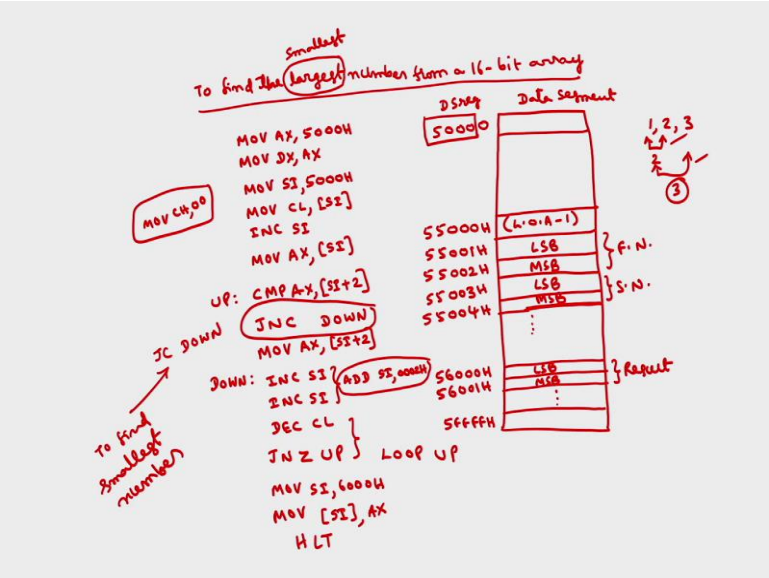
So, if carry flag is set this is false JNC down is false, so it will exchange this AL with SI plus one contents so AL will be having 80. So, then this 02 becomes 01, but comparison is already over. So, this should be 00, so that is why here you have to take length of array minus 1 is it clear? Now already this AL is having largest of the first 5 numbers. So, this if I take here length of array minus 1 this will becomes 00, then it will come out of the loop. Now AL is having the largest number that you have to store in the memory location

whose offset is 6000, so for that I have taken SI with 6000 H and I am storing this largest number in that particular memory location.

So, this is about to find the largest from a series of 8 bit numbers ok. Suppose if I want to find out this smallest instead of largest if I want to find out the smallest. So, the only difference is instead of JNC, if I write JC this will find the smallest number because I want to always keep smallest number in AL ok. So, for that I have to check for jump on carry ok. If already AL is having smallest number smaller of two numbers, then so jump on carry this will be true it will go to the next comparison. Otherwise it will replace with the smaller of these two numbers ok.

So, we can check this operation to find out the smallest we have to replace JNC with JC ok, the remaining all operations will be same. Now this is finding the largest or smallest number from an array of 8 bit numbers. Suppose if I have an array of 16 bit numbers. So, you have almost similar program ok. So, we have only few changes. So, the next program is to find the largest number from a series of 16 bit array.

(Refer Slide Time: 28:33)



Same thing again, so I will start for any program this is the important thing is you have to assume the appropriate memory location before writing the program.

So, data segment data segment register 5 FFFF H, then assume that length of array is of course here also you have to store length of array minus 1, because of the logic that I

have explained in the earlier program. Now store length of array minus 1, I am storing this in 55000 H with the offset of 5000 H and the array starts from the next location onwards which is 55001 H, 55002 H, 55003H, 55004 H and so on. And I am storing the result in 56000 H and 56001 H. Here the first number LSB of this one here MSB is here.

Here second number LSB is here MSB is here and here the result LSB is here MSB is here. So, as we have seen we are going to use the same logic that is why here you have to store length of array minus1, because the number of comparison will be one less than the length of the array ok. Suppose if I want to compare 1 2 3, if I compare with 0 1 2 3 three comparisons ok, if I compare the first two numbers in the first iterations. So, you will find out the larger of this 1 which is 2 this I am going to compare with 3, so you will get 3 result only two comparisons ok. So, always you remember that whenever you want to I mean compare with 0 or in addition also if I directly perform the addition of two first numbers then the number of additions should be less than one less than the length of array.

If I add with 0 if I compare with 0 as a first step, then length of array is equal to the number of additions or comparisons ok. Now what is the program? So, I will repeat the first two instructions MOV SI comma offset of the length of array minus 1 then take into CL, CL will contain length of array minus 1. Then the first number is there in INC SI, so that SI points to the first number. So, we take this onto AX. So, because the destination register AX is 16 bit, if I write contents of SI it will take a word from the SI. Otherwise you can write word ptr here, but it is not required. Then you have to compare with compare the second number which is stored in AX comma contents of SI plus 2.

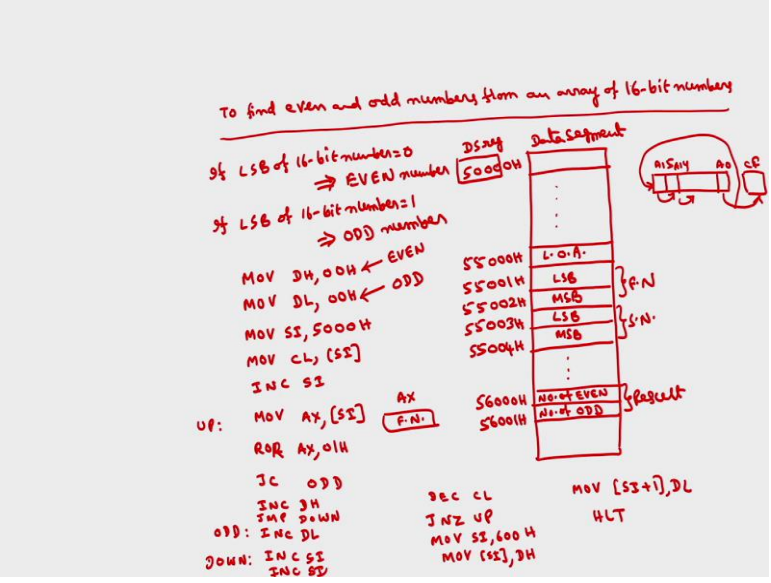
Here SI was 55001 and SI plus 2 become 55003, where the second number is stored ok. Jump not carry down; means already AX is having larger of the first two numbers the same logic ok. Otherwise what you have to do you have to exchange with SI plus 2 contents and down will be here where we are going to because one comparison is over we have to decrement CL, before that we have to increment SI twice. Because this is 16 bit number this is one of the difference INC SI, INC SI or you can write instead of this add SI comma 0002 H these two are same. But we prefer INC SI INC SI only, because this is single byte single byte, here we have this single byte and there are 4 hexa decimal digits means 2 more 2 more bytes ok.

So, INC SI, INC SI then DEC CL here also if I want to use the loop instruction, I can make this into CX CH you can make as 0. So, in that case if I make this CL into this one, now directly contents of SI onto CX. So, here anyhow this will be two bits only. So, if I make here MOV CL CH comma 00. So, that CX will be having lower order 8 bits will be that length of array minus 1, then instead of this DEC CX I can write directly loop also.

So DEC is jump not 0 up, if I use this MOV CH comma 0 0 also instead of these two instructions I can write directly loop up. Because inside the loop we have decrement of CX by 1 ok, up where should be this up this comparison. So, at the end of this loop AX will be having largest number that I want to store in 56000 H. So, I am taking MOV SI comma the offset is 6000 H MOV AX comma contents of SI alright. So, this is the program which finds the largest number from a series of 16 bit numbers ok.

And similarly if I want smallest number, we have to replace JNC with JC ok. So, this is about this program the next program that I am going to discuss is to find the even and odd numbers in a array 16 bit array ok.

(Refer Slide Time: 37:02)



To find the even and odd numbers from an array of 16 bit numbers ok, So you have to assume the appropriate memory locations 50000 H data segment register will be 5000 H this is data segment of 64 kilobyte. Then I am assuming that length of array is in 55000 H.

So, I will come to this point whether this should be a length of array or length of array minus 1 ok. Then the first number I mean next locations 55000 H, 55002 H the first number LSB of this one MSB of the first number. Similarly, second number LSB, MSB and I am going to store result in 56000 H, 56001 H; LSB, MSB. Here basically the logic to find out this even and odd numbers is if I check only the lower LSB bit it is enough ok. If LSB bit of 16 bit number is 0 implies even number. If LSB of 16 bit number is 1 implies odd number, so to store this number of even numbers and odd numbers so I am going to take 2 registers.

So, the number of even and odd numbers can be because I am taking length of array in only single bit 255 only. So, at the most they can be less than I mean 255 ok. So, for that I am taking MOV say DH comma 00; MOV DL comma 00, I want to store here the even numbers number of even numbers, I want to store in DL number of odd numbers ok. So, the first two steps are same I am not repeating here so MOV AX comma that that you can write, then MOV SI comma length of array contents of 5000 MOV CL comma contents of SI.

So, CL will points to length of array or it can be length of array minus 1. So, I will find that value at the end of the program, then we have to find out the first number you take the first 16 bit number into AX. Any other register MOV this is INC SI, so that SI will points to 55001 where LSB of the first number is present. So, I am taking the first 16 bit number directly into register AX or any other register for that matter MOV AX comma contents of SI, so AX will be having first number. So I am going to rotate to get this LSB bit I have rotate instruction ok, so in the rotate instruction if I write ROL AX comma 0 1.

If you want to rotate by only 1 bit, you have to write the register followed by that 1 bit. If you want to rotate by more than 1 bits you take that into CL and you write CL here this we have discussed in the instruction set ok. Normally as the name implies rotate the contents of this AX register to the left ok, here carry flag will be there sorry this should be R because I want LSB bit. So, this should be right here the carry flag will be there. So, this least significant bit which is A0 this is A15. So, this A15 will go to A14 and so on this will go here this will go to carry as well as this will also go to A15.

This is what the operation inside the this ROR AX. So, whatever this LSB bit will go into carry flag now you just check the carry flag status ok. Jump on carry jump on carry

means what does it mean? If this is true is it even number or odd number, if carry is there means LSB bit is 1, LSB bit is 1 means it is a odd number go to odd where I am going to increment DL register, otherwise you have to increment DH register. DH was initially 0 if this is false means, obviously that number is even number. So, we have to increment the even register which is DH INC DH ok.

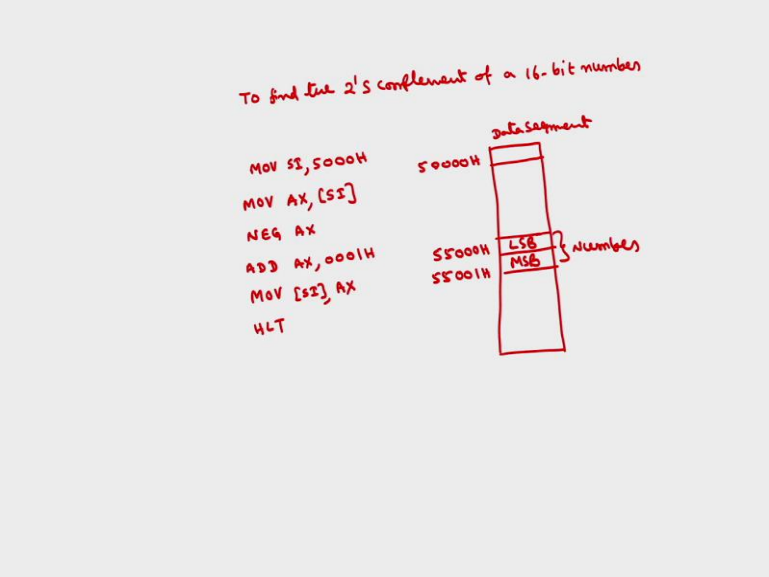
Then odd here, if the number is odd without incrementing the DH it will increment only DL. So, increment DH and then it will jump to the next comparison jump to down, because number can be either even or odd. Once if it is found that this is odd number then no need to increment this DH; it will go to the next comparison, otherwise it will increment the DL and then it will go to the next comparison. Next comparison is here we are going to increment INC SI this is down, again INC SI because this is 16 bit number you have to increment the INC by twice. So, that it will point to the second number which is 55003 ok.

Then we have to write DEC CL, so because you have checked the first number. So, here only checking of the number there is no comparison and addition, so in 55000 you have to store only length of array ok. So, DEC CL jump not 0 up where should be this up now? This, we are taking the second number into AX how take the second number into AX? So MOV AX comma SI so, that the second number will be taken into AX; again you are rotating the second number. So, we will find whether this is even number or odd number, if it is odd number we are going to increment DL. If is even number we are going to decrement only DH and after each I mean checking, we are going to increment the SI by twice and we are going to decrement by CL 1.

So, until that CL is equal to 0 we will repeat. So, I want to store this instead of LSB MSB I will store even and odd otherwise, I want to store the number of even numbers, number of odd numbers. So, at the end of loop this loop where the number of even numbers is there? In DH so, you initiate MOV SI comma 6000 H MOV in 6000 I want to store the even, even is there in DH. So, I will store contents of SI with DH and MOV contents of SI plus 1 with DL and we write here I will write here MOV contents of SI plus 1 with DL then halt. So, this is the program which finds the number of even and odd numbers in a array of 16 bit numbers ok.

And the simple logic is we are going to check the LSB bit, if LSB bit is 1, it is an odd number, if LSB bit is 0 it is an even number. So, like that we are going to write this program.

(Refer Slide Time: 48:10)



So, the next program is to find the twos complement of a 16 bit number ok. Suppose if this 16 bit number is present in a data segment with a offset of same 55000 50000 and in 5000 we have 16 bit number 55000 H and 55001 H, this is LSB this is MSB and the number.

So, in order to find out the twos complement of a number ok. So, I will take first this number onto AX MOV SI comma contents of 5000 sorry, 5000 H MOV AX comma contents of SI ok. Now to get the ones complement of a number there is a instruction called negate. So, I will take the ones complement then I will add 1 and you store the result back onto the same 55000 H and 55001 H ok, you store the result back onto this. So, for that we write NEG negate AX.

So, that AX contents will be complemented, this ones complement, then add AX comma 0001. So, that we will get twos complement and you have to store this again into the same location. So, you have to store back MOV this is reverse operation ok. Here you have transferred this number onto AX now, in AX you have the twos complement number that I have to transfer onto the SI, MOV contents of SI comma AX then halt. This is a simple program which computes the twos complement of 16 bit number and it

replaces the original number with its twos complement ok. So, I will discuss some more programs in the next class ok.

Thank you.