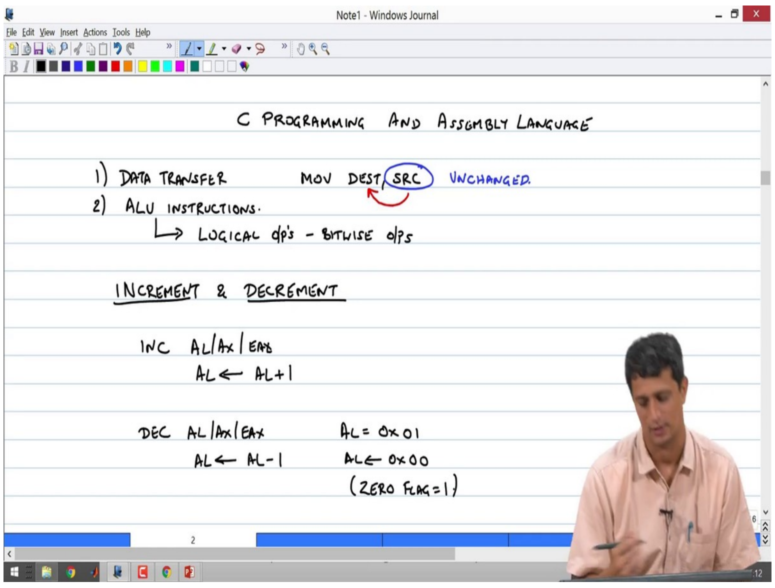
C Programming and Assembly language
Prof. Janakiraman Viraraghavan
Department of Electrical Engineering
Indian Institute of Technology, Madras

Lecture - 04

Welcome back to the course on C Programming and an Assembly language. So, in the last couple of lectures, we discussed two kinds of 8086 instructions.

(Refer Slide Time: 00:21)



C PROGRAMMING AND ASSEMBLY LANGUAGE

1) DATA TRANSFER MOV DEST, SRC UNCHANGED.
2) ALU INSTRUCTIONS.
↳ LOGICAL op's - BITWISE o/ps

INCREMENT & DECREMENT

INC AL/AX/EAX
AL ← AL+1

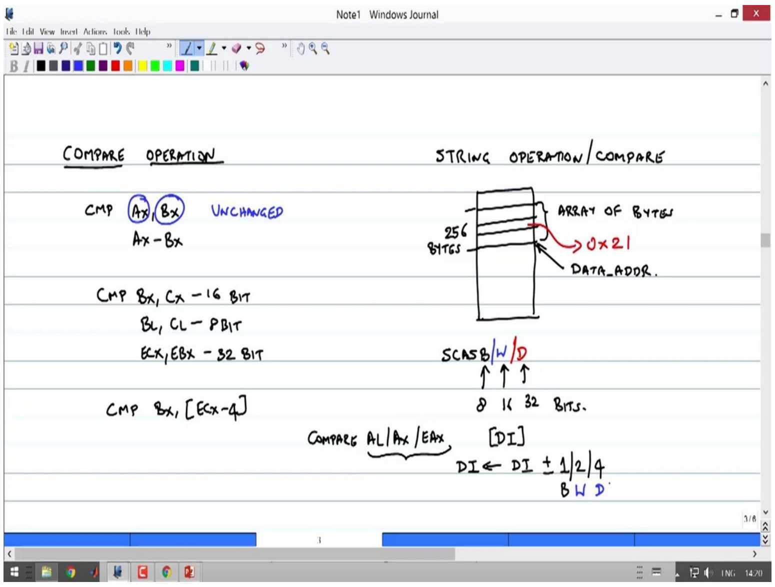
DEC AL/AX/EAX
AL ← AL-1

AL = 0x01
AL ← 0x00
(ZERO FLAG=1)

One is the data transfer. And the key summary for the data transfer is we did MOV destination comma source, and the source was unchanged, unchanged in the process. Data would move from the source to the destination and none of the flag registers would be affected in the process. We then moved on to discuss a few ALU instructions like the ADD, subtract, and XOR, and you know all those instructions were bitwise operations. So, the logical operations were bitwise ops. So, we will continue with the ALU instructions in this discussion, and we will introduce to very simple, but very important instructions here.

So, in this class, we have the increment and decrement instruction. So, what we do here is we could do the following INC AL which means increment the value of AL right. So, what does this do, it just replaces AL by AL plus 1. Alternately you could also do it on AX or you could do it on EAX. So, this is a way of implicitly adding one to the register without calling the ADD instruction. Similarly, you could do a decrement AL or AX or EAX. So, what does this do, it will simply replace AL by AL minus 1. So, this is an ALU operation. So, implicitly the flags will be affected. So, if AL for example happened to be a 0x0001 , oh, this is 8 bit, so it would just be 0x01. And you decremented AL, then after this instruction AL would now become 0x00 and hence the zero flag will be set =1.

(Refer Slide Time: 03:23)


COMPARE OPERATION

CMP AX, BX   UNCHANGED

AX - BX

CMP BX, CX - 16 BIT

BL, CL - 8 BIT

ECX, EBX - 32 BIT

CMP BX, [ECX-4]

STRING OPERATION / COMPARE

SCASB/W/D

8 16 32 BITS.

COMPARE AL/AX/EAX   [DI]

DI ← DI ± 1/2/4

B W D

So, moving on with a few more ALU operations, the next important operation is the compare operation. So, the mnemonic for this is CMP, I could do AX comma BX. What does this do? It simply performs the operation AX minus BX, but does not put the result into any register which means that both AX and BX are unchanged in the process. So, if none of these registers are actually affected then what actually gets affected in the micro processor, it is only the flag registers. So, what is the use of this instruction, it is used when you want to affect you know compare two numbers and then make a branching or looping decision in the assembly language. So, it affects the flags and based on the flags, I could do a branch or a loop which I will cover later in my in another discussion.

So, similarly you could do this with you know compare BX comma CX – this is 16 BIT or BL comma CL-this is 8 BIT , or you could do ECX comma EBX, and this is 32-bit. And with any ALU operation like I mentioned earlier, you can use any kind of addressing which means that you could do CMP BX comma ECX-4. So, this is a register indirect with offset addressing that we are doing here, and we are comparing the 16-bits of BX with 16-bits pointed to by ECX-4 . There are a few more interesting things that you can do with 8086 architecture which happens to be a string operation or a string compare.

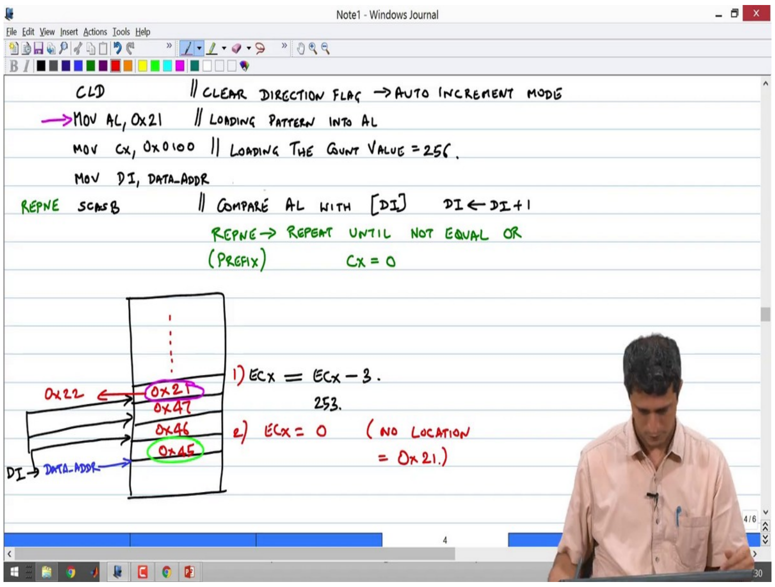
So, let us assume that we have two arrays or you have an array in a in a memory right, and you want to search for a particular pattern. So, let us assume that in the memory in may be 100 consecutive locations, I have a particular string or an array of bytes that have been stored. So, this is my array of bytes, and let us assume that this is may be 256 locations. And what I want to do is to search for a particular byte pattern in this 256 location. So, let us assume that I want to see if some location in this actually has the contents 0x21, this is the very arbitrary number that I am just choosing here, just to illustrate this particular operation.

So, let us assume that we want to search for 0x21 in 256 bytes stored in somewhere in memory. So, let us assume that this address location starts at some address called data underscore address. So, starting from data underscore address, I need to scan through 256 locations in order to see if anyone of them has the pattern 0x21. So, in order to do this, there is something called a scan string SCAS and since we are looking for a byte of information I am going to call it SCASB on the other hand remember that I can also search for a word or Dword. So, as with all our earlier discussions, this is 8-bits, this is 16-bits and this is 32-bits.

So, what does SCASB do? It compares EAX right or if it is I am if I am trying to do a SCASD, or it would compare AX, or it would compare AL with the contents of the destination index register that is pointed to byte. So, whatever value the destination index has, it is going to compare one of these registers with either a byte or word or a Dword pointed to by the destination index.

And after you finish this operation, the destination index register is automatically incremented or decremented by 1, 2 or 4. Now, it is decremented by 1 or increment by 1 if you are talking about a byte; it is incremented or decremented by 2 if you are talking about a word; it is incremented or decremented 4 if you are talking about Dword.

(Refer Slide Time: 09:57)



```
      CLD                   // CLEAR DIRECTION FLAG -> AUTO INCREMENT MODE
   -> MOV AL, 0x21          // LOADING PATTERN INTO AL
      MOV CX, 0x0100        // LOADING THE COUNT VALUE = 256.
      MOV DI, DATA_ADDR
REPNE SCASB                 // COMPARE AL WITH [DI]     DI <- DI+1
                               REPNE -> REPEAT UNTIL NOT EQUAL OR
                               (PREFIX)        CX = 0
```

1) ECX = ECX − 3.
253.

2) ECX = 0 (NO LOCATION = 0x21.)

So, now lets see how we can use this instruction to perform the task that is given to us namely we want to search for an array, search in an array which starts at data underscore address and find if some location has a particular string called 0x21. So, obviously because EAL or AL has to be the register that is going to be compared with , so we load this particular pattern into the AL register. So, what are we doing here we are loading the pattern into AL.

Now, we need to search for the pattern in about 256 locations. So, therefore, we need to load this count into some particular register. So, it turns out that there is a designated register that has to be used for this purpose and that is nothing, but the ECX, C stands for the counter. So, by default any counter has to be loaded into ECX register.

So, we are going to move   CX with 0x0100. What am I doing here loading the count value which is nothing but 256 by the way in decimal. Now, also I have to initialize my destination index to point to that particular starting address. So, MOV DI comma; so, let us go ahead and add this particular instruction that we just discussed which happens to be the SCAS B. So, what is this instruction going to do, it is going to compare AL with contents of DI, and then it is going to either increment or decrement DI.

So, how do you determine if the data I mean the address is going to be incremented or decremented that is determined by the direction flag which can be set or reset. So, therefore, before we start the program, we are going to do this operation called CLD which is nothing, but clear direction flag, so that the string operations work in auto increment mode. So, after you compare AL with the contents pointed to by DI, DI will be incremented by 1 and why, is it 1, because we are doing SCASB, B for byte, and therefore, we are incrementing the address by one. Now, this unfortunately does it only for one particular location .

So, for example, if my memory had,you know let me just illustrate with an example here. These are my n locations. Let us say this data in my data address was pointing here. And the data that is being stored in there is 0x45 , 0x46, 0x47 and 0x21. Remember that we are looking for this particular pattern 0x21 and that is what has been loaded into the AL register. So, when I execute this SCASB, once then it is only going to compare this particular value 0x45 with my AL register which is 0x21. And of course, the two numbers are not equal , but unfortunately we have still not been able to achieve the comparison with the entire array.

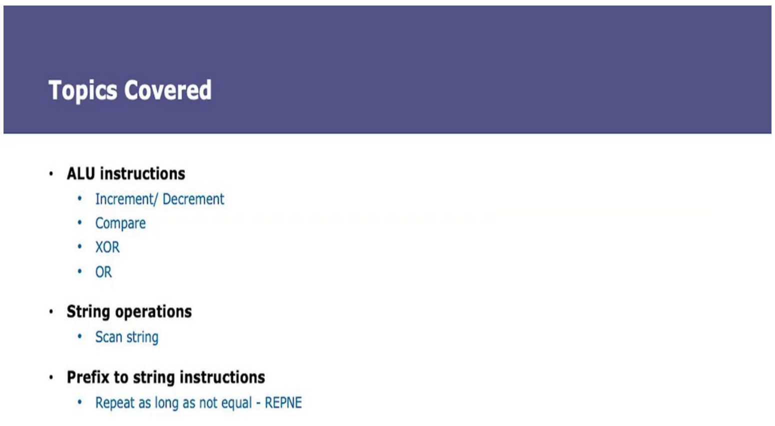
So, how is that achieved, that is achieved with something known as a prefix that we add to this particular instruction REPNE. So, what does REPNE do, REPNE is repeat until not equal or CX equal to 0. So, this is known as a prefix to a string instructions. So, what do we call this, this is a REPNE is a prefix. So, what we are doing here is known as long as none of the locations have 0x21, you continue this SCASB operation again and again. And what is a SCASB do, it just compares then increments the destination pointer. So, after the first operation, my DI , my DI was pointing here 0x45. After the first comparison DI would then point here 0x46; second comparison, it would point here 0x47; the third comparison, it would point here 0x21.

So, when it comes to the third particular value it finds that 0x21 has actually matched with my AL register and hence the comparison will stop. So, what does this mean, at when this comparison stops ECX will simply be ECX minus 3. It will be ECX minus 3 right. So, it will simply be 253. So, the idea here is to use a counter . And if you do not use the counter what happens is, it can go into an infinite loop. So, therefore, you have to tell it how many bytes to go and check with, so that is the limit of the number of locations in the array or the string that we are going to search and that is loaded into the ECX register.

So, you compare you increment the destination index because the direction flag has been reset and you keep doing this operation until you have encountered the particular pattern or the ECX register goes down 0. So, here I showed an example where this 0x21 occurred now what if this 0x21 did not occur. So, instead of 0x21, if it was 0x22, and all the 256 locations did not have any of this any the pattern 0x21, in that case the comparison will fail always and ECX will eventually come down to 0.

So, this is case one, case two ECX simply be 0 because no location equal to 0x21. So, remember that this is not a software loop, but it is a hardware loop which implies that the instruction is just issued once, and the operation finishes in one shot and only then returns to execute the next particular instruction. And therefore, it is extremely fast.

(Refer Slide Time: 19:26)



Topics Covered

- **ALU instructions**
  - Increment/ Decrement
  - Compare
  - XOR
  - OR
- **String operations**
  - Scan string
- **Prefix to string instructions**
  - Repeat as long as not equal - REPNE