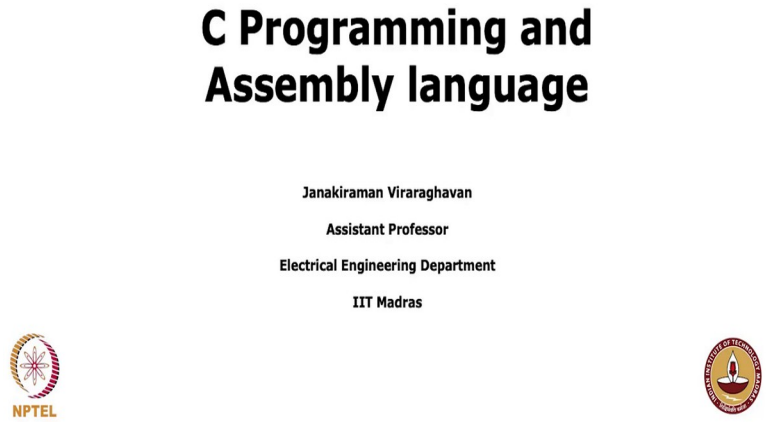
C Programming and Assembly language
Prof. Janakiraman Viraraghavan
Department of Electrical Engineering
Indian Institute of Technology, Madras

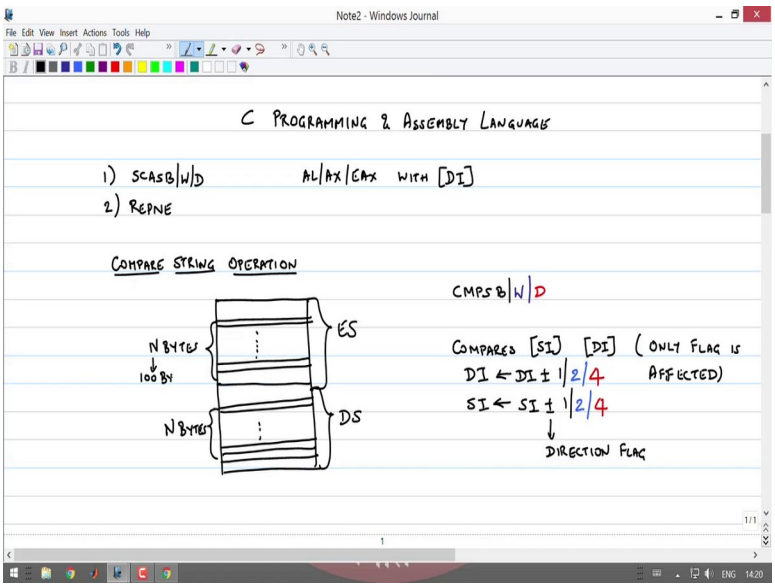
Lecture – 4B

(Refer Slide Time: 00:11)



Welcome back to this course on C Programming and Assembly language. We were in module 1.

(Refer Slide Time: 00:23)



In the last lecture, we concluded by discussing about a particular instruction called the SCAS byte right, we did SCASBword or Dword. And what this basically did was to compare either AL, AX, or EAX with the contents pointed to by DI and we also looked at a prefix to this instruction which is called R E P N E repeat as long as it is not equal. . And we will continue this lecture with some more such similar instructions.

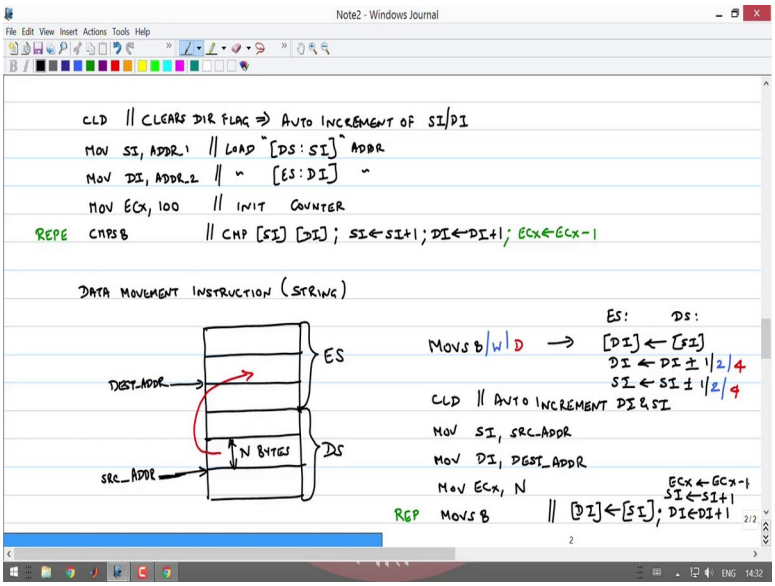
So, in this lecture, we will look at the following instruction which is a compare operation, compare operation on a string. So, let us assume that in my memory I have two different arrays, one sitting in my extra segment , this is my extra segment, and this is my data segment. And let us assume that there are a bunch of bytes here ok, N bytes, and out here as well. So, these are N bytes and so are these.

Now, my job is to write a program to find out if these two arrays have the same data which means that all the locations have to have exactly the same value. So, if I had to do this in regular assembly programming, then I would have to loop through each location, find out if each location matches or mismatches, and then increment the address and so on. It is possible to do this kind of a comparison in a single shot, because the x86 architecture supports string instructions. So, there is an instruction called CMPS byte, word or Dword So, CMPSB would compare a byte, CMPSW would compare a word, and CMPSD would compare a Dword.

So, what does this do? It basically compares the contents of a source index with the contents of destination index. And because it is a compare instruction, it is only going to affect the flag it will not affect either the SI location or the DI location , which means only flag is affected. So, just like the S C A S B the scan byte, after executing this instruction, the source index and the destination index will either be auto incremented or auto decremented.

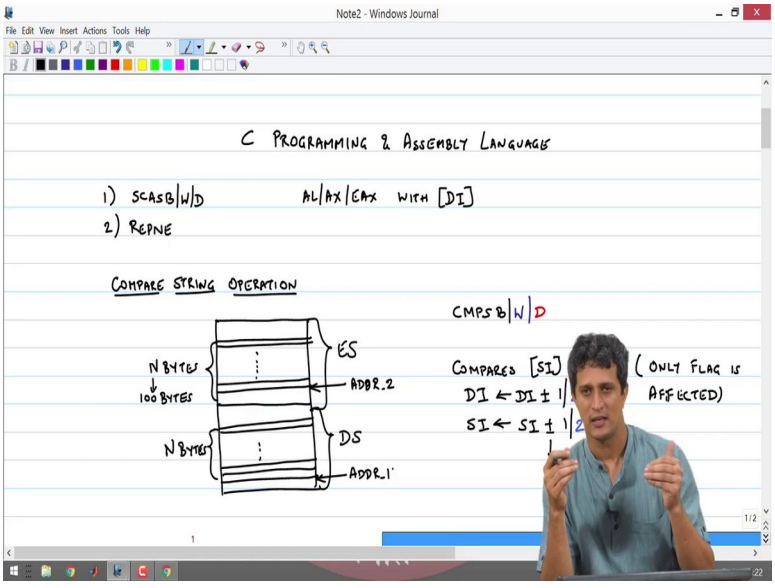
So, it is SI plus minus 1 2 or 4. And this plus or minus is determined by the direction flag . So, similarly DI will become DI plus minus 1/2/ 4 depending on whether we are accessing a word, a byte, a word, or a Dword in memory. So, let us now proceed and you know let us assume that you know in this example N is you know let us say this is about 100 bytes .

(Refer Slide Time: 05:39)



So, what I want to do is to now write a program where I can compare location my location and say if the entire array matches or stop at the first mismatch . So, just like the S C A S B, I am going to go ahead and first clear the direction flag . So, what does this do, it basically clears direction flag implies auto increment, auto increment of SI /DI .

(Refer Slide Time: 06:34)



So, then lets also assume in the previous page that the first location of both arrays is given by ADDR_2, and this is ADDR_ 1. So, starting at ADDR 1 I am now comparing 100 locations with another array starting at ADDR 2 and going through another 100

locations in the extra segment ok. So, obviously, I need to move these addresses into my source index and destination index appropriately MOV SI,ADDR_ 1, and MOV DI,ADDR_2. So, just like we have been mentioning that the segment address and the offset of the register SI or DI gives the full address. The source index is always controlled by the data segment register, this is the full address; and the destination index is controlled by the extra segment ok.

Now, let me MOV my count into the register which is 100. Note that this is in decimal not in hexadecimal now ok. And, I will add this instruction which is CMPSB right. This is basically you know load the address, this is load another address. So, this is INIT counter . So, when I do a CMPSB it is going to compare contents of SI with contents of DI and then auto increment SI+1and DI will be DI+1. Just like we had in the SCASB instruction, this does it only for one location. And if I want to repeat this till all the 100 location have been scanned, I need to add a prefix to it, and that prefix now is as long as the two locations are equal, you keep going .

So, you keep REPE right. So, you keep going either until as long as both the compare you know compare operation results in equal values or the ECX eventually goes down to 0. So now, because of my prefix ECX will become ECX minus 1 after each of these operations. So by the end, when we finish this instruction REPE, CMPSB, and then proceed to the next instruction it would have compared all the locations that are the same. And at the first mismatch, it would have stopped, or it would have or when you are done with all the 100 locations it means that all the locations match exactly .

So, if instead of a byte, if I wanted to compare it with a word, then I would have to do CMPSW in which case my SI and DI would auto increment by value of 2. And if I wanted to compare it with the D word, then I would do CMPSD and the SI and DI would auto increment by a value of 4 each time. So, similar to this there is an other instruction which is known as move instruction , data movement instruction , but string or an array . So, in this case, the example we are dealing with is you have a memory block which is sitting in my data segment data segment, this is my extra segment.

And I have N locations which are starting at ADDR_1. And or this time, I could call it a source address source address you know in the in my data segment, and let us assume that this is sum N bytes. And I have another location in my extra segment which is called

DEST_ADDR. And what I want to do is to copy these n bytes from this to this, which implies from my data segment I want to copy all the N bytes it could be the 100 bytes as an example or 256 bytes from my data segment SRC underscore address offset to extra segment destination underscore ADDR offset and 100 locations there .

So, how would you do this? It turns out that there is an instruction called MOVS byte and has with everything instruction moves word or moves Dword. And so let us go ahead and try to write the simple program for this. If I want to execute this, I would do CLD, which is basically AUTO increment DI and SI. Then I would move my source index with SRC_ADDR. Then I would move my destination index with destination_ADDR. And then of course, I have to load my count, so I would MOV ECX , N, whatever the value of N is . And then I will do MOVS byte ok.

So, what does this MOVS byte do it, it will simply copy to the destination index the contents of source index . And of course, this is going to be ES_:destination index, and this is data segment:source index . And because this is a move operation, and it is not an ALU operation, obviously, a flag will not be set. So, therefore, you cannot use any kind of flag condition to decide if you want to repeat or you want to stop this operation . So, remember that REPE would you know go on as long as they are equal REPNE would go on as long as they are not equal and so on.

But here because no flag is going to be set, I cannot know use any of those conditions. And therefore, I am simply going to put an unconditional prefix just rep which means repeat this MOVS byte operation until ECX has reached 0 . So, and yeah by the way the after doing the MOVS byte, the destination index will get plus minus 1/2/4 and so will the source index 4 ok. So, here when I do this REP MOVS byte it means that destination index will get contents of source index , and then DI will be DI+1, SI will also be SI+1 and ECX let me just ECX will be auto decrement.

So, in summary, we have studied three different string instructions SCASB, which is scan a string for a particular pattern which can be loaded in AL, AX or the EAX; CMPSB, or CMPSWD whatever can be used to compare the contents of SI and DI . And you can use a prefix of REPE to repeat this operation as long as they are equal. In the SCASB case, we repeated as long as they were not equal. And then the third operation is

a move data movement operation from the source index in the data segment to the destination index in the extra segment as long as my ECX is not 0.

(Refer Slide Time: 18:01)