

# Strings and Arrays

## Contents of Lecture

- ❖ Introduction
- ❖ Direction Flag
- ❖ Moving String

### References for this lecture

*Assembly Language for x86 Processors (7th Edition), Chapter 9, Strings and Arrays*

## Introduction:

- ❖ The x86 instruction set has five groups of instructions for processing arrays of bytes, words, and double words. Although they are called ***string primitives***, they are not limited to character arrays.
- ❖ In 32-bit mode, each instruction in following Table implicitly uses ESI, EDI, or both registers to address memory. References to the accumulator imply the use of AL, AX, or EAX, depending on the instruction data size.
- ❖ String primitives execute efficiently because they automatically repeat and increment array indexes.

### String Primitive Instructions.

Instruction	Description
MOVSb, MOVSw, MOVD	Move string data: Copy data from memory addressed by ESI to memory addressed by EDI.
CMPSb, CMPSw, CMPSD	Compare strings: Compare the contents of two memory locations addressed by ESI and EDI.
SCASb, SCASw, SCASD	Scan string: Compare the accumulator (AL, AX, or EAX) to the contents of memory addressed by EDI.
STOSb, STOSw, STOSD	Store string data: Store the accumulator contents into memory addressed by EDI.
LODSb, LODSw, LODSD	Load accumulator from string: Load memory addressed by ESI into the accumulator.

### **Direction Flag:**

- ❖ Direction flag is one of the Control Flags, which determines the direction in which it will be dealing with texts commands.
- ❖ DI, SI the registers are use when dealing with arrays.
- ❖ **There are two ways to deal with the string:**
  - ✓ Deal with string from the beginning, in this case make the registrar DI or SI refers to the first character in the string and therefore the deal is an increase of the contents of the recorders to indicate to the next character, in this case is put the number 0 in DF.
  - ✓ If we put the number 1 in the DF that mean we deal with the string from the end and decrease the contents of the recorders are indexing.
- ❖ The number zero is placed in direction flag using the command; **CLD** (Clear Direction flag)
- ❖ The number 1 is placed in DF using the command; **STD** (Set Direction flag)
- ❖ CLD and STD commands do not affect the other flags.

### **Moving String:**

- ❖ Have the following definition:
 

String1	DB	'Hello'
String2	DB	5 Dup ( ? )
- ❖ And want to make a copy of the first string in the second string, and this usually happens when merging two letters in the program.
- ❖ Will use command **MOVSB** which is without operands.
- ❖ Command is used to transfer the contents of memory at address DS: SI to memory at address ES: DI is not to change the contents of the source.
- ❖ After the transfer of the character command automatically increase the contents of the registered DI: SI If DF contains the number 0.

❖ **Example:**

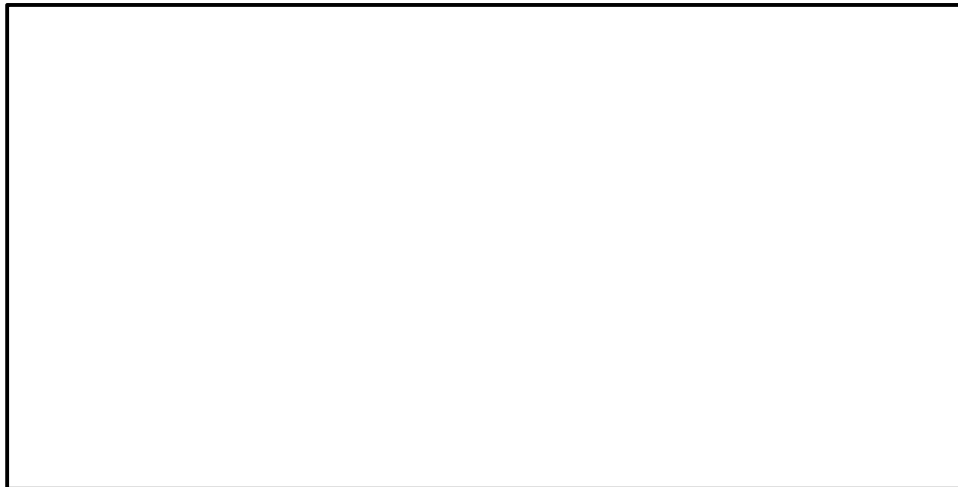
You can copy a string1 In the example on the a string2 the implementation of the following:



- ❖ MOVSB deals with one byte only.
- ❖ To move a number of letters is placed required number of MOVSB or use loops.
- ❖ To deal with loop the number of repeat execute the command MOVSB registered in CX and then the command is executed

**REP MOVSB**

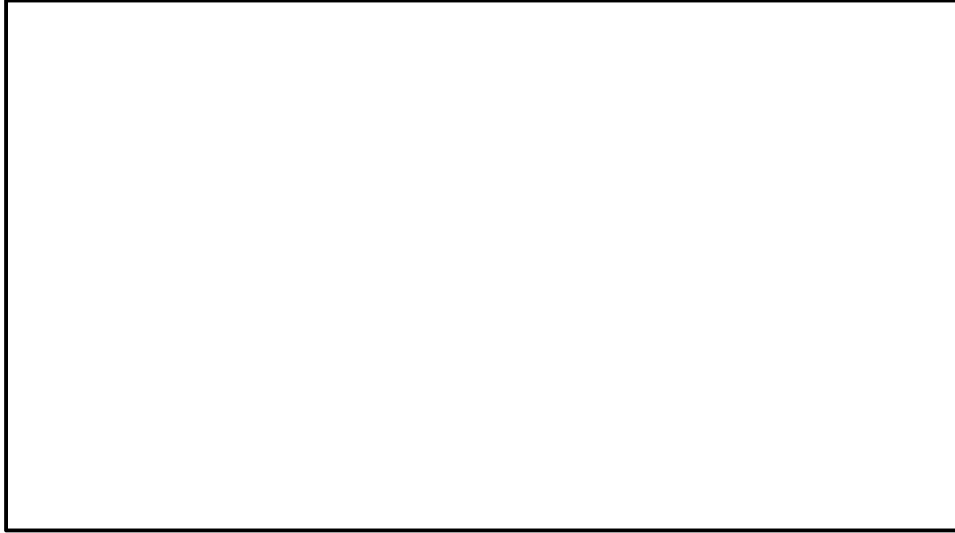
- ❖ Thus, the command is executed MOVSB N number of times. And decreasing the contents of the CX after each time you execute the command MOVSB until the value of CX = 0. And thus can be written for the following example:



❖ **Example:**

Write part of a program that copies String1 to String 2, but in reverse.

**Code:**



- ❖ Make register SI refers to the end of the first string (the last character in it) and DI refers to the beginning of the second string and transform the character.
- ❖ Then decrease SI (put the number 1 in the direction flag) and do not forget to increase the value of DI by 2 after every time as it will decrease the contents by 1 after the execution of the order MOVS and we want to increase it by 1.