Laboratory Exercise #3-6
String Operations

Target Course Outcome:
CO2: Designs a microprocessor-based firmware integrating a microprocessor with supporting peripherals and devices.

Objectives:
To use the different arithmetic and logic operations in creating assembly language programs.
To study and implement the concept of arrays in Assembly Language Programming.

Tools Required:
Emu8086 Emulator

Part 1:

String Operations/Instructions

MOVS – MOVS Destination String Name, Source String Name
MOVSB – MOVSB Destination String Name, Source String Name
MOVSW – MOVSW Destination String Name, Source String Name

This instruction copies a byte or a word from location in the data segment to a location in the extra segment. The offset of the source in the data segment must be in the SI register. The offset of the destination in the extra segment must be in the DI register. For multiple-byte or multiple-word moves, the number of elements to be moved is put in the CX register so that it can function as a counter. After the byte or a word is moved, SI and DI are automatically adjusted to point to the next source element and the next destination element. If DF is 0, then SI and DI will be incremented by 1 after a byte move andby 2 after a word move. If DF is 1, then SI and DI will be decremented by 1 after a byte move and by 2 after a word move. MOVS does not affect any flag.

When using the MOVS instruction, you must in some way tell the assembler whether you want to move a string as bytes or as word. There are two ways to do this. The first way is to indicate the name of the source and destination strings in the instruction, as, for example. MOVS DEST, SRC. The assembler will code the instruction for a byte / word move if they were declared with a DB / DW. The second way is to add a “B” or a “W” to the MOVS mnemonic. MOVSB says move a string as bytes; MOVSW says move a string as words.

Example:

```
MOV SI, OFFSET SOURCE          ; Load offset of start of source string in DS into SI
MOV DI, OFFSET DESTINATION ;Load offset of start of destination string in ES into DI
CLD                            ;Clear DF to auto increment SI and DI after moving
MOV CX, 04H                    ; Load length of string into CX as counter
REP MOVSB                      ;Move string byte until CX = 0
LODS / LODSB / LODSW           ;LOAD STRING BYTE INTO AL OR STRING WORD INTO AX
```

This instruction copies a byte from a string location pointed to by SI to AL, or a word from a string location pointed to by SI to AX. If DF is 0, SI will be automatically incremented (by 1 for a byte string, and 2 for a word string) to point to the next element of the string. If DF is 1, SI will be automatically decremented (by 1 for a byte string, and 2 for a word string) to point to the previous element of the string. LODS does not affect any flag.

Example:

```
CLD                     ; Clear direction flag so that SI is auto-incremented
MOV SI, OFFSET SOURCE   ; Point SI to start of string
LODS SOURCE             ; Copy a byte or a word from string to AL or AX
```

Note:
The assembler uses the name of the string to determine whether the string is of type bye or type word. Instead of using the string name to do this, you can use the mnemonic LODSB to tell the assembler that the string is type byte or the mnemonic LODSW to tell the assembler that the string is of type word.

STOS / STOSB / STOSW (STORE STRING BYTE OR STRING WORD)
This instruction copies a byte from AL or a word from AX to a memory location in the extra segment pointed to by DI. In effect, it replaces a string element with a byte from AL or a word from AX. After the copy, DI is automatically incremented or decremented to point to next or previous element of the string. If DF is cleared, then DI will automatically be incremented by 1 for a byte string and by 2 for a wordstring. If DI is set, DI will be automatically decremented by 1 for a byte string and by 2 for a word string.STOS does not affect any flag.

Example:

```
MOV DI, OFFSET TARGET
STOS TARGET
```

Note: The assembler uses the string name to determine whether the string is of type byte or type word. If it is a byte string, then string byte is replaced with content of AL. If it is a word string, then string word is replaced with content of AX.

STOSB
"B" added to STOSB mnemonic tells assembler to replace byte in string with byte from AL. STOSW would tell assembler directly to replace a word in the string with a word from AX.

CMPS / CMPSB / CMPSW (COMPARE STRING BYTES OR STRING WORDS)
This instruction can be used to compare a byte / word in one string with a byte / word in another string. SI is used to hold the offset of the byte or word in the source string, and DI is used to hold the offset of the byte or word in the destination string.

The AF, CF, OF, PF, SF, and ZF flags are affected by the comparison, but the two operands are not affected. After the comparison, SI and DI will automatically be incremented or decremented to point to the next or previous element in the two strings. If DF is set, then SI and DI will automatically be decremented by 1 for a byte string and by 2 for a word string. If DF is reset, then SI and DI will automatically be incremented by 1 for byte strings and by 2 for word strings. The string pointed to by SI must be in the data segment. The string pointed to by DI must be in the extra segment.

The CMPS instruction can be used with a REPE or REPNE prefix to compare all the elements of a string.

Example:

```
MOV SI, OFFSET FIRST        ; Point SI to source string
MOV DI, OFFSET SECOND       ; Point DI to destination string
CLD                         ; DF cleared, SI and DI will auto-increment after compare
MOV CX, 100                 ; Put number of string elements in CX
REPE CMPSB                  ; Repeat the comparison of string bytes until end of string
                              or until compared bytes are not equal
```

CX functions as a counter, which the REPE prefix will cause CX to be decremented after each compare. The B attached to CMPS tells the assembler that the strings are of type byte. If you want to tell the assembler that strings are of type word, write the instruction as CMPSW. The REPE CMPSW instruction will cause the pointers in SI and DI to be incremented by 2 after each compare, if the direction flag is set.

SCAS / SCASB / SCASW (SCAN A STRING BYTE OR A STRING WORD)
SCAS compares a byte in AL or a word in AX with a byte or a word in ES pointed to by DI. Therefore, the string to be scanned must be in the extra segment, and DI must contain the offset of the byte or the word to be compared. If DF is cleared, then DI will be incremented by 1 for byte strings and by 2 for word strings. If DF is set, then DI will be decremented by 1 for byte strings and by 2 for word strings. SCAS affects AF, CF, OF, PF, SF, and ZF, but it does not change either the operand in AL (AX) or the operand in the string.
The following program segment scans a text string of 80 characters for a carriage return, 0DH, and puts the offset of string into DI.

Example:

```
MOV DI, OFFSET STRING
MOV AL, 0DH                 ; Byte to be scanned for into AL
MOV CX, 80                  ; CX used as element counter
CLD                         ;Clear DF, so that DI auto increments
REPNE SCAS STRING Compare byte in string with byte in AL
```

REP / REPE / REPZ / REPNE / REPNZ (PREFIX)
(REPEAT STRING INSTRUCTION UNTIL SPECIFIED CONDITIONS EXIST)
REP is a prefix, which is written before one of the string instructions. It will cause the CX register to be decremented and the string instruction to be repeated until CX = 0. The instruction REP MOVSB, for example, will continue to copy string bytes until the number of bytes loaded into CX has been copied. REPE and REPZ are two mnemonics for the same prefix. They stand for *repeat if equal* and *repeat if zero*, respectively. They are often used with the Compare String instruction or with the Scan String instruction. They will cause the string instruction to be repeated if the compared bytes or words are equal (ZF = 1) and CX is not yet counted down to zero. In other words, there are two conditions thatwill stop the repetition: CX = 0 or string bytes or words not equal.

REPE CMPSB Compare string bytes until end of string or until string bytes not equal.
REPNE and REPNZ are also two mnemonics for the same prefix. They stand for *repeat if not equal* and *repeat if not zero*, respectively. They are often used with the Compare String instruction or with the Scan String instruction. They will cause the string instruction to be repeated if the compared bytes or words are not equal (ZF = 0) and CX is not yet counted down to zero.

REPNE SCASW Scan a string of word until a word in the string matches the word in AX or until all the string has been scanned. The string instruction used with the prefix determines which flags are affected.

Part II

Write a program using any of the string instructions to manipulate the input string.

Activity #1

Convert lowercase characters to uppercase.

Procedure:

1. Start Emu8086 by selecting its icon from the start menu, or by running Emu8086.exe
2. Create a new file and label it as Exp3-6.asm
3. In the editor window, complete the missing codes (marked as ??) such that the entire contents of the STRING variable are reversed.

```
ORG 100H

 ??? DX, STRING
 MOV AH, ???
 INT ???

??? REVERSE

 LEA ???,???
 ??? ???, 09H
 ??? 21H

RET

REVERSE:
 MOV CX,???

LOOP1:
 XOR ???,???
 MOV AL, ???
 CMP AL, ???
 ??? LABEL1
 PUSH ???
 INC SI
 LOOP ???

LABEL1:
 MOV SI, ??? STRING
 MOV CX, ???

LOOP2:
 POP DX
 MOV ???, DL
 INC ???
 ??? LOOP2

EXIT:
  MOV ???,???
RET

STRING ??? 'THIS IS A SAMPLE STRING$'
```

4. Press the Compile and Emulate button or press F5 key on your keyboard. The emulator window should open with this program loaded, click the Single Step button to test your inputted code if the string reversal works.

5. Input your answers for this activity in Canvas.

Copyright Information

Author : Rosana J. Ferolin (rjferolin@usc.edu.ph)
Date of release : August 7, 2020
Version : 1.0

Change log:

Date	Version	Author	Changes
Aug. 7, 2020	1.0	Rosana J. Ferolin	Initial Draft
Sep. 3, 2025	2.0	Marlowe Edgar C. Burce	Corrected typos and code errors, revised code, activities and procedures