ASSEMBLY - STRINGS

We have already used variable length strings in our previous examples. The variable length strings can have as many characters as required. Generally, we specify the length of the string by either of the two ways −

- Explicitly storing string length
- Using a sentinel character

We can store the string length explicitly by using the $ location counter symbol that represents the current value of the location counter. In the following example −

```
msg  db  'Hello, world!',0xa ;our dear string
len  equ  $ - msg             ;length of our dear string
```

$ points to the byte after the last character of the string variable *msg*. Therefore, ***$-msg*** gives the length of the string. We can also write

```
msg db 'Hello, world!',0xa ;our dear string
len equ 13                 ;length of our dear string
```

Alternatively, you can store strings with a trailing sentinel character to delimit a string instead of storing the string length explicitly. The sentinel character should be a special character that does not appear within a string.

For example −

```
message DB 'I am loving it!', 0
```

String Instructions

Each string instruction may require a source operand, a destination operand or both. For 32-bit segments, string instructions use ESI and EDI registers to point to the source and destination operands, respectively.

For 16-bit segments, however, the SI and the DI registers are used to point to the source and destination, respectively.

There are five basic instructions for processing strings. They are −

- **MOVS** − This instruction moves 1 Byte, Word or Doubleword of data from memory location to another.
- **LODS** − This instruction loads from memory. If the operand is of one byte, it is loaded into the AL register, if the operand is one word, it is loaded into the AX register and a doubleword is loaded into the EAX register.
- **STOS** − This instruction stores data from register $AL, AX, or EAX$ to memory.
- **CMPS** − This instruction compares two data items in memory. Data could be of a byte size, word or doubleword.
- **SCAS** − This instruction compares the contents of a register $AL, AX or EAX$ with the contents of an item in memory.

Each of the above instruction has a byte, word, and doubleword version, and string instructions can be repeated by using a repetition prefix.

These instructions use the ES:DI and DS:SI pair of registers, where DI and SI registers contain valid offset addresses that refers to bytes stored in memory. SI is normally associated with DS *datasegment* and DI is always associated with ES *extrasegment*.

The DS:SI $orESI$ and ES:DI $orEDI$ registers point to the source and destination operands, respectively. The source operand is assumed to be at DS:SI $orESI$ and the destination operand at ES:DI $orEDI$ in memory.

For 16-bit addresses, the SI and DI registers are used, and for 32-bit addresses, the ESI and EDI registers are used.

The following table provides various versions of string instructions and the assumed space of the operands.

Basic Instruction	**Operands at**	**Byte Operation**	**Word Operation**	**Double word Operation**
MOVS	ES:DI, DS:EI	MOVSB	MOVSW	MOVSD
LODS	AX, DS:SI	LODSB	LODSW	LODSD
STOS	ES:DI, AX	STOSB	STOSW	STOSD
CMPS	DS:SI, ES: DI	CMPSB	CMPSW	CMPSD
SCAS	ES:DI, AX	SCASB	SCASW	SCASD

Repetition Prefixes

The REP prefix, when set before a string instruction, for example - REP MOVSB, causes repetition of the instruction based on a counter placed at the CX register. REP executes the instruction, decreases CX by 1, and checks whether CX is zero. It repeats the instruction processing until CX is zero.

The Direction Flag DF determines the direction of the operation.

- Use CLD $ClearDirectionFlag, DF = 0$ to make the operation left to right.
- Use STD $SetDirectionFlag, DF = 1$ to make the operation right to left.

The REP prefix also has the following variations:

- REP: It is the unconditional repeat. It repeats the operation until CX is zero.
- REPE or REPZ: It is conditional repeat. It repeats the operation while the zero flag indicates equal/zero. It stops when the ZF indicates not equal/zero or when CX is zero.
- REPNE or REPNZ: It is also conditional repeat. It repeats the operation while the zero flag indicates not equal/zero. It stops when the ZF indicates equal/zero or when CX is decremented to zero.