# The String Instructions

## Outline

- The  direction  flag
- Move  a  string
- Store  a  string
- Load   a  string
- Scan   a  string
- Compare a string
- General  form  of the  string  instructions

## The Direction Flag

- One of the control flags is the direction flag (DF).
- Its purpose is to determine the direction in which string operations will proceed.
- These operations are implemented by the two index registers SI and DI.
- Suppose, for example, that the following string has been declared STRING1 DB 'ABCDE'. And this string is stored in memory starting at offset 0200h.
- DF = 0, SI and DI proceed in the direction of increasing memory addresses from left to right across the string.
- If DF = 1, SI and DI proceed in the direction of decreasing memory addresses from right to left.

## CLD and STD

- To make DF = 0, use the CLD instruction.

```
CLD     ;clear direction    flag
```

- To make DF = 1, use the STD instruction.

```
STD     ; set direction    flag
```

- CLD and STD have no effect on the other flags.

## MOVSB Instruction

- Syntax:

```
MOVSB          ;move   string   byte
```

- This instruction copies the contents of the byte addressed by DS:SI, to the byte addressed by ES:DI.
- The contents of the source byte are unchanged.
- After the byte has been moved, both SI and DI are automatically incremented if DF = 0 and decremented if DF = 1.
- MOVSB permits a memory- memory operation.
- It also involves the ES register.
- MOVSB have no effect on the flags.

## MOVSW Instruction

- Syntax:

```
MOVSW           ;move   string   word
```

- This instruction copies the contents of the word addressed by DS:SI, to the word addressed by ES:DI.
- The contents of the source word are unchanged.
- After the word has been moved, both SI and DI are automatically incremented by 2 if DF=0 and decremented by 2 if DF= 1.
- MOVSW permits a memory- memory operation.
- It also involves the ES register.
- MOVSW have no effect on the flags.

## The REP Prefix

- The REP prefix causes MOVSB to be executed N times.
- MOVSB moves only a single byte from the source string to the destination string.
- To move the entire string, first initialize CX to the number of bytes N in the source string and execute

```
        REP    MOVSB
```

- After each MOVSB, CX is decremented until it becomes 0.

## STOSB Instruction

- Syntax:

```
STOSB        ; store    string    byte
```

- This instruction moves the contents of the AL register to the byte addressed by ES:DI.
- DI is incremented if DF =0 or decremented if DF = 1.
- STOSB has no effect on the flags.

## STOSW Instruction

- Syntax:

```
STOSW        ; store string word
```

- This instruction moves the contents of AX to the word at address ES:DI and updates DI by 2, according to the direction flag setting.
- STOSW has no effect on the flags.

## LODSB Instruction

- Syntax:

```
        LODSB            ;load  string byte
```

- This instruction moves the byte addressed by DS:SI into AL.
- SI is then incremented if DF = 0 or decremented if DF = 1.
- LODSB can be used to examine the characters of a string.
- LODSB has no effect on the flags.

## LODSW  Instruction

- Syntax:

```
        LODSW              ;load   string   word
```

- This instruction moves  the  word  addressed by  DS:SI  into AX.
- SI  is  incremented by  2  if DF  = 0  or decremented by  2  if DI =  1.
- LODSW has no effect  on the  flags.

## SCASB Instruction

- Syntax:

```
SCASB       ;scan   string   byte
```

- This instruction can be used to examine a string for a target byte.
- The target byte is contained in AL.
- SCASB subtracts the string byte pointed to by ES:DI from the contents of AL and uses the result to set the flags.
- The result is not stored.
- Afterward, DI is incremented if DF =0 or decremented if DF = 1.
- All the status flags are affected by SCASB.

## SCASW Instruction

- Syntax:

```
SCASW      ;scan   string   word
```

- This instruction can be used to examine a string for a target word.
- The target word is in AX.
- SCASW subtracts the word addressed by ES:DI from AX and sets the flags.
- DI is incremented by 2 if DF = 0 or decremented by 2 if DF = 1.
- All the status flags are affected by SCASW.

## REPNZ and REPNE Instruction

- If CX is initialized to the number of bytes in the string, these instructions will repeatedly subtract each string byte from AL, update DI and decrement CX until there is a zero result (the target is found) or CX = 0 (the string ends).
- REPNZ (repeat while not zero) generates the same machine code as REPNE.

## CMPSB Instruction

- Syntax:

```
        CMPSB              ;compare    string   byte
```

- This instruction subtracts the byte with address ES:DI from the byte with address DS:SI and sets the flags.
- The result is not stored.
- Afterward, both SI and DI are incremented if DF = 0 or decremented if DF = 1.
- CMPSB may be used to compare two character strings to see which comes first alphabetically or if they are identical or if one string is a substring of the other.
- All the status flags are affected by CMPSB.

## CMPSW Instruction

- Syntax:

```
CMPSW                    ;compare    string    word
```

- This instruction subtracts the word with address ES:DI from the word whose address is DS:SI and sets the flags.
- If DF= 0, SI and DI are incremented by 2 and if DF= 1, they are decremented by 2.
- CMPSW is useful in comparing word arrays of numbers.
- All the status flags are affected by CMPSW.

## REPE and REPZ Instruction

- String comparison may be done by attaching the prefix REPE (repeat while equal) or REPZ (repeat while zero) to CMPSB or CMPSW.
- CX is initialized to the number of bytes in the shorter string, then

```
REPE    CMPSB  ;compare    string    bytes    while    equal
REPE    CMPSW  ;compare    string    words    while    equal
```

repeatedly executes CMPSB or CMPSW and decrements CX

until

1. There is a mismatch between corresponding string bytes or words
2. Or CX =0

- The flags are set according to the result of the last comparison.

## General Form of the String Instructions

| Explicit Instruction | Implicit Instruction |
|---|---|
| MOVS destination_string, source_string | MOVSB |
| CMPS destination_string, source_string | CMPSB |
| STOS destination_string | STOS STRING2 |
| LODS source_string | LODS STRING1 |
| SCAS destination_string | SCAS STRING2 |

## General Form of the String Instructions

- When the assembler encounters one of these general forms, it checks to see
  1. The source string is in the segment addressed by DS and the destination string is in the segment addressed by ES
  2. In the case of MOVS and CMPS, if the strings are of the same type; that is, both byte strings or word strings.
- An advantage of using the general form of string instructions is that because the operands appear as part of the code, program documentation is improved.
- A disadvantage is that only by checking the data definitions is it possible to tell whether a general string instruction is a byte form or a word form.
- In fact, the operands specified in a general string instruction may not be the actual operands used when the instruction is executed.