Experiment 9

9 String Handling Instructions

Introduction

In this experiment you will deal with string handling instructions, such as moving a block of data from one memory location to another, and comparing two strings.

Objectives:

1. String Handling Instructions.
2. Introduction to the video display.

String handling instructions allow the programmer to manipulate large blocks of data with relative ease. They can be divided in two classes:

- Data Transfer Instructions
- Data Manipulation: such as testing, …

9.1 Registers used with the String Handling Instructions

A number of registers and a flag are needed for the use of string handling instructions: The operands for the string instructions include

- SI (source index) and DI (destination index) register,
- CX (count) register,
- AL/AX register,
- direction flag (DF) in the flag register
- ESI, EDI and EAX registers are used with the Pentium processor

9.1.1 Registers

String handling instructions use the direction flag, CX, SI and DI registers. The CX register is used to indicate the number of elements in the string. SI serves as an offset for the source string, and DI serves as an offset for the destination string.

AL/AX register serves as a temporary register to hold the value of the byte/word to be tested or checked, in case of data manipulation instructions.

9.1.2 Direction Flag

The Direction Flag (DF) selects auto-increment or auto-decrement mode during string operations. for the DI and SI registers. The contents of SI and/or DI is incremented or

decremented by 1 whenever a string instruction transfers a byte. If a word is transferred, the contents of SI and/or DI is incremented or decremented by 2 (Table 9.1)

Format	Operation	Mode	Effect
CLD	Clear DF; (DF) ← 0	Auto-Increment	SI ← SI + 1; DI ← DI +1
STD	Set DF; (DF) ← 1	Auto-Decrement	SI ← SI - 1; DI ← DI -1

Table 9. 1: Direction Flag

9.2 String Transfer Instructions

There is a number of such instructions. They can be further divided into two subclasses: Block instructions (MOVS) and Single element instructions (LODS, STOS)

9.2.1 The MOVS Instruction

The MOVS instruction transfers data from one memory location to another. This is the only memory-to-memory transfer allowed in the Intel family of Microprocessors.

9.2.2 The LODS Instruction:

LODS loads AL or AX with data stored from the DS:SI location. The LODSB loads a byte into AL, and the LODSW loads a word into AX.

9.2.3 The STOS Instruction:

The STOS instruction stores AL or AX in the ES:DI location. Program 9.1 gives an example on the use of STOS instruction to clear the video memory.

Mnemonics	Meaning	Format	Operation As per Direction Flag	Flags affected
LODS	Load string	**LODSB** **LODSW**	(AL or AX) ← [DS:SI] (SI) ← (SI) ± 1 or 2	None
STOS	Store string	**STOSB** **STOSW**	[ES:SI]← (AL or AX)) (DI) ← (DI) ± 1 or 2	None
MOVS	Move string	**MOVSB** **MOVSW**	[ES:SI]← [DS:SI] (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None

Note: B stands for Byte and W for Word.

Table 9. 2: Data Transfer String Instructions

Example 1

Below is a code that moves a block of data using both normal data transfer instructions and handling instructions. The same example is repeated later but with the use of the REP prefix.

1 – Code without String instructions

```
        MOV AX, @DATA
        MOV DS, AX
        LEA SI, BLK1            ; Source address for block1
        LEA DI, BLK2            ; Destination address for block2
        MOV CX, N               ; N = number of bytes to move

NEXT:   MOV  AL, Byte Ptr[SI]   ; Load AL register
        MOV  Byte Ptr[DI], AL   ; Store back in memory
        INC SI
        INC DI
        LOOP NEXT
```

1 – Code using String instructions

```
        MOV AX, @DATA
        MOV DS, AX
        MOV ES, AX              ; Make ES = DS
        LEA SI, BLK1            ; Source address for block1
        LEA DI, BLK2            ; Destination address for block2
        MOV CX, N               ; N = number of bytes to move
        CLD                     ; Set Auto-Increment mode

NEXT:   MOVSB                   ; Move one byte at a time
        LOOP NEXT
```

Example 2

The following code can convert a string from lower case letter to upper case.

```
        MOV AX, @DATA
        MOV DS, AX

        LEA SI, BLK1            ; Source address for block1
        LEA DI, BLK2            ; Destination address for block2
        MOV CX, N               ; N = number of bytes to move
        CLD                     ; Set Auto-Increment mode

NEXT:   LODSB                   ; Move one byte at a time
        SUB AL, 20H
        STOSB
        LOOP NEXT
```

9.3 String Compare Instructions

These instructions are used to compare two strings or look for a specific value in a string.

9.3.1 Searching for a Specific Value

The SCAS instruction is used to compare the content of AL/AX register with a byte (SCASB) or word (SCASW) block of memory (Table 9. 3). Its main purpose is to find a specific byte/word in a string.

9.3.2 Strings Comparing

The CMPS instruction compares two blocks of memory data as bytes (CMPSB), or words (CMPSW). The contents of DS:SI memory indicated by SI are compared with the contents of the ES:DI. The CMPS instruction increment/decrements both SI and DI if the direction flag (DF) is zero, or both of them if DF is set to one.

The CMPS instruction is normally used with either the REPE or REPNE prefixes. Alternates to these prefixes are REPZ (*repeat while zero*) and REPNZ (*repeat while not zero*), though REPE and REPNE are more common (Table 9. 4, 9.4).

Mnemonics	Meaning	Format	Operation	Flags affected
CMPS	Compare strings	**CMPSB** **CMPSW**	Set flags as per: [DS:SI] – [ES:DI] (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	CF,PF,AF, ZF,SF,OF
SCAS	Scan string	**SCASB** **SCASW**	Set flags as per: (AL or AX) – [ES:DI] (DI) ← (DI) ± 1 or 2	CF,PF,AF, ZF,SF,OF

Note: B stands for Byte and W for Word.

Table 9. 3: String Compare Instructions

9.4 Repeat Prefixes

A number of repeat prefixes are used for repetitions with the string instructions.

9.4.1 REP Prefix

The REP prefix can be added to any data transfer or compare instruction except the LODS instruction. The REP prefix causes the CX register to decrement by 1 each time the string instruction executes. If CX reaches 0, the instruction terminates and

the program continues with the next sequential instruction. The following example illustrates the use of a move string with the REP prefix:

```
MOV AX, @DATA
MOV DS, AX
MOV ES, AX          ; Make ES = DS
LEA SI, BLK1        ; Source address for block1
LEA DI, BLK2        ; Destination address for block2
MOV CX, N           ; N = number of bytes to move
CLD                 ; Set Auto-Increment mode
REP MOVSB           ; Move one byte at a time
```

9.4.2 Conditional Prefixes

These are conditional prefixes. Repetition is conditional on the CX register and the zero flag (ZF). CX contains the maximum number of repetitions, while the actual number of repetitions is determined by the ZF flag.

Prefix	Used with	Meaning
REP	**MOVS** **STOS**	Repeat while not end of string CX ≠ 0
REPE **REPZ**	**CMPS** **SCAS**	Repeat while not end of string and strings are equal CX ≠ 0 and ZF = 1
REPNE **REPNZ**	**CMPS** **SCAS**	Repeat while not end of string And strings are not equal CX ≠ 0 and ZF = 0

Table 9. 4: Repeat Prefixes

Example

Find a character in a string and its position.

```
                MOV AX, @DATA
                MOV DS, AX

                MOV ES, AX          ; Make ES = DS
                MOV  AL, Char       ; Char to find
                LEA DI, BLK2        ; Destination address for block2
                MOV CX, N           ; N = number of bytes
                CLD                 ; Set Auto-Increment mode
                REPNE SCASB         ; Search for char
                JNZ not_found
                DEC DI              ; to let DI pointing to the position
not_found:      …
                …
```

Examples on the use of the SCAS and CMPS instructions

Example

The following example shows how to search a memory section of 100 bytes in length and starting at location BLOCK. The program searches if any location contains the value 45H.

```
MOV DI, OFFSET BLOCK   ; address data
CLD                    ; set auto-increment
MOV CX, 100            ; load counter
MOV AL, 45H            ; AL = 45H
REPNE SCASB            ; search
```

Example

This example illustrates a short procedure that compares two sections of memory searching for a match. The CMPSB instruction is prefixed with a REPE. This causes the search to continue as long as an equal condition exists. When the CX register becomes 0, or an unequal condition exists, the CMPSB instruction stops execution. After the CMPSB instruction ends, the CX register is zero or the flags indicate an equal condition when the two strings match. If CX is not zero or the flags indicate a not-equal condition, the strings do not match.

```
MATCH     PROC FAR
          LEA SI, LINE
          LEA DI, TABLE
          CLD              ; set auto-increment
          MOV CX, N        ; load counter
          REPE CMPSB
          RET
MATCH ENDP
```

9.5 Pre Lab Work

9.5.1 Pre-Lab Work

1. Read the manual and understand how the string instructions work.
2. Write programs 9.1, 9.2 and 9.3 and check their functionality.
3. Bring your work to the lab.

9.5.2 Lab Work

1- Show programs 9.1, 9.2 and 9.3 to your lab instructor.
2- Clear the screen using program 9.2 and write the word BUG somewhere on the display. Run program 9.3 and check that it really detects the word BUG on the screen. Clear the screen with program 9.2 and check again with program 9.3.
3- Modify program 9.3 so that it looks for the word MOV on the display, and counts the number of times it occurs. Call it program 9.4.
4- Edit one of your assembly language programs on the screen using the following:

TYPE program.asm

5- Check with program 9.4, how many times you have the word MOV on the screen.

Lab Assignment:

Rewrite the program that reads a password without echo from the keyboard in a more structured way, using Macros and Procedures. To check for password validity, use the string handling instructions CMPSB or SCASB.

TITLE "Program 9.1"

```
;This program clears the videotext display
.MODEL TINY
.CODE
.STARTUP

      CLD                     ;select increment mode
      MOV     AX,0B800H       ;address segment B800H
      MOV     ES,AX
                              ;Video Text Memory = B800:0000
      MOV     DI,0000         ;address offset 0000
      MOV     CX,25*80        ;load count: 25 lines per 80 columns
      MOV     AX,0720H        ;load data AH= 07H = color: white text on black
                              ;background. AL = 20H = space

      REP     STOSW           ;clear the screen

.EXIT                         ;exit to DOS
END                           ;end of file
```

;----------

TITLE "Program 9.2"

```
;This program scrolls the display one line up

.MODEL TINY                   ;select TINY model
.CODE                         ;indicate start of CODE segment
.STARTUP                      ;indicate start of program

        CLD                   ;select increment
        MOV     AX,0B800H     ;load ES and DS with B800
        MOV     ES, AX
        MOV     DS, AX

        MOV     SI,160        ;address line 1: 160  = 2 * 80
        MOV     DI,0          ;address line 0
        MOV     CX,24*80      ;load count
        REP     MOVSW         ;scroll screen

        MOV     DI,24*80*2    ;clear bottom line
        MOV     CX,80
        MOV     AX,0720H
        REP     STOSW
.EXIT                         ;exit to DOS
END                           ;end of file
```

TITLE "Program 9.3"

```
;This program tests the video display for the word BUG
;if BUG appears anywhere on the display the program display Y
;if BUG does not appear, the program displays N
;
.MODEL SMALL                        ;select model SMALL
.DATA                               ;indicate start of DATA segment
        DATA1       DB      'BUG'   ;define BUG
.CODE                               ;indicate start of CODE segment
.STARTUP                            ;indicate start of program
        MOV  AX,0B800H              ;address segment B800 with ES
        MOV  ES,AX
        MOV  CX,25*80               ;set count
        CLD                         ;select increment
        MOV  DI,0                   ;address first display position
L1:
        MOV  SI,OFFSET DATA1        ;address BUG
        PUSH DI                     ;save display address
        CMPSB                       ;test for B
        JNE  L2                     ;if display is not B
        INC  DI                     ;address next display position
        CMPSB                       ;test for U
        JNE  L2                     ;if display is not U
        INC  DI                     ;address next display position
        CMPSB                       ;test for G
        MOV  DL,'Y'                 ;load Y for possible BUG
        JE   L3                     ;if BUG is found
L2:
        POP  DI                     ;restore display address
        ADD  DI,2                   ;point to next display position
        LOOP L1                     ;repeat until entire screen is tested
        PUSH DI                     ;save display address
        MOV  DL,'N'                 ;indicate N if BUG not found
L3:
        POP  DI                     ;clear stack
        MOV  AH,2                   ;display DL function
        INT  21H                    ;display ASCII from DL
.EXIT                               ;exit to DOS
END                                 ;end of file
```