**Lab 04: String Operations**

**OBJECTIVE**

To learn to display, input a string at runtime, copy string and reverse a string.

**String commands in Assembly**

* **Int 21h** has **function # 9** to ‘display strings’. Example: **mov** ah,9; **int** 21h
* The string must ‘end’ with a **$** character e.g. ‘Assembly$’
* **DX** has the ‘offset-address’ of string.
* The **int 21h,** function # 9, expects the ‘offset-address’ of string to be in the DX register. To get it we use **Lea** command.
* **Lea** (Load effective address) command, puts the source offset-address contained in **DX** into destination.
* **Lea** destination, source command adds ‘index-address + base address’ for complete address /offset of string.
* The hexadecimal bytes **0dh** (carriage return) and **0ah** (new line), are called **end-of-line.**

**Example-1:** Using pointer to access an array.

.model small

.data

**arrayB** db 20h, 30h

.code

Main proc

Mov ax,@data

Mov ds,ax

Lea si, **arrayB**

Mov al,[si]

Inc si

Mov al,[si]

Main endp

End main

**Example-2:** Using pointer to access an array. Different data sizes.

.model small

.data

**arrayA** dw 1000h

db ‘A’

.code

Main proc

Mov ax,@data

Mov ds,ax

Lea si,**arrayA**

Mov ax,[si]

Add si,2

Mov bl,[si] ; A = 41h

Main endp

End main

**1. String Display:** To display a string at runtime.

.model small

.stack 100h

.data

**msg** db 'I am a student$'; **Format:** label **db** ‘values$’ (db = define byte)

.code

main proc

mov ax,@data ;to display string

mov ds,ax ;2 lines

lea si,**msg**

mov cx,14 ;14 characters to display

l1:

mov ah,2

mov dl,[si]

int 21h

inc si

loop l1 ;to display 14 characters via loop

main endp

end main

**Code-02:**

.model small

.stack 100h

.data ;data segment (DS)

**message** db ‘Hello World$’

.code

Main proc ;next two lines initialize data segment to starting address

**Mov ax,@data** ;these two lines are used to access variables under DS

**Mov ds,ax** ;bcoz we can’t move a ‘constant’ directly to DS

;two commands translate the name @Data into address & move to DS

Mov ah,9

Mov dx,offset **message** ;offset value added to base address for string

Int 21h

Mov ah,4ch ;terminate program and return control to DOS

Int 21h

Main endp

End main

**2. String Input:** To input a string at prompt, and display a string. If enter key pressed (0dh), then terminate program execution. (see **Flowchart** next)

.model small

.stack 100h

.data

**a** db '?' ;’?’symbol for variables that are not initialized

.code

main proc

mov ax,@data ;these two lines initialize data segment

mov ds,ax

lea si,**a** ; string 'a' stored in index location for SI pointer

l1:

mov ah,1

int 21h

cmp al,0dh ; compare with enter key

je l2 ; conditional jump to display stored string

mov [si],al ;[square brackets] return/store value at memory address.

inc si

jmp l1 ;unconditional jump for new character input

l2:

inc si

mov [si],'$' ; last character to add in string

mov ah,2

mov dl,0dh ;generate new line

int 21h

mov dl,0ah

int 21h

mov ah,9 ; display the string

mov dx,offset **a** ;function 9 expects offset address of string to be in DX

int 21h

mov ah,4ch ;return control to DOS

int 21h

main endp

end main

**Flowchart for ‘String Input’:**

**Input**

**Save**

**Compare**

**False**

**True**

**Display**

**3. String Reversal:** To reverse a string and display it in reverse order.

(See **Flowchart**)

.model small

.stack 100h

.data

**array** db 'abcde$'

.code

main proc

mov ax,@data ;these two lines initialize data segment

mov ds,ax

lea si,**array** ;to access each character in array

mov cx,0 ;initialize counter with zero

label1:

mov al,[si] ;square brackets return value and not address

cmp al,'$' ;if $ is not found then jump to label2

jne label2

jmp rev ;if $ found then jump to rev

label2: ;run this loop until $ sign achieved

inc cx ;increment loop value for each character

inc si ;to get next character

jmp label1 ;continue to run the loop till last character $

rev: ;$ sign has been found

dec si ;so as not to print last character of $

rev1: ;label for displaying in reverse

mov ah,2

mov dl,[si]

int 21h

dec si ;we are going backwards

loop rev1 ;runs the loop cx times

mov ah,4ch ;return control to DOS

int 21h

main endp

end main

**Flowchart for ‘String Reversal’:**

**False**

**True**

**Display**

**Rev**

**Cmp ‘$’**

**4. String Copy:** To copy a source string to destination string. (**Flowchart**)

Here we have used **Si** (Source) and **Di** (Destination) due to two strings.

**DS** has ‘base address’ of ‘string1’, so: **DS+Si** = source location

**ES** has ‘base address’ of ‘string 2’, so **ES+Di** = destination location

**Movsb** does two functions: **Si++** and **Di++** ,it copies **+** inc characters

**Cld** ; Clear Direction Flag, if DF = 0, pointers increment.

.model small

.stack 100h

.data

**st1** db 'hello$' ;source string to be copied

**st2** db '?' ;destination of copy

.code

main proc

mov ax,@data

mov ds,ax

mov es,ax

lea si,**st1** ;these two lines copy 'si' to 'di' till $ sign

lea di,**st2**

cld ;clear direction flag so that string pointers auto increment

mov cx,6 ;6 character string

**rep** movsb ;move string byte using 'repeat' command

mov ah,9 ;string display

mov dx,offset **st2**

int 21h

mov ah,4ch

int 21h

main endp

end main

**The End**

Lab-4 Supporting Definitions

1. **Segment Registers:** Segments are specific areas defined in a program for containing data, code and stack.
2. Code Segment (CS): It contains all the instructions to be executed. A 16-bit code segment register or CS register stores the starting address of the code segment.
3. Data Segment (DS): It contains data, constants and work areas. A 16-bit data segment register or DS register stores starting address of the data segment.
4. Stack Segment (SS): It contains data and return addresses of procedures or subroutines. The stack segment register stores the starting addresses of the stack.
5. Extra Segment (ES): ES provides additional segments for storing data.
6. **Index Registers:** The 32-bit index registers, ESI and EDI and their 16-bit right most portions. SI and DI are used for indexed addressing and sometimes used in addition and subtraction.
7. Source Index (SI): It is used as source index for string operations.
8. Destination Index (DI): It is used as destination index for string operations.
9. **Pointer Registers:** Point to some specific location in memory.
10. Base Pointer (BP): Points to the base element of the stack.
11. Stack Pointer (SP): Always points to the top element of the stack.
12. Instruction Pointer (IP): Stores address of the next instruction to be executed. It is part of the processor register. It keeps track of the next memory address of the instruction that is to be executed, once the execution of the current instruction is executed.
13. **Flag Registers:** The flag register is used to indicate occurrence of a certain condition, during an operation of the CPU. It is a special purpose register with size one byte or two bytes. Each bit of the flag register constitutes a flag, such that the bit value indicates if a special condition was encountered while executing.

Flag Register: Overflow, Direction, Interrupt, Trap, Sign, Zero, Parity, Carry.

<><><><><><><><><><>

**5. String Rev:** To input a string at prompt, and display in reverse order.

.model small

.stack 100h

.data

array db '?'

.code

main proc

mov ax,@data ;these two lines initialize data segment

mov ds,ax

lea si,array ;to access each character in array

mov cx,0 ;initialize counter with zero

l1:

mov ah,1

int 21h

cmp al,0dh ; compare with enter key

je label1 ; conditional jump to display stored string

mov [si],al ;[square brackets] return/store value at memory address.

inc si

inc cx

jmp l1 ;unconditional jump for new character input

inc si

mov [si],'$'

label1:

mov ah,2 ;new line generation

mov dl,0dh

int 21h

mov dl,0ah

int 21h

dec si ;so as not to print last character of $

rev1: ;label for displaying in reverse

mov ah,2

mov dl,[si]

int 21h

dec si ;we are going backwards

loop rev1 ;runs the loop cx times

mov ah,4ch ;return control to DOS

int 21h

main endp

end main

**The End**