Experiment 08

Experiment 8: String & Procedure in 8086 Assembly language programming

8.1 : Check whether a given string is a palindrome or not(String & Procedure in 8086 Assembly language programming)
8.2: Length of the string
8.3 Display the string
8.4 Reverse the string

Roll No.	17
Name	Manav Jawrani
Class	D10-A
Subject	Microprocessor Lab
LO Mapped	LO5: Write programs based on string and procedure for 8086 microprocessors.

Aim: String & Procedure in 8086 Assembly language programming
1: Check whether a given string is a palindrome or not
2: Length of the string
3: Display the string
4: Reverse the string

Introduction:

String -

String is a group of bytes/words and their memory is always allocated in a sequential order. String is either referred to as byte string or word string. Here we will see some instructions which are used to manipulate the string related operations.

Procedure -

Procedures or subroutines are very important in assembly language, as the assembly language programs tend to be large in size. Procedures are identified by a name. Following this name, the body of the procedure is described which performs a well-defined job. End of the procedure is indicated by a return statement.

Theory:

For this experiment we have used various string instructions and procedure to perform the aim of the experiment.

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.

Procedures:

Syntax:

Following is the syntax to define a procedure −

```
proc_name:
   procedure body
   ...
   ret
```

The procedure is called from another function by using the CALL instruction. The CALL instruction should have the name of the called procedure as an argument as shown below −

```
CALL proc_name
```

Stacks Data Structure

A stack is an array-like data structure in the memory in which data can be stored and removed from a location called the 'top' of the stack. The data that needs to be stored is 'pushed' into the stack and data to be retrieved is 'popped' out from the stack. Stack is a LIFO data structure, i.e., the data stored first is retrieved last.

Assembly language provides two instructions for stack operations: PUSH and POP. These instructions have syntaxes like −

```
PUSH    operand
```

POP address/register

The stack implementation has the following characteristics −

- Only words or doublewords could be saved into the stack, not a byte.
- The stack grows in the reverse direction, i.e., toward the lower memory address
- The top of the stack points to the last item inserted in the stack; it points to the lower byte of the last word inserted.

Programs:

A. To check whether the string is Palindrome or not.

Algorithm:

Step 1 - Start the program and consider the string , which is to check that it is palindrome or not (in our case it is ‘civic’).Also write two more output strings like ‘String is Palindrome’ and ‘String is not Palindrome’.

Step 2 - Move the @data to ‘ax’ register and content of ‘ax’ to ‘ds’.

Step 3 - Create a procedure (‘Palindrome’ in our case).

Step 4 - Define the condition to check whether the string is palindrome or not using loops , labels and jump commands.

Step 5 - End the procedure.

Step 6 - Call the procedure and check with the given string.

Step 7 - Terminate the program.

Code:

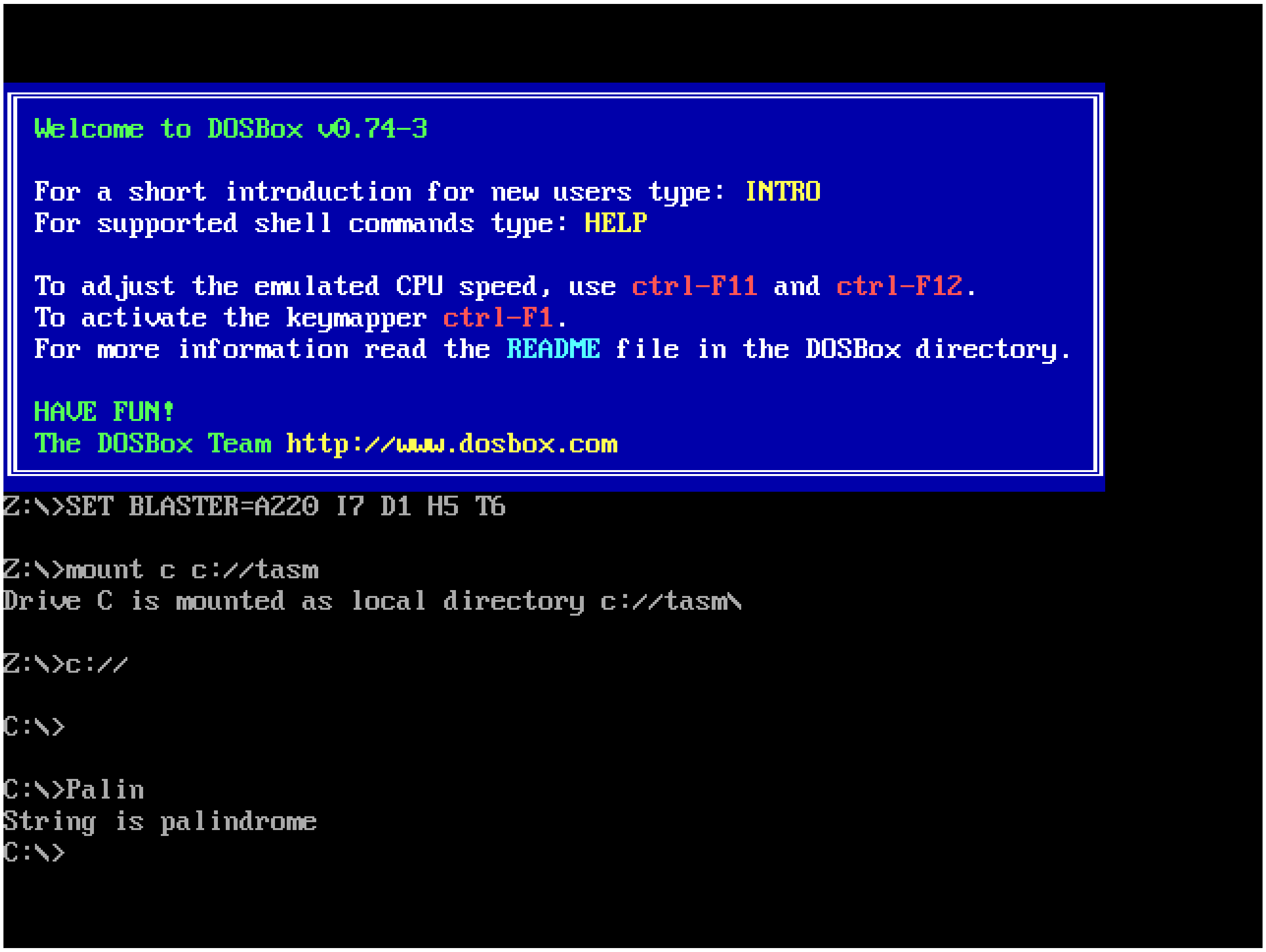
```
.MODEL SMALL
.STACK 100H
.DATA

STRING DB 'civic', '$'
STRING1 DB 'String is palindrome', '$'
STRING2 DB 'String is not palindrome', '$'
```

```
.CODE
MAIN PROC FAR
 MOV AX, @DATA
 MOV DS, AX

 CALL Palindrome

 MOV AH, 4CH
 INT 21H
 MAIN ENDP
 Palindrome PROC

 MOV SI,OFFSET STRING

 LOOP1 :
   MOV AX, [SI]
   CMP AL, '$'
   JE LABEL1
   INC SI
   JMP LOOP1

 LABEL1 :
   MOV DI,OFFSET STRING
   DEC SI

   LOOP2 :
    CMP SI, DI
    JL OUTPUT1
    MOV AX,[SI]
    MOV BX, [DI]
    CMP AL, BL
    JNE OUTPUT2

   DEC SI
   INC DI
   JMP LOOP2

 OUTPUT1:
   LEA DX,STRING1

   MOV AH, 09H
   INT 21H
   RET

 OUTPUT2:
   LEA DX,STRING2

   MOV AH,02H
   INT 21H
   RET

Palindrome ENDP
END MAIN
```

Input:

String - ‘civic’

Output:



```
Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com
Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c c://tasm
Drive C is mounted as local directory c://tasm\

Z:\>c://

C:\>

C:\>Palin
String is palindrome
C:\>
```

B. To check the length of the string.

Algorithm:

Step 1 - Start the program and consider the string whose length is to be checked (‘STUDENT BOX OFFICE’ in our case).

Step 2 - Move the @data to ‘ax’ register and content of ‘ax’ to ‘ds’.

Step 3 - Move ‘cx’ register to 00H memory location.

Step 4 - Move the string to ‘al’ register.And compare the ‘SI’ with ‘al’ register.

Step 5 - Make an increment in ‘SI’ and jump back until the string is over.

Step 6 - Calculate length using the in built length function.

Step 7 - Terminate the program.

Code:

```
ASSUME CS : CODE, DS : DATA
CODE SEGMENT
MOV AX,DATA
MOV DS,AX
MOV AL,"$"
MOV CX,00H
MOV SI,OFFSET STR1
BACK : CMP AL,[SI]
JE GO
INC CL
INC SI
JMP BACK
GO : MOV len,CL
HLT
CODE ENDS
```

DATA SEGMENT

STR1 DB 'STUDENT BOX OFFICE$'

len DB ?

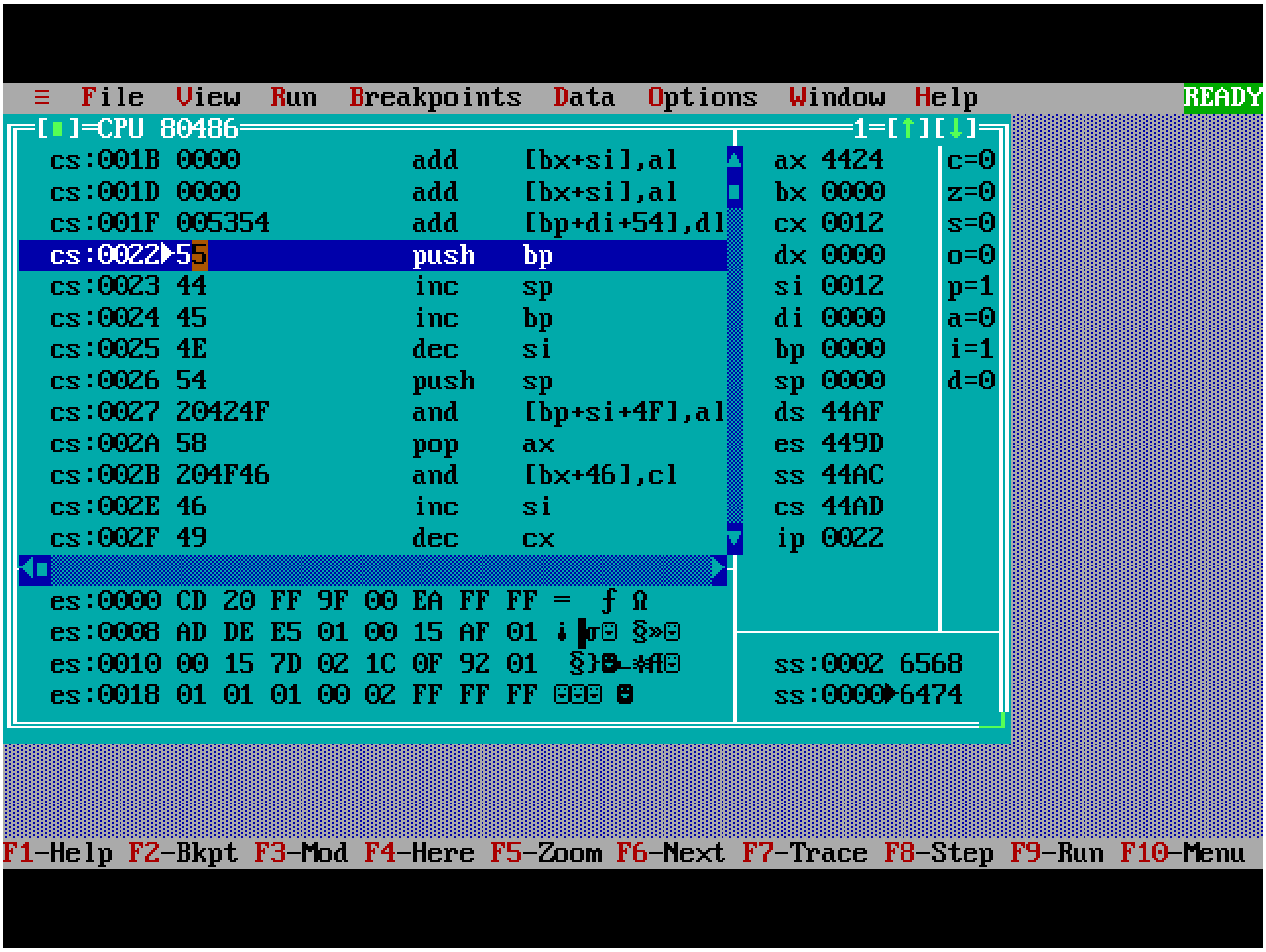
DATA ENDS

END

Input:

String - ‘STUDENT BOX OFFICE’

Output:


≡ File View Run Breakpoints Data Options Window Help READY
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

```
[■]=CPU 80486=====================================1=[↑][↓]=
  cs:001B 0000           add    [bx+si],al      ax 4424   c=0
  cs:001D 0000           add    [bx+si],al      bx 0000   z=0
  cs:001F 005354         add    [bp+di+54],dl   cx 0012   s=0
  cs:0022▸55             push   bp              dx 0000   o=0
  cs:0023 44             inc    sp              si 0012   p=1
  cs:0024 45             inc    bp              di 0000   a=0
  cs:0025 4E             dec    si              bp 0000   i=1
  cs:0026 54             push   sp              sp 0000   d=0
  cs:0027 20424F         and    [bp+si+4F],al   ds 44AF
  cs:002A 58             pop    ax              es 449D
  cs:002B 204F46         and    [bx+46],cl      ss 44AC
  cs:002E 46             inc    si              cs 44AD
  cs:002F 49             dec    cx              ip 0022

  es:0000 CD 20 FF 9F 00 EA FF FF = ƒ Ω
  es:0008 AD DE E5 01 00 15 AF 01 ¡▐σ☺ §»☺
  es:0010 00 15 7D 02 1C 0F 92 01  §}☻∟☼Æ☺        ss:0002 6568
  es:0018 01 01 01 00 02 FF FF FF ☺☺☺ ☻           ss:0000▸6474
```

C. To display the string.

Algorithm:

Step 1 - Start the program and consider the string which is to be displayed (‘Hello Everyone’ in our case).

Step 2 - Move the @data to ‘ax’ register and content of ‘ax’ to ‘ds’.

Step 3 - Offset the message to ‘dx’ register.

Step 4 - Terminate the program.

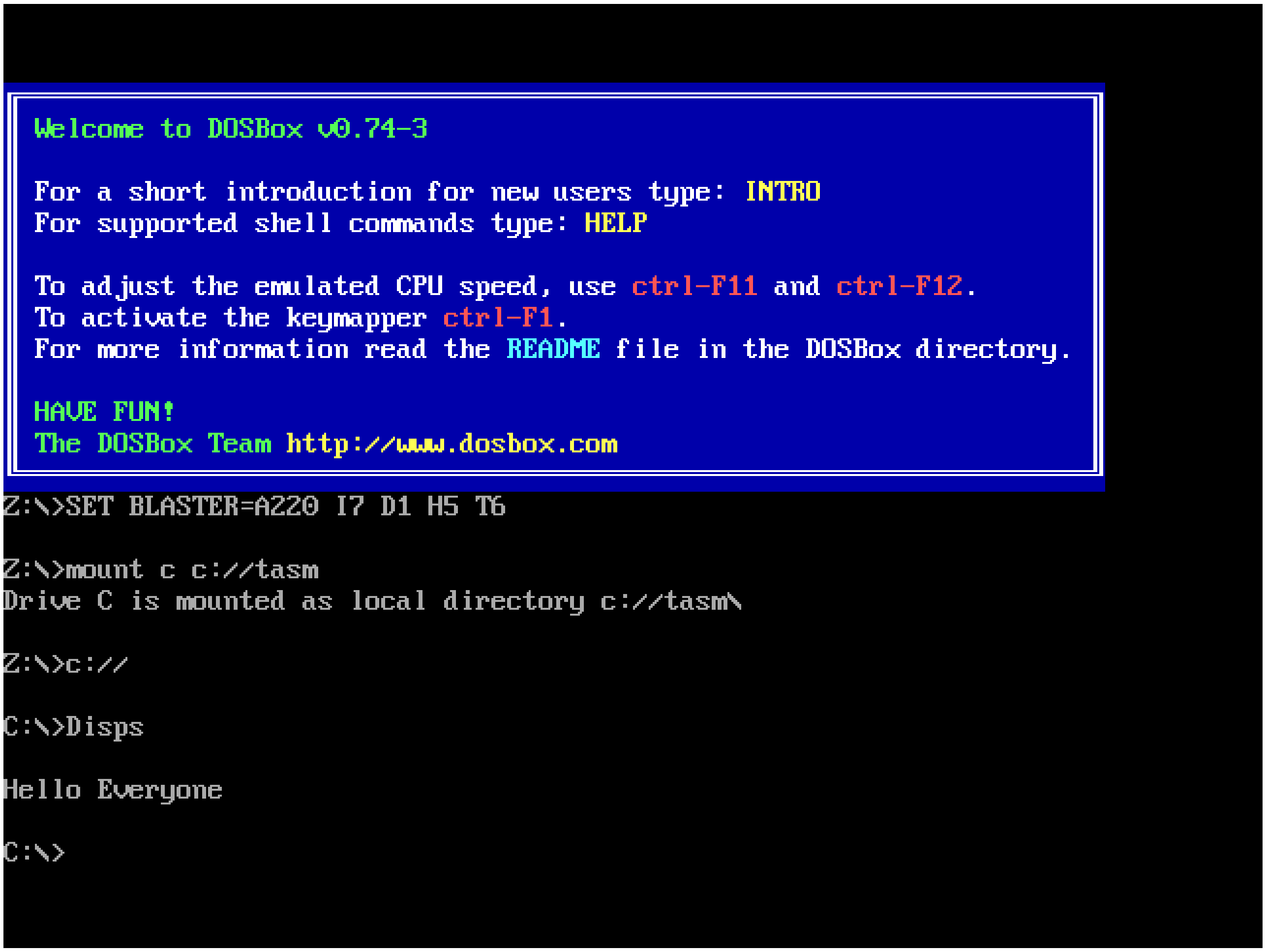
Code:

```
ASSUME CS : CODE, DS : DATA
CODE SEGMENT
MOV AX, DATA
MOV DS, AX
MOV AH, 09H
MOV DX,OFFSET MSG
INT 21H
MOV AH, 4CH
INT 21H
CODE ENDS
DATA SEGMENT
MSG DB 0DH, 0AH, "Hello Everyone", 0DH, 0AH, "$"
DATA ENDS
END
```

Input:

String - ‘Hello Everyone’

Output:



```
Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com
Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c c://tasm
Drive C is mounted as local directory c://tasm\

Z:\>c://

C:\>Disps

Hello Everyone

C:\>
```

D. To reverse a string.

Algorithm:

Step 1 - Start the program and consider the string which is to be reversed (‘Manav’ in our case).

Step 2 - Call the REVERSE function , and load the address of the string.

Step 3 - Create a procedure REVERSE and load the offset of the string.

Step 4 - Count the characters of the string.

Step 5 - Create a loop and compare if this is the last character or else push it in the stack.Then do increment in the pointer.

Step 6 - Again load the starting address of string , if count not equal to zero.Pop the top of stack.

Step 7 - Put the character of the reversed string , increment in ‘si’ and decrement the count.

Step 8 - Exit the procedure.And terminate the program.

Code:

```
.MODEL SMALL
.STACK 100H
.DATA

; The string to be printed
STRING DB 'Manav', '$'

.CODE
MAIN PROC FAR
MOV AX,@DATA
MOV DS,AX
```

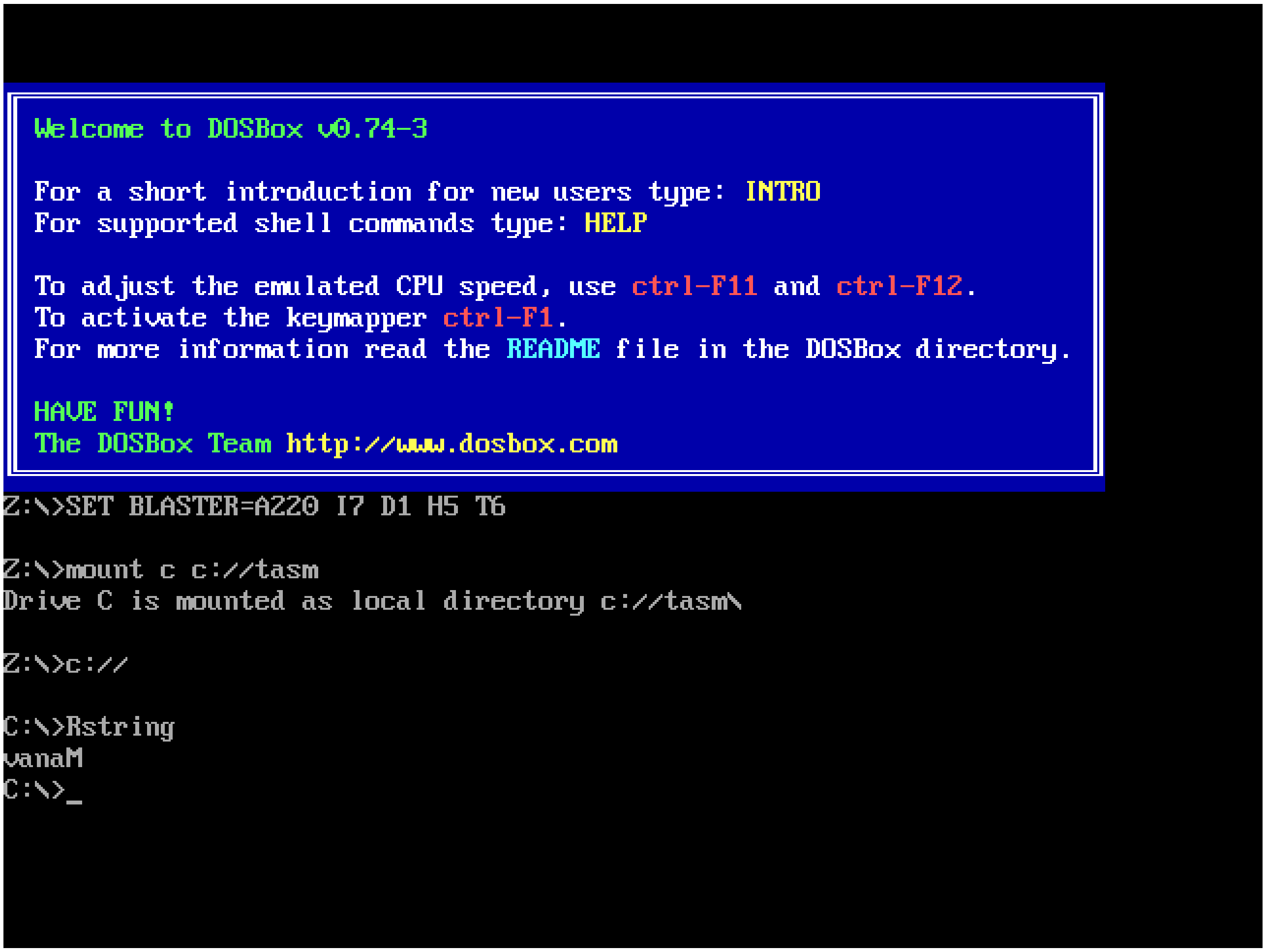
```
; call reverse function
CALL REVERSE

; load address of the string
LEA DX,STRING

; output the string
; loaded in dx
MOV AH, 09H
INT 21H

; interrupt to exit
MOV AH, 4CH
INT 21H

MAIN ENDP
REVERSE PROC
        ; load the offset of
        ; the string
        MOV SI, OFFSET STRING

        ; count of characters of the;
        ;string
        MOV CX, 0H

        LOOP1:
        ; compare if this is;
```

```
;the last character
MOV AX, [SI]
CMP AL, '$'
JE LABEL1

; else push it in the;
;stack
PUSH [SI]

; increment the pointer;
;and count
INC SI
INC CX

JMP LOOP1

LABEL1:
; again load the starting;
;address of the string
MOV SI, OFFSET STRING

        LOOP2:
        ;if count not equal to zero
        CMP CX,0
        JE EXIT

        ; pop the top of stack
        POP DX

        ; make dh, 0
        XOR DH, DH

        ; put the character of the;
        ;reversed string
        MOV [SI], DX

        ; increment si and;
        ;decrement count
        INC SI
        DEC CX

        JMP LOOP2


    EXIT:
    ; add $ to the end of string
    MOV [SI],'$ '
    RET

REVERSE ENDP
END MAIN
```

Input:

String - ‘Manav’

Output:



```
Welcome to DOSBox v0.74-3

For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com
Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c c://tasm
Drive C is mounted as local directory c://tasm\

Z:\>c://

C:\>Rstring
vanaM
C:\>_
```

Conclusion:

We have understood the aim of this experiment and successfully executed it.