



Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science (AI&DS)

Aim: Assembly Language Program to reverse the word in string.

Theory:

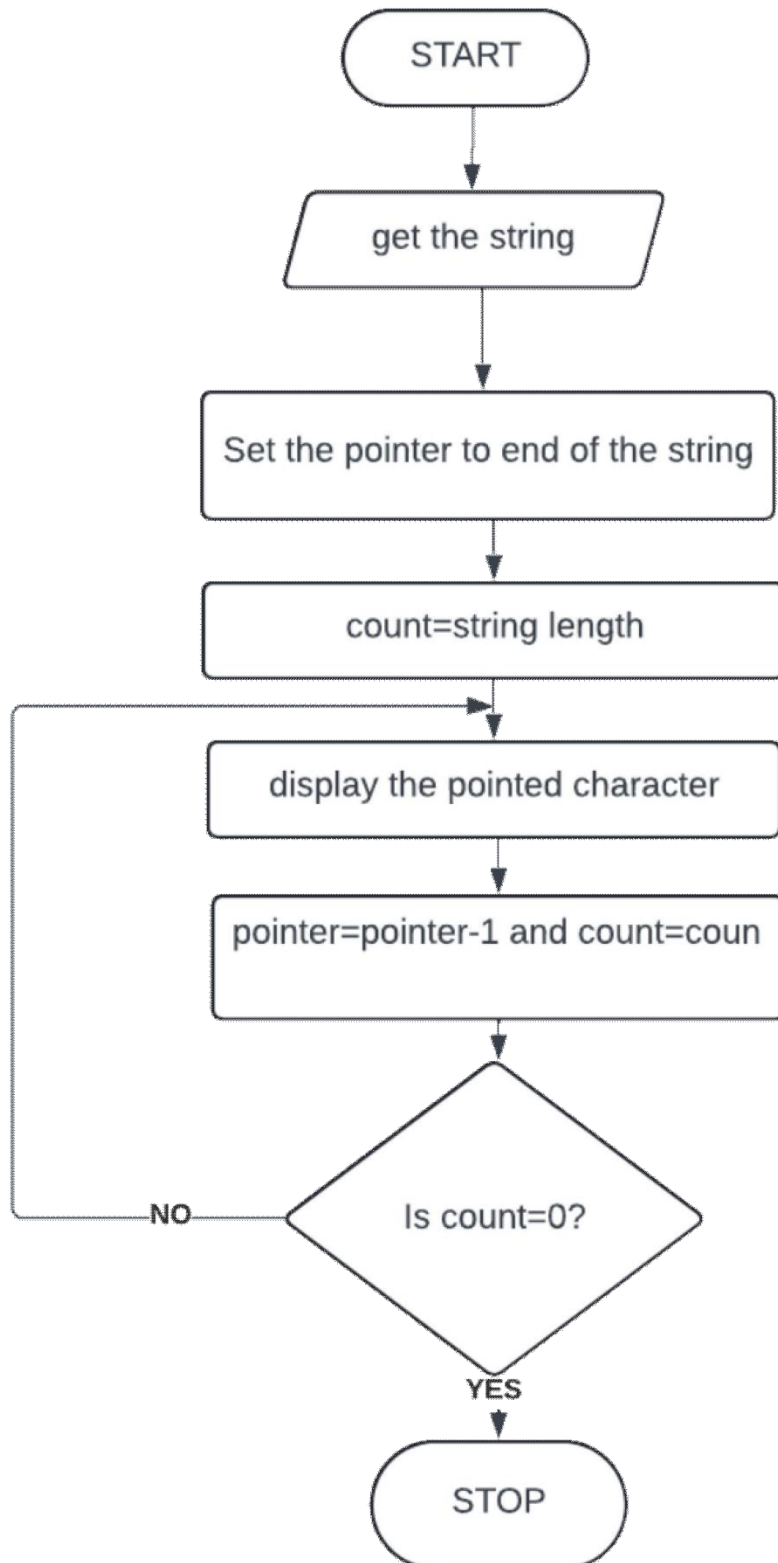
This program will read the string entered by the user and then reverse it. Reverse a string is the technique that reverses or changes the order of a given string so that the last character of the string becomes the first character of the string and so on.

Algorithm:

1. Start
2. Initialize the data segment
3. Display the message -1
4. Input the string
5. Display the message 2
6. Take characters count in DI
7. Point to the end character and read it
8. Display the character
9. Decrement the count
10. Repeat until the count is zero
11. To terminate the program using DOS interrupt
 - a. Initialize AH with 4ch
 - b. Call interrupt INT 21h
12. Stop



Flowchart:





Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science (AI&DS)

Implementation:

Code:

```
org 100h
```

```
.data
```

```
m1 db 10,13,'Enter the string:$'
```

```
m2 db 10,13,'The string is:$'
```

```
buff db 80
```

```
.code
```

```
lea dx,m1
```

```
mov ah,09h
```

```
int 21h
```

```
lea dx,buff
```

```
mov ah,0ah
```

```
int 21h
```

```
lea dx,m2
```

```
mov ah,09h
```

```
int 21h
```

```
mov cl,[buff+1]
```

```
lea bx, buff+2
```

```
L1:mov dx,[bx]
```

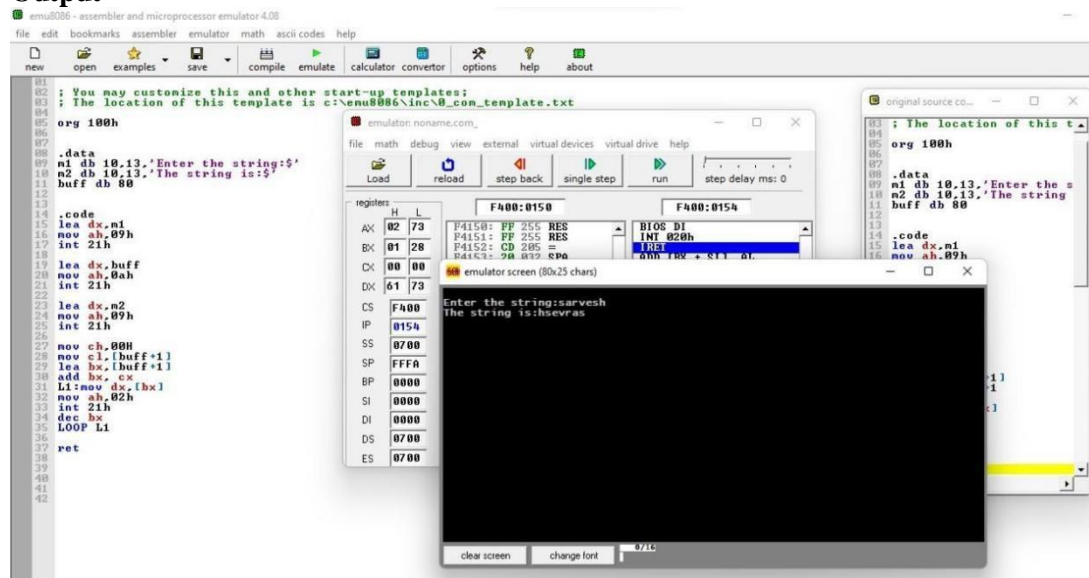
```
mov ah,02h
```

```
int 21h
```

```
inc bx
```

```
LOOP L1
```

Output





1. Explain the difference between XLAT and XLATB.

XLAT (Translate) and XLATB (Translate Byte) are both x86 assembly language instructions used for data manipulation, particularly in converting data from one form to another. Here's the difference:

XLAT:

Operation: The XLAT instruction takes the value in the AL register as an index into a lookup table (often referred to as a translation table) located in memory. It replaces the value in AL with the byte located at the address formed by adding the value in the BX register to the value in the AL register.

Syntax: XLAT

Example:

Let's say AL contains the value 0x02 and BX contains the memory offset where the lookup table starts. If the byte at the memory location BX + AL is 0x67, after executing XLAT, AL would contain 0x67.

XLATB:

Operation: XLATB is similar to XLAT but operates specifically on the AL register. It takes the value in the AL register as an index into a lookup table located in memory and replaces the value in AL with the byte located at the address formed by adding the value in the BX register to the value in the AL register.

Syntax: XLATB

Example:

Similar to XLAT, if AL contains the value 0x02 and BX contains the memory offset where the lookup table starts, after executing XLATB, AL would contain the byte at the memory location BX + 0x02.

In summary, both XLAT and XLATB perform a similar operation of looking up a byte in memory using an index in the AL register, but XLAT operates with BX as an offset while XLATB operates specifically on AL.

2. Explain the instruction LAHF.

The LAHF instruction in x86 assembly language is used to load the lower byte of the flags register (FLAGS) into the AH register. Here's a breakdown:

Operation: LAHF stands for "Load AH from Flags". It copies the lower byte of the flags register (FLAGS) into the AH register. The FLAGS register contains various status flags that reflect the outcome of arithmetic and logical operations performed by the processor.

Usage: After executing arithmetic or logical operations, you might need to inspect the status of the flags to determine the outcome of the operation. LAHF allows you to access these flags indirectly by loading them into a general-purpose register (AH) where they can be manipulated or inspected further.

Syntax : LAHF



Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science (AI&DS)

The LAHF instruction in x86 assembly language is used to load the lower byte of the flags register (FLAGS) into the AH register. Here's a breakdown:

Operation: LAHF stands for "Load AH from Flags". It copies the lower byte of the flags register (FLAGS) into the AH register. The FLAGS register contains various status flags that reflect the outcome of arithmetic and logical operations performed by the processor.

Usage: After executing arithmetic or logical operations, you might need to inspect the status of the flags to determine the outcome of the operation. LAHF allows you to access these flags indirectly by loading them into a general-purpose register (AH) where they can be manipulated or inspected further.

Syntax:

Copy code

LAHF

Affected Flags: The lower byte of the FLAGS register includes the following flags (in order from least significant bit to most significant bit):

Carry flag (CF)

Parity flag (PF)

Auxiliary Carry flag (AF)

Zero flag (ZF)

Sign flag (SF)

Trap flag (TF)

Interrupt flag (IF)

Direction flag (DF)

Overflow flag (OF)

Example:

After executing LAHF, the AH register would contain a copy of the lower byte of the FLAGS register, with each bit corresponding to the state of a particular flag.

Conclusion:

Conclusively, the experiment to perform a program to reverse the word in a string on a microprocessor offers valuable insights into low-level programming and string manipulation techniques. Through this experiment, participants gain a deeper understanding of assembly language instructions, memory management, and algorithmic thinking.

By implementing the reverse word algorithm in assembly language, participants develop skills in:

Instruction Set Understanding: They become familiar with the specific instructions available in the microprocessor's architecture and learn how to leverage them effectively for string manipulation tasks.

Memory Management: Participants grasp the concept of memory addressing and learn how to access and modify data stored in memory locations.

Algorithm Design: Crafting an efficient algorithm to reverse words within a string fosters algorithmic thinking, encouraging participants to consider factors such as time complexity and space complexity.

Debugging and Optimization: As they encounter errors and inefficiencies in their code, participants refine their debugging skills and explore optimization techniques to enhance the performance of their programs.