



# Vidyavardhini's College of Engineering & Technology

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

---

<b>Name:</b>	Saurabh Sushil Vishwakarma
<b>Roll No:</b>	61
<b>Class/Sem:</b>	SE/IV
<b>Experiment No.:</b>	7
<b>Title:</b>	Program to find whether given string is palindrome or not
<b>Date of Performance:</b>	24/02/24
<b>Date of Submission:</b>	06/03/24
<b>Marks:</b>	
<b>Sign of Faculty:</b>	



**Aim:** Assembly Language Program to find given string is Palindrome or not.

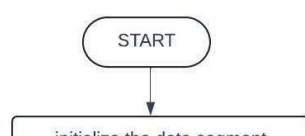
**Theory:**

A palindrome string is a string when read in a forward or backward direction remains the same. One of the approach to check this is iterate through the string till middle of the string and compare the character from back and forth.

**Algorithm:**

1. Initialize the data segment.
2. Display the message M1
3. Input the string
4. Get the string address of the string
5. Get the right most character
6. Get the left most character
7. Check for palindrome.
8. If not Goto step 14
9. Decrement the end pointer
10. Increment the starting pointer.
11. Decrement the counter
12. If count not equal to zero go to step 5
13. Display the message m2
14. Display the message m3
15. To terminate the program using DOS interrupt
  - a. Initialize AH with 4ch
  - b. Call interrupt INT 21h
16. Stop

**Flowchart:**





### Assembly Code:

```
; You may customize this and other start-up templates;
; The location of this template is c:\emu8086\inc\0_com_te
org 100h

.data
m1 db 10,13,'enter string: $'
m2 db 10,13,'your string is a palindrome $'
m3 db 10,13,'your string is not a palindrome $'

buff db 80

.code
lea dx,m1
mov ah,09h
int 21h

lea dx,buff
mov ah,0ah
int 21h

lea bx,buff+1
mov ch,00h
mov cl,[buff+1]
mov di,cx
mov si,01h
sar cl,01h

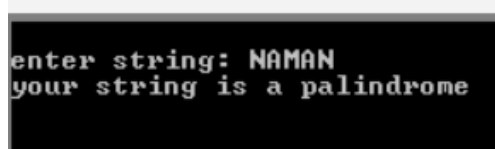
pal:
mov al,[bx+di]
mov ah,[bx+si]
cmp ah,al
JC L1
inc si
dec di
loop pal

lea dx,m2
mov ah,09h
int 21h
jmp L2

L1:
lea dx,m3
mov ah,09h
int 21h
jmp L2

L2:
mov ah,4ch
int 21h
ret
```

### Output:



```
enter string: NAMAN
your string is a palindrome
```

### Conclusion:

#### 1. Explain SAR INSTRUCTION

Ans. The SAR (Shift Arithmetic Right) instruction is a fundamental operation in x86 assembly language used to perform a signed right shift operation on binary data. Here's an explanation of the SAR instruction:

- Purpose:** SAR is used for arithmetic right shifts on binary numbers, shifting all bits to the right by a specified number of positions while preserving the sign of the original number. It's commonly used for division by powers of 2 and signed number manipulation.
- Usage:** SAR typically takes two operands: the data to be shifted and the count specifying the number of bit positions to shift by. The content of the specified register or memory location is then shifted to the right by the specified count.
- Sign Preservation:** SAR ensures that the sign bit, the leftmost bit or the most significant bit, is preserved during the shift operation. This preserves the sign of the number, ensuring that positive numbers remain positive and negative numbers remain negative after the shift.
- Flags:** SAR affects various flags such as overflow (OF), zero (ZF), sign (SF), and parity (PF) flags based on the result of the shift operation.
- Example:** SAR applied to the binary number 10101100 (representing -84 in two's complement notation) by one position results in 11010110 (representing -42), preserving the sign bit during the shift.



# Vidyavardhini's College of Engineering & Technology

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

---

2. Explain DAA instruction.

Ans. The DAA (Decimal Adjust for Addition) instruction is a crucial operation in x86 assembly language, primarily used to correct the result of addition operations involving unpacked decimal numbers stored in the AL register.

Here's a breakdown of its functionality:

- a) Purpose: DAA ensures that the result of an addition operation on BCD (Binary Coded Decimal) numbers in the AL register remains within the valid range of 0 to 9 for each decimal digit.
- b) Usage: Following an addition operation involving BCD digits, DAA is applied to adjust the result in the AL register, making it suitable for further BCD operations or output.
- c) Algorithm: DAA examines the lower nibble of AL and the auxiliary carry flag (AF) to determine if adjustments are necessary. It adds 6 to AL if the lower nibble is greater than 9 or if AF is set. Additionally, if the upper nibble is greater than 9 or if the carry flag (CF) is set, DAA adds 96 to AL.
- d) Flags: DAA may modify the carry flag (CF) and auxiliary carry flag (AF) based on the adjustments performed during the operation.
- e) Example: If AL contains 0x15 after adding two BCD digits, DAA adjusts it to 0x21, correcting the result to the proper BCD representation of 21. Overall, DAA ensures accurate BCD arithmetic by correcting the result of addition operations to comply with BCD formatting requirements.