Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL404**	**Course Name:**	**Microprocessor Lab**

Name of Student:	**Bhagyashri Kaleni Sutar**
Roll No. :	**75**
Experiment No.:	**7**
Title of the Experiment:	**Program to find whether string is palindrome or not**
Date of Performance:	**24/02/2025**
Date of Submission:	**03/03/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Ms. Sweety Patil

Signature :

Date:

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

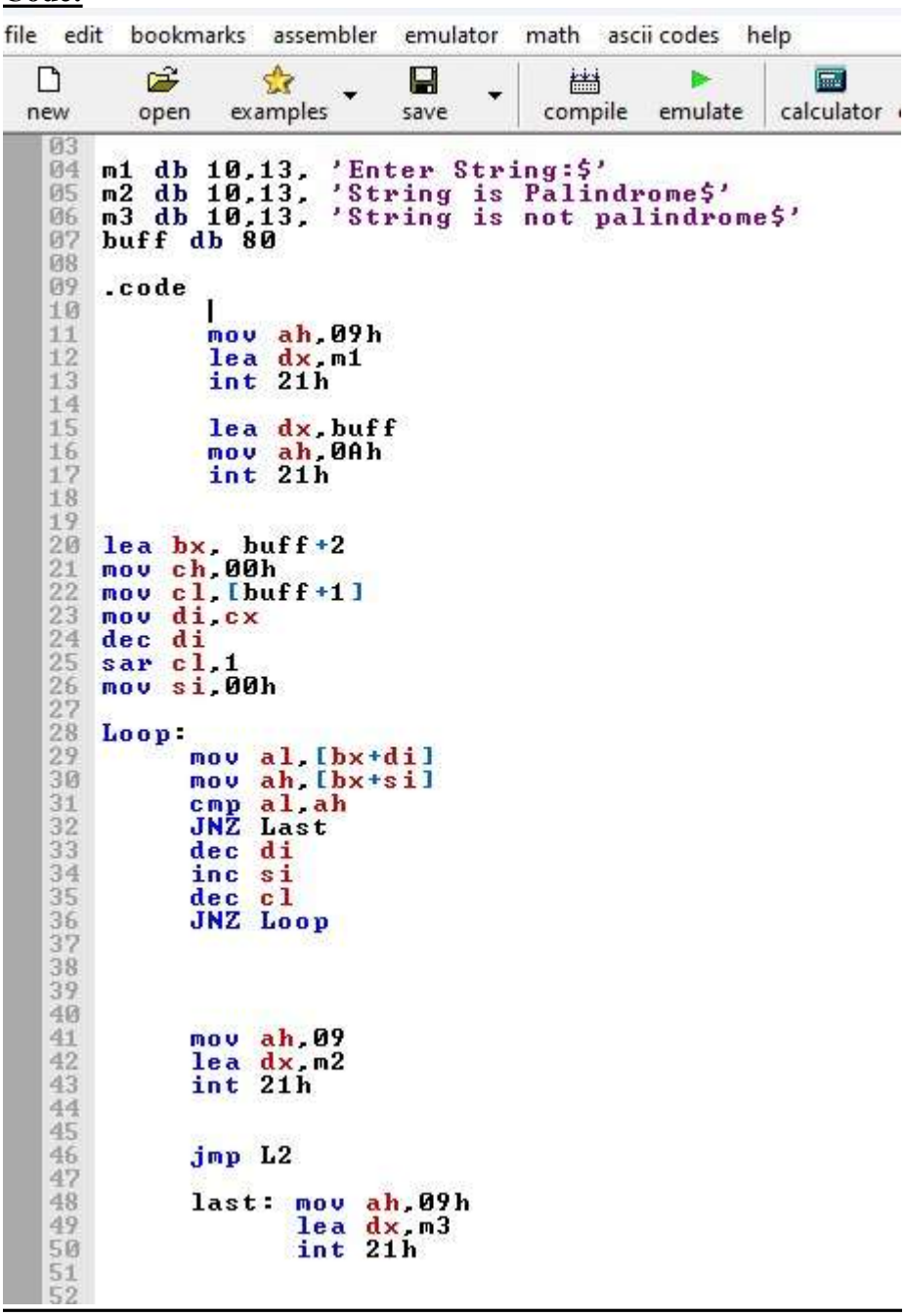
Theory:

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.

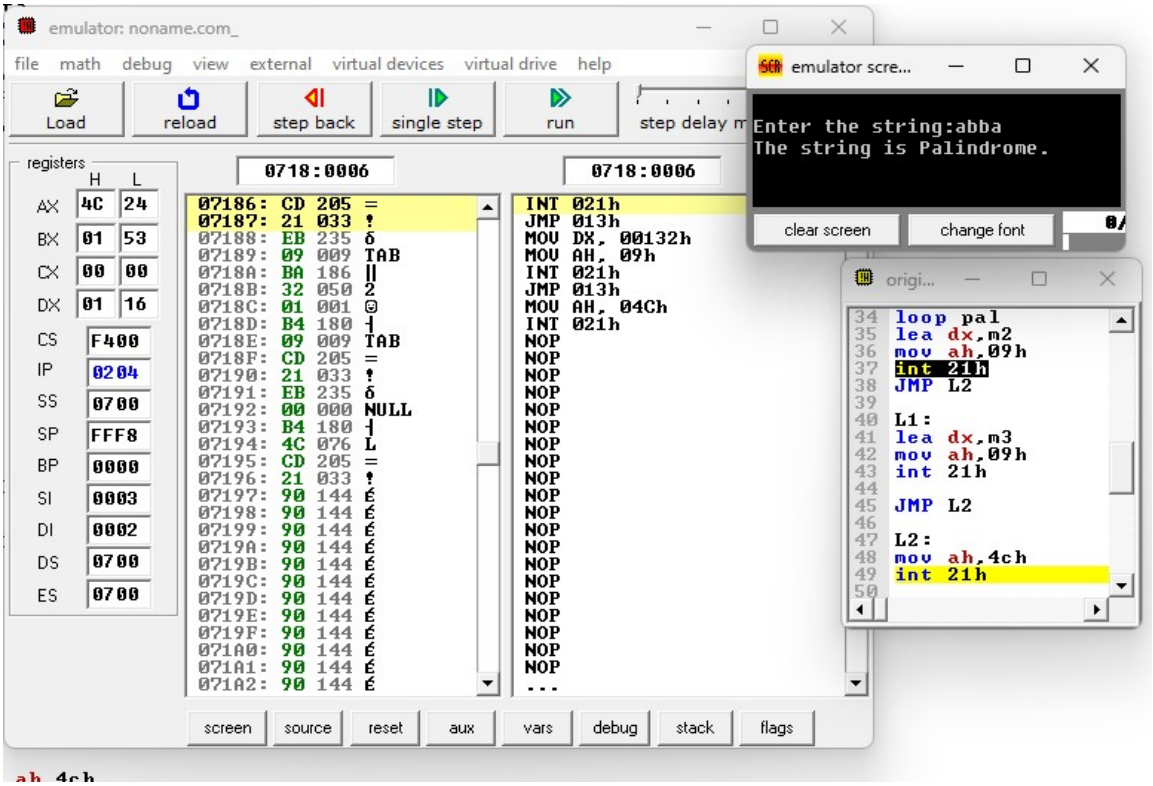
Algorithm:

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

Code:


file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator
m1 db 10,13, 'Enter String:$'
m2 db 10,13, 'String is Palindrome$'
m3 db 10,13, 'String is not palindrome$'
buff db 80
.code
mov ah,09h
lea dx,m1
int 21h
lea dx,buff
mov ah,0Ah
int 21h
lea bx, buff+2
mov ch,00h
mov cl,[buff+1]
mov di,cx
dec di
sar cl,1
mov si,00h
Loop:
mov al,[bx+di]
mov ah,[bx+si]
cmp al,ah
JNZ Last
dec di
inc si
dec cl
JNZ Loop
mov ah,09
lea dx,m2
int 21h
jmp L2
last: mov ah,09h
lea dx,m3
int 21h

Output:



Conclusion:

1. **Explain SAR INSTRUCTION**

Ans:-

The SAR instruction stands for **Shift Arithmetic Right** and is used in **x86 assembly language**. This instruction performs an arithmetic right shift on a value in a register or memory operand. The key feature of **SAR** is that it preserves the **sign** of the number (i.e., it maintains the two's complement representation of signed numbers).

Syntax:

SAR destination, count

- **destination**: This is the operand that will be shifted (can be a register or memory location).
- **count**: This can either be an immediate value (such as a number) or the value in the `CL` register, which indicates the number of positions to shift.

If **count** is not provided, the value in the CL register is used by default.

Example:-

```
SAR AL, 1    ; Shift AL 1 bit right
```

After executing the instruction:

- `AL` will be `11111110` (which represents -2 in 8-bit 2's complement).

2. Explain DAA instruction.

Ans:-

The `DAA` instruction stands for **Decimal Adjust AL After Addition**. It is used in **x86 assembly** to adjust the contents of the **AL register** after performing an addition operation involving BCD (Binary Coded Decimal) values. The `DAA` instruction is specifically useful when dealing with packed decimal numbers (BCD format).

Purpose:

The `DAA` instruction adjusts the **AL** register after adding two packed BCD numbers. It ensures that the result is a valid packed BCD value (i.e., a BCD number where each digit is represented by 4 bits, ranging from 0 to 9).

Syntax:
DAA

Example 1: Adding Two Packed BCD Numbers

Let's consider adding two packed BCD numbers, `AL = 0x25` and `BL = 0x35`.

1.Initial Values:

1.
1. `AL = 0x25` (which represents the number 25 in packed BCD format).
2. `BL = 0x35` (which represents the number 35 in packed BCD format).

2.Addition:

```
ADD AL, BL  ; AL = AL + BL = 0x25 + 0x35 = 0x5A
```

3. Decimal Adjustment:

```
DAA  ; Adjust AL to ensure it holds a valid packed BCD value
```

After the `DAA` instruction, `AL` will contain `0x8A`:

- The **lower nibble** (`0xA`) is greater than `9`, so 6 is added to the lower nibble to bring it within the BCD range (0-9).
- The result is now `0x8A`, which represents the valid packed BCD number `60` (because `0x8` represents the tens digit and `0xA` represents the ones digit).