Name:	Rahul Yadav
Roll No:	76
Class/Sem:	SE/IV
Experiment No.:	5
Title:	Program to display string in Lowercase.
Date of Performance:	07/02/24
Date of Submission:	14/02/2024
Marks:	
Sign of Faculty:	

Aim: Program to display string in Lowercase.

Theory:

The program will take Uppercase string as input and convert it to lowercase string. Int 21h is a DOS interrupt. To use the DOS interrupt 21h load with the desired sub-function. Load other required parameters in other registers and make a call to INT 21h.

INT 21h/AH = 9

output of string at DS: ・String mest be terminated by '"$"

example :

```
org 100h
```

```
mov dx, offset msg

mov ah, 9

int 21h

ret

msg db "hello world $"
```

INT 21h/AH = 0AH – input of string to DS:DX, first byte is buffer size, second byte is number of chars actually read this function does not add '$' in the end of string to print using INT 21h/AH = 9 you must set dollar character at the end of it and start printing from address DS : DX + 2. The function does not allow to enter more characters than the specified buffer siz

Algorithm:

1. Start.
2. Initialize the Data Segment.
3. Display message -1.
4. Input the string.
5. Display message-2.

6 Take the character count in CX.

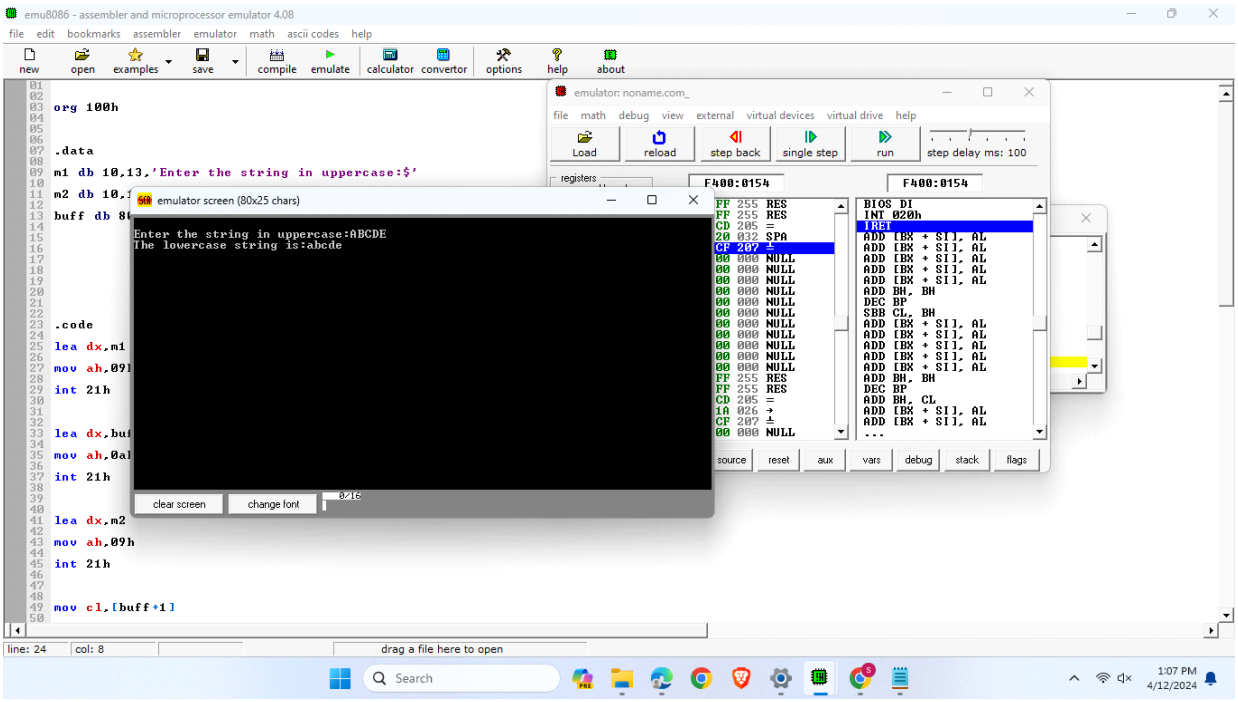
7. Point to the first character.
8. Convert it to Lowercase.
9. Display the character.
10. Decrement the character coun.
11. If not Zero, repeat from step 6.
12. To terminate the program, using the DOS interrupt:
    1) Initialize AH with 4CH
    2) Call interrupt INT 21H.
13. Stop.

Implementation:

Code:

```
org 100h

.data
m1 db 10,13,'Enter the string in uppercase:$'
m2 db 10,13,'The lowercase string is:$'
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]
add dx,20h
mov ah,02h
int 21h
inc bx
loop l1

ret
```

Output:



- **Explain instruction AAA.**

The **AAA** instruction in x86 assembly language stands for "ASCII Adjust after Addition." It is used to adjust the result of an addition operation between two unpacked decimal numbers in the AL and AH registers.

Here's a breakdown of how the **AAA** instruction works:

- **Operation**: **AAA** is typically used after addition operations involving two unpacked decimal digits, where each nibble (4 bits) in the AL and AH registers represents a decimal digit (0-9). After the addition operation, the **AAA** instruction adjusts the result in the AL register to ensure it represents a valid unpacked decimal digit.

- **Usage**: The **AAA** instruction is commonly used in BCD (Binary Coded Decimal) arithmetic operations, where decimal numbers are represented in binary form. It helps maintain the correct format of the result when performing arithmetic operations on BCD numbers.

- **Flags**: The **AAA** instruction affects the following flags:

- **CF (Carry Flag)**: The CF is set to 1 if a carry-out occurs from the low nibble (bits 0-3) to the high nibble (bits 4-7) during the adjustment.
- **AF (Auxiliary Carry Flag)**: The AF flag is undefined after the **AAA** instruction.

- **Adjustment Rules**:

- If the least significant nibble (AL) contains a value in the range 0-9 and there is no carry from this nibble during addition, no adjustment is necessary.
- If the least significant nibble (AL) contains a value in the range 0-9 and there is a carry from this nibble during addition, the AL register remains unchanged.
- If the least significant nibble (AL) contains a value in the range 10-15 (i.e., it holds an invalid unpacked decimal digit), or if the AF flag is set, the AL register is adjusted as follows:
- AL = AL + 6 (to adjust for the addition of 6 in BCD arithmetic)
- AH = AH + 1 (to account for the carry to the higher nibble)
- The CF flag is set to 1 to indicate the carry-out from the adjustment.

- **Explain instruction AAS.**

The **AAS** instruction in x86 assembly language stands for "ASCII Adjust after Subtraction." Similar to **AAA** (ASCII Adjust after Addition), **AAS** is used to adjust the result of a subtraction operation between two unpacked decimal numbers in the AL and AH registers.

Here's an overview of how the **AAS** instruction works:

- **Operation**: **AAS** is typically used after subtraction operations involving two unpacked decimal digits, where each nibble (4 bits) in the AL and AH registers represents a decimal digit (0-9). After the subtraction operation, the **AAS** instruction adjusts the result in the AL register to ensure it represents a valid unpacked decimal digit.

- **Usage**: The **AAS** instruction is commonly used in BCD (Binary Coded Decimal) arithmetic operations, where decimal numbers are represented in binary form. It helps maintain the correct format of the result when performing arithmetic operations on BCD numbers.

- **Flags**: The **AAS** instruction affects the following flags:

- **CF (Carry Flag)**: The CF is set to 1 if a borrow is required from the low nibble (bits 0-3) to the high nibble (bits 4-7) during the adjustment.
- **AF (Auxiliary Carry Flag)**: The AF flag is undefined after the **AAS** instruction.

- **Adjustment Rules**:

- If the least significant nibble (AL) contains a value in the range 0-9 and there is no borrow from the high nibble during subtraction, no adjustment is necessary.
- If the least significant nibble (AL) contains a value in the range 0-9 and there is a borrow from the

- high nibble during subtraction, the AL register remains unchanged.

  If the least significant nibble (AL) contains a value in the range 10-15 (i.e., it holds an invalid unpacked decimal digit), or if the AF flag is set, the AL register is adjusted as follows:
- AL = AL - 6 (to adjust for the subtraction of 6 in BCD arithmetic)
- AH = AH - 1 (to account for the borrow from the higher nibble)
- The CF flag is set to 1 to indicate the borrow-out from the adjustment.

Conclusion:

Conclusively, the experiment to create a program to display a string in lowercase on a microprocessor offers valuable insights into low-level string manipulation and character conversion techniques. Through this experiment, participants gain practical experience in assembly language programming and develop a deeper understanding of microprocessor architecture.