Simple Calculator in 8086 Assembly Language

◈ Introduction

This project presents a simple calculator developed using 8086 Assembly Language. The main objective of this project is to demonstrate the practical implementation of arithmetic operations—addition, subtraction, multiplication, and division—using low-level programming on Intel 8086 microprocessors. The project was designed with a user-friendly interface that interacts with the user via DOS interrupt services and takes numeric input through the keyboard.

◈ Methodology

The calculator was developed using MASM (Microsoft Macro Assembler) syntax and follows a structured, modular programming approach. Key concepts utilized include:

- Input/output through **INT 21h** DOS interrupts.
- Stack-based number entry and conversion.
- Arithmetic instructions like `ADD`, `SUB`, `MUL`, `DIV`.
- Control flow management using conditional jumps (`JE`, `JNE`, etc.).
- Handling multi-digit input and leading-zero suppression.

The methodology involved:

1. Designing modular procedures for input, output, and operations.
2. Handling ASCII to numeric conversion and vice versa.
3. Stack usage to reverse digits during entry.
4. Testing and debugging using emulators like DOSBox.

◈ Implementation

The implementation consists of the following major sections:

1. Menu and User Interaction

- Displays a welcome message and options for four operations: Add, Multiply, Subtract, Divide.
- Takes a single key input to determine the user's choice.

2. Number Input (InputNo)

- Allows the user to input multi-digit numbers.
- Digits are entered one-by-one and formed into the full number using the stack and multiplication.

3. Arithmetic Operations

- **Addition/Subtraction/Multiplication**: Uses basic `ADD`, `SUB`, and `MUL` instructions after retrieving both operands.
- **Division**: Handles quotient display, and prepares for optional remainder (though currently not shown).

4. Display Result (View)

- Converts numeric result into ASCII characters.
- Suppresses leading zeros for a clean output.

5. Program Flow Control

- Conditional jumps manage flow based on user input.
- Graceful exit after result is shown.

◈ Input

- **Input Method**: Keyboard
- **Input Format**: Multi-digit numbers entered one character at a time.
- **Valid Choices**:
  - `1` for Addition
  - `2` for Multiplication
  - `3` for Subtraction
  - `4` for Division

◈ Source Code:

```
org 100h

; add your code here
.model small
.data
.code

jmp start       ; jump over data declaration
msg0:    db      ,0dh,0ah, " ___> Simple calculator <___" ,0dh,0ah,'$'
```

```
msg:    db     0dh,0ah, "1-Add",0dh,0ah,"2-Multiply",0dh,0ah,"3-Subtract",0dh,0ah,"4-Divide",
0Dh,0Ah, '$'
msg1:   db     0dh,0ah, "Enter a number between 1,4 if you want any calculation ::",0Dh,0Ah,'$'
msg2:   db     0dh,0ah,"Enter First No : $"
msg3:   db     0dh,0ah,"Enter Second No : $"
msg4:   db     0dh,0ah,"Choice Error....please Enter any key which is in rang (1-4)" , 0Dh,0Ah,"
$"
msg5:   db     0dh,0ah,"Result : $"
msg6:   db     0dh,0ah ,'thank you for using the calculator! press any key... ', 0Dh,0Ah, '$'

start:  mov ah,9
        mov dx, offset msg0
        int 21h



        mov ah,9
        mov dx, offset msg
        int 21h

         mov ah,9
        mov dx, offset msg1
        int 21h

        mov ah,0
        int 16h
        cmp al,31h
        je Addition
        cmp al,32h  ;
        je Multiply
        cmp al,33h
        je Subtract
        cmp al,34h
        je Divide
        mov ah,09h
        mov dx, offset msg4
        int 21h
        mov ah,0
        int 16h
        jmp start


Addition:   mov ah,09h
            mov dx, offset msg2
            int 21h
```

```
        mov cx,0
        call InputNo
        push dx
        mov ah,9
        mov dx, offset msg3
        int 21h
        mov cx,0
        call InputNo
        pop bx
        add dx,bx
        push dx
        mov ah,9
        mov dx, offset msg5
        int 21h
        mov cx,10000
        pop dx
        call View
        jmp exit

InputNo:    mov ah,0
        int 16h
        mov dx,0
        mov bx,1
        cmp al,0dh
        je FormNo
        sub ax,30h
        call ViewNo
        mov ah,0
        push ax
        inc cx
        jmp InputNo

FormNo:     pop ax
        push dx
        mul bx
        pop dx
        add dx,ax
        mov ax,bx
        mov bx,10
        push dx
        mul bx
        pop dx
        mov bx,ax
        dec cx
        cmp cx,0
        jne FormNo
```

```
        ret



View:  mov ax,dx
      mov dx,0
      div cx
      call ViewNo
      mov bx,dx
      mov dx,0
      mov ax,cx
      mov cx,10
      div cx
      mov dx,bx
      mov cx,ax
      cmp ax,0
      jne View
      ret


ViewNo:    push ax
         push dx
         mov dx,ax
         add dl,30h
         mov ah,2
         int 21h
         pop dx
         pop ax
         ret


exit:   mov dx,offset msg6
       mov ah, 09h
       int 21h


       mov ah, 0
       int 16h

       ret


Multiply:   mov ah,09h
         mov dx, offset msg2
         int 21h
```

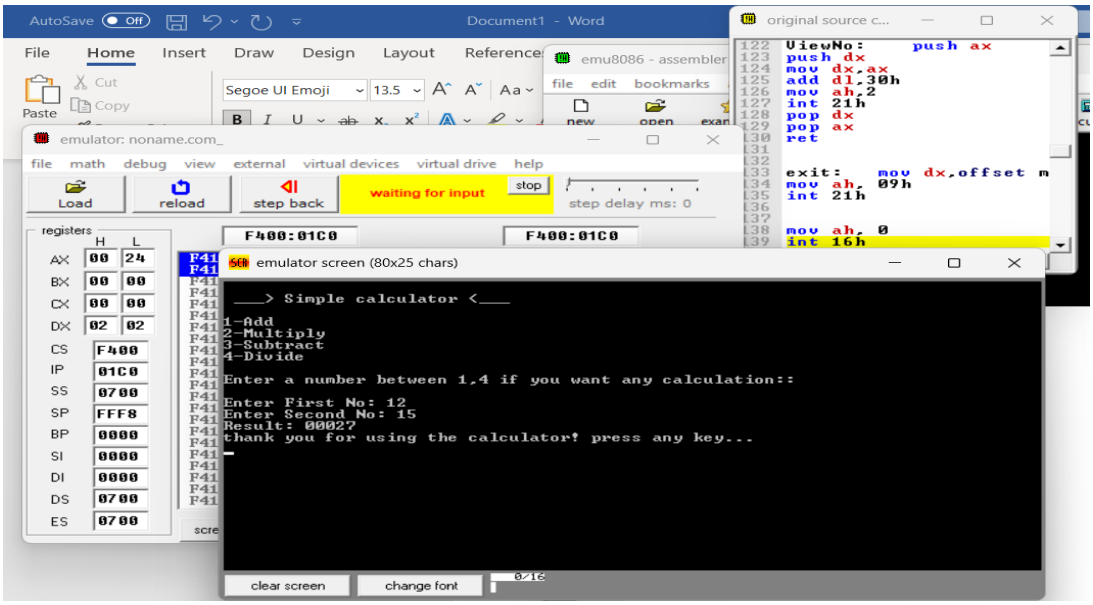
```
        mov cx,0
        call InputNo
        push dx
        mov ah,9
        mov dx, offset msg3
        int 21h
        mov cx,0
        call InputNo
        pop bx
        mov ax,dx
        mul bx
        mov dx,ax
        push dx
        mov ah,9
        mov dx, offset msg5
        int 21h
        mov cx,10000
        pop dx
        call View
        jmp exit


Subtract:   mov ah,09h
        mov dx, offset msg2
        int 21h
        mov cx,0
        call InputNo
        push dx
        mov ah,9
        mov dx, offset msg3
        int 21h
        mov cx,0
        call InputNo
        pop bx
        sub bx,dx
        mov dx,bx
        push dx
        mov ah,9
        mov dx, offset msg5
        int 21h
        mov cx,10000
        pop dx
        call View
        jmp exit
```

```
Divide:     mov ah,09h
            mov dx, offset msg2
            int 21h
            mov cx,0
            call InputNo
            push dx
            mov ah,9
            mov dx, offset msg3
            int 21h
            mov cx,0
            call InputNo
            pop bx
            mov ax,bx
            mov cx,dx
            mov dx,0
            mov bx,0
            div cx
            mov bx,dx
            mov dx,ax
            push bx
            push dx
            mov ah,9
            mov dx, offset msg5
            int 21h
            mov cx,10000
            pop dx
            call View
            pop bx
            cmp bx,0
            je exit
            jmp exit
ret
```

◈ Output

- The output is displayed in decimal format using DOS interrupt.
- Leading zero suppression ensures clean numeric display.
- Output is shown as:

Result : <answer>



◈ Conclusion

This 8086 assembly project successfully demonstrates the design and implementation of a basic calculator using low-level programming principles. The project showcases key aspects of CPU-level arithmetic processing, memory handling, stack usage, and input/output via interrupts. It provides foundational insight into how early computing systems performed basic functions and strengthens understanding of hardware-near programming.

The calculator can be further enhanced by:

- Adding support for negative numbers and floating-point calculations.
- Enhancing user interface with error messages or loop-based interaction.
- Displaying remainders for division operations.

This project not only fulfills the academic requirements but also strengthens low-level programming proficiency.
