

**COMPUTER ORGANIZATION AND ARCHITECTURE LAB  
(15B17CI373)**

**Project Title: CALCULATOR**

**Course Name: Computer Organization And Architecture Lab**

**Batch: F1**



**Submitted by:**

1. Vaishali Ranjan (9920103013)
2. Anya Rathi (9920103001)
3. Khushi Kalra (9920103025)

**Submitted to:**

Dr. Rashmi Kushwah

**Department of CSE  
Jaypee Institute Of Information Technology, Noida**

## **Table of Content**

1. Problem Statement.....	3
2. Introduction.....	4
3. List of technology/data structure used.....	5
4. Detailed design.....	6
5. Implementation details and results.....	8
6. Conclusion.....	10

## **Problem Statement**

The aim of this project is to design a program that simulates the operation of a calculator system using 8086 simulator which can perform all the basic arithmetic operations like addition, subtraction, multiplication and division.

The user will be asked to enter the numbers on which he/she wants to perform the arithmetic operations. The user will be able to decide what arithmetic operation he/she wants to perform by giving some inputs provided to them. The smart calculator which we designed will print the output on the screen.

## **Introduction**

Assembly language, also known as assembler language, is a low-level programming language that's designed to communicate instructions with specific computer hardware and direct the flow of information. It does this using human-readable mnemonics (consisting of mnemonics like "LDA" to represent load accumulator) to form short code that makes it easier for the person trying to complete the work. These short codes are converted into machine learning language (binary, i.e., 1s and 0s) through the use of programs called assemblers.

A calculator is a device that performs arithmetic operations like addition, subtraction, multiplication, and division.

In this project we have computed the assembly language code (8086) to do the some basic arithmetic operations.

## **List of Technology/Data Structure Used**

### **1. Hardware Used**

- 1.1. Processor: Minimum 1 GHz; Recommended 2GHz or more
- 1.2. Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
- 1.3. Hard Drive: Minimum 32 GB; Recommended 64 GB or more
- 1.4. Memory (RAM): Minimum 1 GB; Recommended 4 GB or above

### **2. Software Used**

- 2.1. 8086 Emulator

### **3. Data Structure Used**

- 3.1. Data transfer instructions
- 3.2. Arithmetic instructions
- 3.3. Control transfer instructions

## Detailed Design

```

0001 org 100h
0002
0003 jmp start ; jump over data declaration
0004
0005 msg: db 0dh,0ah,"CALCULATOR",3dh,0ah,"1-Add",0dh,0ah,"2-Multiply",0dh,0ah,"3-Subtract",0dh,0ah,"4-Divide", 0Dh,0Ah, '$'
0006 msg2: db 0dh,0ah,"Enter First No : $"
0007 msg3: db 0dh,0ah,"Enter Second No : $"
0008 msg4: db 0dh,0ah,"Choice Error $"
0009 msg5: db 0dh,0ah,"Result : $"
0010 msg6: db 0dh,0ah,"Thank You for using the calculator! Submitted by Khushi, Anya and Vaishali <Pi>",0dh,0ah, '$'
0011 CONTINUE DB 0dh,0ah,"DO YOU WANT TO CONTINUE",0dh,0ah,'$'
0012 CON DB 7
0013
0014 start: mov ah,9
0015 mov dx, offset msg ;first we will display the first message from which he can choose the operation using int 21h
0016 int 21h
0017 mov ah,0
0018 int 16h ;then we will use int 16h to read a key press, to know the operation he choosed
0019 cmp al,31h ;the keypress will be stored in al so, we will compare to 1 addition .....
0020 je Addition
0021 cmp al,32h
0022 je Multiply
0023 cmp al,33h
0024 je Subtract
0025 cmp al,34h
0026 je Divide
0027 mov ah,09h
0028 mov dx, offset msg4
0029 int 21h
0030 mov ah,0
0031 int 16h
0032 jmp start
0033
0034 Addition: mov ah,09h ;then let us handle the case of addition operation
0035 mov dx, offset msg2 ;first we will display this message enter first no also using int 21h
0036 int 21h
0037 mov cx,0 ;we will call InputNo to handle our input as we will take each number separately
0038 call InputNo ;first we will move to cx 0 because we will increment on it later in InputNo
0039 push dx
0040 mov ah,9
0041 mov dx, offset msg3
0042 int 21h
0043 mov cx,0
0044 call InputNo
0045 pop bx
0046 add dx,bx
0047 push dx
0048
0049 mov ah,9
0050 mov dx, offset msg5
0051 int 21h
0052 mov cx,10000
0053 pop dx
0054 call View
0055 ;for continue
0056 mov ah,09h
0057 mov dx, offset CONTINUE
0058 int 21h
0059 mov ah,01h
0060 int 21h
0061 cmp al,'y'
0062 je start
0063 cmp al,'E'
0064 jmp exit
0065
0066 InputNo: mov ah,0
0067 int 16h ;then we will use int 16h to read a key press
0068 mov dx,0
0069 mov bx,1
0070 cmp al,0dh ;the keypress will be stored in al so, we will compare to 0d which represent the enter key, to know whether he
0071 je FormNo ;if it's the enter key then this means we already have our number stored in the stack, so we will return it back
0072 sub ax,30h ;we will subtract 30 from the value of ax to convert the value of key press from ascii to decimal
0073 call ViewNo ;then call ViewNo to view the key we pressed on the screen
0074 mov ah,0 ;we will mov 0 to ah before we push ax to the stack because we only need the value in al
0075 push ax ;push the contents of ax to the stack
0076 inc cx ;we will add 1 to cx as this represents the counter for the number of digit
0077 jmp InputNo ;then we will jump back to input number to either take another number or press enter
0078
0079 ;we took each number separately so we need to form our number and store in one bit for example if our number 235
0080 FormNo: pop ax
0081 push dx
0082 mul bx
0083 pop dx
0084 add dx,ax
0085 mov ax,bx
0086 mov bx,10
0087 push dx
0088 mul bx
0089 pop dx
0090 mov bx,ax
0091 dec cx
0092 cmp cx,0
0093 jne FormNo
0094
0095 ret
0096
0097
0098 View: mov ax,dx
0099 mov dx,0
0100 div cx
0101 call ViewNo
0102 mov bx,dx
0103 mov dx,0
0104 mov ax,cx
0105 mov cx,10
0106 div cx
0107 mov dx,bx
0108 mov cx,ax
0109 cmp ax,0
0110 jne View
0111 ret
0112
0113 ViewNo: push ax ;we will push ax and dx to the stack because we will change their values while viewing then we will pop them back
0114 push dx ;the stack we will do these so, we don't affect their contents
0115 mov dx,ax ;we will mov the value to dx as interrupt 21h expects that the output is stored in it
0116 add dl,30h ;add 30 to its value to convert it back to ascii
0117 mov ah,2
0118 int 21h
0119 pop dx
0120 pop ax
0121 ret
0122
0123
0124 exit: mov dx,offset msg6
0125 mov ah, 09h
0126 int 21h
0127
0128 mov ah, 0
0129 int 16h
0130
0131 ret
0132
0133
0134 Multiply: mov ah,09h
0135 mov dx, offset msg2
0136 int 21h_

```

```

139      mov cx,0
140      call InputNo
141      push dx
142      mov ah,9
143      mov dx, offset msg3
144      int 21h
145      mov cx,0
146      call InputNo
147      pop bx
148      mov ax,dx
149      mul bx
150      mov dx,ax
151      push dx
152      mov ah,9
153      mov dx, offset msg5
154      int 21h
155      mov cx,10000
156      pop dx
157      call View
158      ;for continue
159      mov ah,09h
160      mov dx, offset CONTINUE
161      int 21h
162      mov ah,01h
163      int 21h
164      cmp al,'y'
165      je start
166      cmp al,'E'
167      jmp exit
168

```

```

169
170 Subtract:  mov ah,09h
171            mov dx, offset msg2
172            int 21h
173            mov cx,0
174            call InputNo
175            push dx
176            mov ah,9
177            mov dx, offset msg3
178            int 21h
179            mov cx,0
180            call InputNo
181            pop bx
182            sub bx,dx
183            mov dx,bx
184            push dx_

```

```

185      mov ah,9
186      mov dx, offset msg5
187      int 21h
188      mov cx,10000
189      pop dx
190      call View
191      ;for continue
192      mov ah,09h
193      mov dx, offset CONTINUE
194      int 21h
195      mov ah,01h
196      int 21h
197      cmp al,'y'
198      je start
199      cmp al,'E'
200      jmp exit
201

```

```

202
203 Divide:    mov ah,09h
204            mov dx, offset msg2
205            int 21h
206            mov cx,0
207            call InputNo
208            push dx
209            mov ah,9
210            mov dx, offset msg3
211            int 21h
212            mov cx,0
213            call InputNo
214            pop bx
215            mov ax,bx
216            mov cx,dx
217            mov dx,0
218            mov bx,0
219            div cx
220            mov bx,dx
221            mov dx,ax
222            push bx
223            push dx
224            mov ah,9
225            mov dx, offset msg5
226            int 21h
227            mov cx,10000
228            pop dx
229            call View
230            pop bx _

```

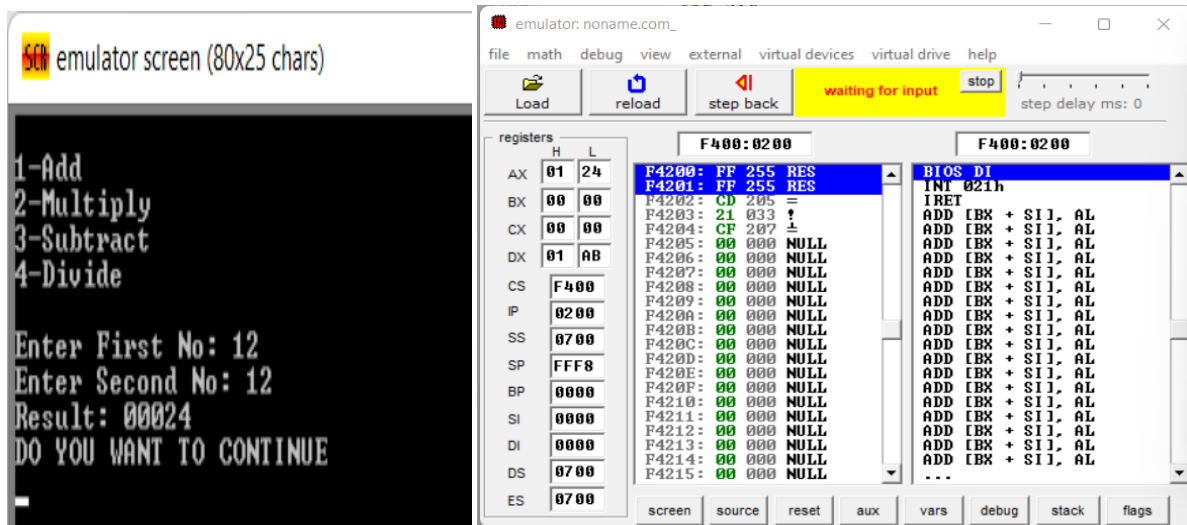
```

231      cmp bx,0
232      ;for continue
233      mov ah,09h
234      mov dx, offset CONTINUE
235      int 21h
236      mov ah,01h
237      int 21h
238      cmp al,'y'
239      je start
240      cmp al,'E'
241      je exit
242      jmp exit

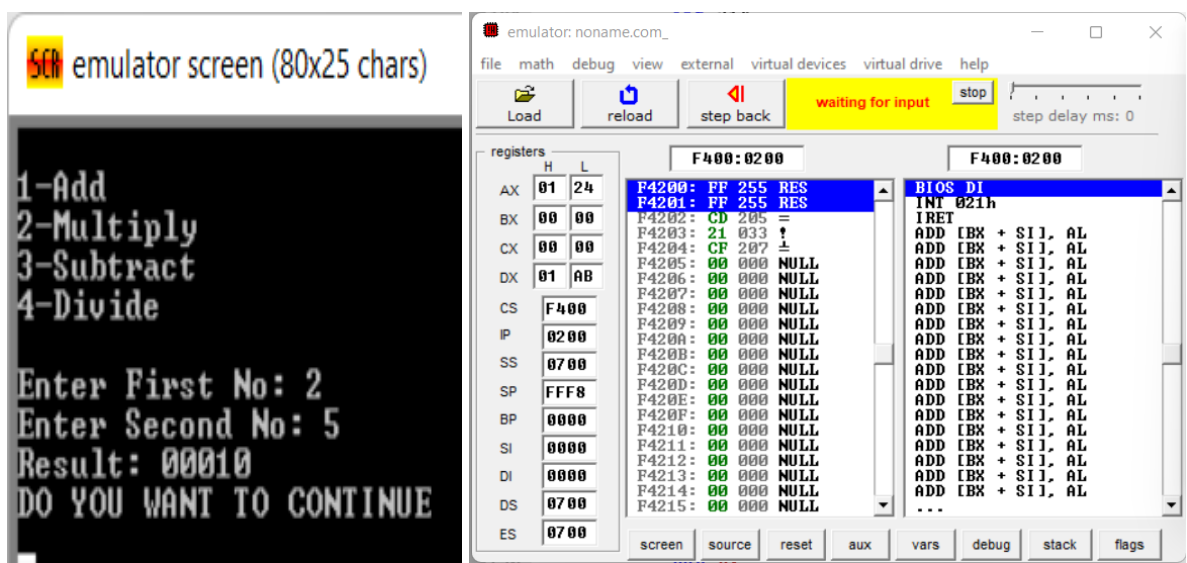
```

## Implementation Details And Results

### ADDITION:



### MULTIPLICATION:





The image shows two windows from the SCA emulator. The left window, titled 'SCA emulator screen (80x25 chars)', displays a simple calculator program with the following text:

```

1-Add
2-Multiply
3-Subtract
4-Divide

Enter First No: 50
Enter Second No: 10
Result: 00040
DO YOU WANT TO CONTINUE
  
```

The right window shows the emulator's internal state. At the top, it says 'emulator: noname.com\_'. Below this are tabs for 'file', 'math', 'debug', 'view', 'external', 'virtual devices', 'virtual drive', and 'help'. There are buttons for 'Load', 'reload', 'step back', and 'stop'. A yellow bar indicates 'waiting for input'. A slider for 'step delay ms: 0' is also present. The 'registers' section shows a table of registers (AX, BX, CX, DX, CS, IP, SS, SP, BP, SI, DI, DS, ES) with their current values. The 'F400:0200' memory window shows the BIOS interrupt vector table (INT 021h) with entries for 'I RET' and 'ADD [BX + SI], AL'. The 'F400:0200' memory window also shows the BIOS interrupt vector table (INT 021h) with entries for 'I RET' and 'ADD [BX + SI], AL'.

[illegible]

```
DO YOU WANT TO CONTINUE
E
Thank You for using the calculator! Submitted by Khushi, Anya and Vaishali (F1)
_
```

```
DO YOU WANT TO CONTINUE
Y
CALCULATOR
1-Add
2-Multiply
3-Subtract
4-Divide
```

## **Conclusion**

After complete execution of code we were able to successfully compute the basic instruction of calculator like addition, subtraction, multiplication, division in 8086 emulator.

Assembly language is not a day-to-day language but a computer engineering developer should know the assembly language so that by the piece of code one can understand what is going in the central processing unit and most important thing about assembly language is where a person wants to work at byte-bit level.