

***KHULNA UNIVERSITY OF ENGINEERING & TECHNOLOGY***

**Department of Electronics and Communication Engineering**

*Project report on designing a calculator using 8086 microprocessor*

**Course no.** ECE 3104

|  |  |
| --- | --- |
| **Submitted by** | **Project Supervisors** |
| Nahian Alam (2009026)  Sadia Islam Mumu (2009027)  Arian Rahman Aditta (2009028)  Ashrafi Zaman Sami (2009029)  Nushrath Jahan Jene (2009030)  Department of Electronics & Communication Engineering,  Khulna University of Engineering & Technology | **Md. Minhajul Islam Arnab Sir**  Lecturer,  Department of Electronics and Communication Engineering,  Khulna University of Engineering & Technology (KUET)  **Nawaz Talukder Sanglap Sir**  Lecturer,  Department of Electronics and Communication Engineering,  Khulna University of Engineering & Technology (KUET |

Objectives:

1. To create a calculator with the 8086 microprocessor.

2. To carry out any one of the four fundamental mathematical operations: division, multiplication, addition, and subtraction.

Introduction:

The conceivable effect of microprocessors on a variety of practical applications, including complicated calculations, automation, control, and interface, makes them an important core topic for students studying electrical engineering. In this paper, we suggest creating an eight-bit calculator using assembly language programming for the Intel 8086. The EMU8086 emulator program was used to create the calculator over the virtual machine for the Intel 8086 CPU. This work implemented several arithmetic and logic operations in addition to trigonometric functions. Additionally, a plot function and function tool integration will be introduced as distinct modules for this design and built accordingly. The capacity of the students to apply numerical approaches and programming algorithms to develop a compact microprocessor-based system, as well as to create a usable application for users, was much enhanced by this effort.

Theory:

Building a calculator with the 8086 microprocessor is like creating a miniaturized math whiz. This powerful chip acts as the brain of the calculator, executing instructions written in assembly language, a low-level language specifically designed for the 8086 architectures. The program interacts with the user through a keyboard, reading numbers and chosen operations (+, -, \*, /). The 8086's built-in arithmetic instructions become the workhorses, performing calculations based on the user's input. The results are then displayed on a screen, often in a simple text format. To achieve this, the program is divided into smaller, more manageable chunks called subroutines. These subroutines handle specific tasks like:

* **Input validation:** Ensuring the user enters valid numbers and operations.
* **Calculation:** Utilizing the 8086's instructions to perform the chosen arithmetic operation accurately.
* **Error handling:** Catching potential issues like division by zero and presenting informative error messages to the user.
* **Output formatting:** Displaying the calculated result in a clear and user-friendly way, potentially considering decimal points for floating-point numbers.

The design process also involves deciding how numbers will be represented (whole numbers or decimals with floating points) and crafting a user interface that is intuitive and easy to navigate. By carefully considering these aspects, we can build a functional and user-friendly calculator that leverages the power of the 8086 microprocessor.

Code:

org 100h

.data ; declaring variable

n1 db 0

n2 db 0

.code

MAIN PROC

MOV DX,OFFSET WhatOption

MOV AH,09H

INT 21H ; Display a new line

MOV AH, 02h

MOV DL, 10

INT 21h

MOV AH,01H

INT 21H

CMP AL,31H

JE ADDITION

CMP AL,32H

JE SUBTRACTION

CMP AL,33H

JE MULTIPLICATION

CMP AL,34H

JE DIVISION

MAIN ENDP

InPut PROC ; Display a new line

MOV AH, 02h

MOV DL, 10

INT 21h ; enter the first number

MOV AH, 09H

MOV DX, OFFSET inPut1

INT 21H

; Read the first number from keyboard

MOV AH, 01H

INT 21H

SUB AL, '0'

MOV n1,AL

; Display a new line

MOV AH, 02h

MOV DL, 10

INT 21h

; Prompt user to enter the second number

MOV AH, 09H

MOV DX, OFFSET inPut2

INT 21H ; Read the second number

MOV AH, 01H

INT 21H

SUB AL, '0' ; Convert ASCII digit to binary

mov n2 , al

; Display a new line

MOV AH, 02h

MOV DL, 10

INT 21h

MOV AH,00H

RET

InPut ENDP

ADDITION PROC

CALL InPut

MOV AX , 0000h

MOV BX , 0000h

MOV AL , n1

MOV BL , n2

ADD AX,BX

MOV CX,AX

;printing

MOV AH, 09H

MOV DX, OFFSET REsult

INT 21H

MOV AX,CX

MOV BL , 10

DIV BL

MOV BH , AH

MOV AH , 02h

MOV DL , AL

ADD DL , '0'

INT 21h

MOV AH , 02h

MOV DL , BH

ADD DL , '0'

INT 21h

RET

ADDITION ENDP

SUBTRACTION PROC

CALL InPut

MOV AX , 0000h

MOV BX , 0000h

MOV AL , n1

MOV BL , n2

SUB AL,BL

JS NEGETIVE

MOV AX , 0000h

MOV BX , 0000h

MOV AL , n1

MOV BL , n2

SUB AX,BX

MOV CX,AX

;printing

MOV AH, 09H

MOV DX, OFFSET REsult

INT 21H

MOV AX,CX

MOV BL , 10

DIV BL

MOV BH , AH

MOV AH , 02h

MOV DL , AL

ADD DL , '0'

INT 21h

MOV AH , 02h

MOV DL , BH

ADD DL , '0'

INT 21h

RET

SUBTRACTION ENDP

NEGETIVE PROC

MOV AX , 0000h

MOV BX , 0000h

MOV AL , n2

MOV BL , n1

SUB AX,BX

MOV CX,AX

;printing

MOV AH, 09H

MOV DX, OFFSET REsult

INT 21H

MOV AH, 09H

MOV DX, OFFSET MINUS

INT 21H

MOV AX,CX

MOV BL , 10

DIV BL

MOV BH , AH

MOV AH , 02h

MOV DL , AL

ADD DL , '0'

INT 21h

MOV AH , 02h

MOV DL , BH

ADD DL , '0'

INT 21h

RET

NEGETIVE ENDP

MULTIPLICATION PROC

CALL InPut

MOV AX , 0000h

MOV BX , 0000h

MOV AL , n1

MOV BL , n2

MUL BX

MOV CX,AX

;printing

MOV AH, 09H

MOV DX, OFFSET REsult

INT 21H

MOV AX,CX

MOV BL , 10

DIV BL

MOV BH , AH

MOV AH , 02h

MOV DL , AL

ADD DL , '0'

INT 21h

MOV AH , 02h

MOV DL , BH

ADD DL , '0'

INT 21h

RET

MULTIPLICATION ENDP

DIVISION PROC

CALL InPut

CMP n2, 0

JE DivisionByZeroError

MOV AX, 0000h

MOV BX, 0000h

MOV AL, n1

MOV BL, n2

DIV BL

MOV CL,AL

MOV CH, AH

; Display quotient

MOV AH, 09H

MOV DX, OFFSET REsult

INT 21H

MOV DL,CL

MOV AH, 02h

ADD DL, '0'

INT 21h ; Display quotient

; Display remainder

MOV AH, 09H

MOV DX, OFFSET rem

INT 21H

MOV DL,CH

MOV AH, 02h

ADD DL, '0' ; Convert remainder to ASCII character

INT 21h ; Display remainder

RET

DIVISION ENDP

DivisionByZeroError PROC

; Handle division by zero error here, such as displaying an error message

MOV AX, 0H

MOV ES, AX

MOV AL, 65H

MOV BL,4H

MUL BL

MOV BX, AX

MOV SI,offset [ZEROO]

MOV ES:[BX],SI

ADD BX,2

MOV AX,CS

MOV ES:[BX],AX

INT 65H ; Trigger division by zero error

RET

DivisionByZeroError ENDP

ZEROO:

MOV AH, 09H

MOV DX, OFFSET intterrupt

INT 21H

MOV AH, 4Ch ;END PROGRAM

INT 21h

WhatOption DB "Which operation to perform - 1.ADDITION 2.SUBTRACTION 3.MULTIPLICATION 4.DIVISION",'$'

inPut1 DB "Enter 1st No:",'$'

inPut2 DB "Enter 2nd No:",'$'

REsult DB "ANSWER : ",'$'

rem DB "REMAINDER : ",'$'

intterrupt DB "I CAN'T DIVIDE BY ZERO! I QUIT...",'$'

MINUS DB "-",'$'

END MAIN

Result Analysis:

In this project, our main concern was to execute arithmetical operations in a designed calculator with 8086 microprocessor. There are too many things which are hidden while designing the calculator. **To display a message on the console using the 8086 microprocessor**-

In this modified code:

- The msg variable stores the string to be displayed, terminated with a '$' character.

- Inside the main procedure:

- mov dx, offset msg loads the offset address of the message into the DX register.

- mov ah, 09h sets AH to 09h, indicating that we want to display a string.

- int 21h is called to trigger interrupt 21h, which will display the string located at the offset address stored in DX.

- Finally, the program terminates using interrupt 21h with AH set to 4Ch.

**There is another term by which the input is retrieved from console in 8086 microprocessor.** To take input from the keyboard using interrupt 21h and ah=02h in 8086 assembly language, we followed these steps:

1. We set the AH register to 02h to indicate that we wanted to read a character from the keyboard.

2. We called interrupt 21h.

3. After the interrupt returns, the AL register will contain the character read from the keyboard.

Here's a simple example code snippet:

assembly

mov ah, 02h ; Set AH to 02h for reading character input

int 21h ; Call interrupt 21h to read character input

mov ah, 00h ; Clear AH to indicate no error

int 21h ; Call interrupt 21h to display character on the screen

There is another and slightly important sector is updating the interrupt vector table. **Updating the 8086 interrupt vector table (IVT) for a specific interrupt** is a delicate process. While you can calculate the interrupt's memory offset and replace its handler address, it's generally discouraged. Modifying the IVT directly can lead to system instability. This is why we followed following steps to update the IVT:

**1. Identify the interrupt number:** The code snippet utilizes interrupt 65H to handle division by zero errors.

**2. Write the custom interrupt handler:** The DivisionByZeroError procedure serves as the custom interrupt handler. It starts by calculating a value that would cause a division by zero error (multiplying zero by 4), then updates the interrupt vector table with the address of the error handling routine, and finally triggers the division by zero error using INT 65H.

**3. Calculate the offset:** The offset address of the custom interrupt handler is calculated using the formula provided in the code.

**4. Update the interrupt vector table:** The offset address of the custom handler is stored in the interrupt vector table at a location determined by the calculated value.

**5. Restore the original handler:** The original handler is not saved or restored in this code snippet. However, in more complex systems, it's common practice to save the original handler address before overwriting it with a custom handler.

**6. Trigger the interrupt:** The INT 65H instruction is used to trigger the division by zero error, which will then invoke the custom interrupt handler.

**7. Handle the interrupt:** The custom interrupt handler, DivisionByZeroError, executes when the division by zero error occurs. It displays an error message and terminates the program using software interrupt 21h (INT 21H).

**8. Resume normal execution:** After handling the interrupt, the program terminates.

The main operations of calculators or arithmetic operations are performed by the 8086 microprocessor.

**1. Addition (ADD):**

This instruction adds two operands and stores the result in the destination.

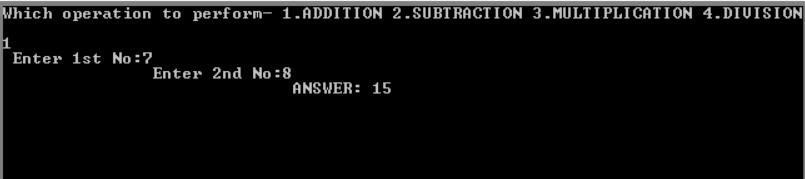
Code:

MOV AX, number1 ; Load first number into AX register

ADD AX, number2 ; Add second number to AX

; AX now contains the sum

**Output:**



**2. Subtraction (SUB):**

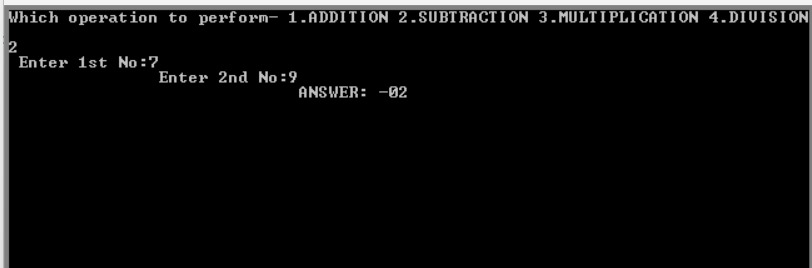
This instruction subtracts the source operand from the destination and stores the result in the destination.

Code:   
MOV AX, number1 ; Load first number into AX register

SUB AX, number2 ; Subtract second number from AX

; AX now contains the difference

**Output:**



**3. Multiplication (MUL):**

This instruction multiplies two operands and stores the 16-bit result in AX and the 16-bit higher-order product in DX. Be mindful of overflow for larger numbers.

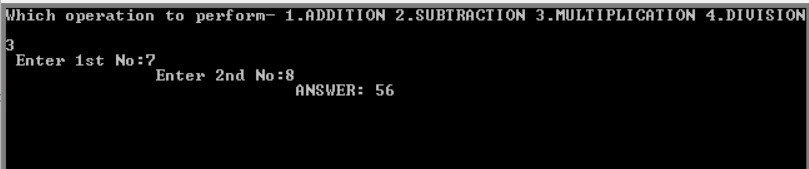
Code:

MOV AX, number1 ; Load first number into AX register

MUL number2 ; Multiply AX by second number

; AX contains the lower 16 bits of the product, DX contains higher 16 bits

**Output:**



**4. Division (DIV):**

This instruction divides the contents of AX by the source operand and stores the quotient in AX and the remainder in DX. Division by zero will cause an exception.

Code:

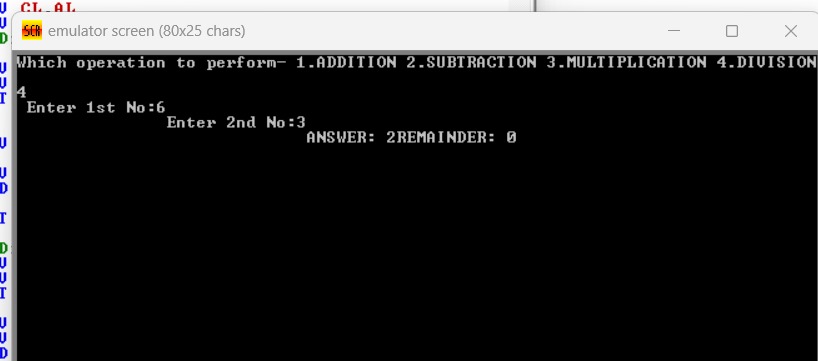
MOV AX, numerator ; Load numerator into AX register

MOV DX, 0 ; Initialize DX to 0 (remainder)

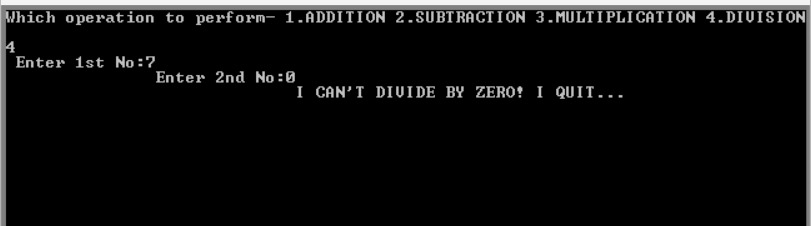
DIV denominator ; Divide AX by denominator

; AX contains the quotient, DX contains the remainder

**Output:**



If the denominator is declared 0(zero) then the calculator will be providing a message, “I CAN’T DIVIDE BY ZERO. I QUIT…” Here is an example,



Discussion:

In this project, we use 8086 microprocessors to design a basic calculator.   
Basic arithmetic operations including addition, subtraction, division, and multiplication may all be properly performed using this calculator. Our calculator accepts input. We made a user the divide by zero interrupt program, which is defined. We have created an ISR program with division zero, and the output result appears flawlessly on the console. First of all, the subtraction operation's negative value was not displayed by our calculator. Once the code in the subtraction section is changed, our software produces negative results. All in all, we successfully finished our assignment.

Conclusion:

In conclusion, we have designed a calculator that can perform fundamental mathematical operations like addition, subtraction, division, and multiplication using an 8086 CPU. It is possible to use signed and unsigned data with this calculator. A calculator will display an error message if we enter data incorrectly. If we divide by zero from any number, it can also display a divide by zero interrupt. We can only hope that having knowledge of these projects will be useful to us later on.

References:

1. <https://www.tutorialspoint.com/microprocessor/microprocessor_8086_overview.htm>
2. <https://www.geeksforgeeks.org/microprocessor-tutorials/>