8086 ASSEMBLY LANGUAGE CALCULATOR

A COURSE PROJECT REPORT

By

KEERTHANA(RA2111003010250)

Under the guidance of

Dr. M. Kanchana
Associate Professor
Computing Technologies

In partial fulfillment for the Course

of

18CSC203J - COMPUTER ORGANIZATION AND ARCHITECTURE Department of

Computing Technologies

FACULTY OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF

SCIENCE AND TECHNOLOGY Kattankulathur, Chenpalpattu District

NOVEMBER 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors. We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We are highly thankful to our my Course project Internal guide **Dr. M. Kanchana, Associate Professor, Computing Technologies** for **her** assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to **Dr. M. Pushpa Latha, Head of Department, Computing Technologies** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

BONAFIDE CERTIFICATE

Certified that this project report "8086 ASSEMBLY LANGUAGE CALCULATOR" is the bonafide work of Student Name (Register no) who carried out the project work under my supervision.

SIGNATURE

Dr. M. Kanchana
Associate Professor
Computing Technologies
SRM Institute of Science and Technology
Potheri, SRM Nagar,Kattankulathur,

TABLE OF CONTENTS

CHAPTERS CONTENTS

ABSTRACT

Microprocessors and their applications course is considered as a significant core course for engineering students due to its potential impact into several real life applications such as complex calculations, interfacing, control and automation technology. We propose an eight bit scientific calculator based Intel 8086 assembly language programming. The calculator were designed over the virtual machine for Intel 8086 microprocessor using EMU8086 emulator software. Several arithmetic and logic operations as well as trigonometric functions were implemented in this paper. Also, a plot function and integration of function tools are to be implemented and added as a separate modules for this design. This work was very beneficial in enhancing the student' skills in mathematics, engineering and computer programming which can be employed in designing a useful applications for users as well as the ability to apply numerical techniques and programming algorithms to design a small microprocessor-based system.The project is about assembly language calculator in which we have used different operations to perform arithmetic operations. I have performed this experiment on the software EMU 8086. The assembly language programming 8086 mnemonics are in the form of op-code, such as MOV, MUL, JMP, and so on, which are used to perform the operations. Different registers have been used to store the values of the operations. The operations used in the project are Addition, Subtraction, Division and Multiplication.

INTRODUCTION

Microprocessors and their applications course is considered as a significant core course for engineering students due to its potential impact into several real life applications such as complex calculations, interfacing, control and automation technology. We propose an eight bit scientific calculator based Intel 8086 assembly language programming. The calculator were designed over the virtual machine for Intel 8086 microprocessor using EMU8086 emulator software. Several arithmetic and logic operations as well as trigonometric functions were implemented in this paper. Also, a plot function and integration of function tools are to be implemented and added as a separate modules for this design. This work was very beneficial in enhancing the student' skills in mathematics, engineering and computer programming which can be employed in designing a useful applications for users as well as the ability to apply numerical techniques and programming algorithms to design a small microprocessor-based system.The project is about assembly language calculator in which we have used different operations to perform arithmetic operations. I have performed this experiment on the software EMU 8086. The assembly language programming 8086 mnemonics are in the form of op-code, such as MOV, MUL, JMP, and so on, which are used to perform the operations. Different registers have been used to store the values of the operations. The operations used in the project are Addition, Subtraction, Division and Multiplication.

REQUIREMENT ANALYSIS

8086 will allow for easy computation on each of the arithmetic operations with MIPS protocol. Addition, subtraction, multiplication and division will have two different operations, one done by MIPS standard instruction arithmetic set and one done by the self-made logical operations which very closely ties to how the processor does this on a hardware level. The project objectives are listed as so: 1. Successfully execute the 8086 simulator program. 2. Implement two modules that successfully achieves Arithmetic operations, one with MIPS Standard Instruction Set and another with only logical operations including binary shifting, masking, Boolean logic. 3. Test the procedures created and verify the results using 8086 simulator. To all the project objectives, provided are simple steps to successfully execute 8086 Simulator, implement these two modules, and finally run and test these implemented procedures

PREPARATION FOR IMPLEMENTATION OF ARITHMETIC OPERATIONS:

A. Installation and Execution 8086 simulator can easily be attained on any of the well-known and popularly used browsers available. While this report was being written, the version used is 8086

B. Loading Project Files into the8086 Environment Provided in the hyperlinked Link: Arithmetic Project Files will download the file “calc.zip” which when extracted will give you a folder containing the files

C. Configuring the 8086 Environment Properly To allow proper compiling and execution of the to be implemented arithmetic modules, turn on ‘Assembles all files in directory’ and ‘Initialize program counter to global main if defined’ options in 8086 settings tab

ARCHITECTURE AND DESIGN

- **ADDING** :

  Adding two numbers is an addition. We may add signed or unsigned numbers. When we add two numbers, say 8 and 5, the result is 13 i.e. while adding two single-digit numbers, we may get a two-digit number in the result. A similar possibility exists in the binary system too. Thumb rule of binary addition is:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10

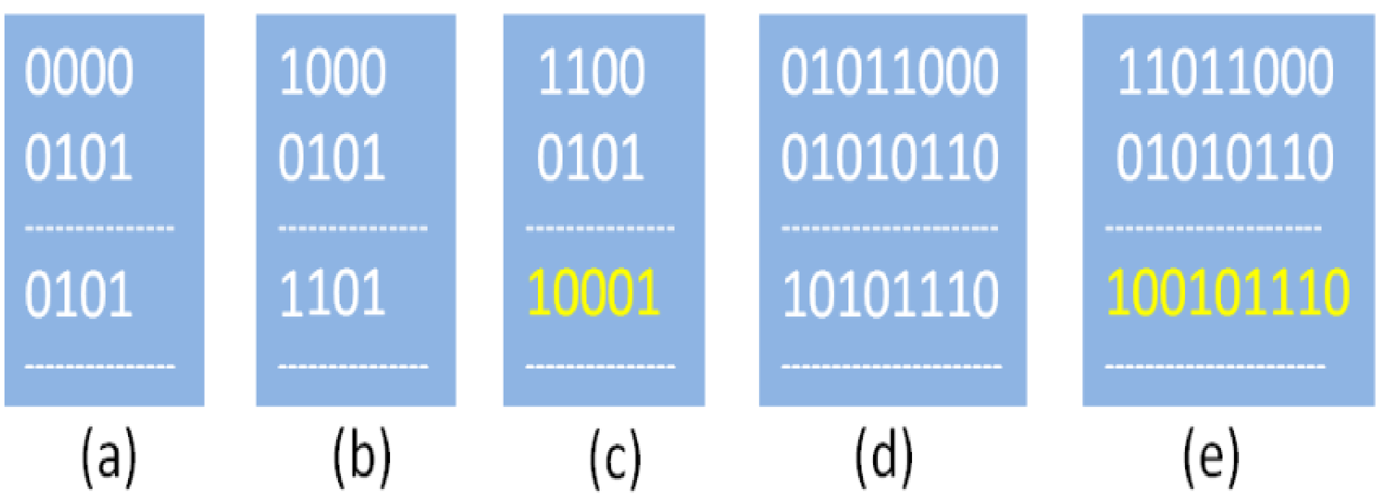
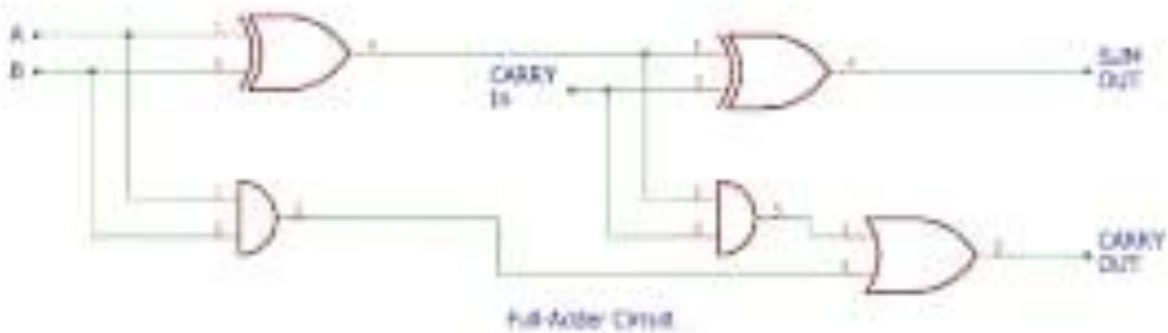


Figure 8.1 Examples of binary Addition

<u>Examples (a –e) of unsigned binary addition are given in figure 8.1.</u>



- **SUBTRACTION** :

  Subtraction is finding the difference of B from A i.e A-B. Basis of binary subtraction is:

  0 - 0 = 0

  0 - 1 = -1

  1 - 0 = 1

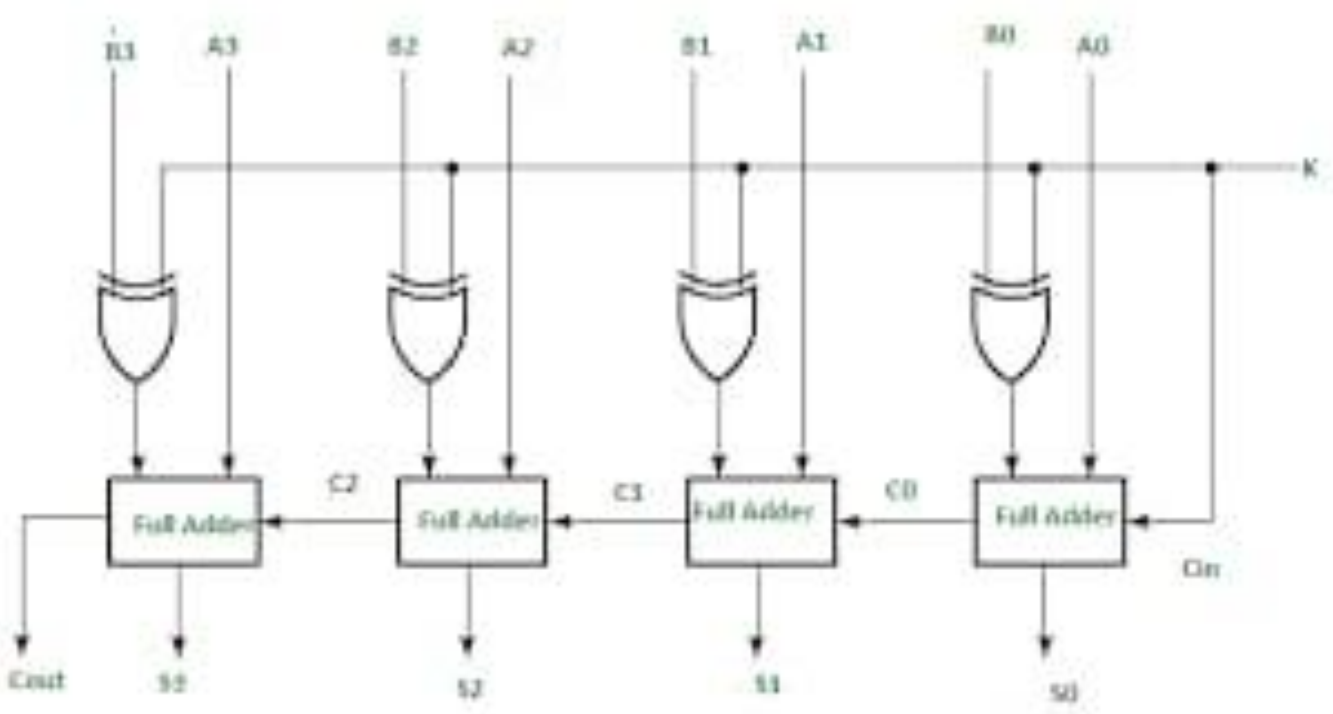
  1 - 1 = 0

  Of course, the usual borrow logic from the adjacent digit is applied as in the case of decimal numbers. Examples of signed binary Subtraction is as below:

2’s complement for subtraction :

"1's complement + 1 = 2's complement"

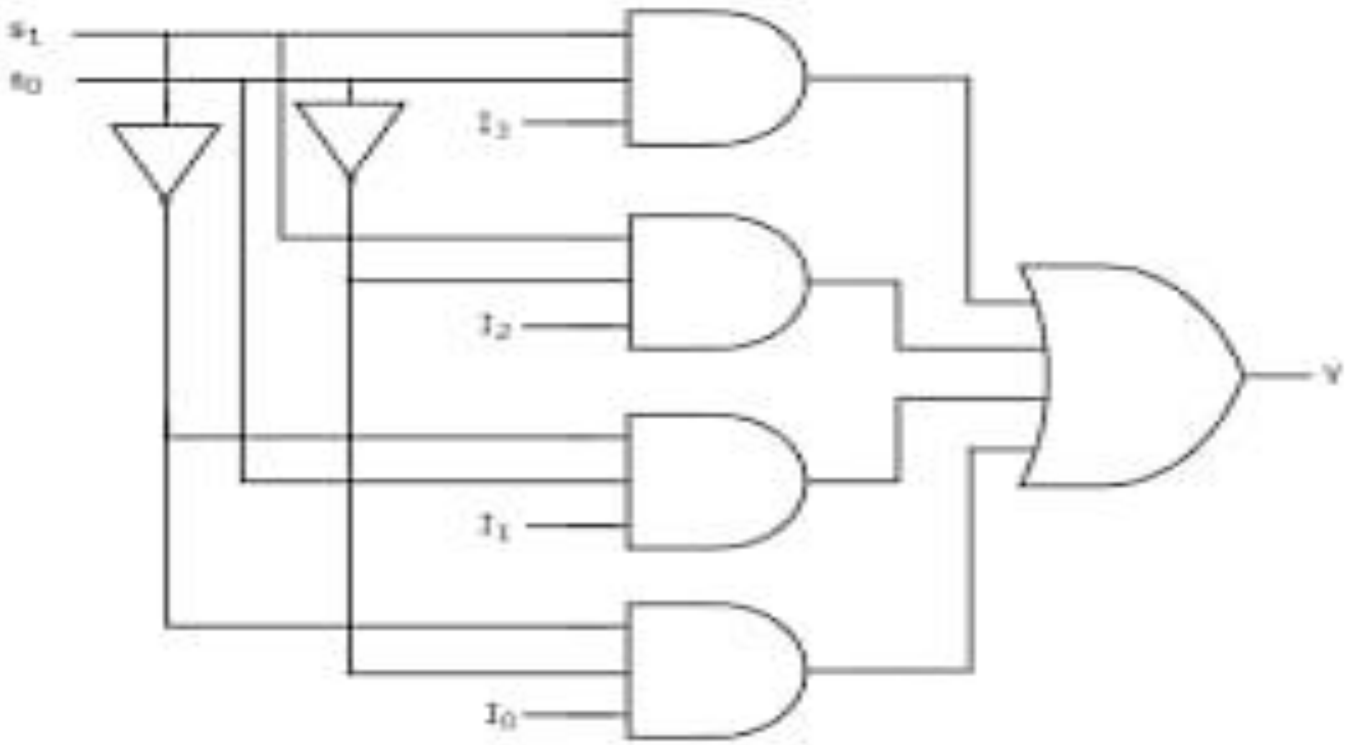
Generating this 2's complement is very simple using an XOR circuit. The XOR circuit will generate 1's complement. A control signal called SUBTRACT is used as add value of 1. This way, an adder executes subtraction. See the example below, where case (b), case (c) and case (e) are worked out as 2's complement representation; and A-B becomes A + (2's complement(B)). The result is obtained in 2's complement form discarding the carry. Observe that this method works for all kind of data.



Full Adder is the adder that adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM. A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to another. we use a full adder because when a carry-in bit is available, another 1-bit adder must be used since a 1-bit half-adder does not take a carry-in bit. A 1-bit full adder adds three operands and generates 2-bit results.

- **MULTIPLICATION** :
  Multiplication of fixed-point binary numbers in signed-magnitude representation is done by successive shift and add operations.
  The process consists of looking at successive Multiplier, least significant bit first. If the Multiplier is 1, the multiplicand is copied down; otherwise, zero is copied. And like we do in standard multiplication, the numbers copied down in successive lines are shifted one position to the left. Finally, all binaries are added, and the total sum is the result. The sign of the product(result) is determined from the signs of multiplicand and Multiplier. If they are alike, the final product sign is positive. If they are unlike, the sign of the product is negative.

- **DIVISION :**
  Division to two fixed-point binary numbers in signed-magnitude representation is done by the process of successive compare, shift, and subtract operations. The binary division is simpler than decimal because the quotient is either 0 or 1. There is no need to calculate how many times the dividend or partial remainder fires into the divisor. You can follow the following steps for binary division.
  Step 1: Compare the divisor with the dividend; if the divisor is greater, place 0 as the quotient, then bring down the second bit of the dividend. If the divisor is smaller, multiply it by one, and the result must be subtracted. Then, subtract the result from the above to get the remainder.
  Step 2: Bring down the next number bit from the dividend and perform step1.
  Step 3: Repeat the whole process until the remainder becomes 0, or the whole dividend is divided.



s_1
s_0
I_3
I_2
I_1
I_0
Y

IMPLEMENTATION

CODE:

```
org 100h
jmp start      ; jump over data declaration
msg1:    db    "1-Add",0dh,0ah,"2-Multiply",0dh,0ah,"3-Subtract",0dh,0ah,"4-
Divide", 0Dh,0Ah, '$'
msg2:    db      0dh,0ah,"Enter First No : $"
msg3:    db      0dh,0ah,"Enter Second No : $"
msg4:    db      0dh,0ah,"Choice Error $"
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 msg ;first we will display hte first message from which he can
choose the operation using int 21h
int 21h
mov ah,0
int 16h  ;then we will use int 16h to read a key press, to know the operation   he
choosed
cmp al,31h ;the keypress will be stored in al so, we will comapre to 1    addition
..........
 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  ;then let us handle the case of addition operation
```

```
 mov dx, offset msg2  ;first we will display this message enter first no also
using int 21h
 int 21h
 mov cx,0 ;we will call InputNo to handle our input as we will take each
number seprately
 call InputNo  ;first we will move to cx 0 because we will increment on it
later in 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 ;then we will use int 16h to read a key press
        mov dx,0
        mov bx,1
        cmp al,0dh ;the keypress will be stored in al so, we will comapre to  0d
        which represent the enter key, to know wheter he finished entering the
        number or not
         je FormNo ;if it's the enter key then this mean we already have our
          number stored in the stack, so we will return it back using FormNo
        sub ax,30h ;we will subtract 30 from the the value of ax to convert the
        value of key press from ascii to decimal
         call ViewNo ;then call ViewNo to view the key we pressed on the
        screen
         mov ah,0 ;we will mov 0 to ah before we push ax to the stack bec we
```

```
          only need the value in al
          push ax  ;push the contents of ax to the stack
          inc cx   ;we will add 1 to cx as this represent the counter for the number
          of digit
          jmp InputNo ;
          then we will jump back to input number to either take another number
          or press enter;we took each number separatly so we need to form our
          number and store in one bit for example if our number 235
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 ;we will push ax and dx to the stack because we will
           change    there values while viewing then we will pop them back from
          push dx ;the stack we will do these so, we don't affect their contents
             mov dx,ax ;we will mov the value to dx as interrupt 21h expect that
           the   output is stored in it
          add dl,30h ;add 30 to its value to convert it back to ascii
          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
```

EXPERIMENT RESULTS & ANALYSIS

Results

Calculator Home Page:

Addition:

Subtraction:

Division:

Result Analysis

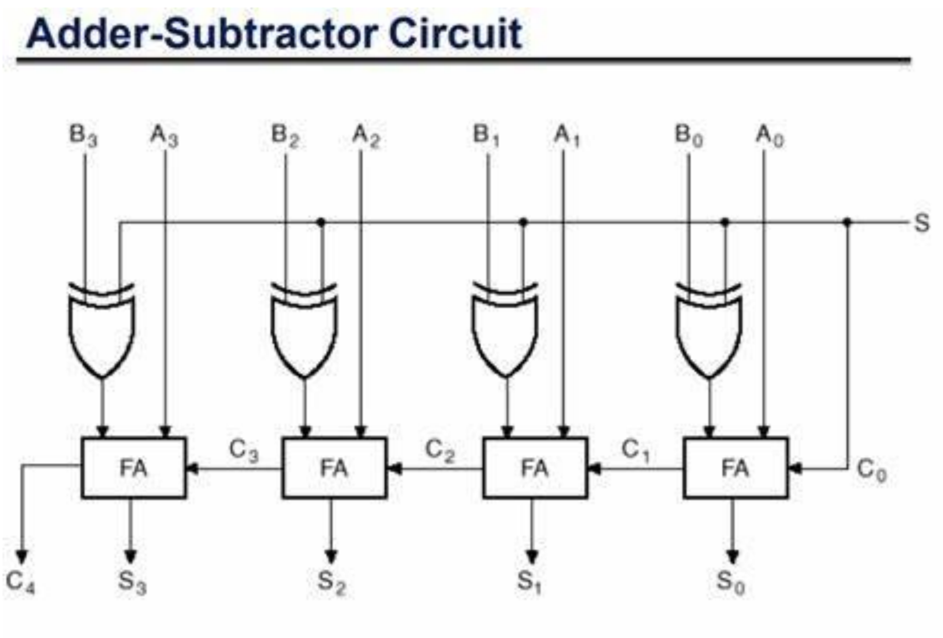
Addition Operation and Subtract Operation

With the frame properly implemented we can continue to implement the addition operation with logical operations. But first let’s cover the logic. In binary, addition needs to take in account a carry in/carry out(Ci/o), the first operand, and the second operand. The (Ci/o) is necessary when you are doing addition and it results in requiring another bit, for example 9 + 1 in decimal results in a Co of 1 and a sum of 0 in current index, resulting in 10 when adding the Co. So same as in decimal, in binary when you add 1 + 1 it requires a Co of 1 and a sum of 0 in that current index

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

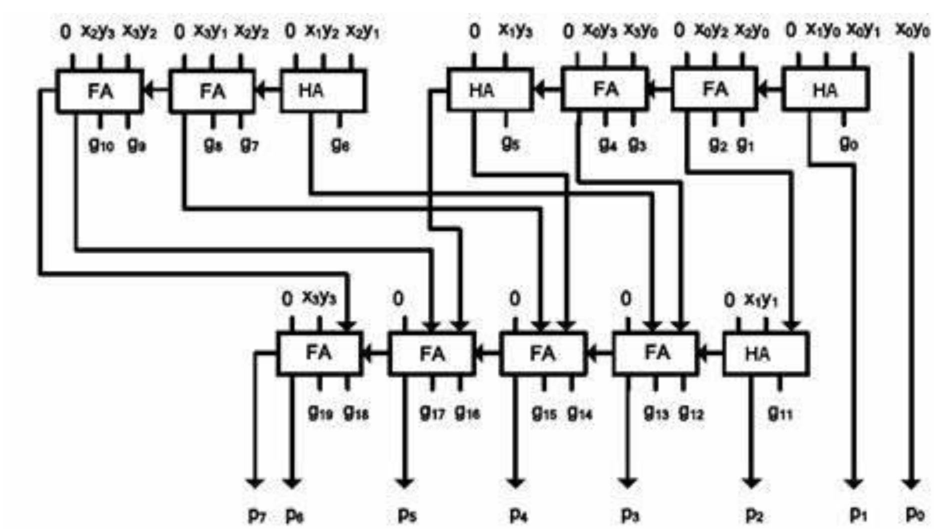
Binary Addition truth table

SIMPLIFICATION OF ADDER AND SUBTRACTOR



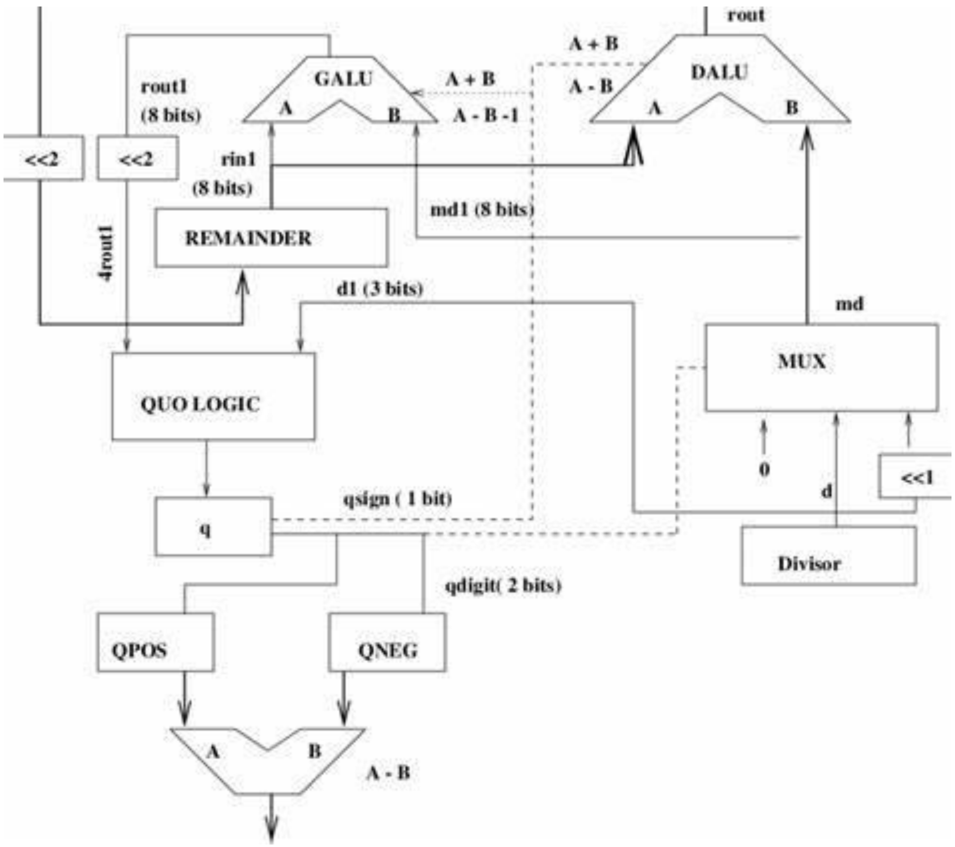
Multiplication Operation

We will first have to create a few helpful procedures, create unsigned multiplication and implement signed multiplication. The reason why we do unsigned multiplication first is because it is simple to check if the result of a signed multiplication should be negative or positive by XOR of the MSB in both the operands. A multiplication operation can be processed



Division Operation

With completion of the Addition, Subtraction and Multiplication operation. The theory for using Division is similar but instead of Addition and shifting to the right, we will now use Subtraction and shift to the left. The process is straightforward as simple division with decimals



CONCLUSION

Overall the Project gave me a better understanding of how:
Assembly language and 8086 Simulator Works This project reinforced almost all of the theories we learned in class and gave reinforced my base knowledge in the Computer Organization and Architecture.

REFERENCE

- Coa modules
- https://www.researchgate.net/publication/259293081_An_8-Bit_Scientific_Calculator_Based_Intel_8086_Virtual_Machine_Emulator
- www.google.com