ALU using 8086 Emulator

Submitted by :

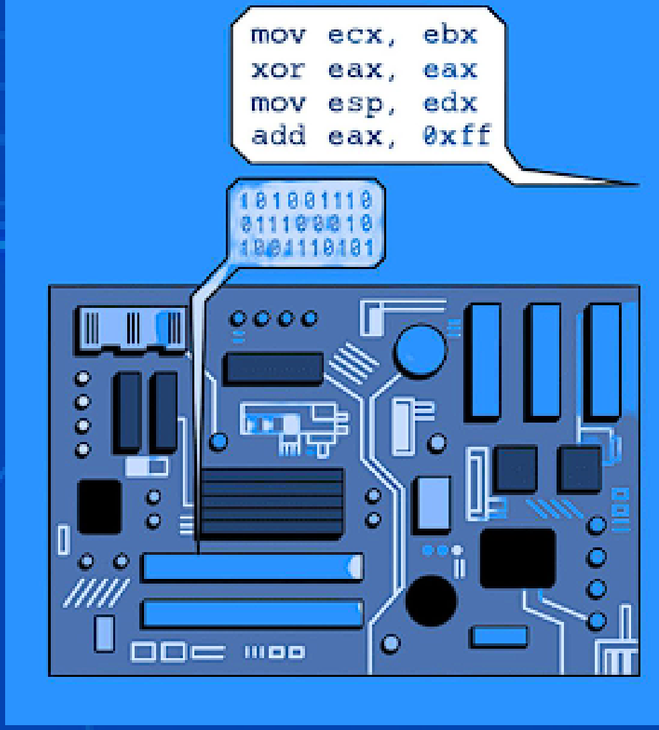
- Aakriti (21104001)
- Anupriya (21104019)
- Chetna (21104029)

Submitted to :

Dr. Kundan Kumar

Title:

A 16-bit arithmetic logic unit coded in Assembly Language which takes numbers as input and performs the selected operations and displays the result. Operations include +, -, x, /, %, and some Binary Operations



Objective:

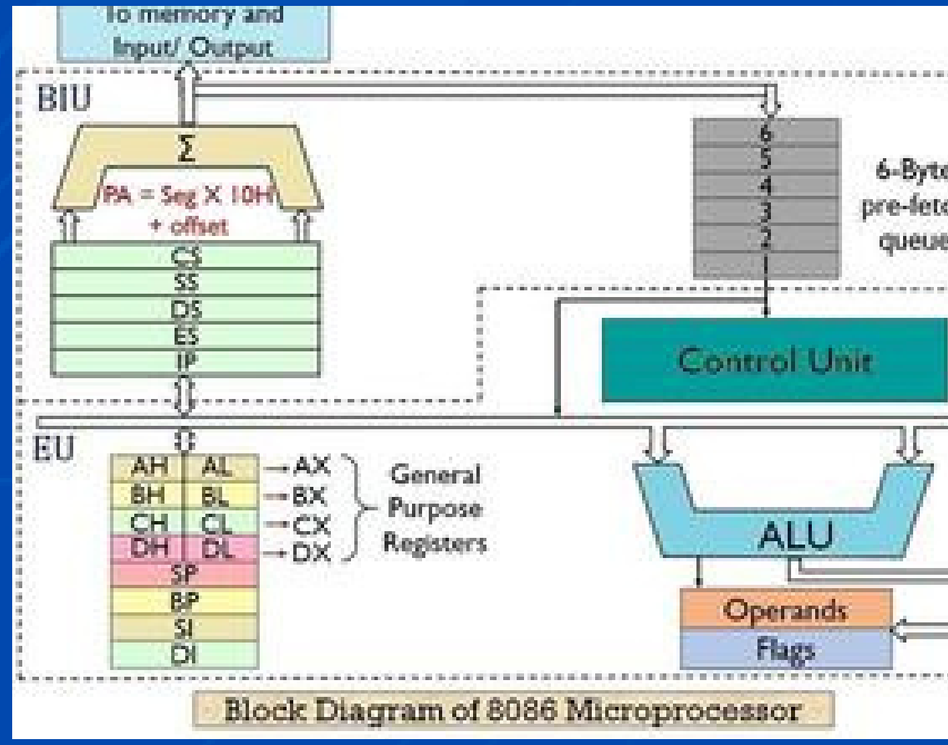
The goal of this project work is to utilize the concepts and knowledge of Assembly Language and use them in the Project. That is what we have learned so far and use the concepts of different Instructions in this project. And to understand the working of instructions and the code of Assembly Language using emulator EMU8086.

8086 ALU

8086 has a powerful set of registers known as general purpose registers and special purpose registers

The register set is categorized into four groups, as follows:

- General data registers – AX, BX, CX and DX used to store 16-bit data
- Segment registers – CS, DS , ES, SS (changes the memory address accessed by 16 bits at a time)
- Pointers and index registers – IP, BP and SP (used as general purpose registers as well as for offset storage in case of indexed)
- Flag register –its status according to the result stored in the accumulator.



Block Diagram of 8086 Microprocessor

Implementation:

We used EMU8086.inc Library which provides built in functions such as:

- print ‘STRING’ – For printing String
- scan_num – For taking Integer Input
- print_num – For displaying value stored in register
- printn – For printing New Line 0dh and 0ah as well as print10 and print13 can also be used to move to new line.

Interrupts used:

- INT 10h / AH = 0 - set video mode.

input:

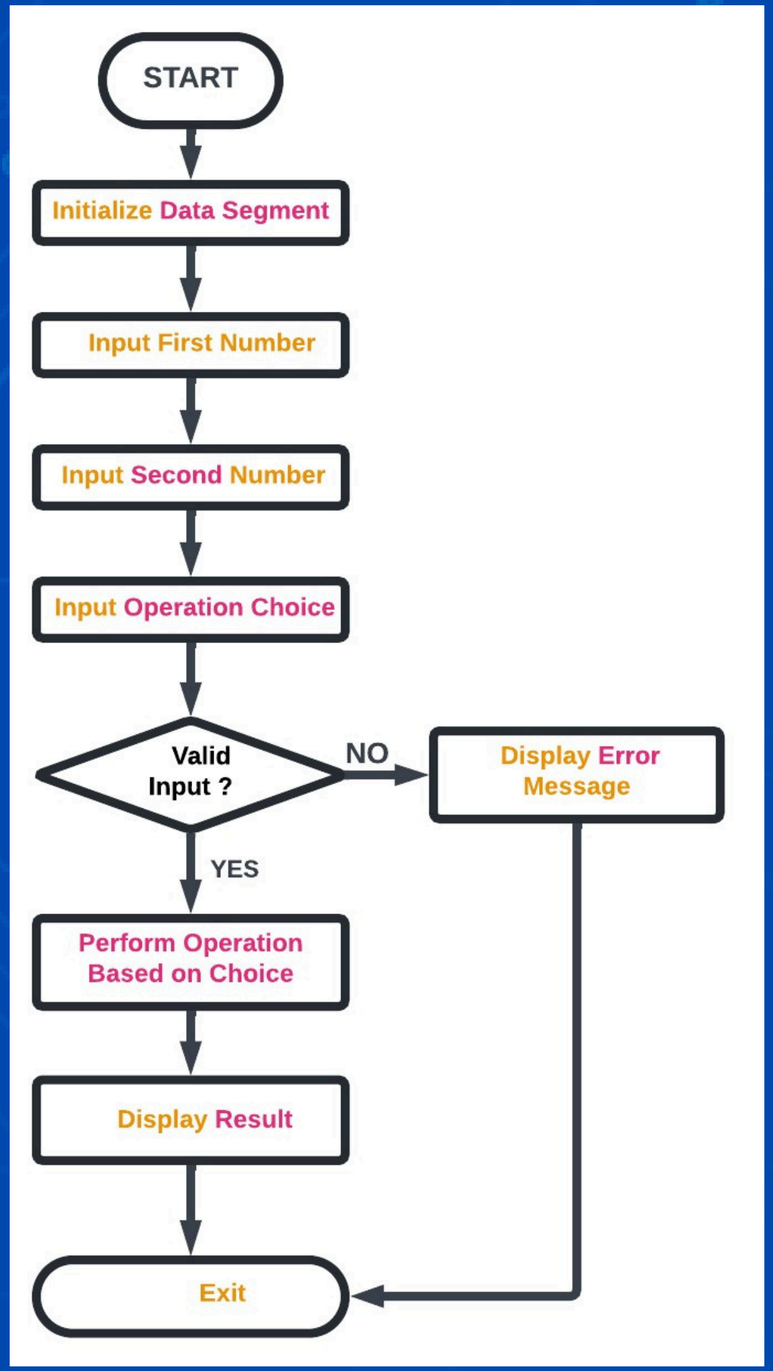
- AL = desired video mode.

These video modes are supported:

- 00h - text mode. 40 * 25 . 16 colors. 8 pages.
- 03h - text mode. 80 * 25 . 16 colors. 8 pages.
- 13h - graphical mode. 40 * 25 . 256 colors. 320 * 200 pixels. 1 page.
- INT 21h / AH = 9 - output of a string at DS:DX. String must be terminated by '$'.

Flowchart of ALU

Flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing.

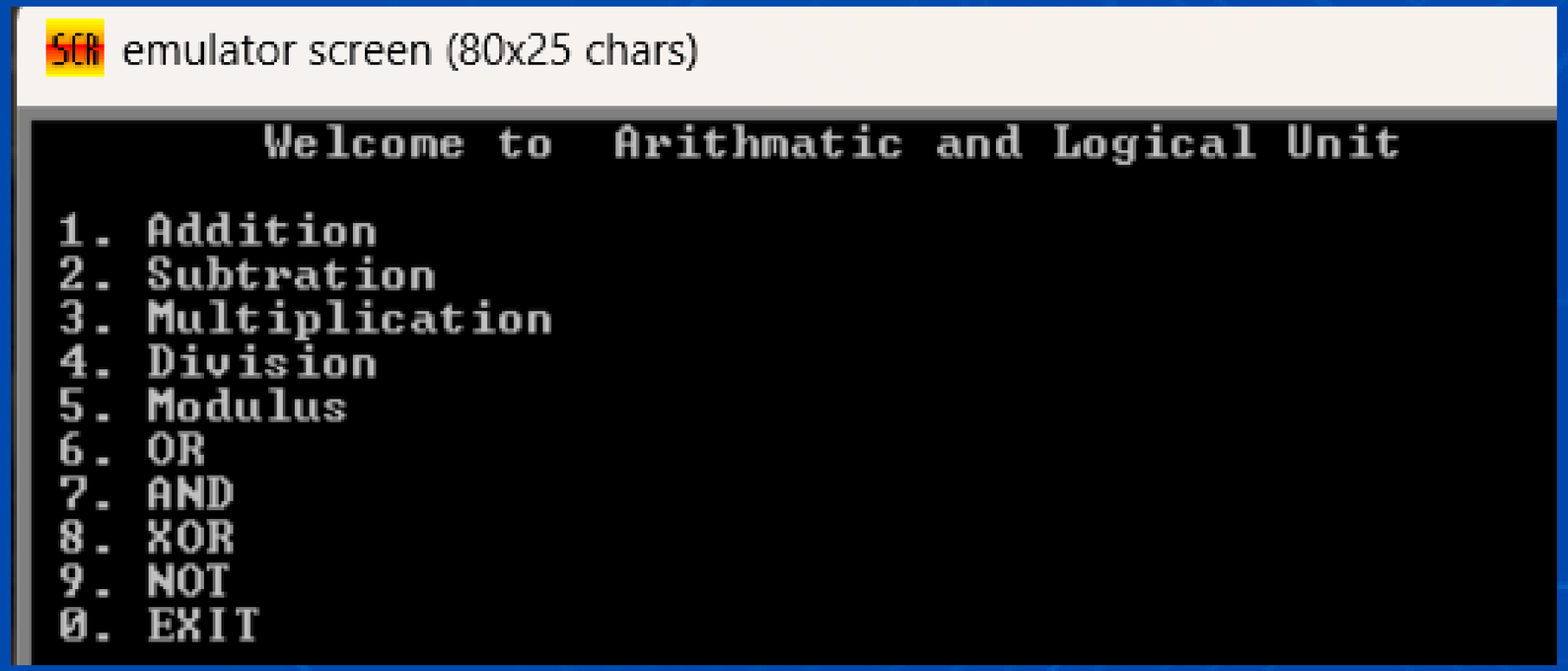


Functionality Expansion

The 8086 ALU offers extensive functionality, supporting 5'digits array of arithmetic and logical operations, including addition, subtraction, multiplication, division, bitwise AND, OR, and XOR etc. .

Debugging-Test-Run

When the user runs the program, a main menu will display as below in graphics mode.



The user will now input the operation number he would like to perform, such as 1 for addition, 2 for subtraction and so on,

Addition:

It involves fetching two operands, performing the addition operation, storing the result and updating flags to reflect the outcome.

```
Choose the Operation: 1

      Addition

Enter First Number: 22
Enter Second Number: 32

The SUM of two Numbers  = 54

Do you want to continue? (Yes  = 1 / No  = 0) :
```

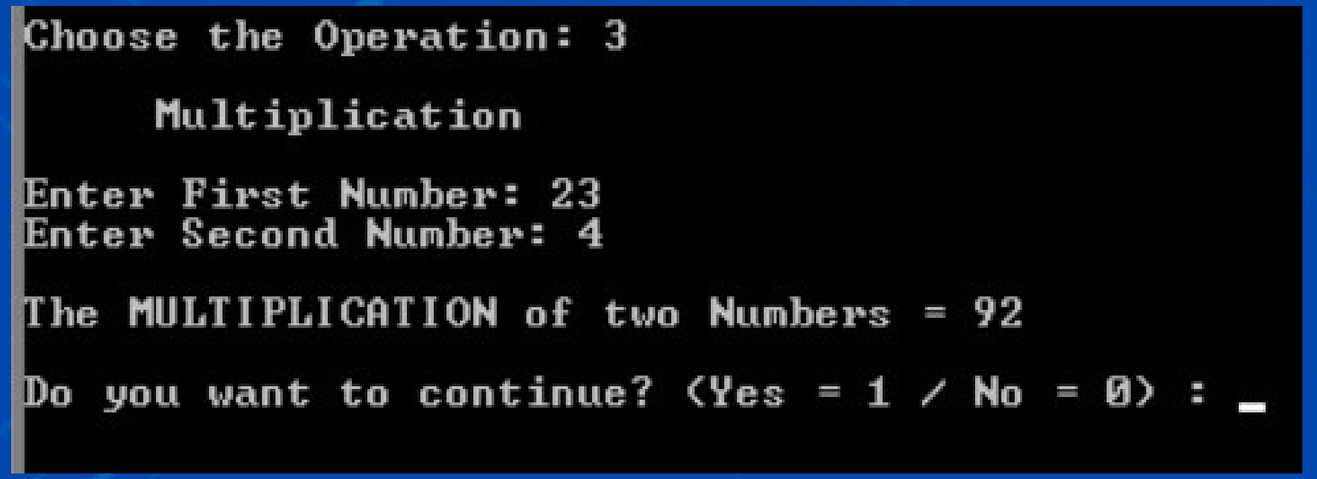
Subtraction:

It involves converting the subtrahend to its two's complement, adding it to the minuend, and updating flags to reflect the result, such as setting the Carry Flag (CF) if borrowing occurs.

```
Choose the Operation: 2

      Subtraction

Enter First Number: 34
Enter Second Number: 12
The SUBTRACTION of two Numbers  = 22

Do you want to continue? (Yes  = 1 / No  = 0) :
```

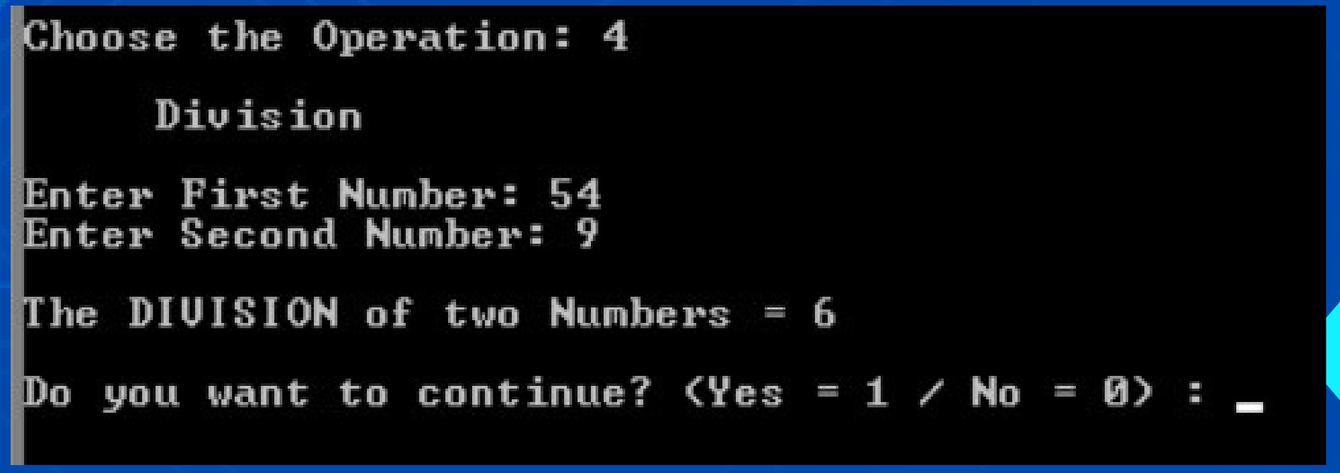
Multiplication:

It involves converting the subtrahend to its two's complement, adding it to the minuend, and updating flags to reflect the result, such as setting the Carry Flag (CF) if borrowing occurs.



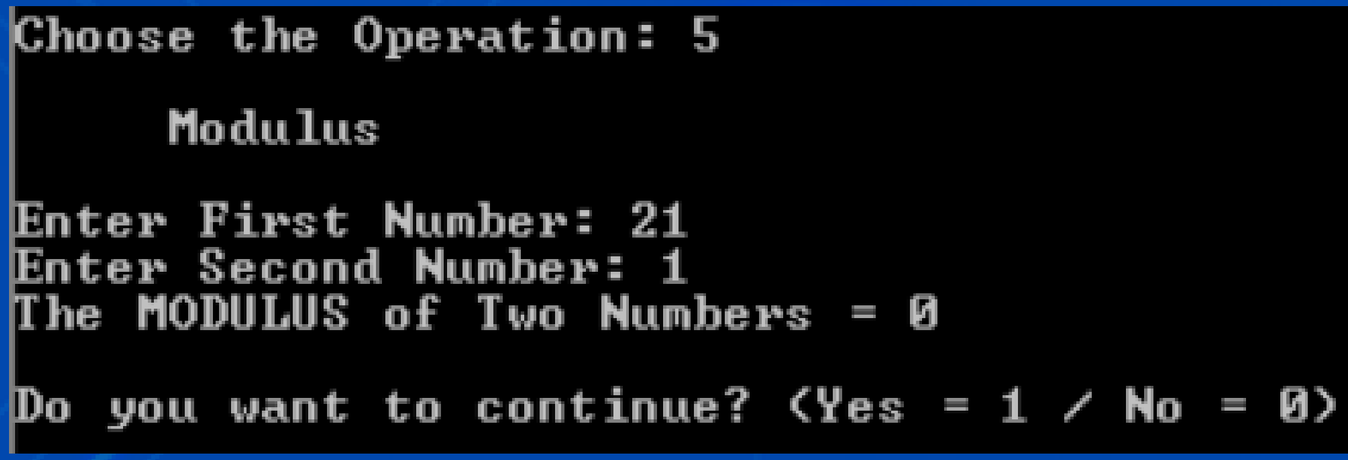
Division:

The division operation is executed using the DIV instruction, dividing the 16-bit dividend in the AX register by an 8-bit or 16-bit divisor, with the quotient stored in AL and the remainder in AH.



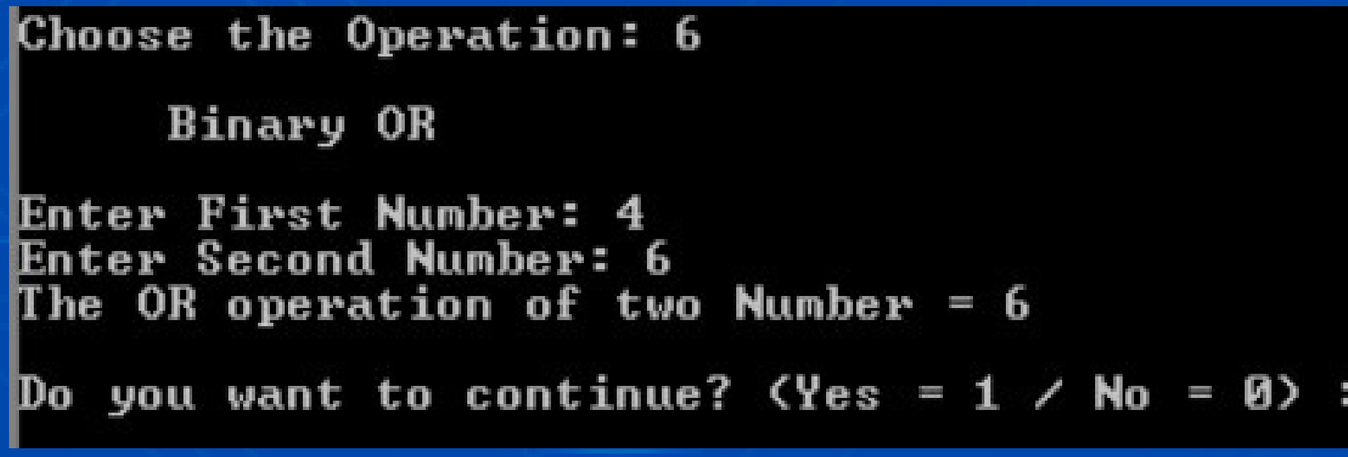
Modulus:

It computes the remainder of a division operation, often used for tasks like finding odd or even numbers in assembly language programming. It involves the DIV instruction followed by retrieval of the remainder from the DX register.



Binary Or:

The OR operation performed using the OR instruction, combining two operands by setting each bit in the result to 1 if either corresponding bit in the operands is 1; otherwise, it sets the bit to 0.



Binary And:

AND operation performs bitwise AND between operands using the AND instruction, setting each bit in the result to 1 only if both corresponding bits in operands are 1; otherwise, it sets the bit to 0.

```
Choose the Operation: 7

     Binary AND

Enter First Number: 12
Enter Second Number: 5

The AND operation of two  Number = 4
```

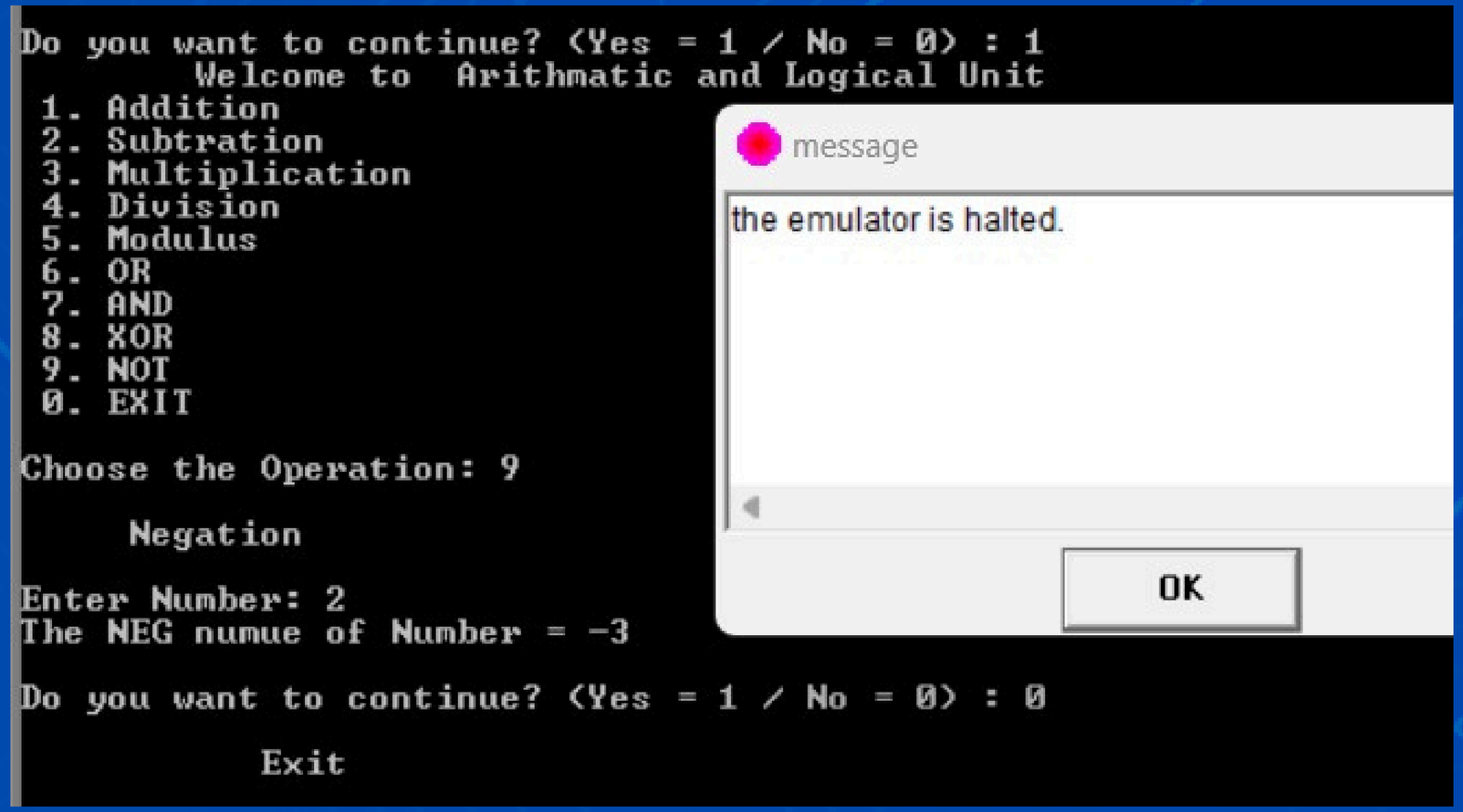
Binary Xor:

The XOR operation performed using the XOR instruction, combining two operands by setting each bit in the result to 1 if the corresponding bits in the operands are different; otherwise, it sets the bit to 0.

```
Choose the Operation: 8

     Binary XOR

Enter First Number: 12
Enter Second Number: 21

The XOR operation of two Number = 25
```

Negation:

The negation operation is performed using the NEG instruction, which changes the sign of the operand by taking its two's complement and then adding 1 to it, updating the appropriate flags accordingly.



Limitations

- **Data Handling Limitations:**
    - *Limited Data Size*: The project likely supports only 16-bit integers. This restricts the range of numbers it can handle. Real-world applications often require calculations with larger data types (32-bit, 64-bit).
    - *Lack of Floating*-Point Support: The project might not handle floating-point numbers, which are crucial for calculations involving decimals.
- **Functionality Limitations:**
    - Limited Operation Set: The implemented arithmetic and logical operations might be a basic set (addition, subtraction, multiplication, etc.). It might lack more complex mathematical functions like trigonometry, logarithms, or exponentiation.

Future scope

- **Expand Data Handling Capabilities**:
  - *Increase data size*: Currently, the project seems to handle 16-bit integers. You could explore extending it to support 32-bit or even 64-bit integers for wider numeric ranges.
  - *Introduce Floating-Point Support:* Implement calculations for floating-point numbers, allowing for operations with decimals.
- **Enhance Functionality**:
  - *Add more complex operations*: Include advanced mathematical functions like trigonometry (sin, cos, tan), logarithms, or exponentiation.
  - *Implement bitwise operations*: Explore bit shifting, rotation, and other bit manipulation instructions for low-level programming tasks.

Conclusion

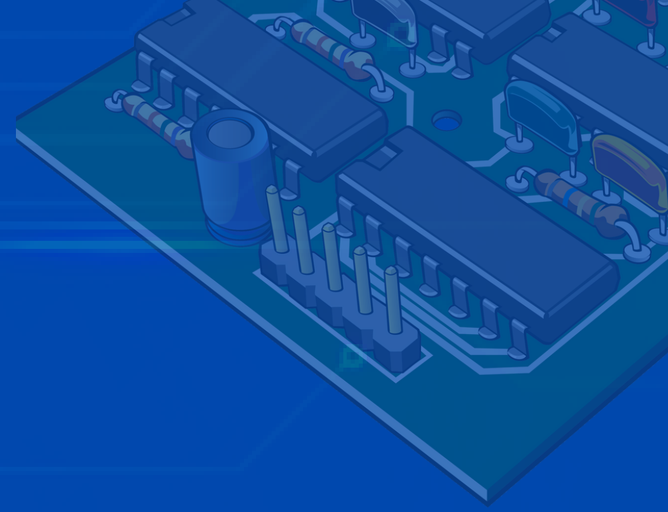
In conclusion, this project successfully developed a functional 16-bit ALU using 8086 assembly language and the EMU8086 emulator. The ALU offers basic arithmetic and logical operations, providing a valuable platform to explore assembly language concepts and 8086 architecture.

However, there is scope for future advancements. Expanding data handling capabilities to include larger data types (32-bit, 64-bit) or floating-point numbers would broaden its applicability. Additionally, incorporating more complex mathematical functions or bitwise operations would enhance its functionality.

References :-

1. https://www.philadelphia.edu.jo/academics/qhamarsheh/uploads/emu8086.pdf

2. https://yassinebridi.github.io/asm-docs/



THANK YOU