Course Title: **Computer Architecture**

Course Code: **CSE360**

Section-**3**

Project Title:

Build a system with the fewest possible instructions such that it can perform the fundamental logic and arithmetic operations.

Submitted To

Dr. Md. Nawab Yousuf Ali
Professor, Department of computer science and engineering
East West University

Submission Date:03.01.2023

Build a system with the fewest possible instructions such that it can perform the fundamental logic and arithmetic operations.

Fatema Akter
(ID: 2020-1-60-115)
Department of Computer Science and Engineering
East West University,
Dhaka, Bangladesh

Abdullah al Tamim
(ID: 2020-1-60-127)
Department of Computer Science and Engineering
East West University,
Dhaka, Bangladesh

Abin Roy
(ID: 2020-1-60-141)
Department of Computer Science and Engineering
East West University,
Dhaka, Bangladesh

Eva Islam
(ID:2020-1-60-273)
Department of Computer Science and Engineering
East West University,
Dhaka, Bangladesh

***Abstract: In* this paper we have discussed how we can construct a processor with the fewest possible instructions such that it can perform the fundamental logic and arithmetic operationsusing with C++ language. In this project, we tried to make a processor with some function like for arithmetic operations, it will be able to do additions, subtractions, multiplications, and divisions. And for logical operations, our processor will be able to do OR, AND, NOT, XOR, XNOR, NAND, and NOR.In our processor, we will extend some more arithmetic operations features like Mod,Power,Root,Logarithm ,Trigonometry.**

***Keywords*:additions,subtractions,multiplications,divisions,OR,AND,NOT,XOR,XNOR,NAND,NOR,Mod,Power,Root,Logarithm,Trigonometry,Infix, Postfix,Ascending and Descending sort.**

I. Introduction

The arithmetic instructions define the set of operations performed by the processor Arithmetic Logic Unit (ALU).

code instructs the ALU what operation to carry out.

For instance, two operands could be logically compared or added together. The format and opcode can be mixed, and it indicates things like whether an instruction is fixed-point or floating-point.The output comprises settings that show if the operation was successful as well as a result that is written to a storage The arithmetic instructions are further classified into binary, decimal, logical, shift/rotate, and bit/byte manipulation instructions.The binary arithmetic instructions perform basic binary integer computations on byte, word, and double word integers located in memory and/or the general-purpose registers.

The component of a central processing unit known as an arithmetic-logic unit performs arithmetic and logic operations on the operands in computer instruction words.The ALU in certain processors is split into an arithmetic unit (AU) and a logic unit (LU). One AU may be for fixed-point operations while another may be for floating-point operations in some CPUs.

In computer systems, a floating-point unit (FPU) on a separate chip known as a numeric coprocessor will occasionally do floating-point calculations.An instruction word, also known as a machine instruction word, makes up the input. It contains an operation code, often known as an "opcode," one or more operands, and occasionally a format code. The operands are used in the operation, and the operation.

II. RELETED WORKS

ALUs are a type of combinational digital circuit used in computing to perform arithmetic and bitwise operations on binary values. This serves as a fundamental building block for arithmetic logic circuits in many different kinds of control and computing units, including as a central register. If it isn't, a status of some kind will be permanently kept in what is referred to as the machine status word.

The ALU typically has storage areas for input operands, adding operands, the accumulated result (stored in an accumulator), and results that have been shifted. Gated circuits regulate the movement of bits and the operations that are carried out on them in the ALU's subunits.

A sequence logic unit that employs a specific algorithm or sequence for each operation code controls the gates in these circuits.Multiplication and division in the arithmetic unit are accomplished using a series of adding, subtracting, and shifting operations.Negative numbers can be expressed in a variety of ways. One of the 16 possible logic operations can be carried out in the logic unit, such as comparing two operands and determining which bits don't match.

A crucial component of the processor's architecture is the ALU design, and new strategies for accelerating instruction handling are always being created by any number of bits in a single operation; complex ALUs use barrel shifters.

number of 20 while 2+3*4 should, according to the conventional calculator, equal 14.

Because it simply multiplies the two operands with the operator sandwiched in between, a simple calculator gets it completely wrong. We simply cannot place the blame for this incorrect result on the simple calculator. After all, simplicity was the whole point.
Our primary objective is to rearrange the operators to produce the desired effect. In order to do that, we also need to be familiar with two other computer science concepts: infix expression and postfix expression. The mathematical expression that the computer understands is the Postfix expression, whereas we understand the Infix expression. Any mathematical expression will have the correct output if we use the Infix to Postfix technique.

III. PROPOSED WORK

We are all familiar with the order of operators: addition and subtraction have the processing units (CPUs), FPUs, and graphics processing units.ALUs first assisted in supporting microprocessors and transistors in the 1970s, a long time before current PCs.
Basic arithmetic operations and bitwise logical operations enabled by ALUs include the following:

- Addition: incorporates A and B into Y's carry-in or carry-out sum.
- Subtraction: Y and carry-in or carry-out the difference after subtracting B from A or vice versa.
- Increment: where Y is the new value and A or B are both increased by one.
- Decrement: where Y is the new value and A or B are both reduced by one.
- AND: Y represents the bitwise logic AND of A and B.
- OR: Y represents the bitwise logic OR of A and B.
- Exclusive-O:. Y represents the bitwise XOR of the two variables A and B.

ALU shift functions cause the right or left shifting of A or B operands, with Y serving as the new operand. To shift A or B operands

A project to construct a multi-function calculator with arbitrary precision is currently under development. With the help of this project, we will develop a calculator that performs a variety of operations, including addition, subtraction, multiplication, division, square roots, and percentage calculations. With this project,the same priority as division, whereas multiplication has a greater priority. Most calculations made using a basic calculator are incorrect. For instance, the simple calculator returns a

string. We can use a common expression to create our paintings.

- Postfix Expression

  We proposed four characteristics for a 4-characteristic calculator based on postfix expression: addition, subtraction, multiplication, and division. The programmer turns characters to digits and pushes them so we can accomplish our objective. With the help of this project, we can simply carry out calculations more quickly. For instance, we may compute the sum, root, percentage, etc., of the values of A and B. Overall, we can state that the following tasks will be completed more quickly in our project.

- Infix Expression:

An initial string and a stack were suggested based on the entered expression. Any other string can be used to convert a saved enter string to postfix. The program examines the first string separately and uploads any digits it finds inside the second string. However, if the programmer discovers operators, then ships them inside the stack if the stack's pinnacle operator's priority is lowered. Otherwise, the new operator will be sent. If not, the stack's top detail might be transmitted inside the 2D string and emerge from the stack as well. Once more, as the first brace approaches, this detail may be driven into the stack, and as the primary brace's termination approaches, each operator may emerge from the stack until the first brace may emerge. The stack will pop the operators' relaxation from the top of the stack after scanning the general first wrong equations can be used. Push the stack to create it last. The outcome might be the stack's crowning detail.

We implemented another two number operation. Like, Summation, Division, Subtraction, Multiplication, Square, Root and percentage.within the stack by scanning each solitary postfix string with the aid of one. If someone is an operator, consider whether they fit our four criteria to determine which paintings will work.

- Addition:

  The new feature will upload and pop numbers from the top of the stack. Push the stack to create it last.

- Subtraction:

  The subtraction functionality will remove digits from the top of the stack and deduct the first digit from the last digit removed. Push the stack to create it last.

- Multiplication:

  This function will take the highest number in the stack and multiply it. Push the stack to create it last.

- Division:

  If the first digit that is popped is less than the last digit that is popped, this feature will pop digits from the top of the stack and divide. Otherwise, the wrong equation can be used. Push the stack to create it last.

IV. EXPERIMENTAL RESULT

After opening this program,the program will ask the user what they want to do. Let's see the result of this program.

- *Software*

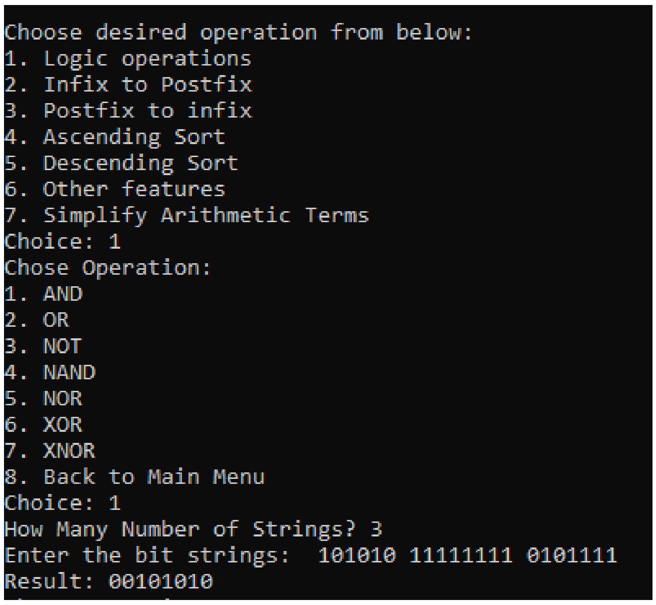
  1. Software: Codeblocks

  2. Language: C++

As part of this project, we created a straightforward calculator in C++. This project was a complete success. The Code Block finds the file project.cpp first when we wish to open this project, and then it enters the following code. The complexity in time and space is decreased by this effort. The space complexity is always O in the worst scenario (1). However, the temporal complexity is O in the worst scenario (n).

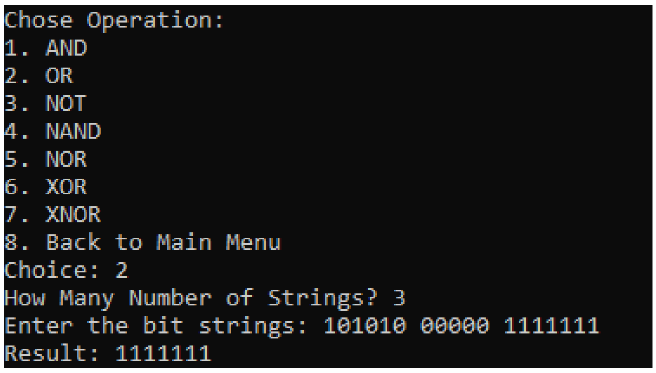
1. Logical operations

a. And operation



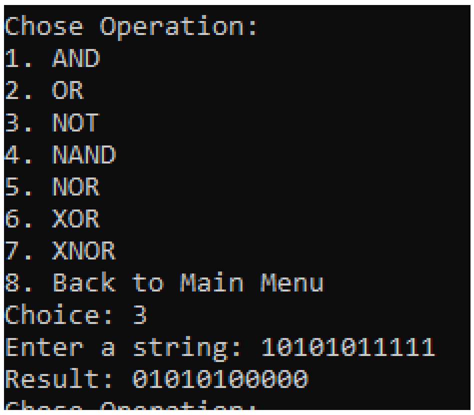
```
Choose desired operation from below:
1. Logic operations
2. Infix to Postfix
3. Postfix to infix
4. Ascending Sort
5. Descending Sort
6. Other features
7. Simplify Arithmetic Terms
Choice: 1
Chose Operation:
1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. XNOR
8. Back to Main Menu
Choice: 1
How Many Number of Strings? 3
Enter the bit strings:  101010 11111111 0101111
Result: 00101010
```

b. OR operation



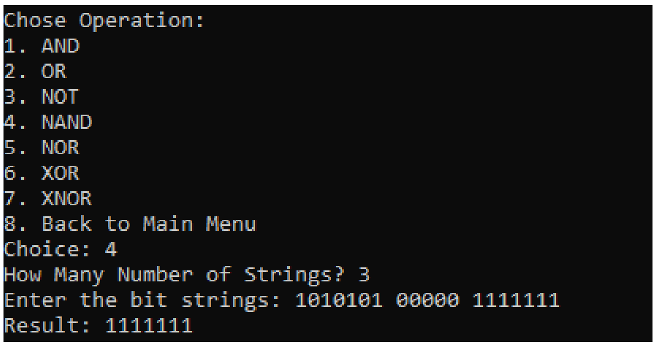
```
Chose Operation:
1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. XNOR
8. Back to Main Menu
Choice: 2
How Many Number of Strings? 3
Enter the bit strings: 101010 00000 1111111
Result: 1111111
```

c. NOT operation



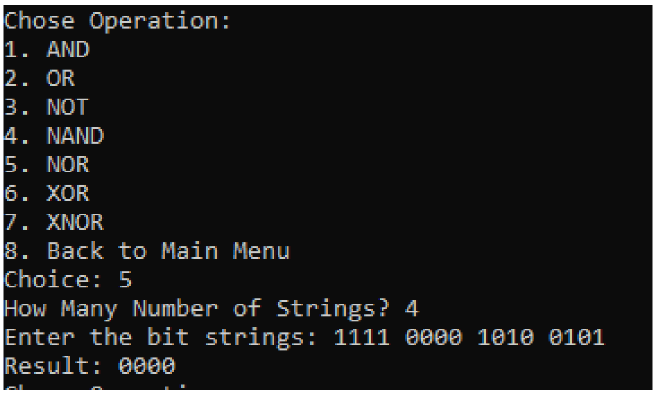
```
Chose Operation:
1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. XNOR
8. Back to Main Menu
Choice: 3
Enter a string: 10101011111
Result: 01010100000
```

d. NAND operation



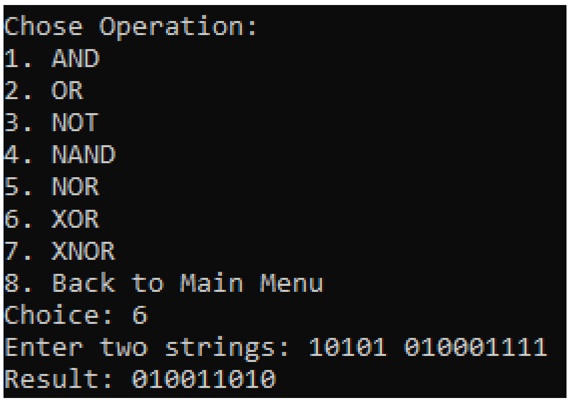
```
Chose Operation:
1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. XNOR
8. Back to Main Menu
Choice: 4
How Many Number of Strings? 3
Enter the bit strings: 1010101 00000 1111111
Result: 1111111
```

e. NOR operation



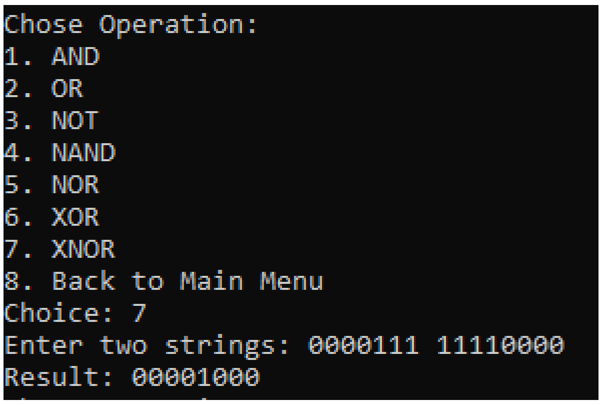
```
Chose Operation:
1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. XNOR
8. Back to Main Menu
Choice: 5
How Many Number of Strings? 4
Enter the bit strings: 1111 0000 1010 0101
Result: 0000
```

f. XOR operation



```
Chose Operation:
1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. XNOR
8. Back to Main Menu
Choice: 6
Enter two strings: 10101 010001111
Result: 010011010
```

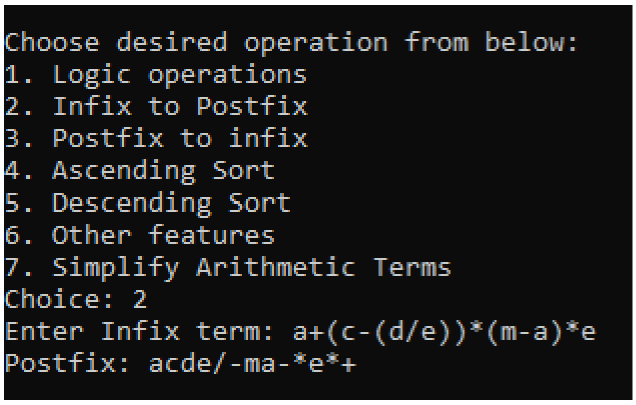
g. XNOR operation



```
Chose Operation:
1. AND
2. OR
3. NOT
4. NAND
5. NOR
6. XOR
7. XNOR
8. Back to Main Menu
Choice: 7
Enter two strings: 0000111 11110000
Result: 00001000
```

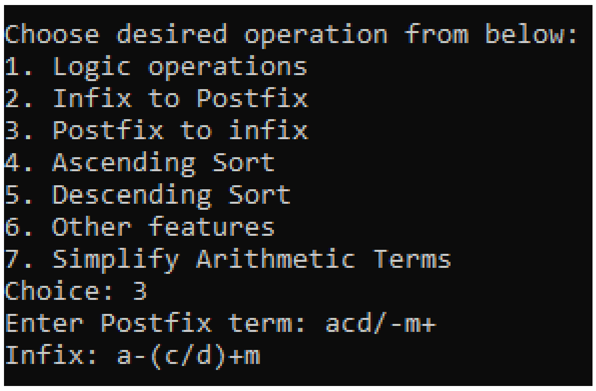
2. Arithmetic operation

a. Infix to postfix



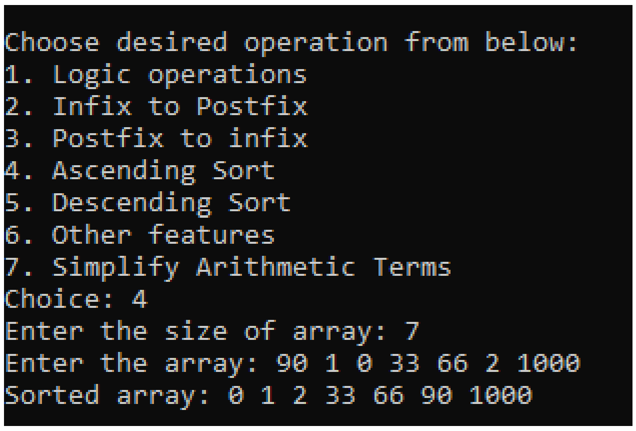
```
Choose desired operation from below:
1. Logic operations
2. Infix to Postfix
3. Postfix to infix
4. Ascending Sort
5. Descending Sort
6. Other features
7. Simplify Arithmetic Terms
Choice: 2
Enter Infix term: a+(c-(d/e))*(m-a)*e
Postfix: acde/-ma-*e*+
```

b. Postfix to infix



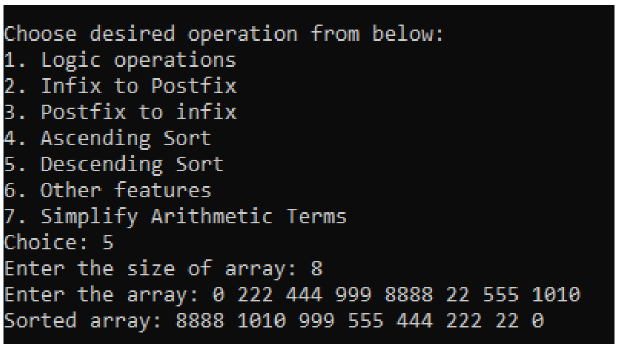
```
Choose desired operation from below:
1. Logic operations
2. Infix to Postfix
3. Postfix to infix
4. Ascending Sort
5. Descending Sort
6. Other features
7. Simplify Arithmetic Terms
Choice: 3
Enter Postfix term: acd/-m+
Infix: a-(c/d)+m
```

c. Ascending sort



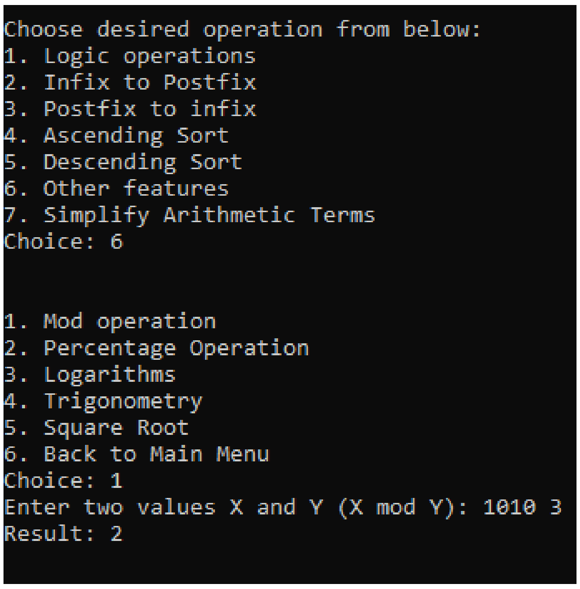
```
Choose desired operation from below:
1. Logic operations
2. Infix to Postfix
3. Postfix to infix
4. Ascending Sort
5. Descending Sort
6. Other features
7. Simplify Arithmetic Terms
Choice: 4
Enter the size of array: 7
Enter the array: 90 1 0 33 66 2 1000
Sorted array: 0 1 2 33 66 90 1000
```

d. Descending sort

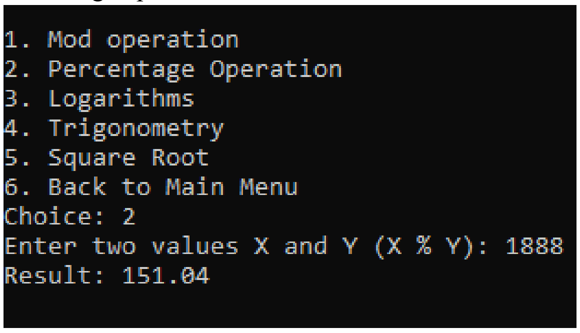

```
Choose desired operation from below:
1. Logic operations
2. Infix to Postfix
3. Postfix to infix
4. Ascending Sort
5. Descending Sort
6. Other features
7. Simplify Arithmetic Terms
Choice: 5
Enter the size of array: 8
Enter the array: 0 222 444 999 8888 22 555 1010
Sorted array: 8888 1010 999 555 444 222 22 0
```

e. Other features

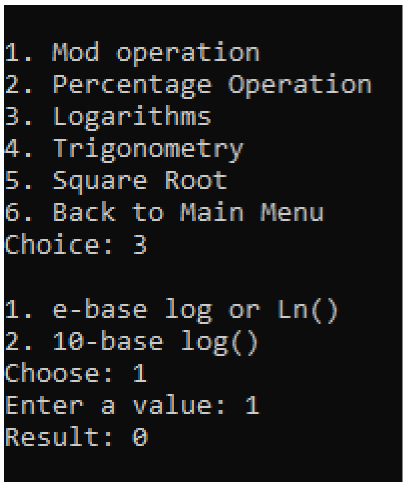
- Mod


```
Choose desired operation from below:
1. Logic operations
2. Infix to Postfix
3. Postfix to infix
4. Ascending Sort
5. Descending Sort
6. Other features
7. Simplify Arithmetic Terms
Choice: 6


1. Mod operation
2. Percentage Operation
3. Logarithms
4. Trigonometry
5. Square Root
6. Back to Main Menu
Choice: 1
Enter two values X and Y (X mod Y): 1010 3
Result: 2
```

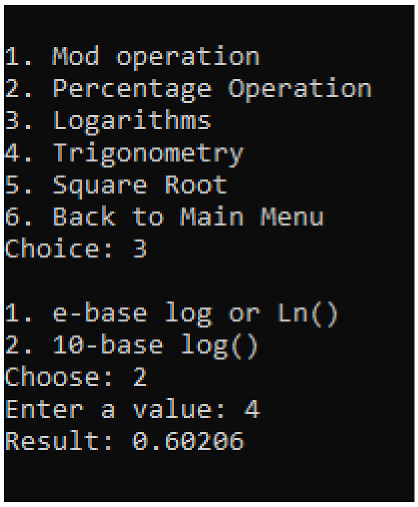
- Percentage operation


```
1. Mod operation
2. Percentage Operation
3. Logarithms
4. Trigonometry
5. Square Root
6. Back to Main Menu
Choice: 2
Enter two values X and Y (X % Y): 1888
Result: 151.04
```

- Logarithms

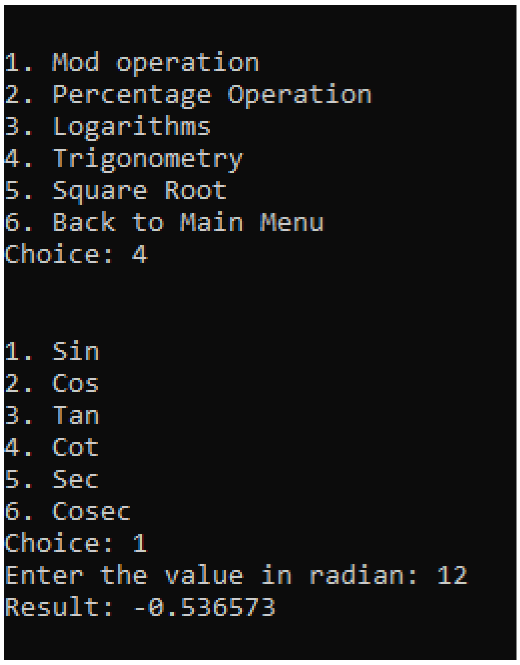

```
1. Mod operation
2. Percentage Operation
3. Logarithms
4. Trigonometry
5. Square Root
6. Back to Main Menu
Choice: 3

1. e-base log or Ln()
2. 10-base log()
Choose: 1
Enter a value: 1
Result: 0
```

#(e-base log or ln())

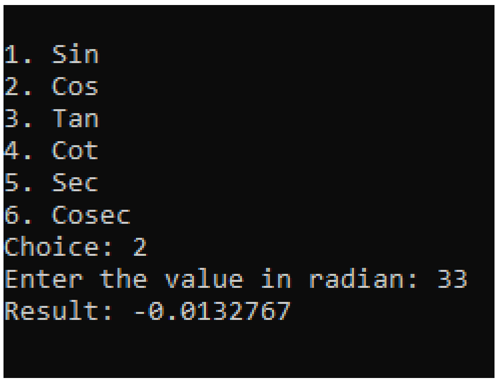

```
1. Mod operation
2. Percentage Operation
3. Logarithms
4. Trigonometry
5. Square Root
6. Back to Main Menu
Choice: 3

1. e-base log or Ln()
2. 10-base log()
Choose: 2
Enter a value: 4
Result: 0.60206
```

# (10-base log())

- Trigonomerty

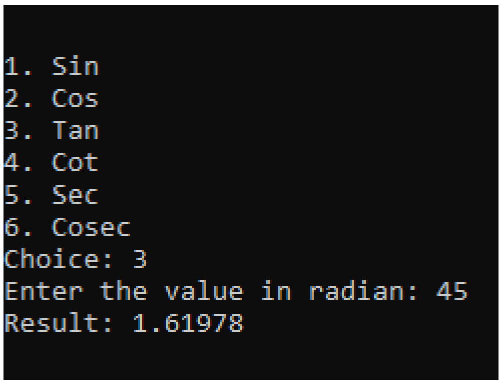

```
1. Mod operation
2. Percentage Operation
3. Logarithms
4. Trigonometry
5. Square Root
6. Back to Main Menu
Choice: 4


1. Sin
2. Cos
3. Tan
4. Cot
5. Sec
6. Cosec
Choice: 1
Enter the value in radian: 12
Result: -0.536573
```

#Sin



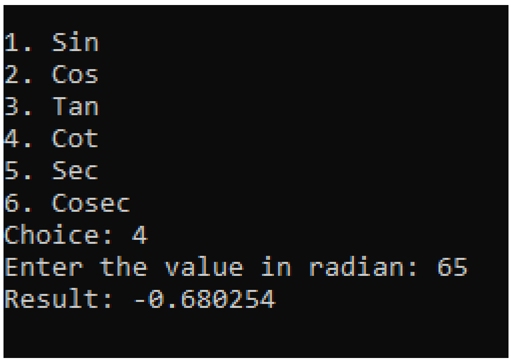
```
1. Sin
2. Cos
3. Tan
4. Cot
5. Sec
6. Cosec
Choice: 2
Enter the value in radian: 33
Result: -0.0132767
```

#Cos



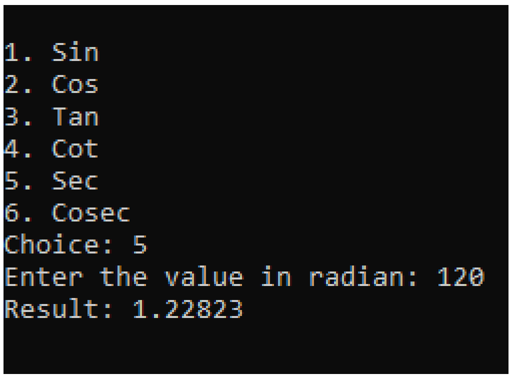
```
1. Sin
2. Cos
3. Tan
4. Cot
5. Sec
6. Cosec
Choice: 3
Enter the value in radian: 45
Result: 1.61978
```

#Tan



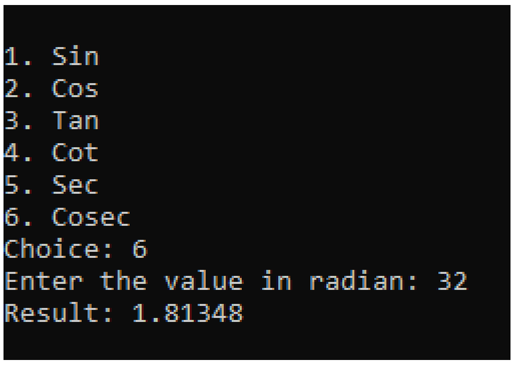
```
1. Sin
2. Cos
3. Tan
4. Cot
5. Sec
6. Cosec
Choice: 4
Enter the value in radian: 65
Result: -0.680254
```

#Cot



```
1. Sin
2. Cos
3. Tan
4. Cot
5. Sec
6. Cosec
Choice: 5
Enter the value in radian: 120
Result: 1.22823
```

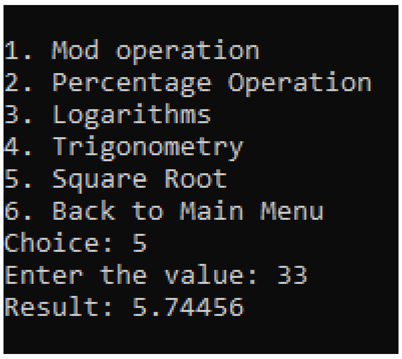
#Sec



```
1. Sin
2. Cos
3. Tan
4. Cot
5. Sec
6. Cosec
Choice: 6
Enter the value in radian: 32
Result: 1.81348
```

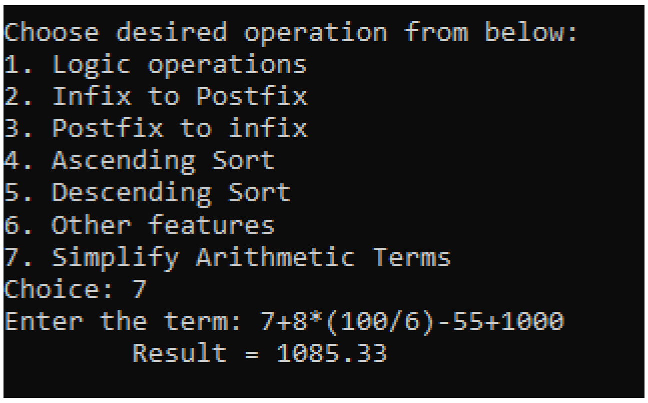
#Cosec

- Square root



```
1. Mod operation
2. Percentage Operation
3. Logarithms
4. Trigonometry
5. Square Root
6. Back to Main Menu
Choice: 5
Enter the value: 33
Result: 5.74456
```

f. Simplification



```
Choose desired operation from below:
1. Logic operations
2. Infix to Postfix
3. Postfix to infix
4. Ascending Sort
5. Descending Sort
6. Other features
7. Simplify Arithmetic Terms
Choice: 7
Enter the term: 7+8*(100/6)-55+1000
        Result = 1085.33
```

V. CONCLUSION

Utilizing this CPU is really easy. We made every effort to fix any bug that surfaced while writing this code for our project. In conclusion, we were able to make the code entirely functional without making a lot of mistakes.

This project was completed entirely in C++. Simple mathematical operations including addition, subtraction, multiplication, division, square roots, and percentages can be calculated using it for this project. However, the main drawback of our approach is the inability to calculate any binary number operation. This represented our whole project effort.

VI. References

1. https://www.techtarget.com/whatis/definition/arithmetic-logic-unit-ALU
2. https://www.bartleby.com/questions-and-answers/develop-a-processor-with-the-fewest-possible-instructions-that-can-perform-fundamental-arithmetic-an/ab4b9ac9-09fb-42be-a46b-46301aa94b0b
3. https://www.sciencedirect.com/topics/computer-science/arithmetic-instruction

VII. APPENDIX

- Code:

```
 //#include <iostream>
#include <bits/stdc++.h>
using namespace std;
#define maxx 100000000
double operation(double a, double b, char op)
{
   switch (op)
   {
   case '+':
      return a + b;
   case '-':
      return a - b;
   case '*':
      return a * b;
   case '/':
      return a / b;
   default:
      return pow(a, b);
   }
}

int precedence(char c)
{
   if (c == '^')
      return 3;
   else if (c == '/' || c == '*')
      return 2;
   else if (c == '+' || c == '-')
      return 1;
   else
      return -1;
}

double simplify(string term)
{
   stack<double> operand;
   stack<double> opertor;
   for (int i = 0; i < term.length(); i++)
   {
      if (term[i] == ' ')
      {
         continue;
      }
      else if (isdigit(term[i]))
      {
         // determining whether the number is double or
contains multiple digits
         string temp = "";
         while (i < term.length() and (isdigit(term[i]) || term[i]
== '.'))
         {
            temp += term[i];
            i++;
         }
         operand.push(std::stod(temp));
         i--;
      }
      else if (term[i] == ')')
      {
         while (!opertor.empty() and opertor.top() != '(')
         {
            double oprnd1 = operand.top();
            operand.pop();

            double oprnd2 = operand.top();
            operand.pop();

            char op = opertor.top();
            opertor.pop();
            // cout << "T: " << oprnd1 << " " << oprnd2 << "
" << op << endl;
            operand.push(operation(oprnd2, oprnd1, op));
         }
         if (opertor.top() == '(')
         {
            opertor.pop();
         }
      }
      else if (term[i] == '(')
      {
         opertor.push(term[i]);
      }
      else
      {

         while (!opertor.empty() &&
precedence(opertor.top()) >= precedence(term[i]))
         {
            double operand1 = operand.top();
            operand.pop();

            double operand2 = operand.top();
            operand.pop();

            char op = opertor.top();
            opertor.pop();

            operand.push(operation(operand2, operand1, op));
         }

         opertor.push(term[i]);
      }
   }
```

```
    while (!opertor.empty())
    {
        double oprnd1 = operand.top();
        operand.pop();

        double oprnd2 = operand.top();
        operand.pop();

        char op = opertor.top();
        opertor.pop();
        // cout << "T: " << oprnd1 << " " << oprnd2 << " " <<
op << endl;
        operand.push(operation(oprnd2, oprnd1, op));
    }
    return operand.top();
}

string InToPost()
{
    string s;
    cout << "Enter Infix term: ";
    cin >> s;
    stack<char> st;
    string res;

    for (int i = 0; i < s.length(); i++)
    {
        if ((s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <=
'Z'))
        {
            res += s[i];
        }
        else if (s[i] == '(')
        {
            st.push(s[i]);
        }
        else if (s[i] == ')')
        {
            while (!st.empty() && st.top() != '(')
            {
                res += st.top();
                st.pop();
            }
            if (!st.empty())
            {
                st.pop(); // Popping '(' here
            }
        }
        else
        {
            while (!st.empty() && precedence(st.top()) >=
precedence(s[i]))
            {
                res += st.top();
                st.pop();
            }
            st.push(s[i]);
        }
    }

    while (!st.empty())
    {
        res += st.top();
        st.pop();
    }

    return res;
}

string PostToIn()
{
    string term;
    cout << "Enter Postfix term: ";
    cin >> term;

    stack<string> stack;

    for (int i = 0; i < term.length(); i++)
    {
        if ((term[i] >= 'a' && term[i] <= 'z') || (term[i] >= 'A'
&& term[i] <= 'Z'))
        {
            // the following line is assigning term[i]'th element
into the variable op
            // term[i] = 'a'
            // op(1, term[i]) = 'a'
            // op(2, term[i]) = 'aa'
            string op(1, term[i]);
            stack.push(op);
        }
        else
        {
            string op1 = stack.top();
            stack.pop();
            string op2 = stack.top();
            stack.pop();
            if (stack.size() == 0)
            {
                stack.push(op2 + term[i] +
                           op1);
            }
            else
            {
                stack.push("(" + op2 + term[i] +
                           op1 + ")");
            }
        }
    }

    return stack.top();
}

void swap(double *a, double *b)
{
    double t = *a;
    *a = *b;
    *b = t;
}

int ascend_partition(double arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++)
    {

        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

int descend_partition(double arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {

        if (arr[j] > pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void ascend_quickSort(double arr[], int low, int high)
{
    if (low < high)
    {

        int pi = ascend_partition(arr, low, high);

        ascend_quickSort(arr, low, pi - 1);
        ascend_quickSort(arr, pi + 1, high);
    }
}

void descend_quickSort(double arr[], int low, int high)
{
    if (low < high)
    {

        int pi = descend_partition(arr, low, high);

        descend_quickSort(arr, low, pi - 1);
        descend_quickSort(arr, pi + 1, high);
    }
}

string strBitwiseAND(string arr[], int n)
{
    string res;

    int smallest_size = INT_MAX;
    int largest_size = INT_MIN;

    for (int i = 0; i < n; i++)
    {
        reverse(arr[i].begin(), arr[i].end());

        smallest_size = min(smallest_size, (int)arr[i].length());
        largest_size = max(largest_size, (int)arr[i].length());
    }

    for (int i = 0; i < smallest_size; i++)
    {
        bool all_ones = true;

        for (int j = 0; j < n; j++)
        {
            if (arr[j][i] == '0')
            {

                all_ones = false;
                break;
            }
        }

        res += (all_ones ? '1' : '0');
    }

    for (int i = 0; i < largest_size - smallest_size; i++)
        res += '0';

    reverse(res.begin(), res.end());
    return res;
}

string strBitwiseOR(string *arr, int n)
{
    string res;

    int max_size = INT_MIN;

    for (int i = 0; i < n; i++)
    {
        max_size = max(max_size, (int)arr[i].size());
        reverse(arr[i].begin(), arr[i].end());
    }

    for (int i = 0; i < n; i++)
    {

        string s;
        for (int j = 0; j < max_size - arr[i].size(); j++)
            s += '0';

        arr[i] = arr[i] + s;
    }

    for (int i = 0; i < max_size; i++)
    {
        int curr_bit = 0;
        for (int j = 0; j < n; j++)
            curr_bit = curr_bit | (arr[j][i] - '0');

        res += (curr_bit + '0');
    }
    // Reverse the resultant string
    // to get the final string
    reverse(res.begin(), res.end());

    // Return the final string
    return res;
}

string strBitwiseXor(string a, string b, int n)
{
    string ans = "";

    if (a.length() < b.length())
    {
        while (a.length() != b.length())
        {
            a = "0" + a;
        }
    }
    else
    {
        while (a.length() != b.length())
        {
            b = "0" + b;
        }
    }
    for (int i = 0; i < n; i++)
    {
        if (a[i] == b[i])
            ans += "0";
        else
            ans += "1";
    }
    return ans;
}

string strNot(string term)
{
    string result = term;
    for (int i = 0; i < term.length(); i++)
    {
        if (term[i] == '0')
        {
            result[i] = '1';
        }
        else
        {
            result[i] = '0';
        }
    }
    return result;
}

void logic_operations()
{
    while (true)
    {
        cout << "Chose Operation: " << endl;
        cout << "1. AND" << endl;
        cout << "2. OR" << endl;
        cout << "3. NOT" << endl;
        cout << "4. NAND" << endl;
        cout << "5. NOR" << endl;
        cout << "6. XOR" << endl;
        cout << "7. XNOR" << endl;
        cout << "8. Back to Main Menu" << endl;
        cout << "Choice: ";
        int choice;
        cin >> choice;
        //      string inp1, inp2;

        // if (choice == 3){
        //    cin >> inp1;
        // }else{
        //    cin >> inp1 >> inp2;
        // }

        if (choice == 1)
        {
            int n;
            cout << "How Many Number of Strings? ";
            cin >> n;
            string arr[n];
            cout << "Enter the bit strings: ";
            for (int i = 0; i < n; i++)
            {
                cin >> arr[i];
            }
            cout << "Result: " << strBitwiseAND(arr, n) <<
endl;
        }
        else if (choice == 2)
        {
            int n;
            cout << "How Many Number of Strings? ";
            cin >> n;
            string arr[n];
            cout << "Enter the bit strings: ";
            for (int i = 0; i < n; i++)
            {
                cin >> arr[i];
            }
            cout << "Result: " << strBitwiseOR(arr, n) << endl;
        }
        else if (choice == 3)
        {
            string inp;
            cout << "Enter a string: ";
            cin >> inp;
            cout << "Result: " << strNot(inp) << endl;
        }
        else if (choice == 4)
        {
            int n;
            cout << "How Many Number of Strings? ";
            cin >> n;
            string arr[n];
            cout << "Enter the bit strings: ";
            for (int i = 0; i < n; i++)
            {
                cin >> arr[i];
            }
        cout << "Result: " << strNot(strBitwiseAND(arr, n))
<< endl;
      }
      else if (choice == 5)
      {
        int n;
        cout << "How Many Number of Strings? ";
        cin >> n;
        string arr[n];
        cout << "Enter the bit strings: ";
        for (int i = 0; i < n; i++)
        {
          cin >> arr[i];
        }
        cout << "Result: " << strNot(strBitwiseOR(arr, n))
<< endl;
      }
      else if (choice == 6)
      {
        cout << "Enter two strings: ";
        string inp1, inp2;
        cin >> inp1 >> inp2;
        int mx = max(inp1.length(), inp2.length());
        cout << "Result: " << strBitwiseXor(inp1, inp2, mx)
<< endl;
      }
      else if (choice == 7)
      {
        cout << "Enter two strings: ";
        string inp1, inp2;
        cin >> inp1 >> inp2;
        int mx = max(inp1.length(), inp2.length());
        cout << "Result: " << strNot(strBitwiseXor(inp1,
inp2, mx)) << endl;
      }
      else if (choice == 8)
      {
        return;
      }
    }
}

void other_features()
{
  while (true)
  {
    cout << "\n\n1. Mod operation" << endl;
    cout << "2. Percentage Operation" << endl;
    cout << "3. Logarithms" << endl;
    cout << "4. Trigonometry" << endl;
    cout << "5. Square Root" << endl;
    cout << "6. Back to Main Menu" << endl;
    cout << "Choice: ";
    int choice;
    cin >> choice;
    if (choice == 1)
    {
      cout << "Enter two values X and Y (X mod Y): ";
      int inp1, inp2;
      cin >> inp1 >> inp2;
      cout << "Result: " << inp1 % inp2 << endl;
    }
```

```
else if(choice==6){
  return;
}
else if (choice == 3)
{
  cout << "\n1. e-base log or Ln()" << endl;
  cout << "2. 10-base log()" << endl;
  cout << "Choose: ";
  int choice2;
  cin >> choice2;
  if (choice2 == 1)
  {
    cout << "Enter a value: ";
    double val;
    cin>>val;
    cout << "Result: " << log(val) << endl;
  }
  else if (choice2 == 2)
  {
    cout << "Enter a value: ";
    double val;
    cin>>val;
    cout << "Result: " << log10(val) << endl;
  }
}
else if (choice == 5)
{
  cout << "Enter the value: ";
  double inp1;
  cin >> inp1;
  cout << "Result: " << sqrt(inp1) << endl;
}
else if (choice == 2)
{

  cout << "Enter two values X and Y (X % Y): ";
  double inp1, inp2;
  cin >> inp1 >> inp2;
  cout << "Result: " << inp1 * (inp2/100) << endl;
}
else if (choice == 4)
{

  cout << "\n\n1. Sin" << endl;
  cout << "2. Cos" << endl;
  cout << "3. Tan" << endl;
  cout << "4. Cot" << endl;
  cout << "5. Sec" << endl;
  cout << "6. Cosec" << endl;
  cout << "Choice: ";
  int ch;
  cin >> ch;
  if(ch==1){
   cout<<"Enter the value in radian: ";
   double value;
   cin>>value;
   cout<<"Result: "<<sin(value)<<endl;
  }
   else if(ch==2){
   cout<<"Enter the value in radian: ";
   double value;
   cin>>value;
```

```
            cout<<"Result: "<<cos(value)<<endl;
           }
           else if(ch==3){
            cout<<"Enter the value in radian: ";
            double value;
            cin>>value;
            cout<<"Result: "<<tan(value)<<endl;
           }
           else if(ch==4){
            cout<<"Enter the value in radian: ";
            double value;
            cin>>value;
            cout<<"Result: "<<1/tan(value)<<endl;
           }
           else if(ch==5){
            cout<<"Enter the value in radian: ";
            double value;
            cin>>value;
            cout<<"Result: "<<1/cos(value)<<endl;
           }
           else if(ch==6){
            cout<<"Enter the value in radian: ";
            double value;
            cin>>value;
            cout<<"Result: "<<1/sin(value)<<endl;
           }
        }
    }
}

int main()
{
    while (true)
    {
        cout << "\nChoose desired operation from below:" <<
endl;
        cout << "1. Logic operations" << endl;
        cout << "2. Infix to Postfix" << endl;
        cout << "3. Postfix to infix" << endl;
        cout << "4. Ascending Sort" << endl;
        cout << "5. Descending Sort" << endl;
        cout << "6. Other features" << endl;
        cout << "7. Simplify Arithmetic Terms" << endl;
        cout << "Choice: ";
        int option;
        // getline(cin, op);
        cin >> option;
        // cout << op << endl;
        if (option == 1)
        {
            logic_operations();
        }
        else if (option == 2)
        {
            string res = InToPost();
            cout << "Postfix: " << res << endl;
        }
        else if (option == 3)
        {
            string res = PostToIn();
            cout << "Infix: " << res << endl;
        }
        else if (option == 4)
        {
            int n;
            cout << "Enter the size of array: ";
            cin >> n;
            cout << "Enter the array: ";
            double arr[n + 5];
            for (int i = 0; i < n; i++)
            {
                cin >> arr[i];
            }
            ascend_quickSort(arr, 0, n - 1);
            cout << "Sorted array: ";
            for (int i = 0; i < n; i++)
            {
                cout << arr[i] << " ";
            }
            cout << endl;
        }
        else if (option == 5) // descending sort
        {
            int n;
            cout << "Enter the size of array: ";
            cin >> n;
            cout << "Enter the array: ";
            double arr[n + 5];
            for (int i = 0; i < n; i++)
            {
                cin >> arr[i];
            }
            descend_quickSort(arr, 0, n - 1);
            cout << "Sorted array: ";
            for (int i = 0; i < n; i++)
            {
                cout << arr[i] << " ";
            }
            cout << endl;
        }
        else if (option == 6)
        {
            other_features();
        }
        else if (option == 7)
        {
            string inp;
            cout << "Enter the term: ";
            getline(cin, inp);
            double res = simplify(inp);
            cout << "\tResult = " << res << endl;
        }
    }
}
```