Objective

The objective of this lab is to implement a C program that multiplies two decimal numbers using Booth's algorithm, handling negative numbers using two's complement representation.

Algorithm

1. Start.
2. Initialize and take number inputs in decimal.
3. Initialize arrays of specific bit size Q, M, and M_{comp}. Also, initialize the accumulator A to 0, an integer $(Q-1)=0$, and set the counter to the number of bits.
4. Convert the multiplicand (first decimal number) to binary. If the number is negative, compute its two's complement and store it in M.
5. Convert the multiplier (second decimal number) to binary. If the number is negative, compute its two's complement and store it in Q.
6. Compute the two's complement of M and store it in M_{comp}.
7. For n bits of data:
   - If $Q[0](Q-1)$ is 01, perform $A \leftarrow A+M$ and then perform an arithmetic right shift on A, Q, $(Q-1)$.
   - Else if $Q[0](Q-1)$ is 10, perform $A \leftarrow A-M$ and then perform an arithmetic right shift on A, Q,$(Q-1)$.
   - Else, perform an arithmetic right shift on A, Q, $(Q-1)$.
8. Decrement the counter by 1.
9. If the counter is greater othan 0, go to step 7.
10. Stop. The result is stored in AQ.

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void decimalToBinary(int n, int *binary, int size)
{
    n = abs(n);
    for (int i = size - 1; i >= 0; i--)
    {
        binary[i] = n & 1;
        n >>= 1;
    }
}

void twosComplement(int *binary, int size)
{
    int carry = 1;
```

```
    for (int i = 0; i < size; i++)
    {
        binary[i] = ~binary[i] & 1;
    }

    for (int i = size - 1; i >= 0; i--)
    {
        binary[i] += carry;
        if (binary[i] == 2)
        {
            binary[i] = 0;
            carry = 1;
        }
        else
        {
            carry = 0;
        }
    }
}

void arithmeticRightShift(int *binary1, int *binary2, int *a, int size)
{
    *a = binary2[size - 1];

    for (int i = size - 1; i > 0; i--)
    {
        binary2[i] = binary2[i - 1];
    }

    binary2[0] = binary1[size - 1];

    int msb = binary1[0];
    for (int i = size - 1; i > 0; i--)
    {
        binary1[i] = binary1[i - 1];
    }
    binary1[0] = msb;
}

void addTwoBinaries(int *binary1, const int *binary2, int size)
{
    int carry = 0;
    for (int i = size - 1; i >= 0; i--)
    {
        int sum = binary1[i] ^ binary2[i] ^ carry;
        carry = (binary1[i] & binary2[i]) | (binary2[i] & carry) | (carry & binary1[i]);
        binary1[i] = sum;
    }
}

void printBinary(const int *binary, int size)
{
```

```
    for (int i = 0; i < size; i++)
    {
        printf("%d", binary[i]);
    }
}

void printRow(int count, const int *accumulator, const int *temp, int a, const char *op
{
    printBinary(accumulator, size);
    printf(" | ");
    printBinary(temp, size);
    printf(" |   %d   |   %d   | %s\n", a, count, operation);
}

int main()
{
    int x, y, a = 0, count;
    int size;

    printf("Enter the size: ");
    scanf("%d", &size);

    int *first = (int *)malloc(size * sizeof(int));
    int *second = (int *)malloc(size * sizeof(int));
    int *accumulator = (int *)calloc(size, sizeof(int));
    int *complementSecond = (int *)malloc(size * sizeof(int));
    int *temp = (int *)malloc(size * sizeof(int));

    printf("Enter the first number: ");
    scanf("%d", &x);
    printf("Enter the second number: ");
    scanf("%d", &y);

    decimalToBinary(x, first, size);
    if (x < 0)
    {
        twosComplement(first, size);
    }
    printf("First number in binary: ");
    printBinary(first, size);
    printf("\n");

    decimalToBinary(y, second, size);
    if (y < 0)
    {
        twosComplement(second, size);
    }
    printf("Second number in binary: ");
    printBinary(second, size);
    printf("\n\n");

    for (int i = 0; i < size; i++)
    {
```

```
    complementSecond[i] = second[i];
}

twosComplement(complementSecond, size);

for (int i = 0; i < size; i++)
{
    temp[i] = first[i];
}

count = size;
printf("|   A   |   Q   | Q-1 | COUNT | Remarks\n");
printf("|-------|-------|-----|-------|-----------\n");

printRow(count, accumulator, temp, a, "Initialization", size);

while (count > 0)
{
    if ((temp[size - 1] == 0) && (a == 1))
    {
        addTwoBinaries(accumulator, second, size);
        printRow(count, accumulator, temp, a, "Q[0]Q-1=10\tAddition A=A+M", size);
        arithmeticRightShift(accumulator, temp, &a, size);
        count--;
        printRow(count, accumulator, temp, a, "Arithematic Right Shift A Q Q-1", si
        printf("\n");
    }
    else if ((temp[size - 1] == 1) && (a == 0))
    {
        addTwoBinaries(accumulator, complementSecond, size);
        printRow(count, accumulator, temp, a, "Q[0]Q-1=01\tSubtraction A=A-M", size
        arithmeticRightShift(accumulator, temp, &a, size);
        count--;
        printRow(count, accumulator, temp, a, "Arithematic Right Shift A Q Q-1", si
        printf("\n");
    }
    else
    {
        arithmeticRightShift(accumulator, temp, &a, size);
        count--;
        printRow(count, accumulator, temp, a, "Arithematic Right Shift A Q Q-1", si
        printf("\n");
    }
}

printf("Result after Booth's multiplication:\n");
printBinary(accumulator, size);
printBinary(temp, size);
printf("\n");

free(first);
free(second);
free(accumulator);
```

```
    free(complementSecond);
    free(temp);

    return 0;
}
```

Sample Input/Output

- **Input:** 6 and -5
- **Output:**

```
Enter the size: 5
Enter the first number: 6
Enter the second number: -5
First number in binary: 00110
Second number in binary: 11011
```

A	Q	Q-1	COUNT	Remarks
00000	01001	0	5	Initialization
00111	01001	0	5	Q[0]Q-1=01 Subtraction A=A-M
00011	10100	1	4	Arithmetic Right Shift A Q Q-1
11100	10100	1	4	Q[0]Q-1=10 Addition A=A+M
11110	01010	0	3	Arithmetic Right Shift A Q Q-1
11111	00101	0	2	Arithmetic Right Shift A Q Q-1
00110	00101	0	2	Q[0]Q-1=01 Subtraction A=A-M
00011	00010	1	1	Arithmetic Right Shift A Q Q-1
11100	00010	1	1	Q[0]Q-1=10 Addition A=A+M
11110	00001	0	0	Arithmetic Right Shift A Q Q-1

Result after Booth's Multiplication

A	Q
11101	01000

Discussion

The program demonstrates the Booth's algorithm for multiplication of two numbers, taking care of the sign by converting the numbers to their two's complement if negative. The algorithm handles the bitwise operations step by step, providing a clear multiplication process in binary format.