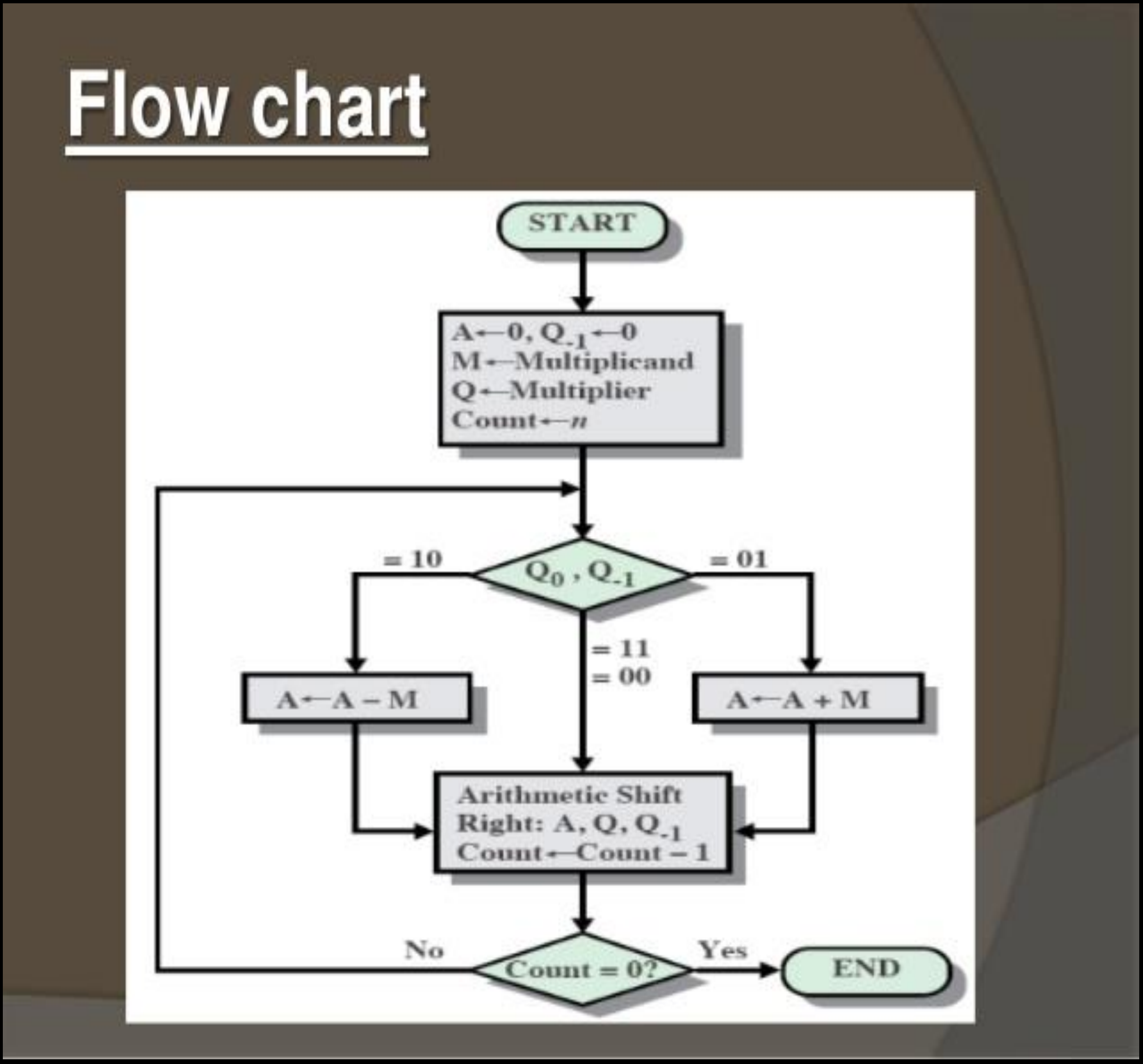
Booth's Algorithm

Documentation

-By Manvik Arya

The Booth's algorithm gives an efficient procedure for multiplying binary integers in signed 2's complement representation.



Variables/Terms Used –

1. M – Multiplicand
2. Q – Multiplier
3. Q0 – Least Significant Bit of Multiplier
4. ac – Accumulator
5. cycles – Number of steps involved to calculate the result

```
class Main {
        static String M,Q;
        static String Q1="0";
        static String Q0;
        static String ac= "0";
        static int cycles=0;
        static String neg_M;
        static boolean ism1Negative = false;
        static boolean ism2Negative = false;
        static String result;
```

Assumptions –

- The provided Multiplicand and Multiplier are both integers and not fractions, irrational numbers or decimal numbers.
- The numbers are provided in integer format and not in the form of strings.
- Both the Multiplicand and the Multiplier can have digits up to 11 digits in the integer form.

Constraints –

- Number of bits allowed for Multiplicand = 32 bits
- Number of bits allowed for Multiplier = 32 bits
- Number of bits allowed for the Final Result = 64 bits

Functions involved in Code –

1. The matchBits() function alters the Binary representation of the multiplicand and the multiplier and makes them equal in terms of the number of bits.

```
static void matchBits()
{
    while(M.length() - Q.length() > 0)
    {
        Q = "0" + Q;
    }
    while(M.length() - Q.length() < 0)
    {
        M = "0" + M;
    }
    M = "0" + M;
    Q = "0" + Q;

}

static void matchAc()
{
    while(ac.length()-Q.length()<0)
    {
        ac = "0" + ac;
    }

}
```

2. The Complement() function takes a String as a parameter and provides the 2s complement of the given string.

```
static String Complement(String k)
{
      //ones
    StringBuilder temp= new StringBuilder();
    StringBuilder temp2= new StringBuilder();
    for(int i = 0;i<k.length();i++)
    {
        if(k.charAt(i)=='0')
        {               temp.append("1");
        }
        else if(k.charAt(i)=='1')
        {               temp.append("0");
        }
    }
    //twos
    for(int i = k.length()-1;i>=0;i--)
    {
        if(temp.charAt(i)=='0')
        {               temp.replace(i, end: i+1, str: "1");
            break;
        }           else if(temp.charAt(i)=='1')
        {               temp.replace(i, end: i+1, str: "0");
        }
    }
return temp.toString();
}
```

3. The addBinary() function takes two Strings as parameter provides the corresponding additive result of the two.

```
static String addBinary(String a, String b) {
    // Initialize result
    String result = "";
    // Initialize digit sum
    int s = 0;
    int i = a.length() - 1, j = b.length() - 1;
    while (i >= 0 || j >= 0 || s == 1) {

        // Compute sum of last
        // digits and carry
        s += ((i >= 0) ? a.charAt(i) - '0' : 0);
        s += ((j >= 0) ? b.charAt(j) - '0' : 0);

        result = (char) (s % 2 + '0') + result;

        // Compute carry
        s /= 2;
        // Move to next digits
        i--;
        j--;
    }
    if(result.length()!=a.length())
    {            result = result.substring(1);
    }
    return result;
}
```

4. The rightShift() function performs the Arithmetic right shift on ac, Q, Q0.

```
static void rightShift()
{

    Q1 = Q0;
    Q = String.valueOf(ac.charAt(ac.length()-1)) + Q.substring(0,Q.length()-1);
    ac = ac.charAt(0) + ac.substring(0,ac.length()-1);
    Q0 = String.valueOf(Q.charAt(Q.length()-1));
}
```

5. The booths() function is the implementation of the booth's algorithm and consists of three cases.

a) Case1: When both the multiplicand and multiplier are positive.

```
static String booths()
{

    matchAc();
    cycles = ac.length();

    for(int i = 0;i<cycles;i++)
    {
        // Case 1 = When both the given numbers are positive
        if(!ism1Negative && !ism2Negative)
        {

            if ((Q0.equals("0") && Q1.equals("0")) || (Q0.equals("1") && Q1.equals("1"))) {
                rightShift();
            }
            else if (Q0.equals("1") && Q1.equals("0")) {
                ac = addBinary(ac, neg_M);
                rightShift();
            }
            else if (Q0.equals("0") && Q1.equals("1")) {
                ac = addBinary(ac, M);
                rightShift();
            }
        }
```

b) When the either the multiplicand or the multiplier is negative.

```
// Case 2 = When one of the given numbers if negative
else if((ism1Negative && !ism2Negative) || (!ism1Negative && ism2Negative))
{
    if ((Q0.equals("0") && Q1.equals("0")) || (Q0.equals("1") && Q1.equals("1"))) {
        rightShift();
    }
    else if (Q0.equals("1") && Q1.equals("0")) {
        ac = addBinary(ac, neg_M);
        rightShift();
    }
    else if (Q0.equals("0") && Q1.equals("1")) {
        ac = addBinary(ac, M);
        rightShift();
    }


}
```

c) When both the multiplicand and the multiplier are negative.

```
        // Case 3 = When both the numbers are negative:-
        else if(ism1Negative && ism2Negative)
        {
            if ((Q0.equals("0") && Q1.equals("0")) || (Q0.equals("1") && Q1.equals("1"))) {
                rightShift();
            }
            else if (Q0.equals("1") && Q1.equals("0")) {
                ac = addBinary(ac, neg_M);
                rightShift();
            }
            else if (Q0.equals("0") && Q1.equals("1")) {
                ac = addBinary(ac, M);
                rightShift();
            }
        }
        System.out.println("AC = " + ac + "     Q = " + Q  + "     Q1 = " + Q1);
    }
    result = ac + Q;
    if((ism1Negative && !ism2Negative) || (!ism1Negative && ism2Negative))
    {
        result = Complement(result);
    }
    System.out.println("The Binary Result = " + result);
    return result;
}
```

Code's Working –

The code follows the Booth's algorithm in order to multiply two integers and produce the result along with the calculation involved.

1. The user provides the Multiplicand and the Multiplier which are converted to Binary Strings.
2. The signs of the Multiplicand and the Multiplier (i.e. if they are positive or negative) are evaluated.
3. According to the signs, the corresponding 2s complement of the binary string is calculated.
4. The bits of the corresponding Multiplicand (M) and Multiplier (Q) are matched so, that they have equal number of bits.
5. The Booth's function is called that carries out the calculation and gives the detailed information of the values involved in the process (AC, Q, Q0, Q-1).
6. Finally, the result is expressed in the Binary as well as the Integer form.

Example –

The Multiplicand = 23

The Multiplier = -9

```
Run:   Main

"C:\Program Files\Java\jdk-13.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
Enter the 1st decimal number :
23
Enter the 2nd decimal number :
-9
The Given Multiplicand = 010111
The Given Multiplier = 110111

-------------------Initializing Booth's Algorithm-------------------
AC = 110100     Q = 111011     Q1 = 1
AC = 111010     Q = 011101     Q1 = 1
AC = 111101     Q = 001110     Q1 = 1
AC = 001010     Q = 000111     Q1 = 0
AC = 111001     Q = 100011     Q1 = 1
AC = 111100     Q = 110001     Q1 = 1
The Binary Result = 000011001111
The Final Integer Result = -207

Process finished with exit code 0

2: Favorites   4: Run   6: TODO   Terminal   0: Messages
```

The Final Binary Result = 000011001111

The Final Integer Result = -207