REPORT

Vaibhav Jindal (2018115)

Mayank Rawat (2018049)

**Multiplication**

**Time Complexity**

Time Complexity of algorithm implemented is (N)^2 where N is the maximum length of the binary of the input provided.

**Input**

Enter two integers in the range of -(1000) to (+1000)

**Output**

Output is stored in output.txt in binary and decimal form in a nice and readable format.

**Algorithm**

Flowchart has been attached as pdf. Dry run is as follow :

1. Take input a,b and convert into binary; a\_bin and b\_bin
2. For negative numbers, store their two’s complement in the corresponding binary.
3. Make the length of both the binaries same
4. Initialize ac with ‘0’. Length(ac) = length(a\_bin) ; Counter = length(a\_bin); Q\_nplusone = 0
5. While (Counter > 0)
   1. last = a\_bin[-1]
   2. if last, Q\_nplusone == 1,0
      1. ac = ac + twos\_complement(b\_bin)
   3. else if last, Q\_nplusone == 1,0
      1. ac = ac + b\_bin
   4. RightShift(ac, a\_bin)
   5. Counter = Counter-1
6. Answer = ac + a\_bin { Note : this is string addition }

**Test Cases Used and corresponding outputs :**

1. 3 4
   1. Answer in Binary is : 00001100 Answer in Decimal is : + 12
2. -3 4
   1. Answer in Binary(2's complement) is : 00001100 Answer in Decimal is : - 12
3. -3 -4
   1. Answer in Binary is : 00001100 Answer in Decimal is : + 12
4. 1000 1000
   1. Answer in Binary is : 0011110100001001000000 Answer in Decimal is : + 1000000

**DIVISION**

**TIME COMPLEXITY**

Time Complexity for algorithm ON HARDWARE implemented is N where N is max number of bits need to represent the number

THE TIMECOMPLEXTITY OF THE PROGRAM AS IT IS N^2

**INPUT**

Enter two integers in range (-1000) to (1000)

**OUTPUT**

Shown on the terminal in Binary 2’s Complement format and Decimal representation as well

**ALGORITHM**

1. TAKES INPUT FOR TWO NUMBER THE DIVIDEND AND DIVISOR
2. IF DIVISOR =0 ERROR IS GENREATED AND PROGRAM IS TERMINATED
3. THE NUMBER ARE CONVERTED TO BINARY 2’S COMPLEMENT NEGLECTING THE SIGN AND SIGN BIT IS STORED FOR LATER USE
4. ACC ,REG\_Q AND REG\_M ARE INITALIZE TO ZERO COUNTER INITALIZED TO ZERO
5. ACC AND REG\_Q STORES THE DIVIDEND BINARY (22BITS)
6. REG\_M STORES THE DIVISOR BINARY
7. PREVIOUS\_ACC REGISTER IS INTIALIZED TO THE VALUE OF ACC
8. IF THE SIGN OF ACC AND REG\_M ACC<- ACC -REG\_M
9. ELSE ACC <- ACC +REG\_M
10. IF THE SIGN OF (ACC AND PREVIOUS ACC) ARE NOT EQUAL AND ACC IN NOT ZERO AND REG\_Q IS NOT ZERO ACC<- PREVIOUSACC
11. ELSE REG\_Q <- REG\_Q[:-1] +”1”
12. 7-12 STEP ARE REPEATED TILL COUNTER != NUMBER OF BITS IN DIVISOR =11 BITS
13. QUOITENT AND REMAINDER ARE DISPLAYED

**TEST CASE USED**

**TESTCASE 1**

**DIVIDENT 5; DIVISOR 3**

**O/P**

**QUOTIENT: 00000000001 REMAINDER: 00000000010**

**QUOTIENT IN DECIMAL: 1 REMAINDER IN DECIMAL: 2**

**TESTCASE 2**

**DIVIDENT -5; DIVISOR 3**

**O/P**

**QUOTIENT: 11111111111 REMAINDER: 11111111110**

**QUOTIENT IN DECIMAL: -1 REMAINDER IN DECIMAL: -2**

**TESTCASE 3**

**DIVIDENT 10; DIVISOR 2**

**QUOTIENT: 00000000101 REMAINDER: 00000000000**

**QUOTIENT IN DECIMAL: 5 REMAINDER IN DECIMAL: 0**

**TESTCASE 4**

**DIVIDENT 10; DIVISOR -2**

**O/P**

**QUOTIENT: 11111111011 REMAINDER: 00000000000**

**QUOTIENT IN DECIMAL: -5 REMAINDER IN DECIMAL: 0**

**TESTCASE 5**

**DIVIDENT 10; DIVISOR 0**

**O/P**

**Cannot Divide by 0**