Design Documentation

Overview

The aim of this project is to perform multiplication and division of signed binary numbers using booth algorithms. These algorithms were invented by Andrew Donald Booth in 1950 as one of his research works. Our report has been divided into two sections, multiplication algorithm, and division algorithm. We'll go through the algorithms that we used in detail along with their complexity analysis.

Booth's Restoring Division Algorithm

This division algorithm takes two numbers as it's input, the dividend, and the divisor. And, the algorithm throws two values as its output, the quotient and the remainder. The algorithm is called a restoring algorithm because it keeps restoring the values of the remainder register after every iteration.

This algorithm uses three registers for the following:

- The dividend (which is restored with every iteration and finally gets converted to the quotient)
- The divisor
- The remainder

Let's name these registers as Q, M, A respectively. We also take a seq_ctr, which keeps track of the number of iterations.

Note: Booth's algorithm works for unsigned number division, we have added separate cases to handle signed division, by performing unsigned division on the numbers and then adding the required sign.

The Algorithm

1. The initial values are loaded into the register, Q stores the dividend, M stores the divisor, A is initialized to 0 and the seq_ctr is initialized to the number of bits in the dividend.
2. A shift left operation takes place, the values of the register A and Q are concatenated (A||Q) and shifted left as a single unit. Their post-shifted values are stored in the registers A and Q. The least significant bit of the value in register Q is stored as 0 for now.
3. The value of register M is subtracted from the register A and register A is updated.

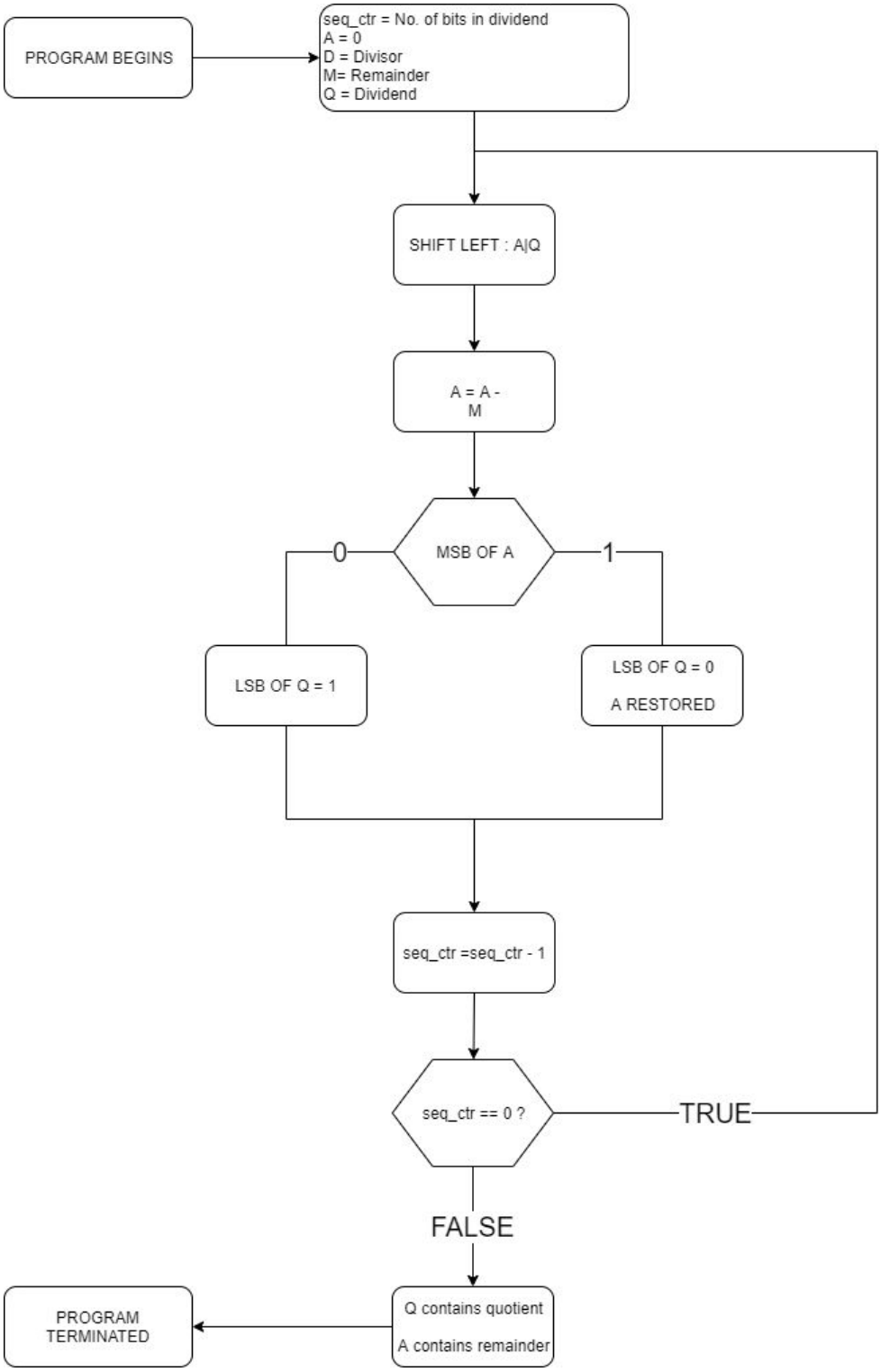
A <- A - M

4. Now, a conditional test takes place.
    - If the most significant bit of the value of Register A is '1', the value of the least significant bit of the register Q is set as 0 (default value). The value of register A is restored to its old value (before step 3) [Restoration Step].
    - But, if the most significant bit of the value of Register A is '0', the value of the least significant bit of the register Q is updated to '1'.
5. The seq_counter value is decremented. Here one iteration is complete.

seq_ct r = seq_ctr-1

6. Now if the value of the sequence counter(seq_ctr) is 0. The program exits the loop. Otherwise, steps 2, 3, 4 and 5 are repeated.
7. In the end, we receive the Quotient in the register Q and the remainder in the register A.
8. We explicitly handle cases with negative numbers, assuming remainder to be a positive number.

Flow Chart


PROGRAM BEGINS
seq_ctr = No. of bits in dividend
A = 0
D = Divisor
M= Remainder
Q = Dividend
SHIFT LEFT : A|Q
A = A - M
MSB OF A
0
1
LSB OF Q = 1
LSB OF Q = 0
A RESTORED
seq_ctr =seq_ctr - 1
seq_ctr == 0 ?
TRUE
FALSE
Q contains quotient
A contains remainder
PROGRAM TERMINATED

Test Cases Handled

S. no.	Input	Output
1	1 1	Binary equivalent of quotient 01 Binary equivalent of remainder00 quotient 1 remainder 0
2	-1 1	Binary equivalent of quotient 11 Binary equivalent of remainder00 quotient -1 remainder 0
3	1 -1	Binary equivalent of quotient 11 Binary equivalent of remainder00 quotient -1 remainder 0
4	0 1	Binary equivalent of quotient 00 Binary equivalent of remainder00 quotient 0 remainder 0
5	9 5	Binary equivalent of quotient 00001 Binary equivalent of remainder 00100 quotient 1 remainder 4
6	-9 -5	Binary equivalent of quotient 00010 Binary equivalent of remainder 00001 quotient 2 remainder 1
8	1 0	Invalid Operation
7	123 234	Binary equivalent of quotient 000000000 Binary equivalent of remainder 001111011 quotient 0 remainder 123

Complexity analysis

Let n and m be the two numbers that are being divided. (n/m)

Let us assume n>m, then the binary representation of the number n is of length log n. The booth's algorithm repeats itself log n times (length of divisor). In each iteration a subtraction operation takes place. If a full adder circuit (Carry look-ahead adder) is present in hardware, this can be done in constant time, else it would take time proportional to the length of numbers in binary representation, i.e. log n (length of dividend). Further, an arithmetic shift left operation is then performed which would take O(1) time if the special register is present else can be done in log n time logically.

Thus our complexity of the current program which assumes the absence of a full adder and special registers is O(log n) * O(log n) = O((log n)^2).

Booth's signed Multiplication Algorithm

This multiplication algorithm takes two numbers as it's input, the multiplier, and the multiplicand. And, the algorithm throws the multiplication result as the output. This algorithm uses three registers for the following:

- Product(A) which finally stores the first n bits of the answer (n most significant bits)
- Multiplicand(Q), which finally stores the last n bits of the answer (n least significant bits)
- Multiplier(M)

The Algorithm

1. The initial values are loaded into the register, 'Q' stores the multiplicand, 'M' stores the multiplier, 'A' is initialized to 0 and the sequence_counter is initialized to the number of bits in the multiplicand.
2. If Q[n] == 1 and Q_{-1} == 0, that is the last bit of Register Q is 1 and the Q_{-1} flag is 0, then the value of register M is subtracted from the register A.

   A <- A - M

3. If Q[n] == 0 and Q_{-1} == 1, that is the last bit of Register Q is 0 and Q_{-1} flag is 1, then the value of register M is added to the register A.

   A <- A + M

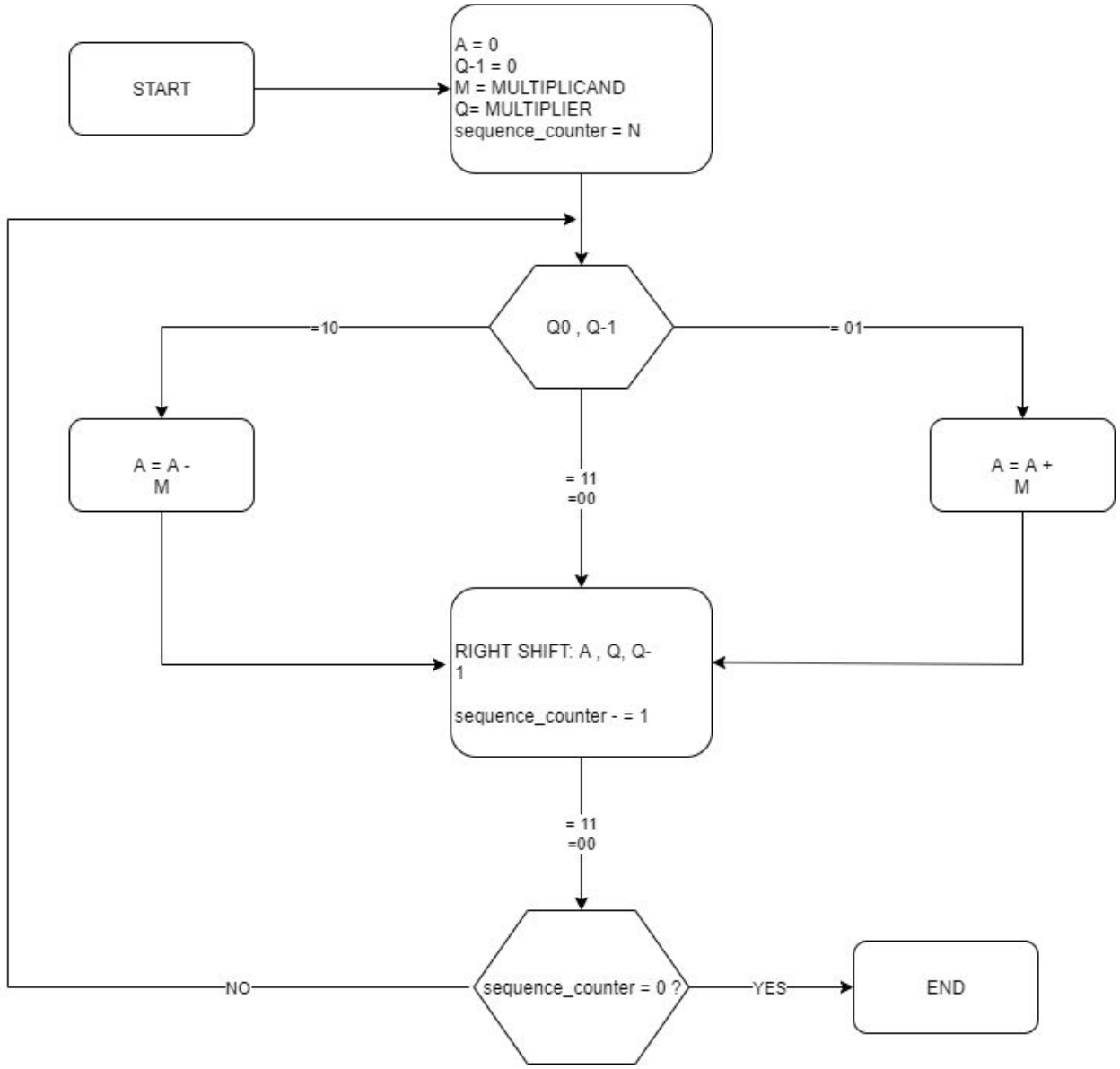
4. An arithmetic shift right operation takes place, the values of the register A, Q, and Q_{-1}. It can be thought of as A, Q and Q_{-1} are concatenated and a shift right is performed and then separated, even though in actual implementation, concatenation isn't required. Their post-shifted values are stored in registers A, Q, and Q_{-1}. The most significant bit of the value in register A remains the same.
5. The seq_counter value is decremented. Here one iteration is complete.

   seq_ct r = seq_ctr-1

6. Now if the value of the sequence counter(seq_ctr) is 0. The program exits the loop. Otherwise, steps 2, 3 and 4 are repeated.

7. In the end, we receive the product split in 2 registers, A and Q (Q_{-1} is ignored). Product is a concatenation of the registers A and Q. Register A stores the n most significant bits of the product(2n bits).

Flowchart



Test Cases Handled

S.no	Input	Output
1	1 1	Binary equivalent: 0001 Answer: 1
2	1 -1	Binary equivalent: 1111 Answer: -1
3	-1 1	Binary equivalent: 1111 Answer: -1
4	-1 -1	Binary equivalent: 0001 Answer: 1
5	2 2	Binary equivalent: 000100 Answer: 4
6	123 234	Binary equivalent: 0001110000011011 10 Answer: 28782
7	-123 234	Binary equivalent: 1110001111100100 10 Answer: -28782
8	-7 -07	Binary equivalent: 00110001 Answer: 49
9	1 0	Binary equivalent: 0000 Answer: 0
10	1 -0	Binary equivalent: 0000 Answer: 0

Complexity analysis

Let n and m be the two numbers that are being multiplied. (n and m in decimal notation)

Without loss of generality, let us assume n>m, then the binary representation of the number n is of length log n. The booth's algorithm repeats itself log n times (length of multiplicand times). In each iteration, depending on the state, an addition or subtraction operation takes place. If a full adder circuit (Carry look-ahead adder) is present in hardware, this can be done in constant time, else it would take time proportional to the length of numbers in binary representation, i.e. log n. Further, an arithmetic shift right operation is then performed which would take O(1) time if the special register is present else can be done in log n time logically.

Thus our complexity of the current program which assumes the absence of a full adder and special registers is O(log n) * O(log n) = O((log n)^2). This program can we updated to give complexity of O(log n * log m) if change the length of register A to size M. Note this has no meaning in actual representation as the length of registers is fixed on a circuit due to high costs. In practice, the two complexities give similar performance. Further, if proper circuitry is present, that is in presence of adders and special registers capable of performing arithmetic shift right operation, we can get a complexity of O(log n) from Booth's algorithm.