* Binary Multiplication :-

1) Unsigned Number Multiplication :-

The product of two n-digit no.s can be accommodated in 2n digits, so the product of two 4-bit no.s fits into 8-bits. In binary system multiplication of the multiplicand (M) by one bit of the multiplier (Q) is easy. when the multiplier bit (Q_n) is 0, the partial product is 0. when the bit is 1, the ~~parti~~ partial product is the multiplicand (M). The final product (P) is produced by adding all the partial products.

e.g. 12 x 08 =

	1 1 0 0	(12) M
X	1 0 0 0	(08) Q
	0 0 0 0	P1
	0 0 0 0 X	P2
	0 0 0 0 X X	P3
	1 1 0 0 X X X	P4
	0 1 1 0 0 0 0 0	= (96) Product.

② 14 x 10 =

	1 1 1 0	(14) M
x	1 0 1 0	(10) Q
	0 0 0 0	P1
	1 1 1 0 X	P2
	0 0 0 0 X X	P3
	1 1 1 0 X X X	P4
	1 0 0 0 1 1 0 0	= (140)

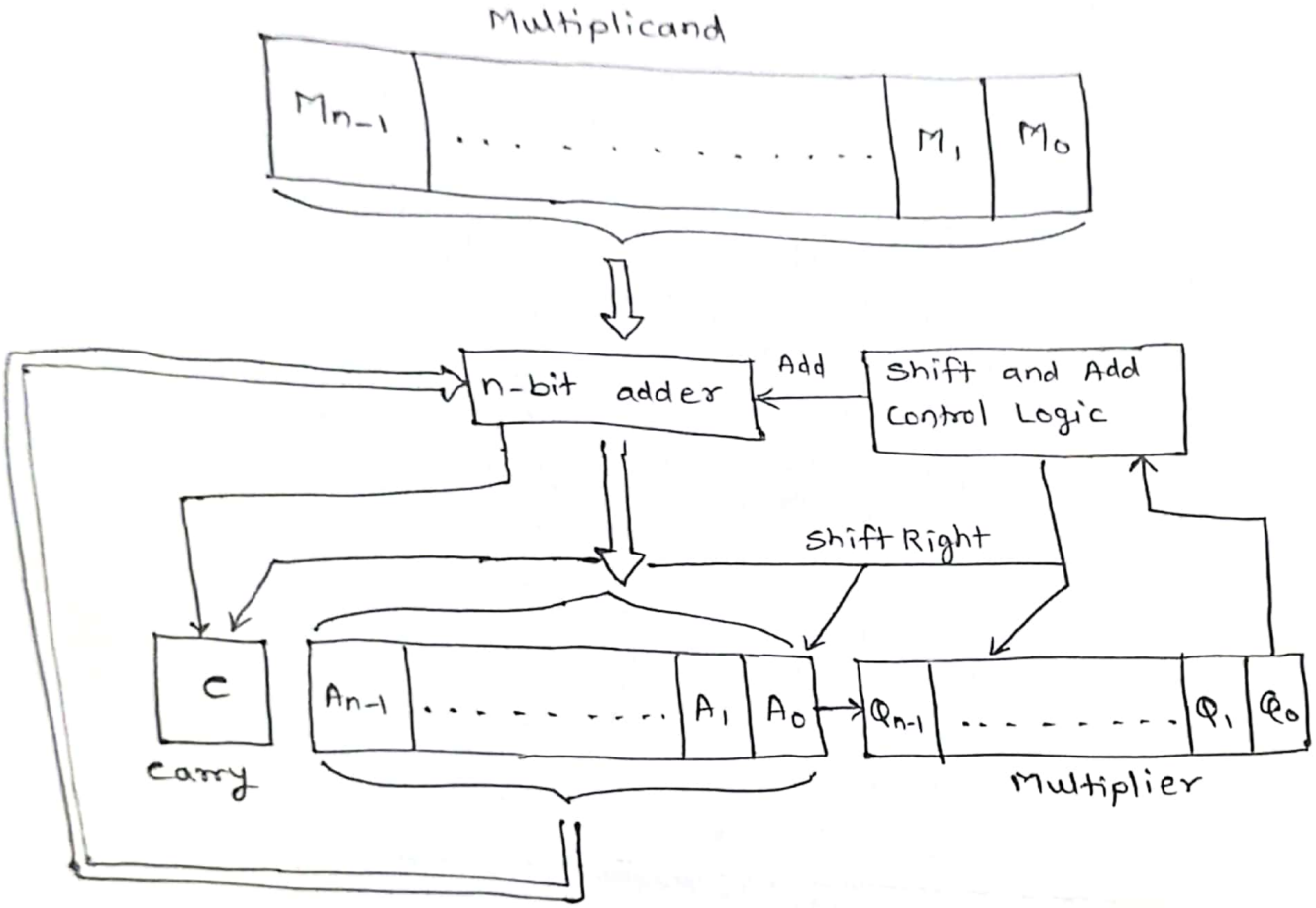
Hardware Implementation :-

Fig. shows a p hardware implementation. The multiplicand & multiplier are loaded into two registers (M & Q). A third register, the A register (Accumulator) is needed & is set to 0. There is a 1 bit C register; (carry) is set to 0, which holds a potential carry bit resulting from addition.

Step 1 :- Control logic reads the bits of the multiplier one at a time.

Case 1 : If Q_0 is 1, the multiplicand is added to the A register & the result is stored in the A register (partial product).

Block diagram of Unsigned multiplication :-

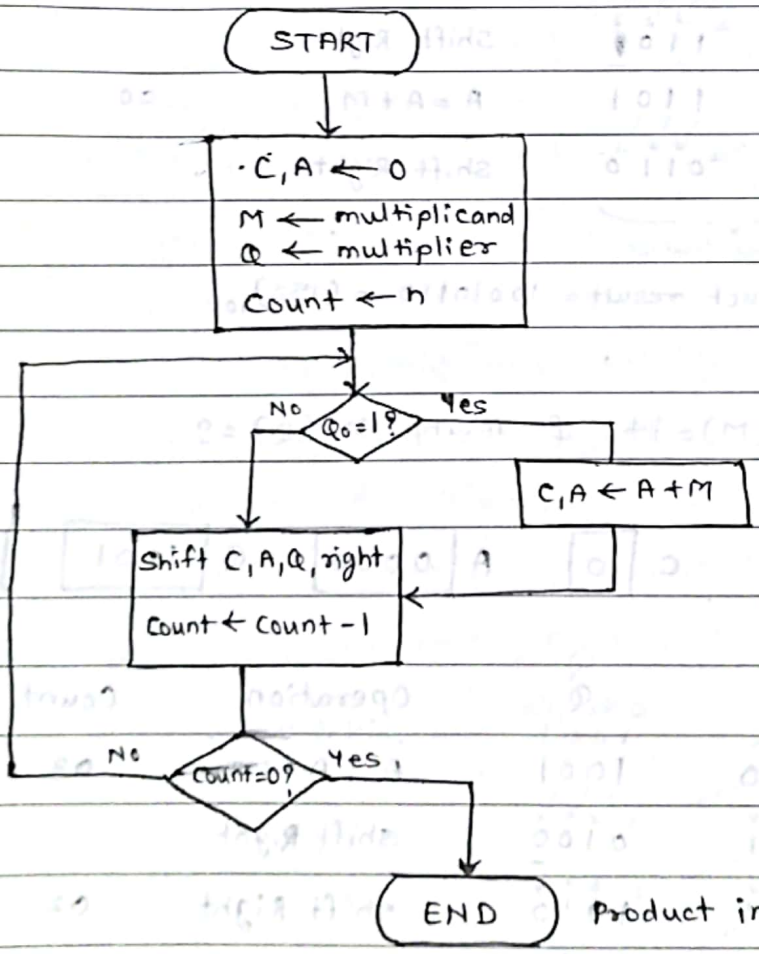


Step 2 :- All the bits of C, A & Q registers are shifted to the right by one bit, so that C bit goes into A_{n-1}, A_0 goes into Q_{n-1} & Q_0 is lost.

Case 2 :- If Q_0 is 0, then no addition is performed, just the shift.

Step 3 :- This process is repeated for each bit of the original multiplier. The resulting ~~n~~ 2n bit product is contained in the A & Q registers.

Flowchart for unsigned binary multiplication is -



The final product results appears in A & Q register which is 2n bits. Multiplier (Q) is of 4 bits in length, it takes 4 cycles to complete this process.

e.g. ① Multiplicand (M) = 10 & multiplier (Q) = 15

M	C	A	Q	
1010	0	0000	1111	Initial values

	C	A	$Q_3Q_2Q_1Q_0$	Operation	Count
	0	0000	1111	Initial values	
A = 0000, m = 1010, New A = 1010 →	0	1010	1111	A = A + M	03
	0	0101	0111	Shift Right	
A = 0101, m = 1010, New A = 1111 →	0	1111	0111	A = A + M	02
	0	0111	1011	Shift Right	
A = 0111, m = 1010, New A = 10001 →	1	0001	1011	A = A + M	01
	0	1000	1101	Shift Right	
A = 1000, m = 1010, New A = 10010 →	1	0010	1101	A = A + M	00
	0	1001	0110	Shift Right	

Final Product.

Final product result = $10010110 = (150)_{10}$

② Multiplicand (M) = 14 & multiplier (Q) = 9

M	C	A	Q	
1110	0	0000	1001	Initial values.

	C	A	$Q_3Q_2Q_1Q_0$	Operation	Count
	0	0000	1001	Initial values	
A = 0000, m = 1110, New A = 1110 →	0	1110	1001	A = A + M	03
	0	0111	0100	Shift Right	
	0	0011	1010	Shift Right	02
	0	0001	1101	Shift Right	01
A = 0001, m = 1110, New A = 1111 →	0	1111	1101	A = A + M	00
	0	0111	1110	Shift Right	

Final product result = $01111110 = (126)_{10}$

Signed Multiplication :-

Multiplication of signed operands generates a double-length product in 2's complement no. sys. For example, 5-bit signed operand -13 is the multiplicand & it is multiplied by +11, we get 10-bit product as -143.

```
    10011   (-13)   2's complement of 13
  x 01011   (+11)
  ---------
    10011   P1
   10011    P2
  00000     P3
 10011      P4
00000       P5
---------
1101110001  (-143)
```

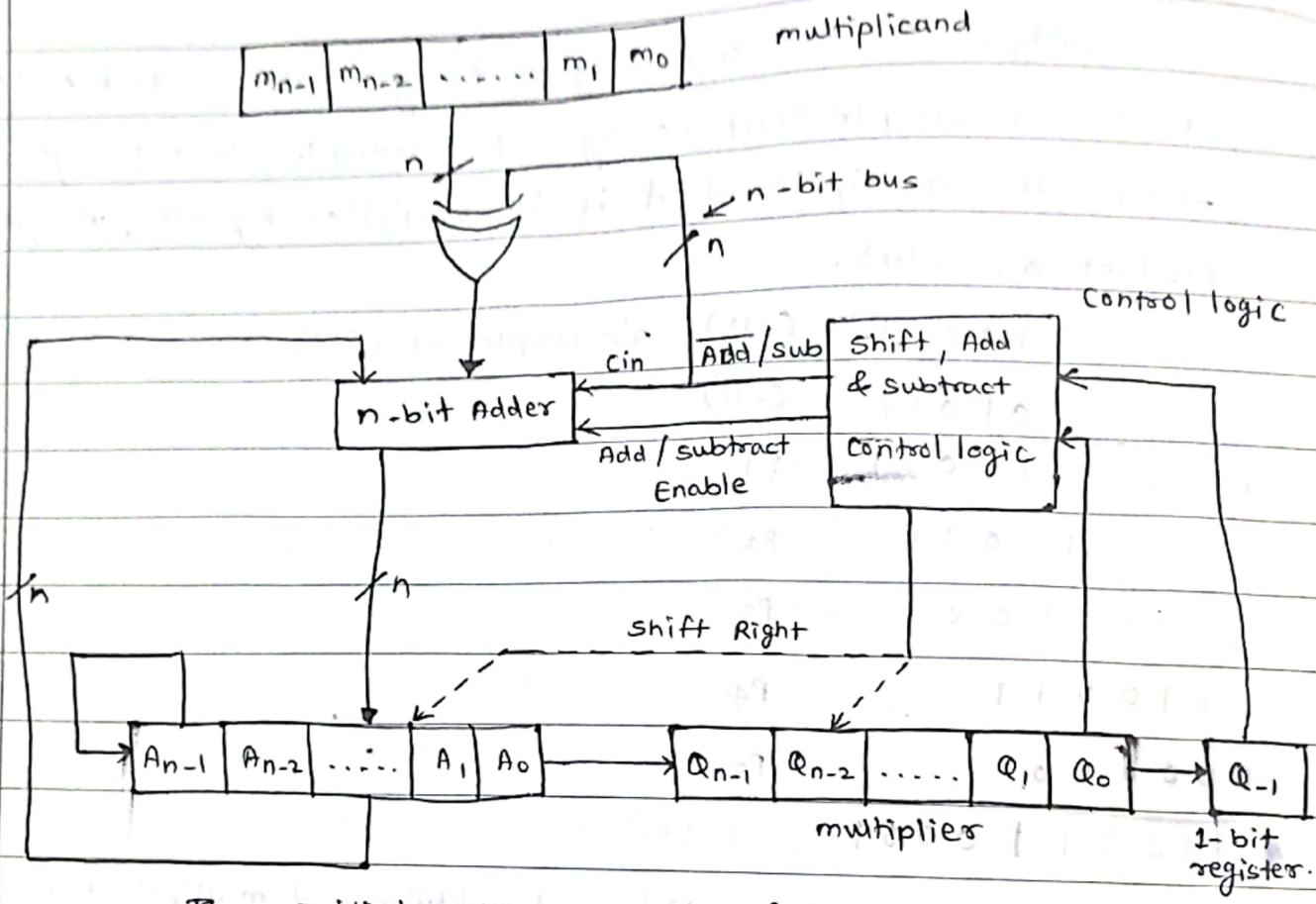
If multiplier is negative, straightforward multiplication will not work, because the bits of the multiplier no longer corresponds to the shifts or multiplications that must take place.

A powerful algorithm for signed no. multiplication is a Booth's algorithm. It is used for 2's complement multiplication. Booth's algorithm contains both addition & subtraction, but it treats positive & negative operands uniformly, no special action required for negative no.s.

* Booth's Algorithm :-

Hardware Implementation :-

The h/w implementation of Booth's algorithm is shown in fig. It consist of n-bit adder, shift, Add & subtract control logic & 4 registers A, M, Q & Q_{-1}. The multiplier & multiplicand are loaded into register Q & M respectively, & register A & Q_{-1} are set to be 0.



Initial setting: A ← 0 & $Q_{-1} = 0$

The n-bit adder performs addition of two inputs. One i/p is the A register & other i/p is multiplicand.

Case 1 :- In case of addition Add/sub line is 0, therefore Cin=0 & multiplicand is directly applied as second i/p to the n-bit adder.

Case 2 :- In case of subtraction, Add/sub line is 1 therefore, Cin=1 & multiplicand is complemented & then applied to the n-bit adder. As a result 2's complement of multiplicand is added in the A register.

An algorithm :-

Step 1 :- The multiplicand & multiplier are placed in M & Q registers respectively. A 1-bit register is placed to the right of the LSB (Q_0) of Q register & named as Q_{-1}. The result of the

multiplication will appear in A & Q register. A & Q_{-1} are initialized to 0. Count is initialized to number of bits in Q.

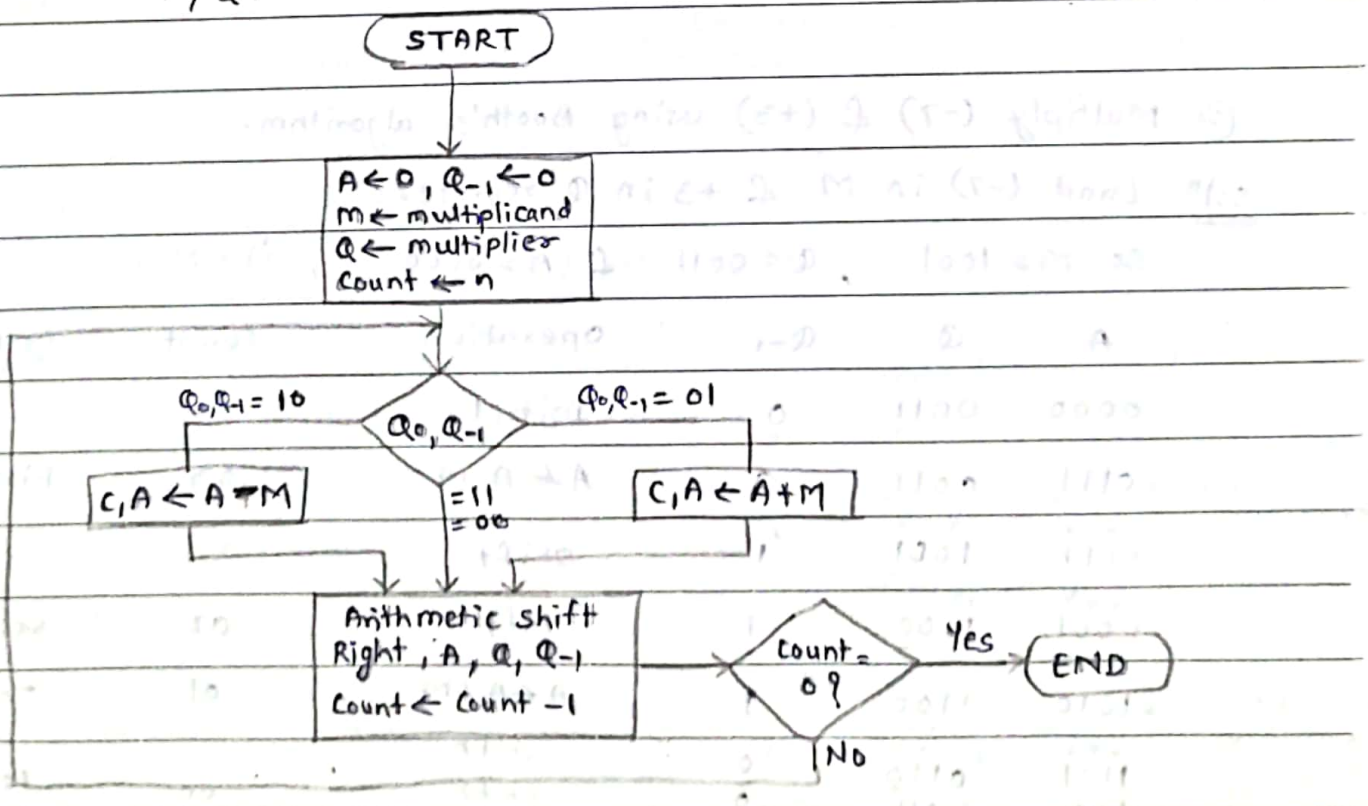
step 2 :- Control logic scans the bits of the multiplier Q & Q_{-1}.

Case 1 :- If the two bits are same (1-1 or 0-0), then all of the bits of A, Q & Q_{-1} registers are shifted to right by 1-bit.

Case 2 :- If two bits are (0-1 or 1-0), then multiplicand (M) is added to or subtracted from register A. Following addition or subtraction, the right shift is done. In either case, the right shift is such that the leftmost bit of A, namely A_{n-1} not only is shifted into A_{n-2} but also remains in A_{n-1}. This is required to preserve the sign of the no. in A & Q. It is known as 'Arithmetic shift' as it preserves the sign bit.

step 3 :- The count is decremented by 1 after case 1 or 2 & the step 2 is repeated till the count is not equal to 0.

step 4 :- When the count is 0, the final product result appears in A, Q.



(1) Multiply 7 & 3 using Booth's algorithm :-

Soln Multiplicand (7) = 0111 = (M)

Multiplier (3) = 0011 = (Q)

(-M) or (M') = 1001

Initial settings A = 0000 Q_{-1} = 0

A	Q $Q_3 Q_2 Q_1 Q_0$	Q_{-1}	Operation	Count	Cycle
0000	0011	0	Initial	04	
A = 0000 + (-M) = 1001 → 1001	0011	0	A ← A-M		First cycle
1100	1001	1	shift		
1110	0100	1	Shift	03	second cycle
A = 0110 + M = 0111 → 0101	0100	1	A ← A+M	02	Third cycle
0010	1010	0	Shift		
0001	0101	0	Shift	01	Fourth cycle.

Product Result in A & Q.

Final product result of (7) x (3) = (21) which is in A & Q register (00010101).

(2) Multiply (-7) & (+3) using Booth's algorithm.

Soln Load (-7) in M & +3 in Q register.

So M = 1001 , Q = 0011 & A = 0000 , (M') = 0111

A	Q $Q_3 Q_2 Q_1 Q_0$	Q_{-1}	Operation	Count	Cycle
0000	0011	0	Initial	04	
A = 0000 + M' = 0111 → 0111	0011	0	A ← A-M	03	First cycle.
0011	1001	1	Shift		
0001	1100	1	Shift	02	second cycle.
A = 0001 + M = 1001 → 1010	1100	0	A ← A+M	01	Third cycle
1101	0110	0	shift		
1110	1011	0	shift	00	Fourt cycle.

Final product result of $(-7) \times (+3) = (-21)$ which in A & Q register after taking 2's complement. So 2's complement of 11101011 is (00010101) is the product result (-21).

③ Multiply $-7 = 1001$ & $-3 = 1101$

Soln Load -7 in M, -3 in Q & A = 0000

A	Q	Q-1	Operation	Count	cycle
0000	1101	0	Initial	04	
0111	1101	0	A ← A-M	03	First cycle
0011	1110	1	shift		
1100	1110	1	A ← A+M	02	second cycle
1110	0111	0	shift		
0101	0111	0	A ← A-M	01	Third cycle
0010	1011	1	shift		
0001	0101	1	shift	00	Fourth cycle.

Product Result in A & Q register.

Final product result of (-7) & (-3) is in A & Q register which is (00010101) = (+21).

* Binary Division :-

We know that binary multiplication can be carried out as a series of addition & shift operation, division can be carried out as by a series of subtraction & shift operation.

```
          14   ← Quotient
Division → 12 ) 169  ← Dividend
           -012
           ----
            049
           - 48
           ----
             01  ← Remainder.
```

```
            1110
        ------------
1100 ) 10101001
       -1100
       ------
        10010
        1100
        ----
         1100
         1100
         ----
         00001
```

So, Dividend = Quotient × Divisor + Remainder

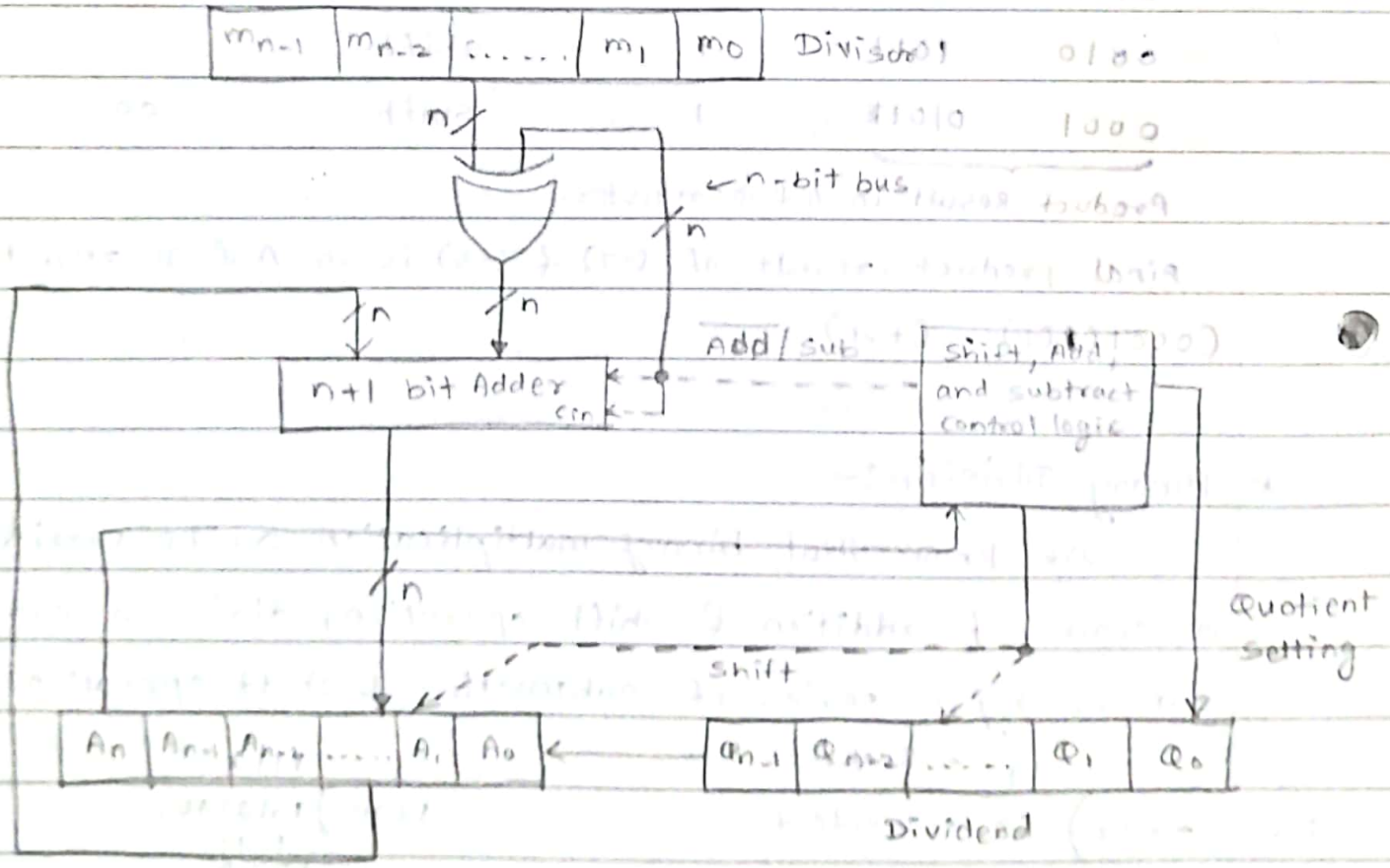
* Restoring Division Technique :-

Case1 :- If remainder is zero or +ve, a quotient is 1 of bit 1 is determined, the remainder is extended by another bit of the dividend, the divisior is repositioned & another subtraction is performed.

Case2 :- If remainder is -ve, a quotient bit of 0 is determined, the dividend is restored by adding back the divisor & the divisor is repositioned for another subtraction.

H/w implementation :-

Fig. H/w implementation of Restoring Binary Division.



An n-bit positive divisor is loaded into register M & an n-bit positive dividend is loaded into register Q at the start of the operation. Register A is set to 0. After division is complete, n-bit quotient is

in register Q & the remainder is in register A. The required subtractions are done by using 2's complement arithmetic.

Restoring Division Algorithm :-

The divisor is placed in M register, the divided placed in Q register. The A & Q registers together are shifted to the left by 1- bit. M is subtracted from A to determine whether A divides the partial remainder. If it does, then Q_0 gets a bit. otherwise, Q_0 gets a bit & M must be added back to A to restore the previous value. The count is decremented & the process continues for n steps. At the end, quotient is in the Q register & the remainder is in the A register.

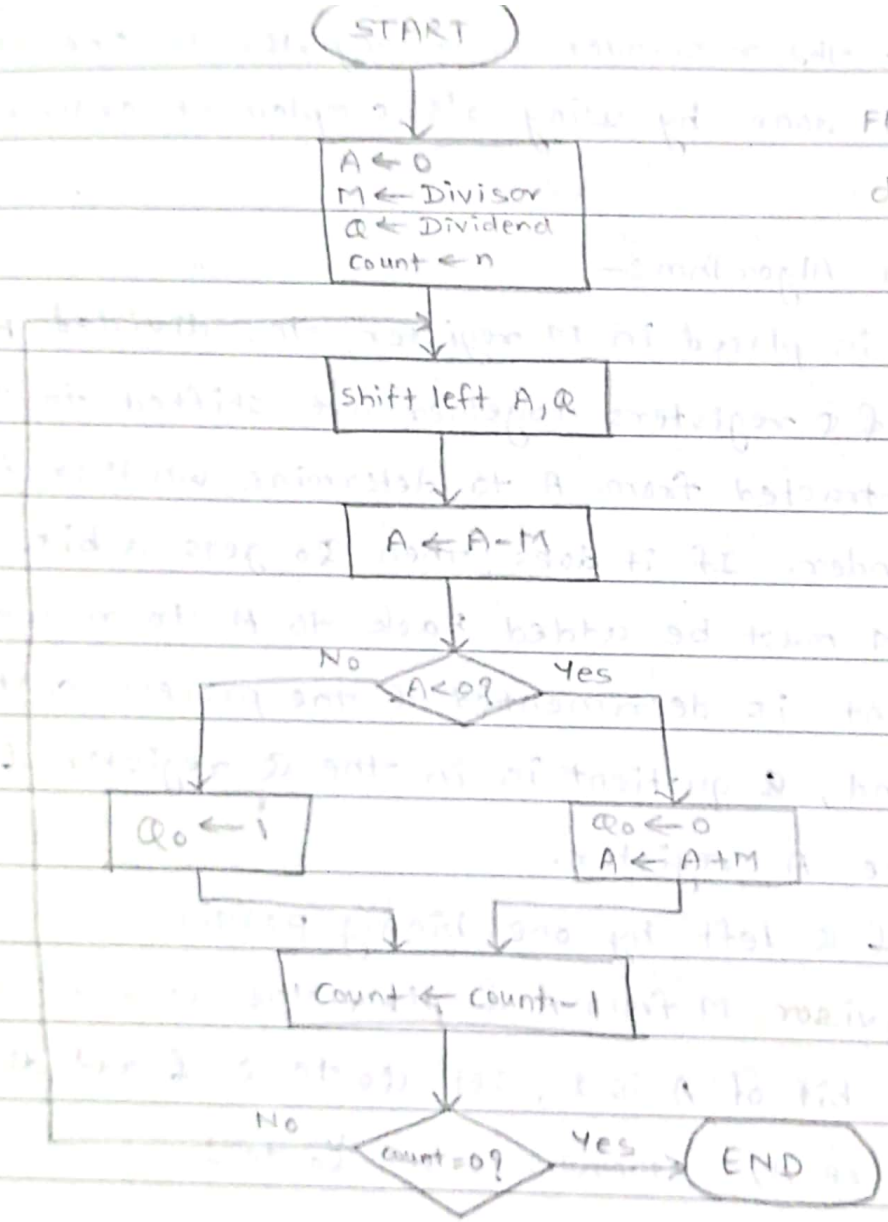
Step 1 :- Shift A & Q left by one binary position.

Step 2 :- Subtract divisor M from A & place the answer in A ($A \leftarrow A-M$).

Step 3 :- If the sign bit of A is 1, set Q_0 to 0 & add divisor back to A. (i.e. restore A), otherwise set Q_0 to 1.

Step 4 :- Repeat above steps n times.

This algorithm needs restoring of register A after each unsuccessful subtraction (subtraction is said to be unsuccessful if the result is negative). This algorithm is referred to as 'restoring division algorithm'



Flowchart for restoring division operation.

Quotient in Q.
Remainder in A.

e.g. ① Divide $\underset{8}{1000}$ by $\underset{3}{11}$

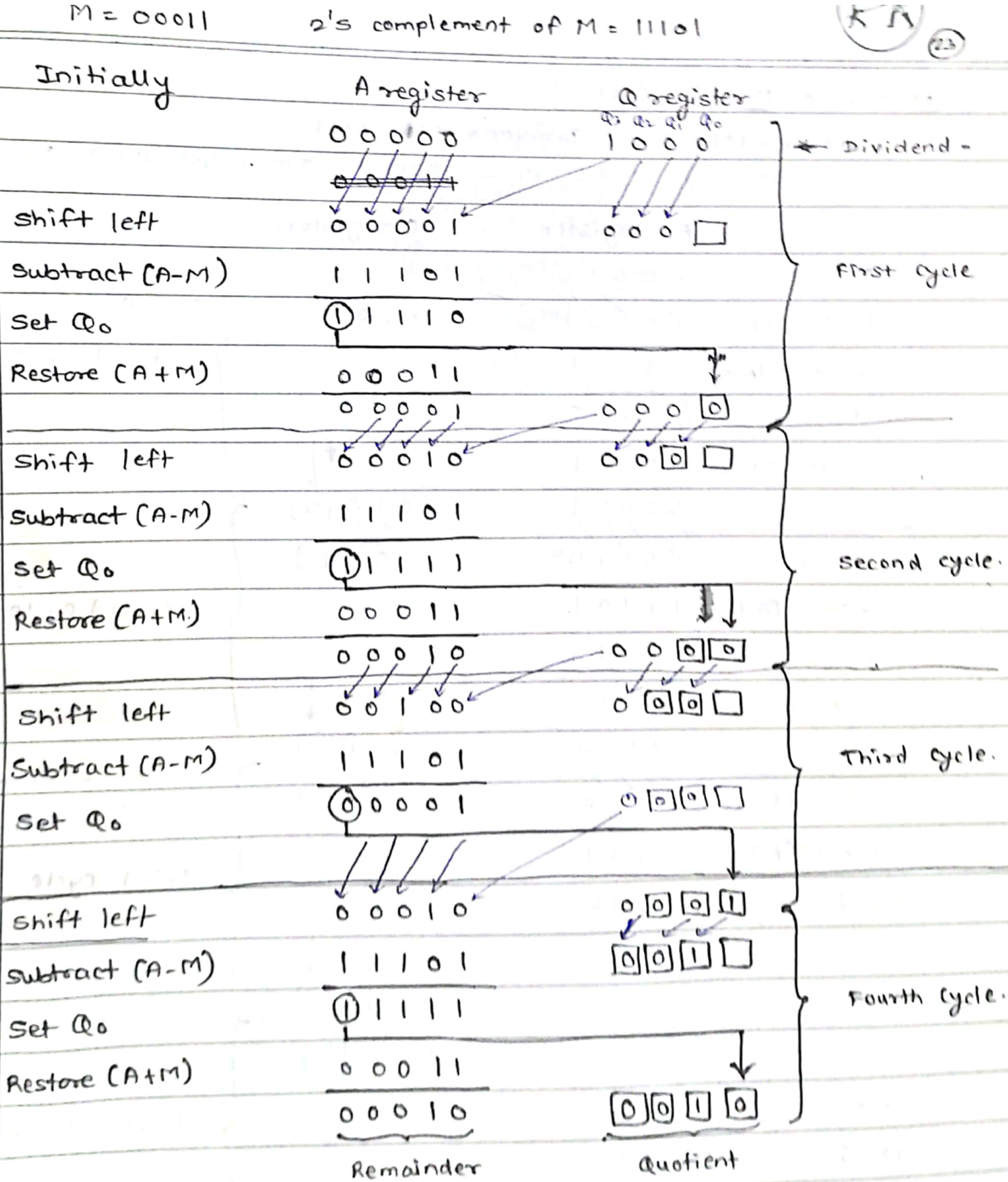
Soln

$$11 \overline{)\,1000\,} \quad 10$$
$$11$$
$$010$$

Dividend = 1000 → Q

Divisor = 11 = 0011 → M

M = 00011 2's complement of M = 11101



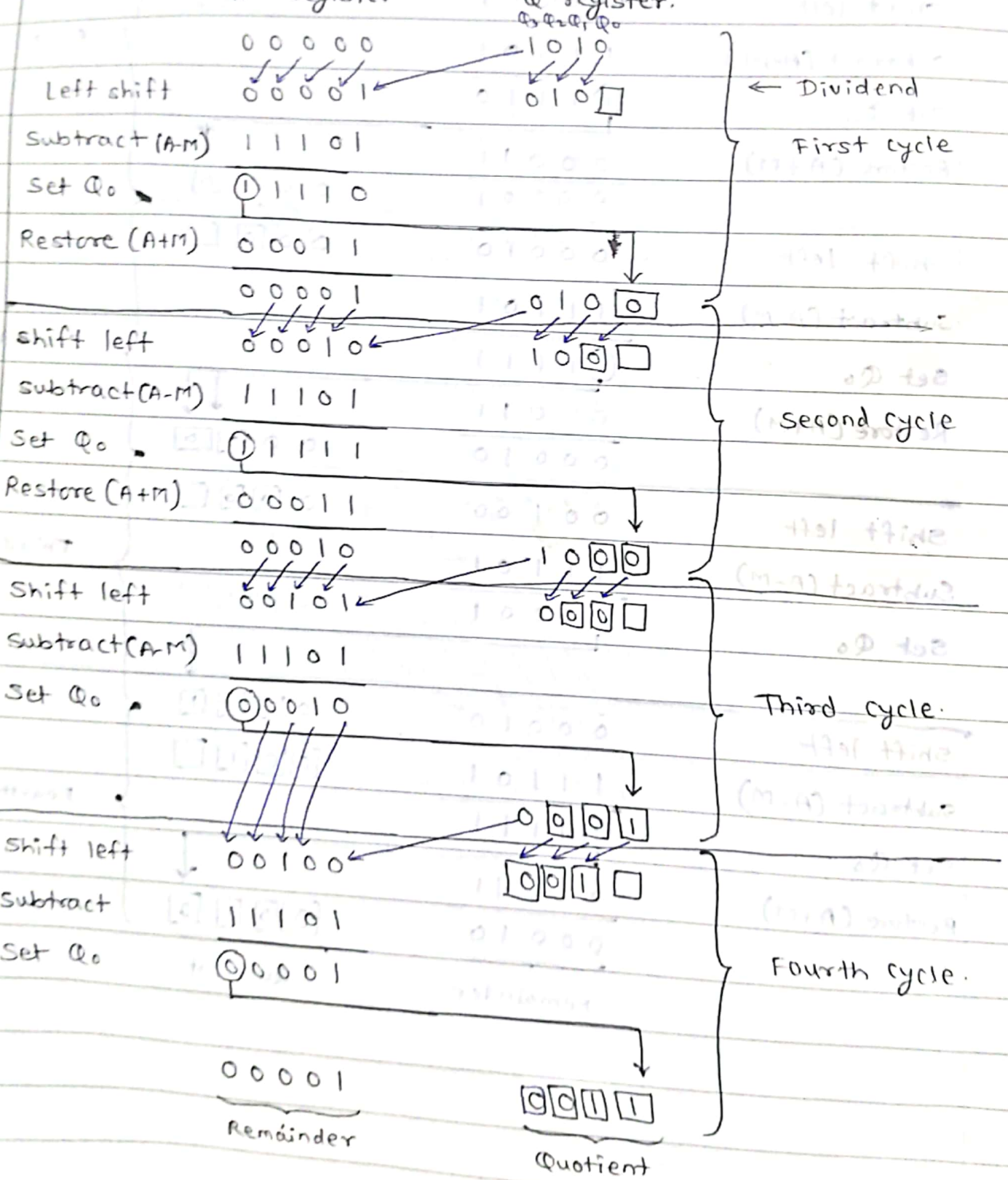
(3) Divide 1010_{10} by 0011_3.

Sol^n Dividend = 1010

Q = 1010

Divisor = ~~0010~~ 0011

M = 00011 Two's complement = 11101



Non-Restoring Division Technique :-

Consider the sequence of operations that takes place after the subtraction operation in the preceding algo. If A is +ve, we shift left & subtract M, i.e. we perform 2A-M. If A is -ve, we store it by performing A+M & then we shift it left & subtract M. This is equivalent to performing 2A+M. The Q_0 bit is set to 0 or 1, after the correct operation has been performed.

Algorithm :-

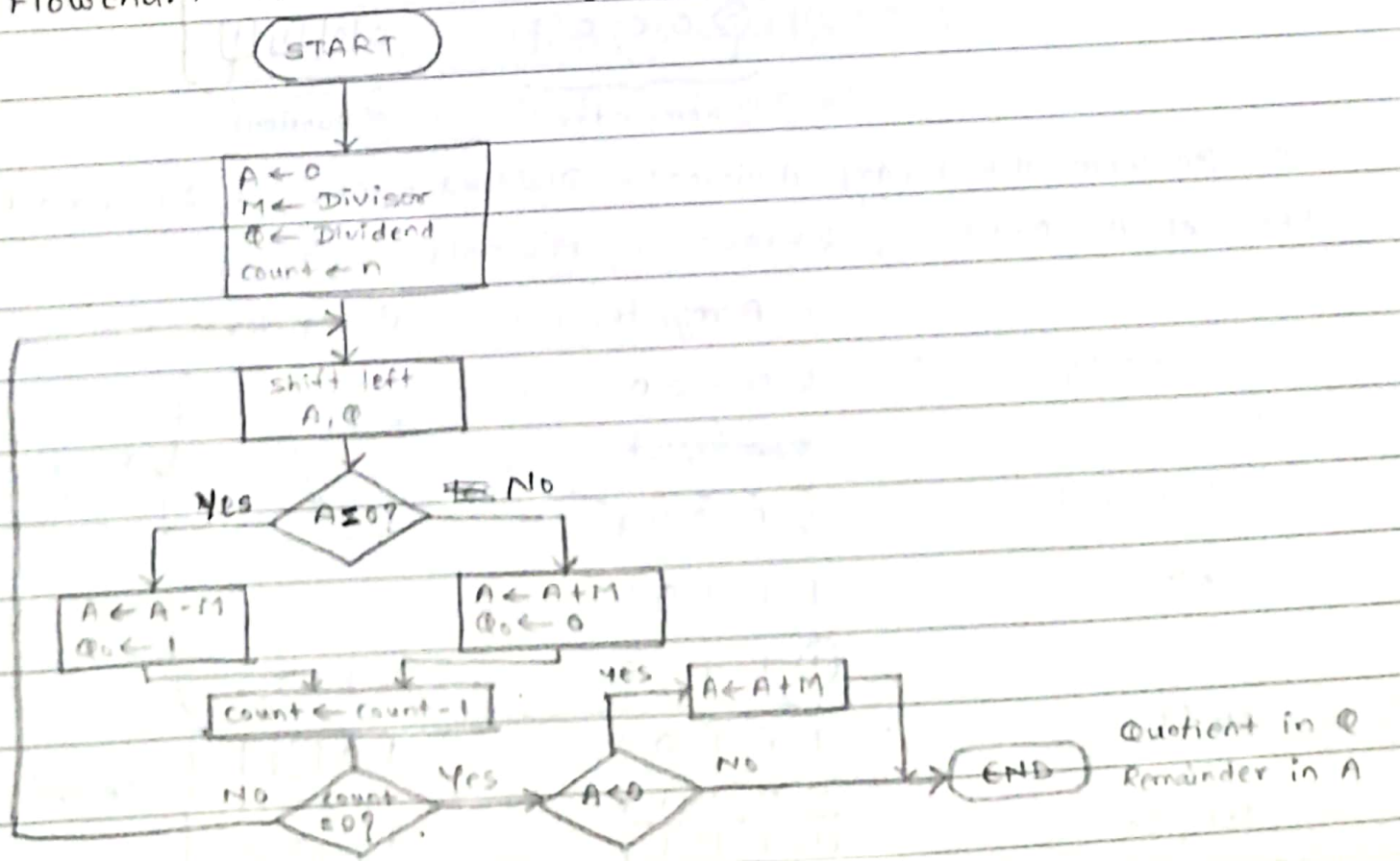
Step 1 :- If sign of A is 0, shift A & Q left by one bit position & subtract divisor from A, otherwise shift A & Q left & add divisor to A.

Step 2 :- If sign of A is 0, set Q_0 to 1, otherwise set Q_0 to 0.

Step 3 :- Repeat steps 1 & 2 for n times.

~~Step 4 :- If sign of A is ±, add divisor to A.~~

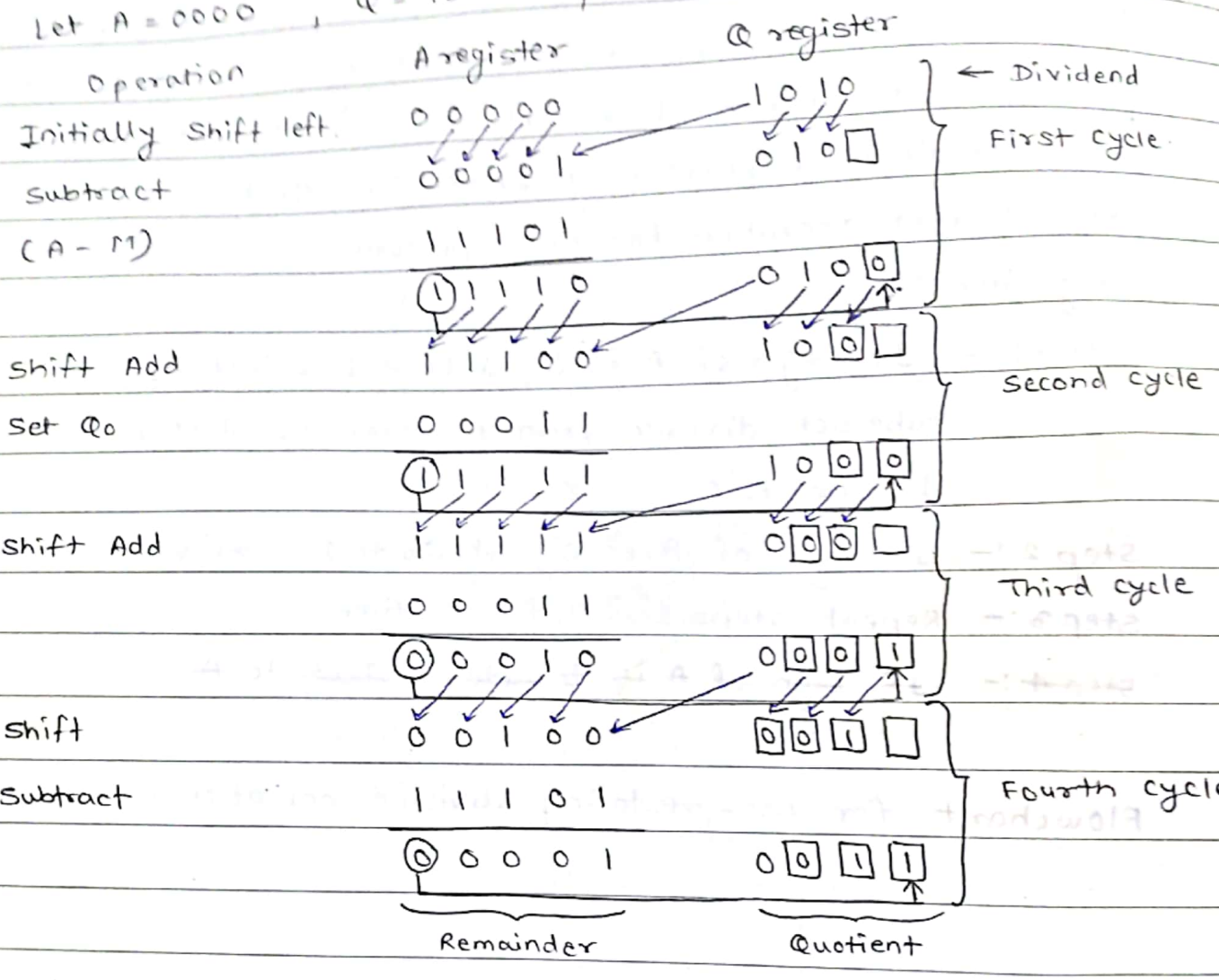
Flowchart for non-restoring division operation is -



eg ① Perform the binary division (10/3)

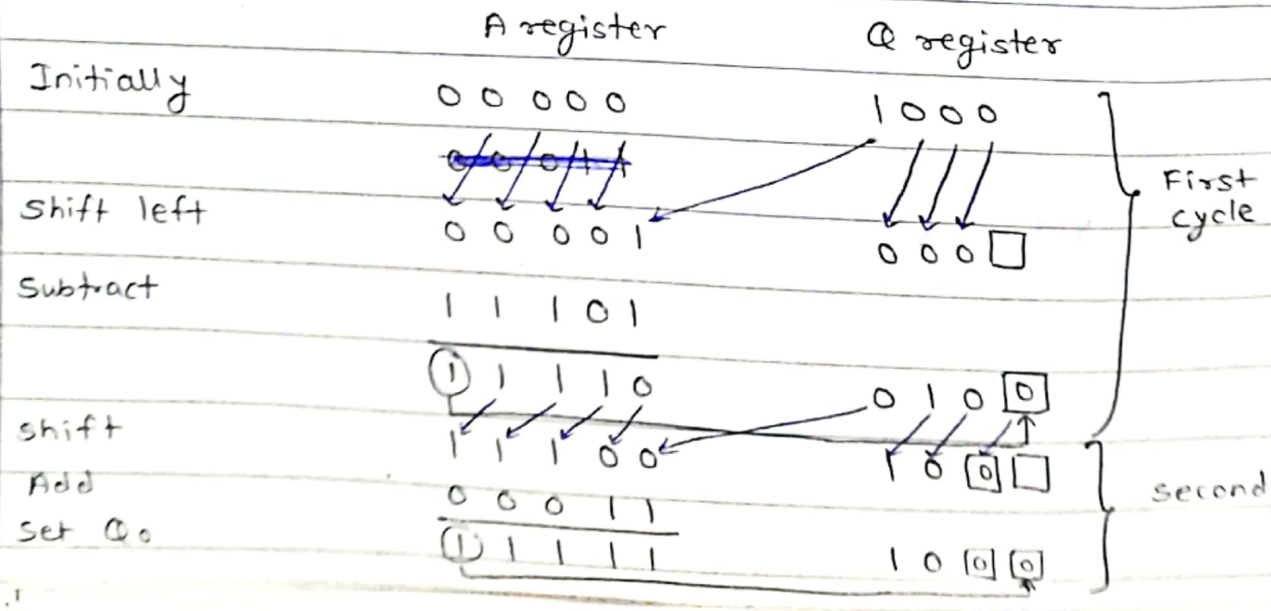
Dividend = 1010 = Q & Divisor = 0011 = M

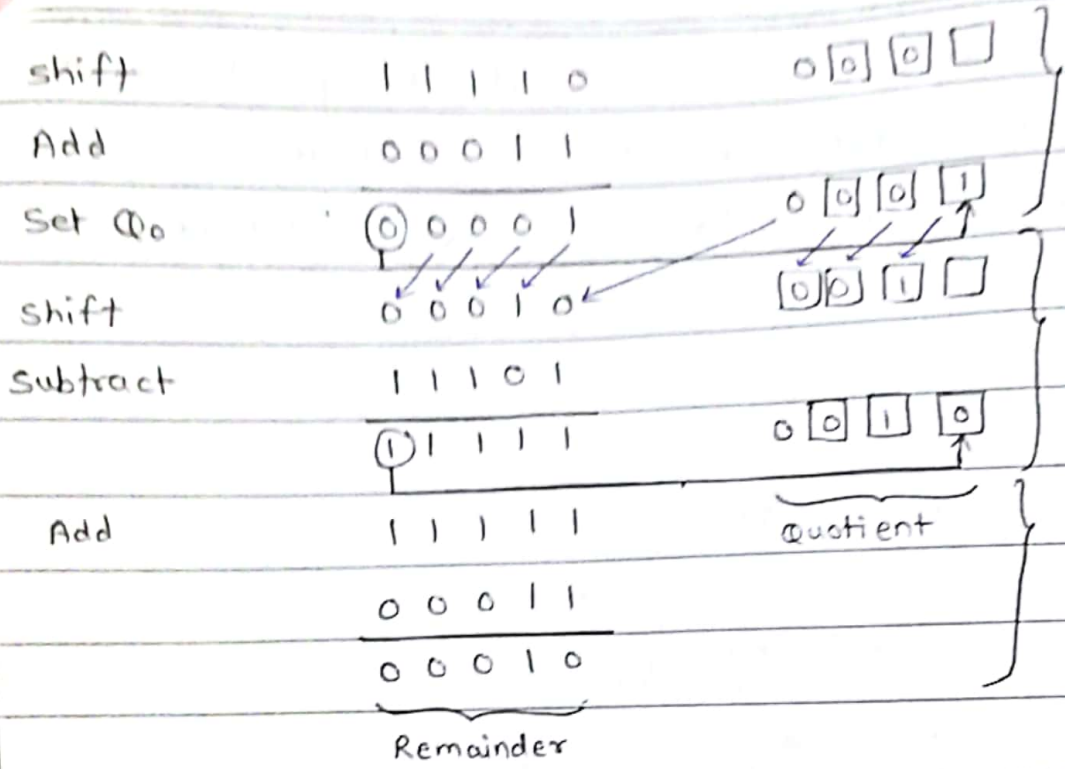
Sol^n Let A = 0000 , Q = 1010 , M = 0011 M' = 11101



② Perform the binary division :- Dividend = 1000 & Divisor = 0011

Sol^n Let A = 0000 , Q = 1000 , M = 0011





Division of signed Numbers :-

The restoring & non-restoring division techniques can be extended to signed no.s in the same way, as multiplication. By using two's complement no. approach the ~~ea~~ algorithm can be summerized as follows :-

Algorithm :-

Step 1 :- Load the divisor into M register & dividend into A & Q registers. The dividend must be a 2n-bit two's complement no. e.g. 4-bit 0010 becomes 00000010 & 1010 becomes 11111010.

Step 2 :- Shift A, Q left 1-bit position.

Step 3 :- If M & A have same signs, perform $A \leftarrow A-M$, otherwise $A \leftarrow A+M$.

Step 4 :- The above operation is successful if the sign of A is same before & after the operation.

(a) If operation is successful or $(A=0$ & $Q=0)$, then set $Q_0 \leftarrow 1$.

(b) If operation is unsuccessful & $(A=0$ OR $Q=0)$, then set $Q_0 \leftarrow 0$ & restore the previous value of A.

Step 5 :- Repeat step 2 through 4 as many times as there are bit positions in Q.

Step 6 :- The remainder is in A. If the sign of the divisor & dividend were same, then Quotient ~~in~~ is in Q, otherwise the quotient is ~~in~~ the two's complement of Q.