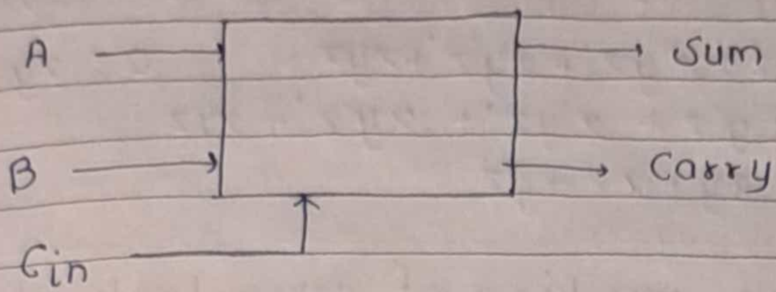
Assignment No. 2

GoodLuck | Page No.
Date

Q.1. Design full adder and write the equations for sum and carry.

The full adder is used to add three 1-bit binary numbers A, B and carry C. The full adder has three input states and two output states i.e. sum and carry.



Truth table

Inputs			Outputs	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

In the above table,

1. 'A' and 'B' are the input variables. These variables represent the two significant bits which are going to be added.
2. 'Cin' is the third input which represents the carry from the previous lower significant position. the carry bit is fetched.
3. The 'Sum' and 'carry' are the output variables that define the output values.

4 The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

x \ yz	00	01	11	10
0		1		1
1	1		1	

x \ yz	00	01	11	10
0			1	
1		1	1	1

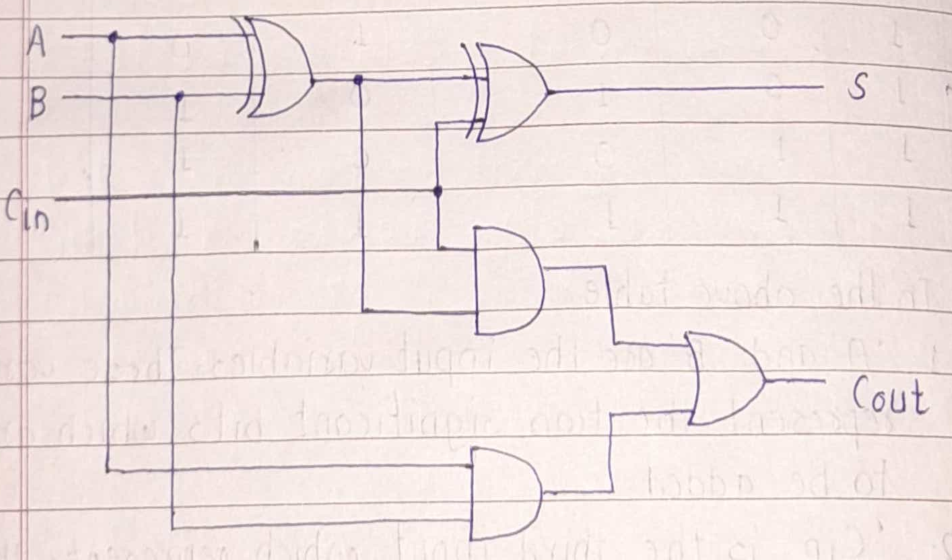
$S = x'y'z + x'yz' + xy'z' + xyz$ $\quad$ $C = xy + xz + yz$

Sum : $x'y'z + x'yz' + xy'z' + xyz$

Carry : $xy + xz + yz$

Q.2. Explain the working of carry look ahead adder with neat diagram

1. Carry look-ahead adder reduces the propogation delay by introducing more complex hardware.
2. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic.



A	B	C	C+1	Condition
0	0	0	0	No carry generate
0	0	1	0	
0	1	0	0	

A	B	C	C+1	condition
0	1	1	1	No carry
1	0	0	0	propogate
1	0	1	1	
1	1	0	1	Carry
1	1	1	1	propogate

3. Consider the full adder circuit shown above with corresponding truth table we define two variables as 'carry generate (G_i)' and 'carry propogate (P_i)' then.

$P_i = A_i \oplus B_i \qquad G_i = A_i B_i$

4. The sum output and carry output can be expressed in terms of carry generate G_i and carry propogate P_i as

$C_{i+1} = G_i + (P_i \cdot C_i)$

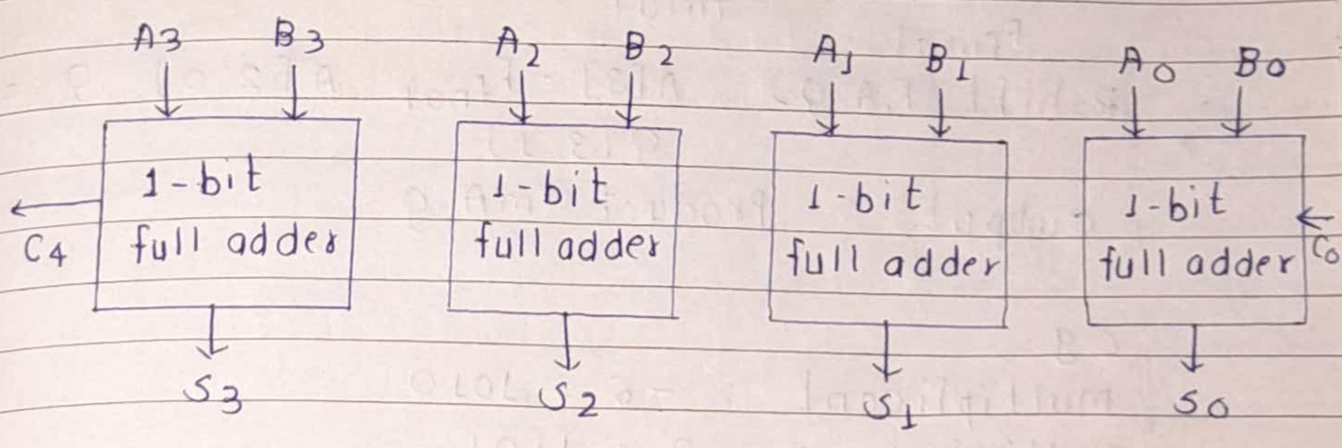
5. where G_i produces the carry when both $A_i . B_i$ are 1 regardless of the input carry.
6. The carry output boolean function of each stage in a 4 stage carry look-ahead adder can be expressed as.

$C_1 = G_0 + P_0 C_{in}$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$

$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$



Block diagram of carry look-ahead adder

Q.3 Explain Robertson's algorithm with example
Robertson's algorithm is used for two's complement multiplication
Depending on the sign of the two operands X and Y, there are 4 cases to be considered:

1. $x_0 = y_0 = 0$ i.e. both X and Y are positive. multiplication of these numbers is similar to the multiplication of unsigned numbers
2. $x_0 = 1$, $y_0 = 0$ i.e. X is negative and Y is positive. In this case, the partial product is always positive
3. $x_0 = 0$, $y_0 = 1$ i.e. X is positive and Y is negative is this case the product is positive
4. $x_0 = 1$. $y_0 = 1$ i.e. both x and y are negative

Robertson's algorithm is as follow:

Begin : A = 0, count = 0, F := 0
Input : M = multiplicant, Q = multiplier
ADD : A [3:0] := A [3:0] + (M[3:0] + Q[0]);
Calculate F_{new} : F_{new} := (M[3] and Q[0] or F_{old});
Rshift (F,A,Q) : A[3] := F_{new}, A [2:0], Q = A : Q [3:1]
count = count + 1;

Test : If count ≠ 3 then go to ADD.
Substract : A [3:0] := A[3:0] - (M[3:0] × Q[0]);
Calculate F_{final} : F_{final} := M[3] xor Q[0];
Rshift (F, A, Q) : A[3] = F_{final}, A [2:0], Q := A. Q [3:1]

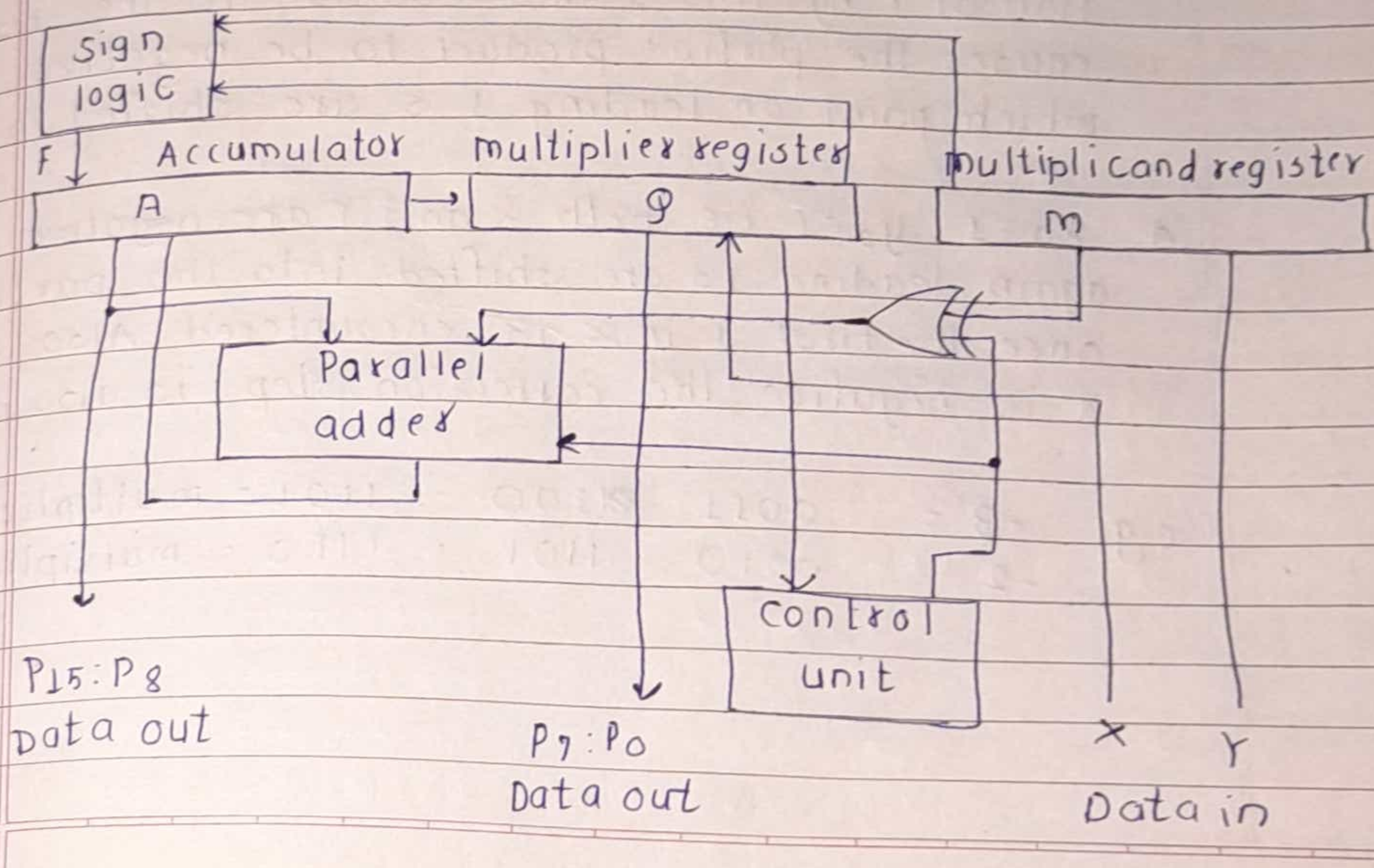
Output : Product MA Q

e.g.
multiplicant : -6 = 1010
Multiplier : -3 = 1101

F	A	Q	count	Action
0	0000	1101	0	Initialization
	1010			ADD M
	1010			
1	1101	0110	1	calculate Fnew
	0000			Rshif, count = count+1 ADD 0
1	1101			
	1110	1011	2	Rshift FAQ
	1010			
1	1000			
	1100	0101	3	
	0110			ADD -M
0	1 0010			Calculate F final
	0001	0010		Rshift FAQ

The answer is 00010010

Q.4. With neat diagram explain the working of 2's complement multiplication using Robertson's method.



Depending on the sign of the two operands X and Y there are 4 cases to be considered:

1. $x_0 = y_0 = 0$, i.e. both X and Y are positive. Hence multiplication of these numbers is similar to the multiplication of unsigned numbers. In other words, the product P is computed in a series of add-and-shift steps of the form

$P_i \leftarrow P_i + x_j + Y$

$P_{i+1} \leftarrow P_i + 2^{-1}$ and the process repeats

2. $x_0 = 1$, $y_0 = 0$, that is, X is negative and Y is positive. In this case, the partial product is always positive (till the sign bit x_0 is used). In the final step a subtraction is performed. i.e. $P \leftarrow P - Y$

3. $x_0 = 0$, $y_0 = 1$, i.e. X is positive and Y is negative. In this case, the partial product is positive and hence leading 0s are shifted into the partial product until the first 1 in X is encountered. Multiplication of Y by this 1 and addition to the result causes the partial product to be negative, from which point on leading 1s are shifted in

4. $x_0 = 1$, $y_0 = 1$ i.e. both X and Y are negative. Once again, leading 1s are shifted into the partial product once the first 1 in X is encountered. Also, since X is negative, the correction step is also performed.

e.g.	-3 =	0011	~~0~~1100	= 1101	= multiplicant
	-2 =	0010	1101	= 1110	= multiplier

GoodLuck Page No.
Date / /

F	A	Q	count	Action
0	0000	1110	0	
	0000			
	0000			
0	0000	0111	1	
	1101			
	1101			
1	1110	1011	2	
	1101			
	1 1011			
1	1101	1101	3	
	0011			
	1 0000			
0	0000	0110		

The answer is 00000110

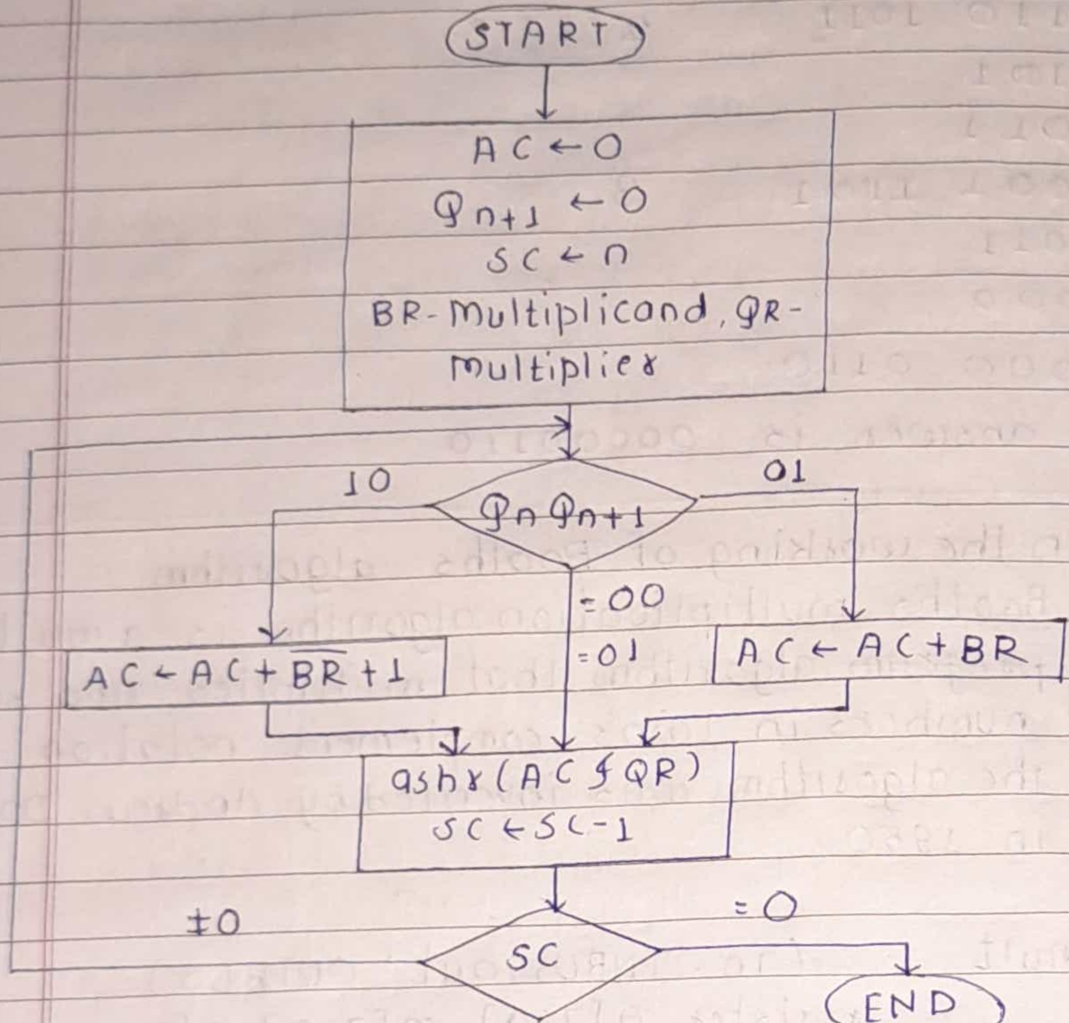
Q.5. Explain the working of Booth's algorithm

Booth's multiplication algorithm is a multiplication ~~program~~ algorithm that multiplies two signed binary numbers in two's complement notation.

The algorithm was invented by Andrew Donald Booth in 1950.

```
Boothmult      (in: INBUS; out: OUTBUS);
               register A[7:0], m[7:0], q[7:-1], COUNT
               [2:0];
               bus INBUS[7:0], OUTBUS[7:0];
BEGIN :        A := 0, COUNT := 0
INPUT :        m := INBUS;
               Q[7:0] := INBUS, q[-1] := 0;
SCAN :         if Q[1]q[0]=01 then A[7:0] := A[7:0]
               + m[7:0], go to
               TEST;
               else if Q[1]q[0]=10 then
                  A[7:0] := A[7:0] - m[7:0];
```

TEST: if COUNT = 7 then go to OUTPUT.
RSHIFT: A[7] := A[7], A[6:0]. Q = A.Q[7:0];
INCREMENT: COUNT := COUNT + 1 go to SCAN;
OUTPUT: OUTBUS := A, Q[0] := 0;
OUTBUS := Q[7:0];
end BoothMult;



step	Accumulator	Register Q (Multiplier)	Action
0	00000000	10110011	Initialize registers
	00000000	101100110	set Q[-1] to 0
1	11010101 Multiplicant		
	00101011	101100110	Subtract M from A
	00010101	110110011	Rshift A.Q
2	00010101	110110011	skip add/subtract
	00001010	111011001	Right-shift A.Q

	Action	A	Q
		11010101	
3	ADD m to A	11011111	111011001
	Rshift A.Q	11101111	111101100
	Skip add/subtract	11101111	111101100
4	Rshift A.Q	11110111	111110110
		11010101	
5	Subtract m from A	00100010	111110110
	Rshift A.Q	00010001	011111011
	Skip add/subtract	00010001	011111011
6	Rshift A.Q	00001000	101111101
		11010101	
7	Add m to A	11011101	101111101
	Rshift A.Q	11101110	110111110
		11010101	
8	subtract m from A	00011001	110111110
	Set Q[0] to 0	00011001	110111100 = Product P.

Q.6 Write a non-restoring method for division of 2's complement number.

Non-restoring Division Algorithm comes from the restoring division. The basis of this algorithm is based on paper and pencil approach and the operation involve repetitive shifting with addition and subtraction.

The Non-restoring algorithm works with any combination of positive and negative numbers.

A fast division algorithm for 2's complement numbers base

The algorithm for non-restoring method for division is as follow:

```
NRdivider        (in: INBUS ; out: OUTBUS);
                 register S, A[n-1;0], M[n-1;0], Q[n-1;0],
                 COUNT [log2 n]:0 ;
                 bus INBUS [n-1:0], OUTBUS[n-1:0];
```

GoodLuck Page No. Date

BEGIN : COUNT := 0, S := 0

INPUT : A := INBUS; {Input the left half of the dividend D}

Q := INBUS; {Input the right half of the dividend D}

m := INBUS; {Input the divisor v}

SUBTRACT : S.A := S.A − m; {S is the sign of the result}

TEST : if S = 0 then

begin Q[0] = 1;

if COUNT = n−1 then go to CORRECTION; else

begin COUNT := COUNT+1, S.A.Q [n−1 : 1] := A.Q; end

S.A := S.A − m, go to TEST; end

else { if S = 1 }

begin Q[0] = 0;

if COUNT = n − 1 then go to CORRECTION

else

begin COUNT := COUNT + 1, S.A.Q[n−1:1] = A.Q; end

S.A := S.A + m go to TEST; end

CORRECTION : if S = 1 then S.A := S.A + m;

OUTPUT : OUTBUS := Q; {output the quotient Q}

OUTBUS := A; {output the remainder R}

end NRdivider;

Q.7 Explain non-restoring division algorithm with example

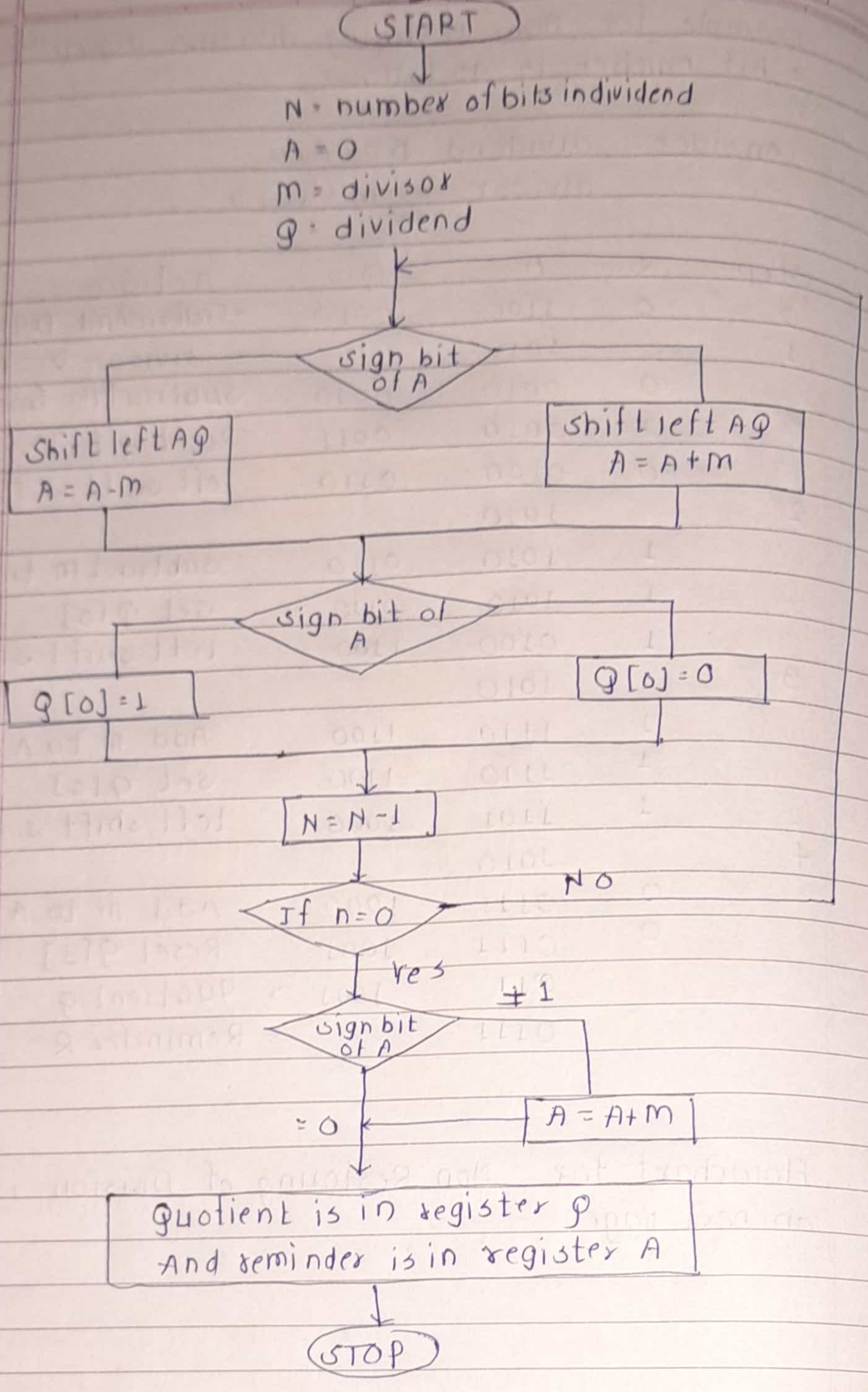
A fast division algorithm for 2's complement numbers based on the nonrestoring approach was devised independently in 1958 by Dura W. Sweeney, James E. Robertson, and Keith D. Tocher.

Example for Non-restoring division algorithm for 4-bit number is as fallow

consider dividend D = 0010
divisor V = m = 1010

Step	S	A	Q	Action
0	0	1100	0010	Initialize registers
1		1010		
	0	0010	0010	subtract m from A
	0	0010	0011	Reset Q [0]
	0	0100	0110	Left shift S.A.Q
2		1010		
	1	1010	0110	subtract m from A
	1	1010	0110	Set Q [0]
	1	0100	1100	Left shift S.A.Q
3		1010		
	1	1110	1100	Add m to A
	1	1110	1100	set Q [0]
	1	1101	1000	Left shift S.A.Q
4		1010		
	0	0111	1000	Add m to A
	0	0111	1001	Reset Q [3]
			1001	= quotient Q
		0111		= Reminder R

Flowchart for Non Restoring of Division is given on next page.


START
N = number of bits in dividend
A = 0
M = divisor
Q = dividend
sign bit of A
Shift left AQ
A = A - M
shift left AQ
A = A + M
sign bit of A
Q[0] = 1
Q[0] = 0
N = N - 1
If n = 0
NO
Yes
sign bit of A
≠ 1
= 0
A = A + M
Quotient is in register Q
And reminder is in register A
STOP