UNIT 6

COMPUTER ARITHMETIC

INTRODUCTION

- Computer Arithmetic includes the arithmetic operation like addition, subtraction, multiplication and division.
- These operations are performed usually in signed 2’s complement.
- However, the processing can be preceded with signed magnitude, signed 1’s complement and signed 2’s complement.
- For every process, we design a hardware and analyze the corresponding algorithm used.

We all are familiar with integer representation and signed magnitude representation.

If not we will discuss later.

+18 = 00010010
1’s complement = 11101101
2’s complement = 11101110 = -18

5.1 Addition Algorithm

5.2 Subtraction Algorithm

1001 = -7
$\underline{\text{0101 = +5}}$
1110 =-2
(a) (-7)+(+5)

1100 = -4
$\underline{\text{0100 = +4}}$
10000 = 0
(b) (-4)+(4)

0011 = 3
$\underline{\text{0100= 4}}$
0111= 7
(c) (+3)+(+4)

1100 = -4
$\underline{\text{1111 = -1}}$
11011 = -5
(d) (-4)+(-1)

0101 =5
$\underline{\text{0100 =4}}$
1001=overflow
(e) (+5)+(+4)

1001 = -7
$\underline{\text{1010 = -6}}$
10011 = overflow
(f) (-7)+(-6)

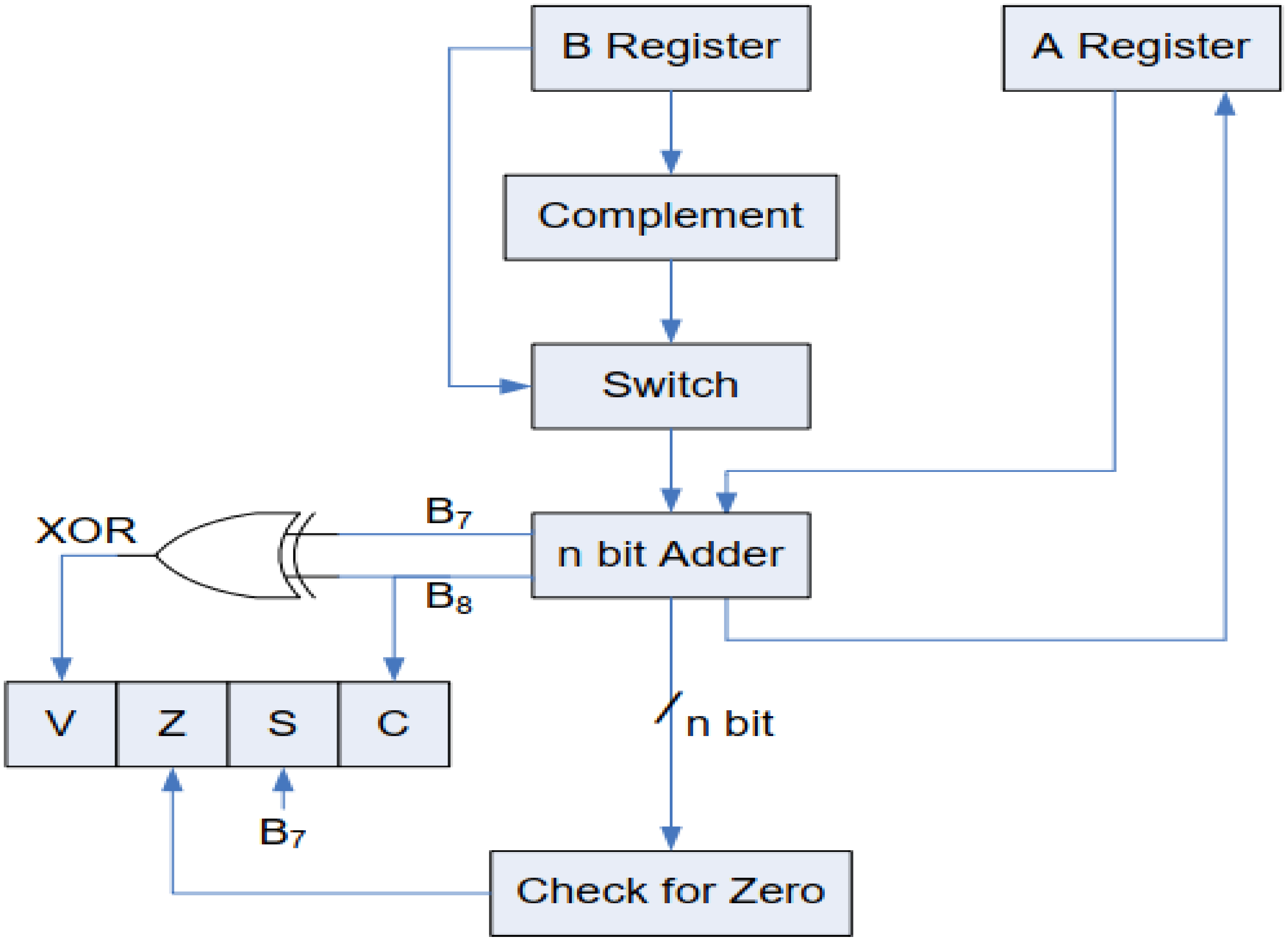


Fig: Block diagram of hardware for addition / subtraction

Multiplication Algorithm(Pen and Pencil)

- The multiplier and multiplicand bits are loaded into two registers **Q and M**. A third register A is initially set to zero.
- **C** is the 1-bit register which holds the carry bit resulting from addition.
- Now, the control logic reads the bits of the multiplier one at a time.
- If **Q_0** is 1, the multiplicand is added to the register **A** and is stored back in register **A** with **C** bit used for carry.
- Then all the bits of **CAQ** are shifted to the right 1 bit so that C bit goes to A_{n-1}, A0 goes to Q_{n-1} and Q_0 is lost.
- If Q_0 is 0, no addition is performed just do the shift.
- The process is repeated for each bit of the original multiplier.
- The resulting 2n bit product is contained in the **QA** register.

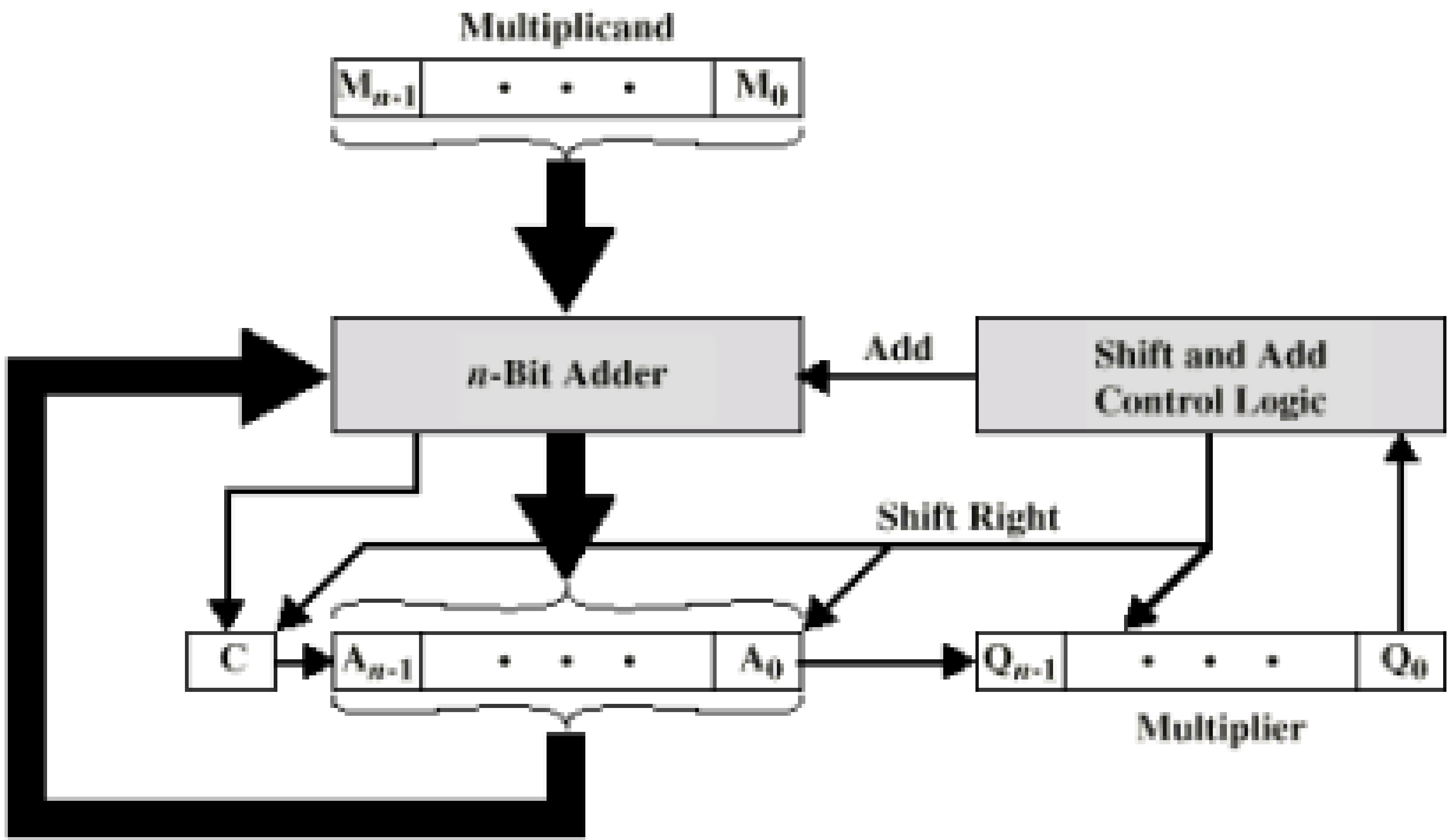
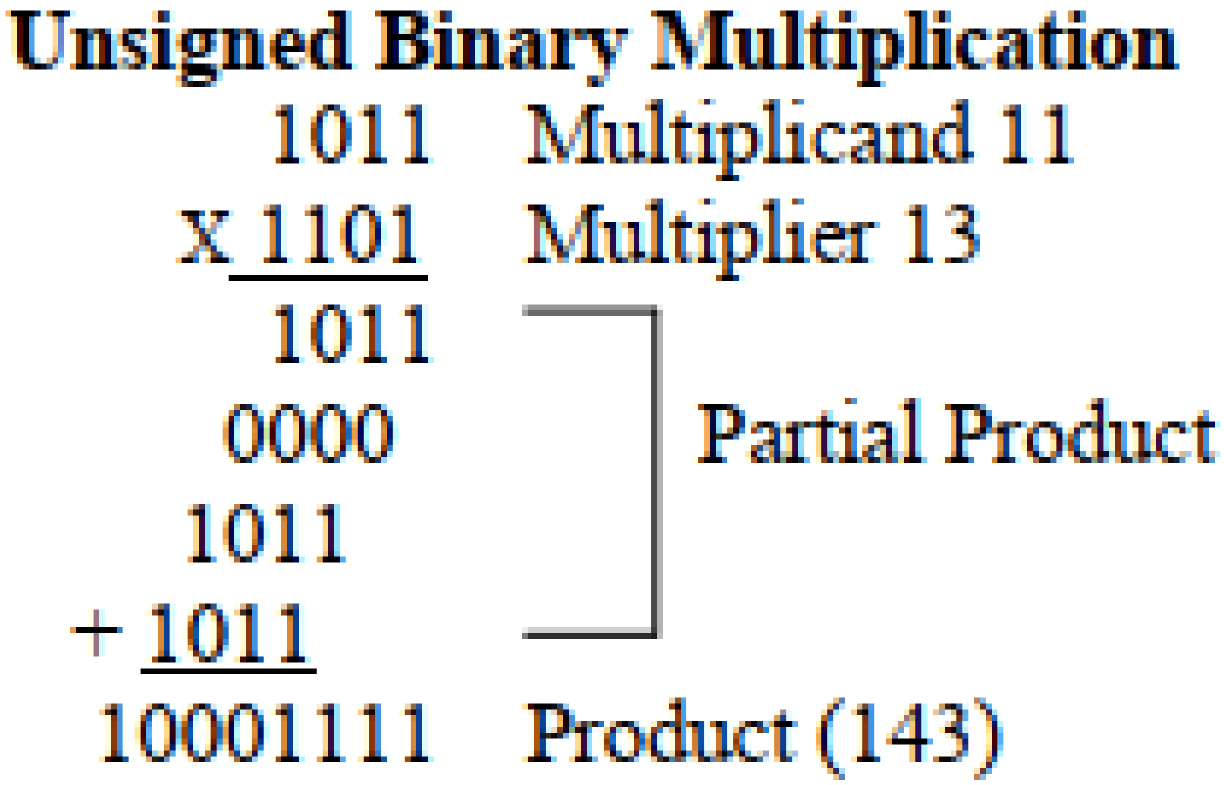


Fig: Block diagram of multiplication

There are three types of operation for multiplication.

- It should be determined whether a multiplier bit is 1 or 0 so that it can designate the partial product.
- If the multiplier bit is 0, the partial product is zero; if the multiplier bit is 1, the multiplicand is partial product.
- It should shift partial product.
- It should add partial product



Unsigned Binary Multiplication

```
       1011    Multiplicand 11
     x 1101    Multiplier 13
       1011  ┐
      0000   │
     1011    │  Partial Product
  + 1011     ┘
   10001111    Product (143)
```

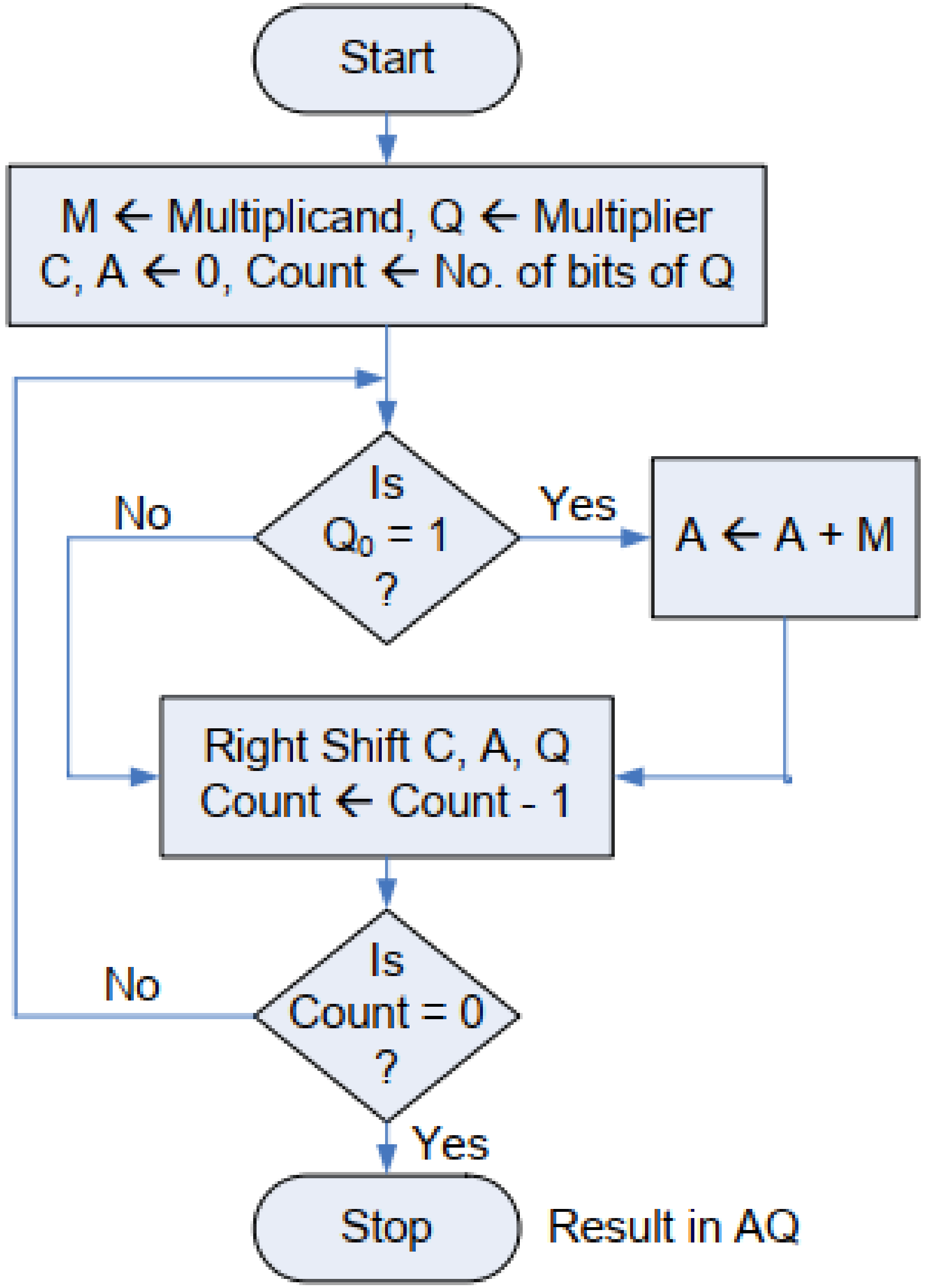


Fig. : Flowchart of Unsigned Binary Multiplication

Example: Multiply 15 X 11 using unsigned binary method

C	A	Q	M	Count	Remarks
0	0000	1011	1111	4	Initialization
0	1111	1011	-	-	Add (A ← A + M)
0	0111	1101	-	3	Logical Right Shift C, A, Q
1	0110	1101	-	-	Add (A ← A + M)
0	1011	0110	-	2	Logical Right Shift C, A, Q
0	0101	1011	-	1	Logical Right Shift C, A, Q
1	0100	1011	-	-	Add (A ← A + M)
0	1010	0101	-	0	Logical Right Shift C, A, Q

Result = 1010 0101 = $2^7 + 2^5 + 2^2 + 2^0$ = 165

Signed Multiplication (Booth Algorithm) – 2's Complement Multiplication

- Multiplier and multiplicand are placed in **Q and M** register respectively.
- There is also one bit register placed logically to the right of the least significant bit **Q_0** of the Q register and designated as **Q_{-1}**.
- The result of multiplication will appear in **A and Q** register.
- **A and Q_{-1}** are initialized to zero if two bits (Q_0 and Q_{-1}) are the same (11 or 00) then all the bits of A, Q and Q_{-1} registers are shifted to the right 1 bit.
- If the two bits differ then the multiplicand is added to or subtracted from the A register depending on weather the two bits are 01 or 10. Following the addition or subtraction the arithmetic right shift occurs.
- When count reaches to zero, result resides into AQ in the form of signed integer $[-2^{n-1}*a_{n-1} + 2^{n-2}*a_{n-2} + \ldots\ldots\ldots\ldots\ldots + 2^1*a_1 + 2^0*a_0]$

Hardware Implementation

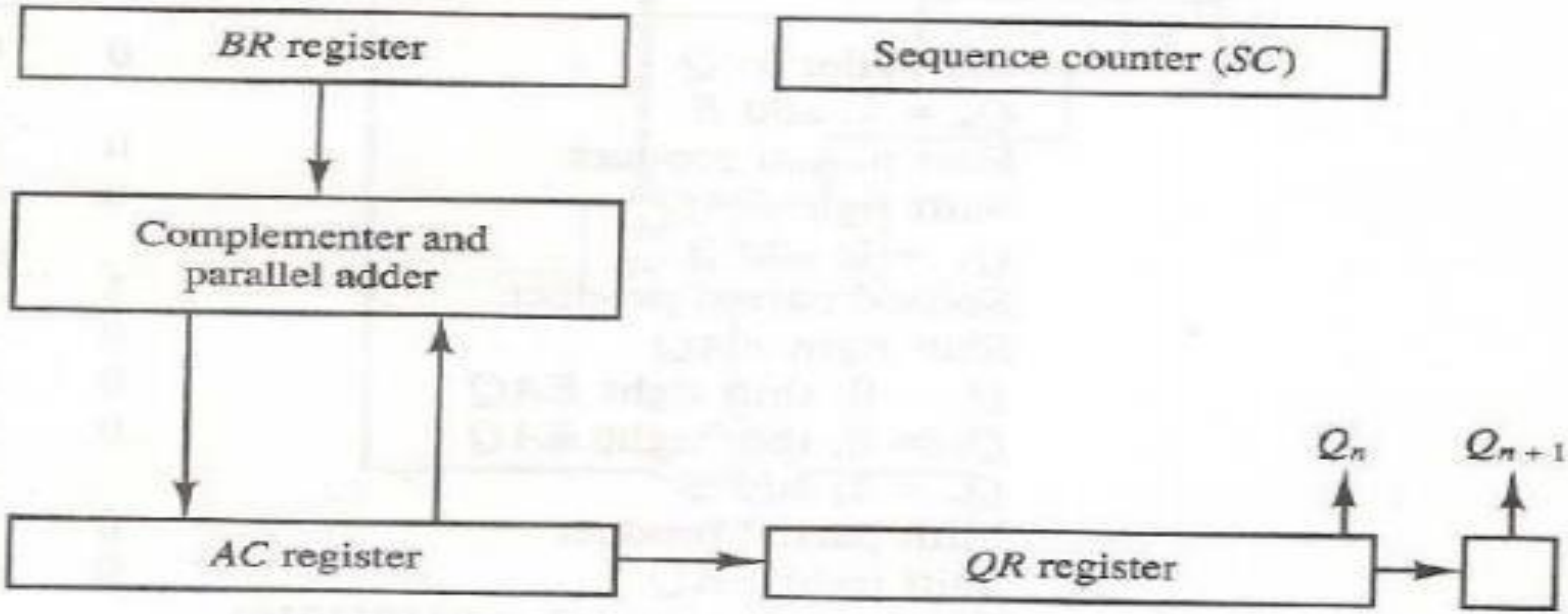


Fig. Hardware for Booth Algorithm

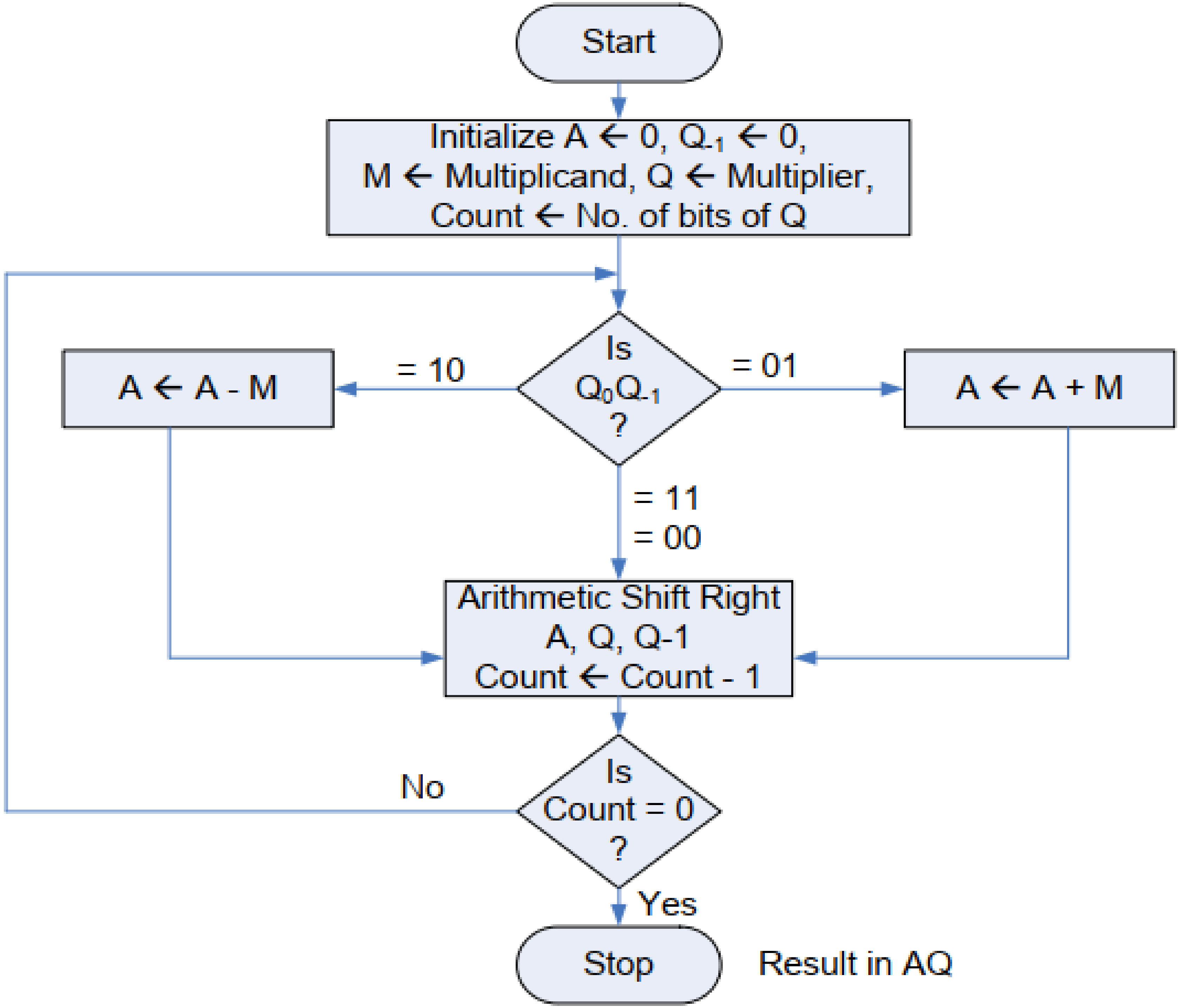


Fig.: Flowchart of Signed Binary Numbers (using 2's Complement, Booth Method)

Example: Multiply 9 X -3 = -27 using Booth Algorithm

+3 = 00011, -3 = 11101 (2's complement of +3)

A	Q	Q_{-1}	Add (M)	Sub ($\overline{M}$ +1)	Count	Remarks
00000	11101	0	01001	10111	5	Initialization
10111	11101	0	-	-	-	Sub (A ← A - M) as Q_0Q_{-1} = 10
11011	11110	1	-	-	4	Arithmetic Shift Right A, Q, Q_{-1}
00100	11110	1	-	-	-	Add (A ← A + M) as Q_0Q_{-1} = 01
00010	01111	0	-	-	3	Arithmetic Shift Right A, Q, Q_{-1}
11001	01111	0	-	-	-	Sub (A ← A - M) as Q_0Q_{-1} = 10
11100	10111	1	-	-	2	Arithmetic Shift Right A, Q, Q_{-1}
11110	01011	1	-	-	1	Arithmetic Shift Right A, Q, Q_{-1} as Q_0Q_{-1} = 11
11111	00101	1	-	-	0	Arithmetic Shift Right A, Q, Q_{-1} as Q_0Q_{-1} = 11

Result in AQ = 11111 00101 = $-2^9+2^8+2^7+2^6+2^5+2^2+2^0$ = -512+256+128+64+32+4+1 = -27

EXERCISE

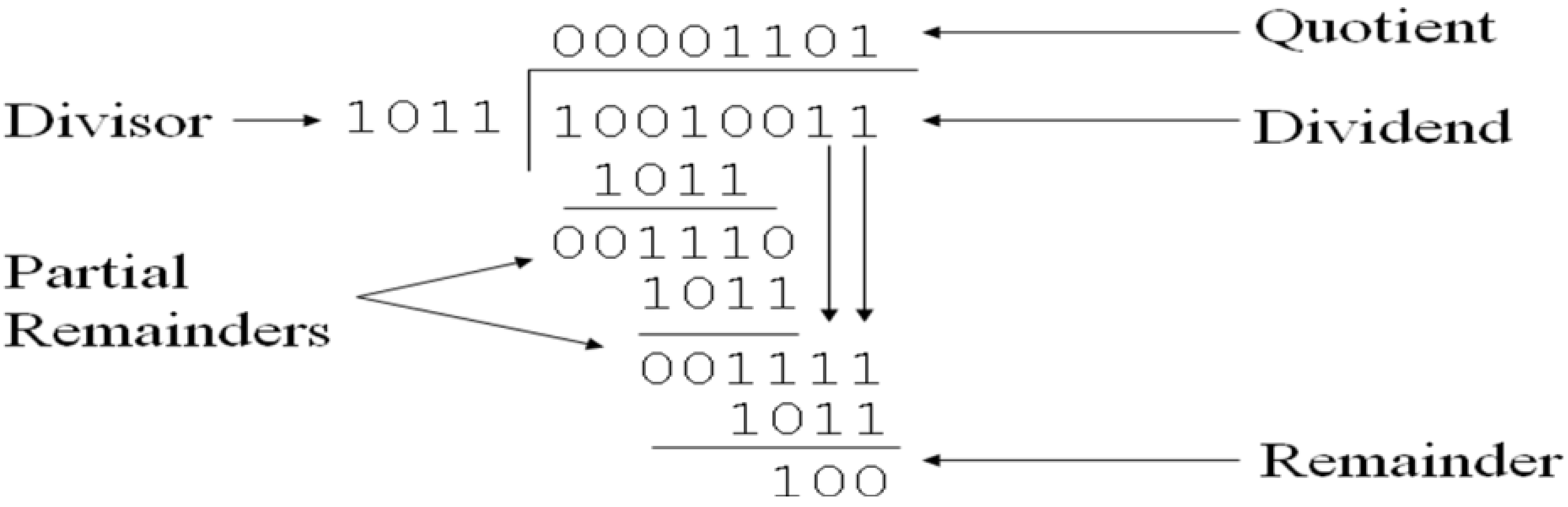
Multiply following using Booth Algorithm:

- -7*19
- -10*20
- -25*-24

Division

- Division is somewhat more than multiplication but is based on the same general principles.
- The operation involves repetitive shifting and addition or subtraction.
- First, the bits of the dividend are examined from left to right, until the set of bits examined represents a number greater than or equal to the divisor; this is referred to as the divisor being able to divide the number. Until this event occurs, 0s are placed in the quotient from left to right.
- When the event occurs, a 1 is placed in the quotient and the divisor is subtracted from the partial dividend. The result is referred to as a partial remainder. The division follows a cyclic pattern.

- At each cycle, additional bits from the dividend are appended to the partial remainder until the result is greater than or equal to the divisor.
- The divisor is subtracted from this number to produce a new partial remainder. The process continues until all the bits of the dividend are exhausted



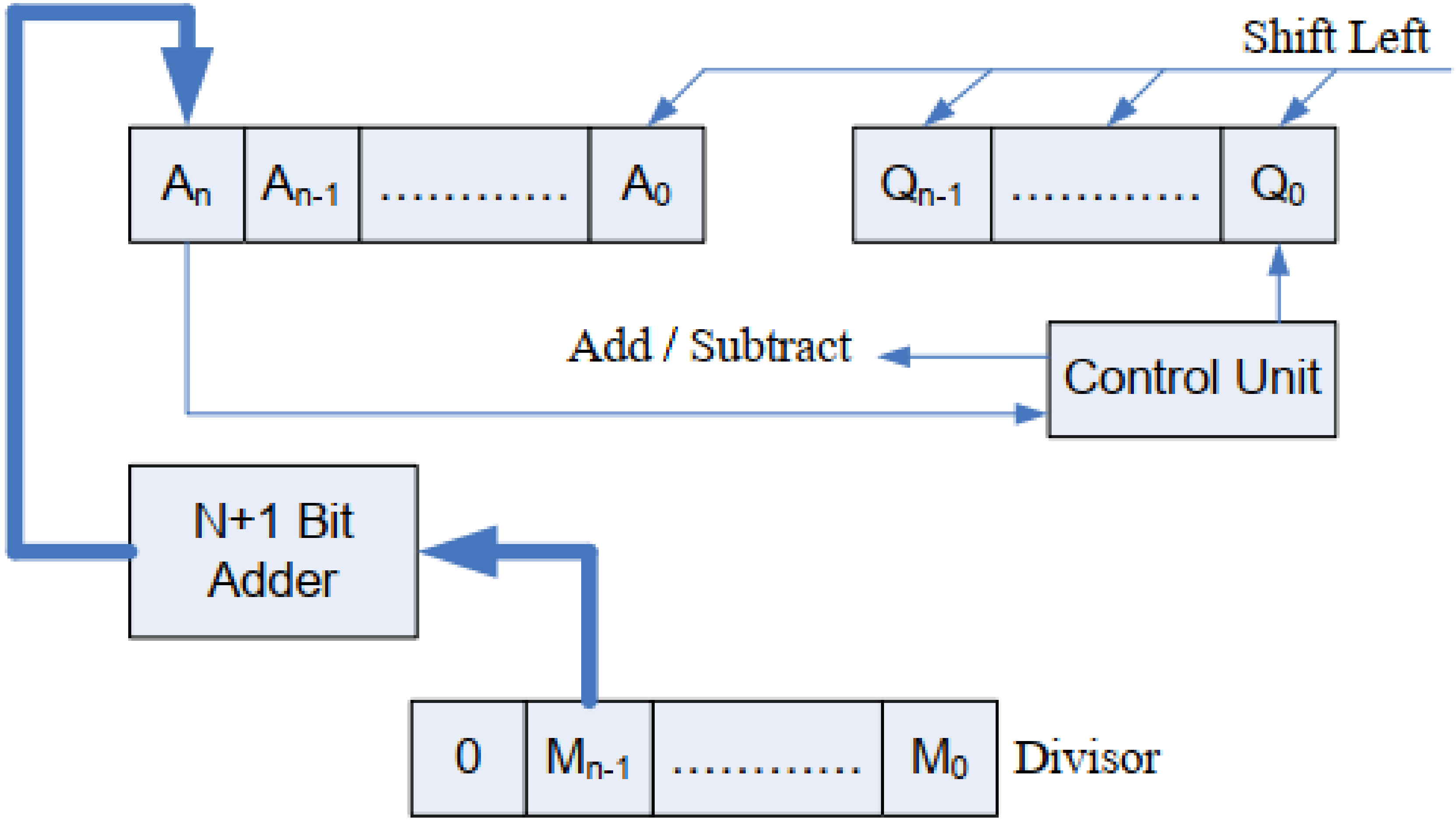
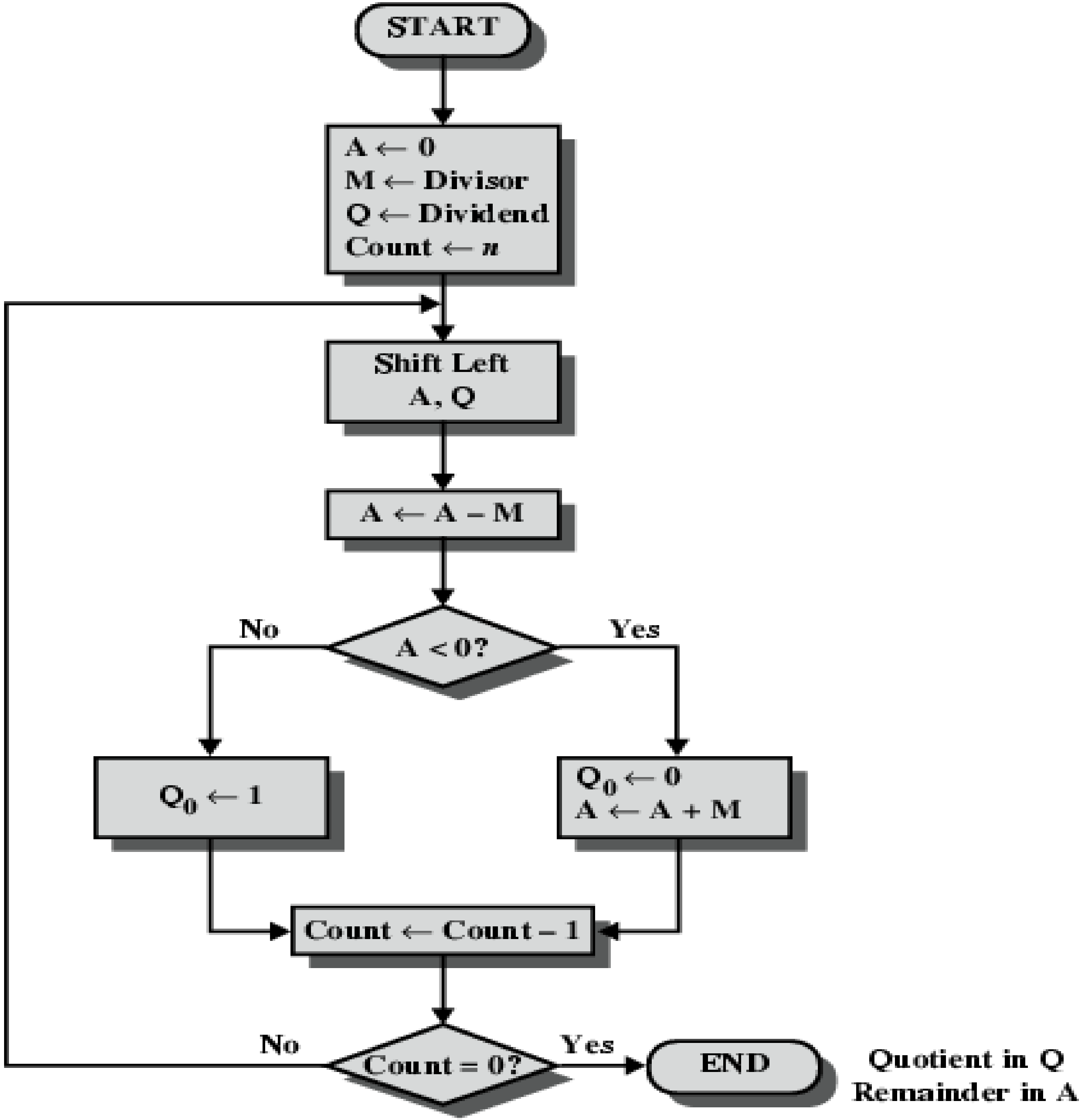


Fig.: Block Diagram of Division Operation

Restoring Division (Unsigned Binary Division)



Algorithm

Step 1: Initialize A, Q and M registers to zero, dividend and divisor respectively and counter to n where n is the number of bits in the dividend.

Step 2: Shift A, Q left one binary position.

Step 3: Subtract M from A placing answer back in A. If **sign of A is 1**, **set Q0 to zero** and **add M back to A (restore A).** If **sign of A is 0**, set Q0 to 1.

Step 4: Decrease counter; if counter > 0, repeat process from step 2 else stop the process. The **final remainder will be in A and quotient will be in Q.**

Example: Divide 15 (1111) by 4 (0100)

A	Q	M	$\overline{M}+1$	Count	Remarks
00000	1111	00100	11100	4	Initialization
00001 11101 00001	111□ 111□ 1110	- - -	- - -	- - 3	Shift Left A, Q Sub (A ← A – M) Q_0 ← 0, Add (A← A + M)
00011 11111 00011	110□ 110□ 1100	- - -	- - -	- - 2	Shift Left A, Q Sub (A ← A – M) Q_0 ← 0, Add (A← A + M)
00111 00011 00011	100□ 100□ 1001	- - -	- - -	- - 1	Shift Left A, Q Sub (A ← A – M) Set Q_0 ← 1
00111 00011 00011	001□ 001□ 0011	- - -	- - -	- - 0	Shift Left A, Q Sub (A ← A – M) Set Q_0 ← 1

Quotient in Q = 0011 = 3

Remainder in A = 00011 = 3

Algorithm

Step 1: In this step, the corresponding value will be initialized to the registers, i.e., register A will contain value 0, register M will contain Divisor, register Q will contain Dividend, and count[N] is used to specify the number of bits in dividend.

Step 2: In this step, we will check the sign bit of A.

Step 3: If this bit of register A is 1, then shift the value of AQ through left, and perform A = A + M. If this bit is 0, then shift the value of AQ into left and perform A = A - M. That means in case of 0, the 2's complement of M is added into register A, and the result is stored into A.

Step 4: Now, we will check the sign bit of A again.

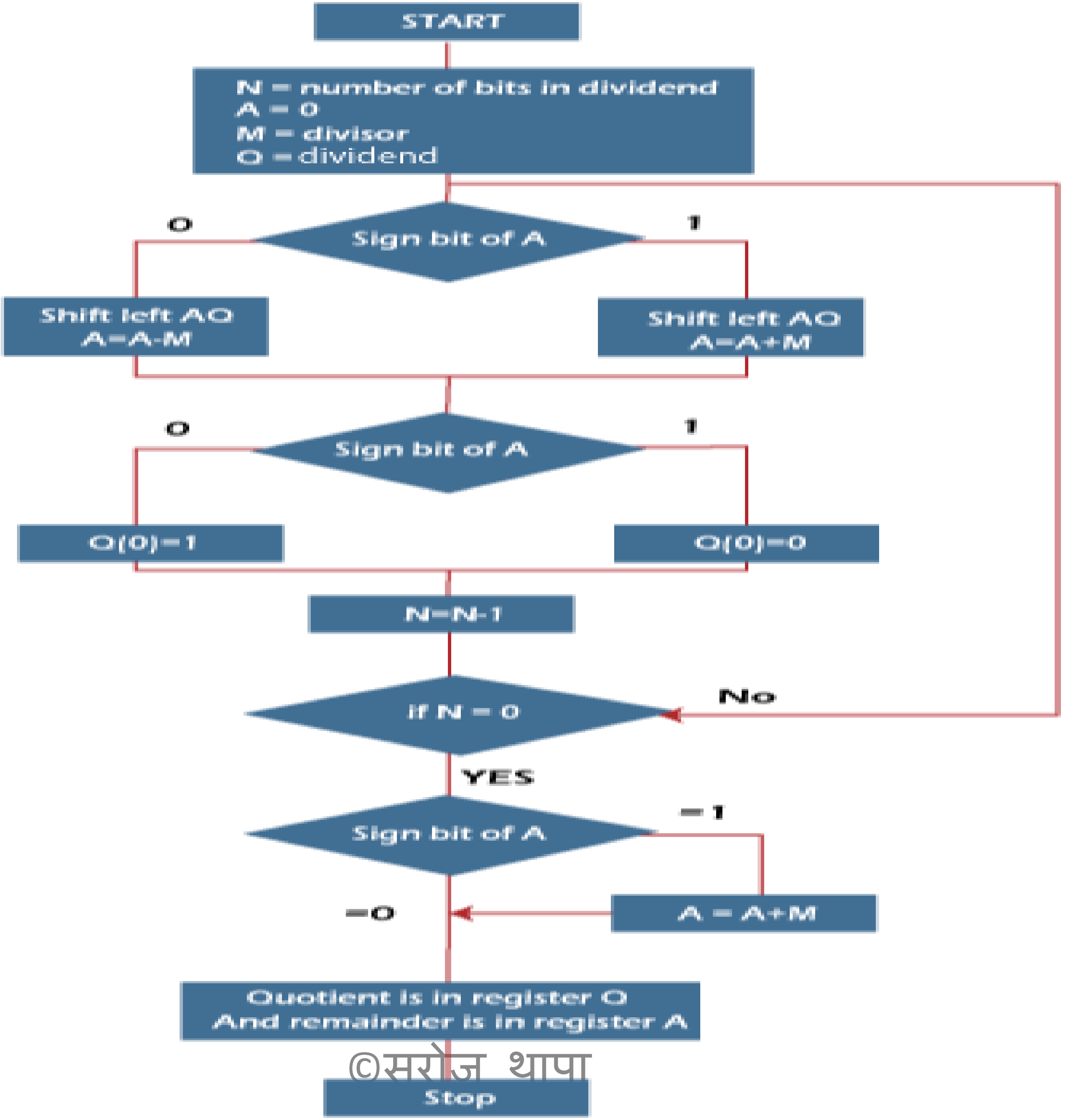
Step 5: If this bit of register A is 1, then Q[0] will become 0. If this bit is 0, then Q[0] will become 1. Here Q[0] indicates the least significant bit of Q.

Step 6: After that, the value of count will be decremented. Here count is used as a counter.

Step 7: If the value of count = 0, then we will go to the next step. Otherwise, we have to again go to step 2.

Step 8: We will perform A = A + M if the sign bit of register A is 1.

Step 9: This is the last step. In this step, register A contains the remainder, and register Q contains the quotient.



START

N = number of bits in dividend
A = 0
M = divisor
Q = dividend

Sign bit of A

0

1

Shift left AQ
A=A-M

Shift left AQ
A=A+M

Sign bit of A

0

1

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 remainder is in register A

Stop

Example: Divide 1110 (14) by 0011 (3) using non-restoring division.

A	Q	M	$\overline{M}+1$	Count	Remarks
00000	1110	00011	11101	4	Initialization
00001	110□	-	-	-	Shift Left A, Q
11110	110□	-	-	-	Sub (A ← A – M)
11110	1100	-	-	3	Set Q_0 to 0
11101	100□	-	-	-	Shift Left A, Q
00000	100□	-	-	-	Add (A ← A + M)
00000	1001	-	-	2	Set Q_0 to 1
00001	001□	-	-	-	Shift Left A, Q
11110	001□	-	-	-	Sub (A ← A – M)
11110	0010	-	-	1	Set Q_0 to 0
11100	010□	-	-	-	Shift Left A, Q
11111	010□	-	-	-	Add (A ← A + M)
11111	0100	-	-	0	Set Q_0 to 0
00010	0100	-	-	-	Add (A ← A + M)

Quotient in Q = 0011 = 3

Remainder in A = 00010 = 2

Comparison between restoring and Non-Restoring Method

Floating Point Representation

The floating point representation of the number has two parts. The first part represents a signed fixed point numbers called mantissa or significand. The second part designates the position of the decimal (or binary) point and is called exponent.

For example, the decimal no + 6132.789 is represented in floating point with fraction and exponent as follows.

Fraction	Exponent
+0.6132789	+04

This representation is equivalent to the scientific notation $+0.6132789 \times 10^{+4}$

The floating point is always interpreted to represent a number in the following form $\pm M \times R^{\pm E}$.

Only the mantissa M and the exponent E are physically represented in the register (including their sign). The radix R and the radix point position of the mantissa are always assumed.

A floating point binary no is represented in similar manner except that it uses base 2 for the exponent.
For example, the binary no +1001.11 is represented with 8 bit fraction and 0 bit exponent as follows.

0.1001110×2^{100}

Fraction	Exponent
01001110	000100

The fraction has zero in the leftmost position to denote positive. The floating point number is equivalent to $M \times 2^E = +(0.1001110)_2 \times 2^{+4}$

Floating Point Arithmetic

The basic operations for floating point arithmetic are

Floating point number	*Arithmetic Operations*
$X = Xs \times B^{XE}$	$X + Y = (Xs \times B^{XE-YE} + Ys) \times B^{YE}$
$Y = Ys \times B^{YE}$	$X - Y = (Xs \times B^{XE-YE} - Ys) \times B^{YE}$
	$X * Y = (Xs \times Ys) \times B^{XE+YE}$
	$X / Y = (Xs / Ys) \times B^{XE-YE}$

A floating point operation may produce one of these conditions:

- Exponent Overflow: A positive exponent exceeds the maximum possible exponent value.
- Exponent Underflow: A negative exponent which is less than the minimum possible value.
- Significand Overflow: The addition of two significands of the same sign may carry in a carry out of the most significant bit.
- Significand underflow: In the process of aligning significands, digits may flow off the right end of the significand

Floating Point Addition and Subtraction

In floating point arithmetic, addition and subtraction are more complex than multiplication and division. This is because of the need for alignment. There are four phases for the algorithm for floating point addition and subtraction.

1. Check for zeros:

Because addition and subtraction are identical except for a sign change, the process begins by changing the sign of the subtrahend if it is a subtraction operation. Next; if one is zero, second is result.

2. Align the Significands:

Alignment may be achieved by shifting either the smaller number to the right (increasing exponent) or shifting the large number to the left (decreasing exponent).

3. Addition or subtraction of the significands:

The aligned significands are then operated as required.

4. Normalization of the result:

Normalization consists of shifting significand digits left until the most significant bit is nonzero.

Example: Addition

$X = 0.10001 * 2^{110}$

$Y = 0.101 * 2^{100}$

Since $E_Y < E_X$, Adjust Y

$$Y = 0.00101 * 2^{100} * 2^{010} = 0.00101 * 2^{110}$$

So, $E_Z = E_X = E_Y = 110$

Now, $M_Z = M_X + M_Y = 0.10001 + 0.00101 = 0.10110$

Hence, $Z = M_Z * 2^{EZ} = 0.10110 * 2^{110}$

Example: Subtraction

$X = 0.10001 * 2^{110}$

$Y = 0.101 * 2^{100}$

Since $E_Y < E_X$, Adjust Y

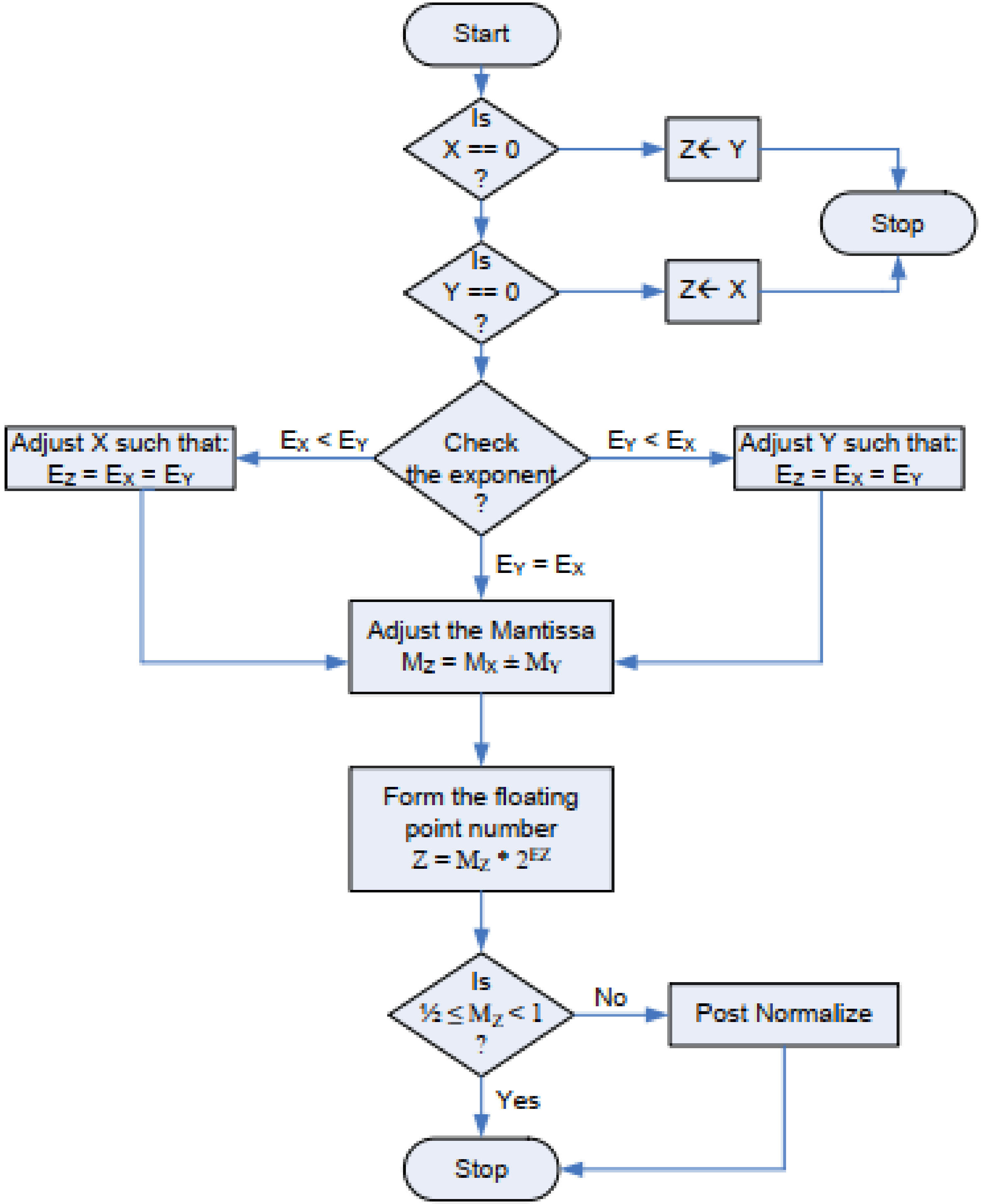
$$Y = 0.00101 * 2^{100} * 2^{010} = 0.00101 * 2^{110}$$

So, $E_Z = E_X = E_Y = 110$

Now, $M_Z = M_X - M_Y = 0.10001 - 0.00101 = 0.01100$

$Z = M_Z * 2^{EZ} = 0.01100 * 2^{110}$ (Un-Normalized)

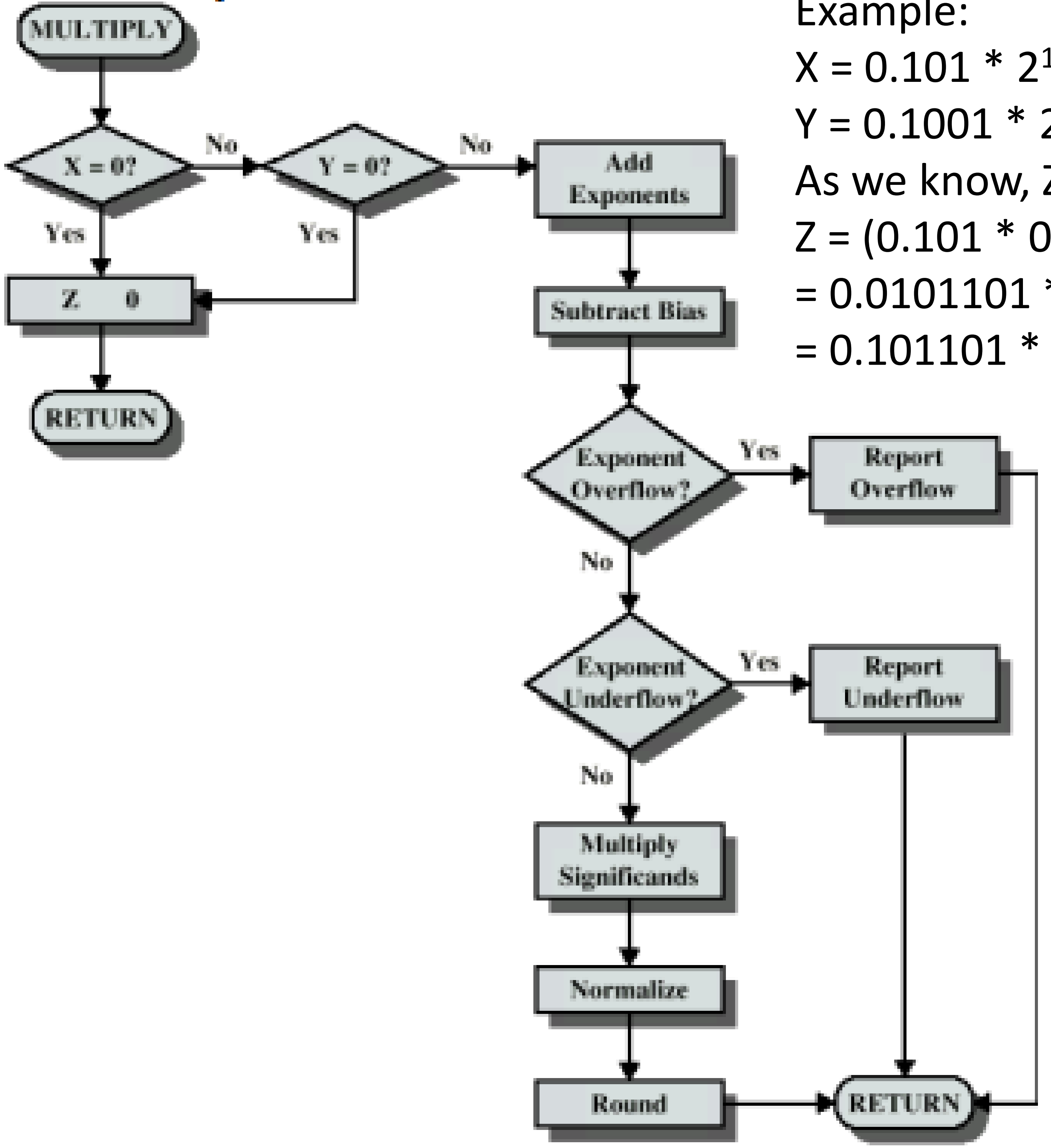
Hence, $Z = 0.1100 * 2^{110} * 2^{-001} = 0.1100 * 2^{101}$


Start
Is
X == 0
?
Z← Y
Stop
Is
Y == 0
?
Z← X
Adjust X such that:
$E_Z = E_X = E_Y$
$E_X < E_Y$
Check
the exponent
?
$E_Y < E_X$
Adjust Y such that:
$E_Z = E_X = E_Y$
$E_Y = E_X$
Adjust the Mantissa
$M_Z = M_X \pm M_Y$
Form the floating
point number
$Z = M_Z * 2^{EZ}$
Is
$½ \leq M_Z < 1$
?
No
Post Normalize
Yes
Stop

Floating Point Multiplication

The multiplication can be subdivided into 4 parts.

1. Check for zeroes.
2. Add the exponents.
3. Multiply mantissa.
4. Normalize the product.



Example:

X = 0.101 * 2^{110}

Y = 0.1001 * 2^{-010}

As we know, Z = X * Y = (M_X * M_Y) * $2^{(EX + EY)}$

Z = (0.101 * 0.1001) * $2^{(110-010)}$

= 0.0101101 * 2^{100}

= 0.101101 * 2^{011} (Normalized)

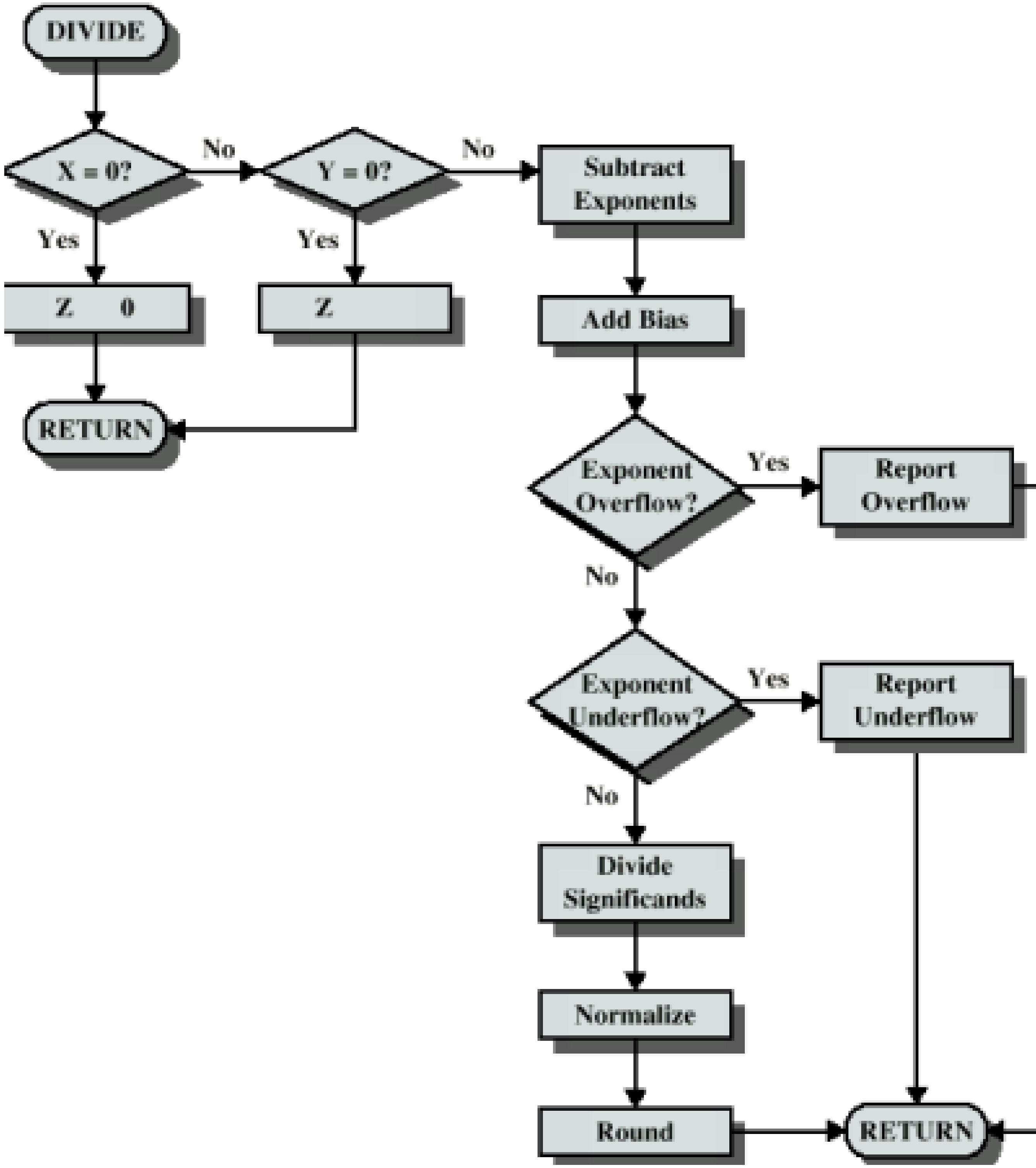
```
   0.1001
  *0.101
   1001
  0000*
 +1001**
 101101 =
 0.0101101
```

।पा

Floating Point Division

The division algorithm can be subdivided into 5 parts

1. Check for zeroes.
2. Initial registers and evaluates the sign.
3. Align the dividend.
4. Subtract the exponent.
5. Divide the mantissa.



Example:

$X = 0.101 * 2^{110}$

$Y = 0.1001 * 2^{-010}$

As we know, $Z = X / Y = (M_X / M_Y) * 2^{(EX - EY)}$

$M_X / M_Y = 0.101 / 0.1001 = (1/2 + 1/8) / (1/2 + 1/16$

$= 1.11 = 1.00011$

$0.11 * 2 = 0.22 \rightarrow 0$

$0.22 * 2 = 0.44 \rightarrow 0$

$0.44 * 2 = 0.88 \rightarrow 0$

$0.88 * 2 = 1.76 \rightarrow 1$

$0.76 * 2 = 1.52 \rightarrow 1$

$E_X - E_Y = 110 + 010 = 1000$

Now, $Z = M_Z * 2^{EZ} = 1.00011 * 2^{1000}$

$= 0.100011 * 2^{1001}$

END OF Chapter 6