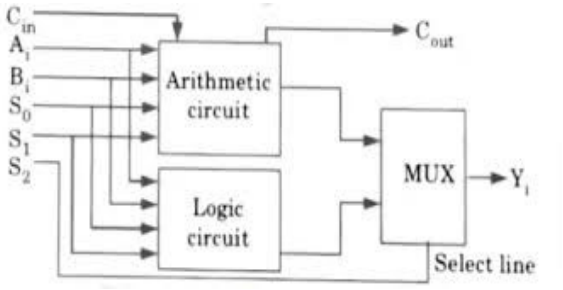
Author :
ABHAY KUMAR SINGH

COA UNIT -2

Sequential Artihmetic and Logic Unit

- ALU is a component within a computer's CPU.
- It performs arithmetic and logical operations on binary data.
- **Arithmetic Operations**: Includes addition, subtraction, multiplication, and division.
- **Logical Operations:** Handles logical functions like AND, OR,NOT.
- **Sequential Execution**: Operations are carried out one after another, not simultaneously.



Arithmetic Mode (s2 = 0):

- ALU functions as an arithmetic circuit.
- The output of the arithmetic circuit is transferred as the final output.
- This implies that arithmetic operations, such as addition, subtraction, multiplication, or division, are performed.

Logic Mode (s2 = 1):

- In this mode, the ALU acts as a logic circuit.
- The output of the logic circuit is transferred as the final output.
- Unlike in arithmetic operations, carry input or carry output is not required in logic operations.

Half adder

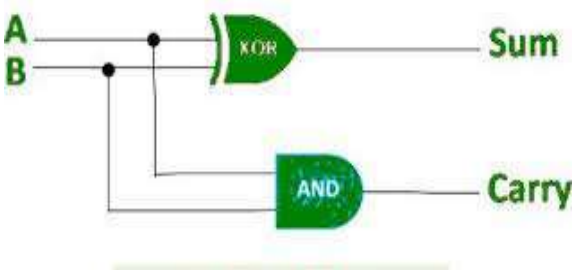
Sum (S): The XOR of A and B.

S = A XOR B

Carry (C): The AND of A and B.

$Cout_t$ = A AND B

Truth Table			
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



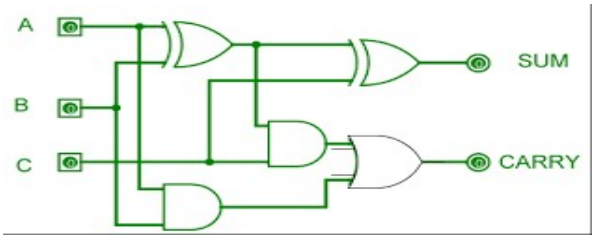
Full adder

sum (S): The XOR of the three inputs (A, B, and Cin).

S=A XOR B XOR Cin

Carry (Cout):

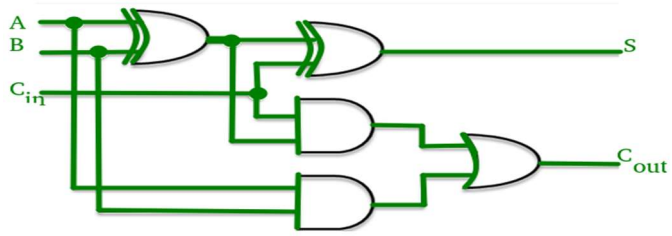
Cout= (A AND B) + (A XOR B) AND Cin



Input			Output	
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

Carry Lookahead Adder (CLA)

- A Carry Lookahead Adder (CLA) is a type of adder in digital logic.
- It improves speed by reducing the time needed to determine carry bits.
- CLA calculates carry bits before the sum, minimizing waiting time for the result of higher-value bits.
- Two variables, propagator and generator, are used in CLA for efficient carry computation.



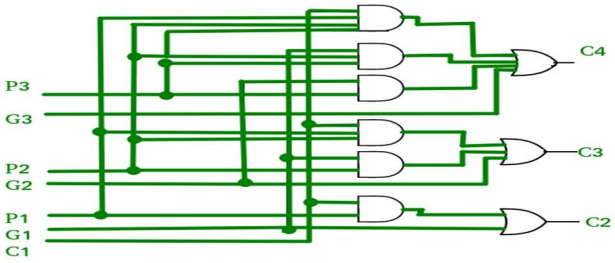
- Addition of two binary numbers in parallel allows all bits to be available for computation simultaneously.
- Carry propagation time is a critical attribute of adders.
- Reducing carry delay time is a key advantage of CLA, contributing to overall speed improvement.

$$P_i = A_i \oplus B_i \qquad S_i = P_i \oplus C_i$$
$$G_i = A_i B_i \qquad C_{i+1} = G_i + P_i C_i$$

where G_i produces the carry when both A_i, B_i are 1 regardless of the input carry. P_i is associated with the propagation of carry from C_i to C_{i+1}.

$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

- C4 is computed simultaneously with lower-order carries .
- No waiting for carry propagation; they are determined in parallel.
- Boolean expressions for carries use a sum of products approach.
- Implemented with one level of AND gates for product terms and an OR gate.
- Boolean functions are designed for minimal delay.
- Carry-out is ready for the next bit without waiting for sequential propagation.

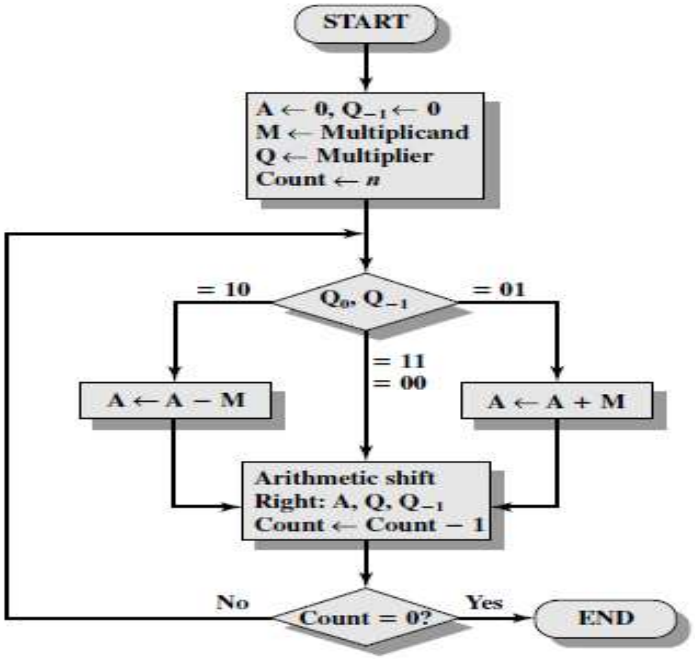


Booths algo

- Booth's Algorithm is a multiplication algorithm that efficiently performs binary multiplication using a series of steps.

  - Initialize variables: Multiplier (Q), Multiplicand (M), Accumulator (A), and a counter (N).
  - Set the counter (count) to the bit length of the multiplier (Q).
  - Start a loop that iterates N times.
  - Check the rightmost two bits of the multiplier (Q0 ,Q-1).
    - If Q0Q-1 = 10, perform the operation A=A−M.
    - If Q0 Q-1 = 01, perform the operation A=A+M.
  - Right shift Q and A by 1 bit.
  - Decrement (count) by 1.
  - Check if the counter (count) is greater than 0.
  - If true, go back to the "Loop Start" step; otherwise, exit the loop.

- The final product is in the Accumulator (A) and the Multiplier (Q).



Example : Multiply (7 x 3)

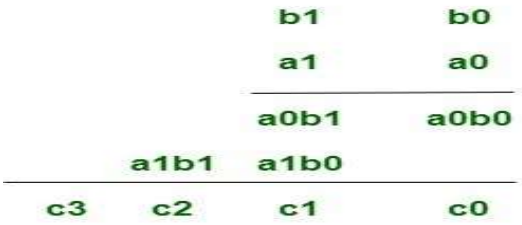
Initialized value :

A=0000
Q=0011
Q-1=0
M=0111
-M = 1001

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial values	
1001	0011	0	0111	A ← A − M	First cycle
1100	1001	1	0111	Shift	First cycle
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A ← A + M	Third cycle
0010	1010	0	0111	Shift	Third cycle
0001	0101	0	0111	Shift	Fourth cycle

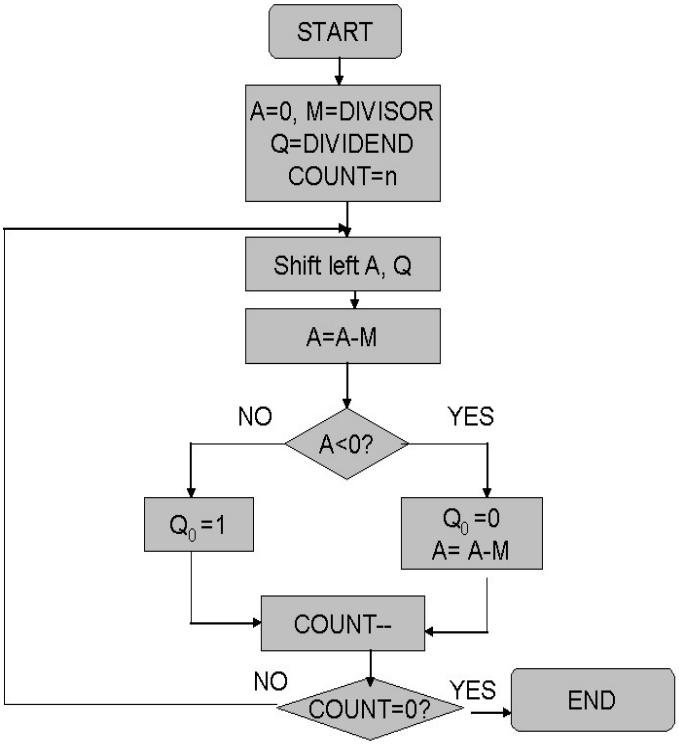
Array multiplier

- An array multiplier is a digital circuit that performs binary multiplication using an array of logic gates. The most common array multiplier architecture is the Wallace Tree Multiplier.
- **Partial Products:**



Restroing division algorithm

- Restoring division is a binary division algorithm that involves restoring partial remainders during each step of the division process.



Example : divide (11 / 3) using restoring

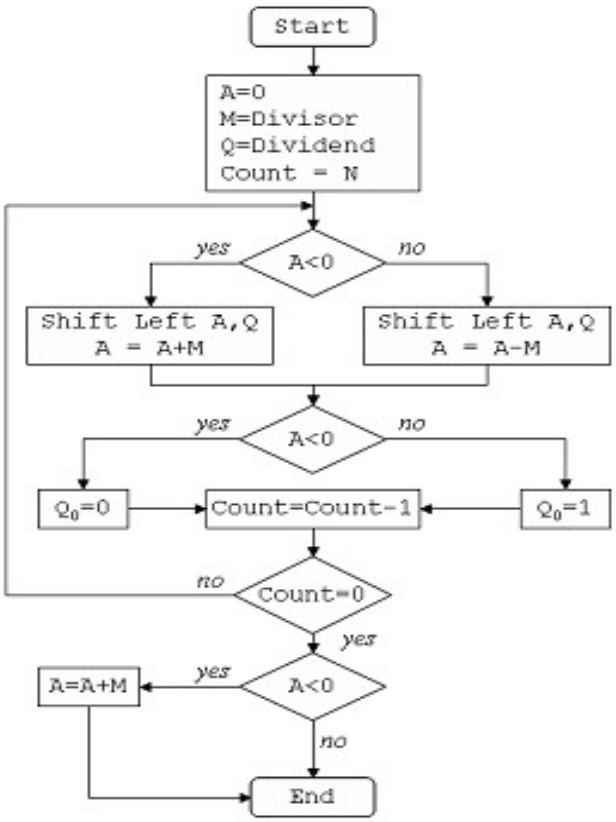
Initialized value :

A=0000
Q=1011
M=00011 , -M = 11101

N	A	Q	ACTION
4	00000	1011	initialize
	00001	011_	ShL AQ
	11110	011_	A=A-M
3	00001	0110	Restore Q[0]=0
	00010	110_	ShL AQ
	11111	110_	A = A - M
2	00010	1100	Restore, Q[0] = 0
	00101	100_	ShL AQ
	00010	100_	A = A - M
1	00010	1001	Q[0] =1
	00101	001_	ShL AQ
	00010	001_	A = A - M
0	00010	0011	Q[0] = 1

Non restoring method

- Non-restoring division is more complex than restoring division algorithmically.
- However, its hardware implementation simplifies the process.
- Non-restoring division involves only one decision and addition/subtraction per quotient bit.
- After subtraction, there are no additional restoring steps, leading to a simpler hardware design.



Example : divide (11 / 3) using non-restoring

Initialized value :

A=0000
Q=1011
M=00011 , -M = 11101

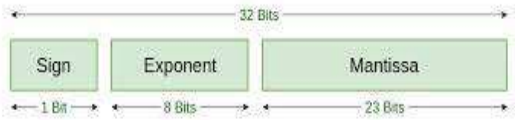
N	A	Q	ACTION
4	00000	1011	initialize
	00001	011_	ShL AQ
	11110	011_	A=A-M
3	11110	0110	Q[0]=0
	11100	110_	ShL AQ
	11111	110_	A = A + M
2	11111	1100	Q[0] = 0
	11111	100_	ShL AQ
	00010	100_	A = A +M
1	00010	1001	Q[0] = 1
	00101	001_	ShL AQ
	00010	001_	A = A - M
0	00010	0011	Q[0] = 1

IEEE standard for floating point arithmetic

- IEEE standard for floating point arithmetic is a technical standard for floating point computation .

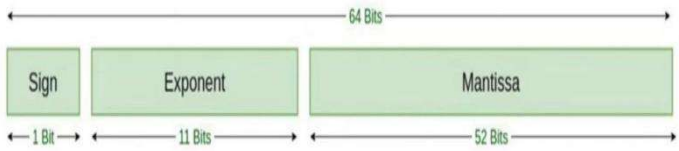
Single Precision (32 bit):

- The format consists of three components: the sign bit, the exponent, and the fraction (also known as the significand or mantissa).
- 1. **1 sign bit** 2. **8 bit exponent** 3. **23 bit mantissa**
- The formula for the value of a single-precision floating-point number is: $(-1)^S x 1.f \times 2^{(e-127)}$



Double Precision (64 bit):

- 1. **1 sign bit** 2. **11 bit exponent** 3. **52 bit mantissa**
- The formula for the value of a single-precision floating-point number is: $(-1)^S x 1.f \times 2^{(e-1023)}$



Example: ***Suppose we want to represent the decimal number 6.75 in IEEE 754 single-precision format.***

number is positive, then sign bit **s = 0**

Convert the absolute value to binary:

$(6.75)_{10}$ = $(110.11)_2$ in binary.

Normalize the binary representation:

$110.11 = 1.1011 \times 2^2$.

exponent (e): e=2+127=129.

exponent in binary: $129_{10} = 10000001_2$

Single precision format :

0	10000001	1011……..

Example: ***Suppose we want to represent the decimal number -6.75 in IEEE 754 double-precision format.***

number is positive, then sign bit **s = 1**

Convert the absolute value to binary:

$(6.75)_{10}$ = $(110.11)_2$ in binary.

Normalize the binary representation:

$110.11 = 1.1011 \times 2^2$.

exponent (e): e=2+1023=1025.

exponent in binary: $1025_{10} = (10000000001)_2$

Double precision format :

1	10000000001	1011……..

OVERFLOW :

- Overflow occurs when the result of an arithmetic operation is too large (in absolute value) to be represented within the available number of bits.
- There are two primary types of overflow: signed overflow and unsigned overflow.
- **Signed Overflow**: Occurs when the result of an operation exceeds the maximum representable positive value or falls below the minimum representable negative value for the given number of bits.
- **Unsigned Overflow**: Occurs when the result of an operation exceeds the maximum representable value for the given number of bits, considering all values as non-negative.