UNIT 1

Data Representations

Arithmetic

- Binary Addition
- Binary Subtraction
- Binary Multiplication
- Binary Division

Binary Addition

- Rules:-

0 + 0 = Sum of 0 with a carry of 0

0 +1 = Sum of 1 with a carry of 0

1 + 0 = Sum of 1 with a carry of 0

1 + 1 = Sum of 0 with a carry of 1

Binary Subtraction

- Rules:-

  0 - 0 = 0

  1 -1 = 0

  1 - 0 = 1

  10 - 1 =1

  0 – 1 with a borrow of 1

Binary Multiplication

- Rules:-

0 * 0 = 0

0 * 1 = 0

1 * 0 = 0

1 * 1 = 1

SIGNED NUMBERS(Integer Representation)

- **Signed numbers can be represented as both *positive* and *negative* numbers**

- There are **3 representations for Signed Numbers**
- **Signed magnitude representation**
- **Signed 1's complement representation**
- **Signed 2's complement representation**

- **Example: Represent +9 and -9 in 8 bit-binary number**
- **Only one way to represent +9 ==> 0000 1001**
- **Three different ways to represent -9:**
- **In signed-magnitude: 1 0001001**
- **In signed-1's complement: 1 1110110**
- **In signed-2's complement: 1 1110111**

- Represent the following numbers in **signed-magnitude, signed-1's complement, and signed-2's complement**
- 1. -13 and +13
- in signed 2s complement 8 bit,
- -13 = 2s complement(00001101) = 11110011
- 2. -17 and +17
- 3. -25 and +25

Complement

- Computers uses complemented numbers to perform subtraction.
- In binary number system there are 2 types of complement – 1’s and 2’s complement
- Similarly, in decimal number system there are 2 types of complement – 10’s and 9’s complement

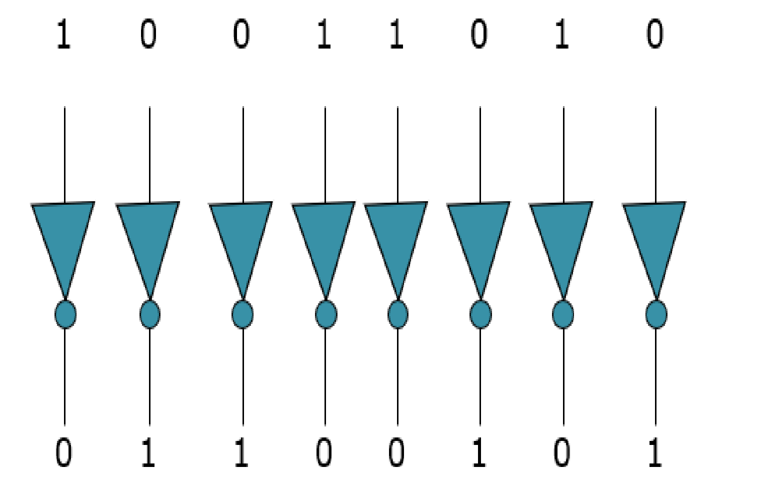
1's Complement

- To get 1's complement convert 0's to 1 and 1's to 0's.
- Eg: 1's complement of 10010 is 01101

1's complement of 1111 is 0000

Application Example

- The Simplest way to obtain the 1's complement of a binary number with digital circuits is :
- To use parallel inverters (NOT circuits)
  - **Eg: an 8 bit no:**



Convert the following to 1's complement

1. 1001010
2. 10001100
3. 1110001
4. 11100110
5. 10101110
6. 111010
7. 1101100

2's complement

- To find 2's complement
  - **Find 1's complement**
  - **Add 1 to the LSB of 1's complement**
- Eg:- 1010

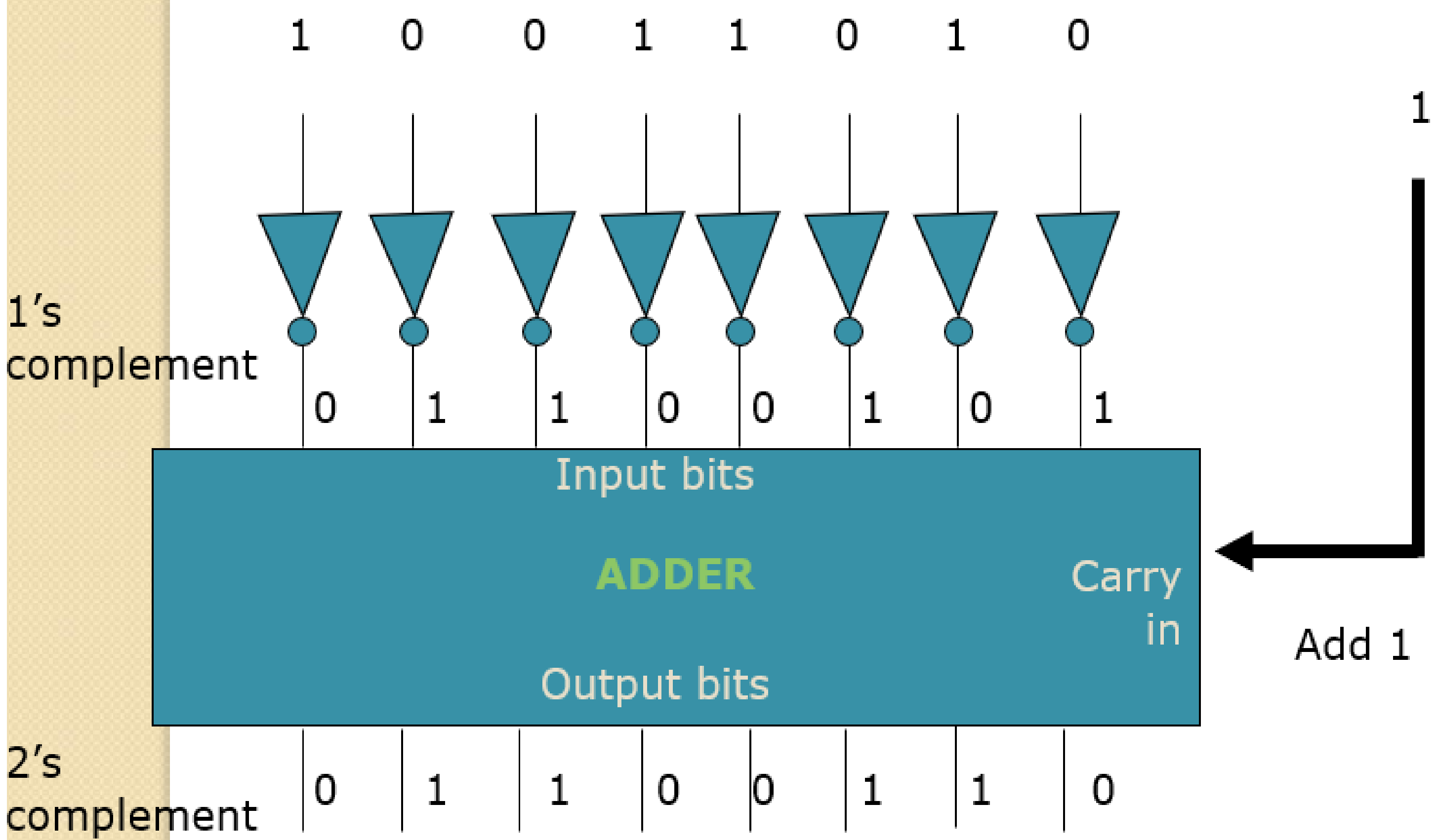
0101+

1

0110

Application Example

- The 2's complement can be realized using:
- Inverters
- Adder
  - **An Eg: shows how an 8bit no can be converted to 2's complement**


1 0 0 1 1 0 1 0
1
1's complement
0 1 1 0 0 1 0 1
Input bits
ADDER
Carry in
Output bits
Add 1
2's complement
0 1 1 0 0 1 1 0

Convert the following to 2's complement

1. 1001010
2. 10001110
3. 11100011
4. 11100110
5. 1010111
6. 1110101
7. 11011001

9's Complement

- The 9's complement of a decimal number is obtained by subtracting each digit of the number from 9.
- Eg:- 9's complement of 2 is 7 and
- 9's complement of 123 is 876

999 -

123

876

Convert the following to 9's complement

1. 25
2. 456
3. 122
4. 101
5. 111
6. 345
7. 342

10's Complement

- 10's complement of a decimal number is obtained by adding one to the 9's complement of that number.
- Eg: 10's complement of 2

9-

2

7+

1

8

Convert the following to 10's complement

1. 254
2. 453
3. 123
4. 102
5. 1110
6. 324
7. 376

- Find 1's and 2's complement
- 1. 11010101
- 2.11011011
- 3. 10110010

Logic Operations

- Perform the following Logic operations

1. 10110101 v 1100110
2. 11001101 v 1000110
3. 10010101 Λ 1100111
4. 11101001 Λ 1101110
5. 11010011 ⊕ 1101110
6. 11101101 ⊕ 1101100

IEEE754 STANDARD

- Computers have various architectures designed by different vendors like Intel, AMD, ARM, etc.
- How do computers interpret various number formats?
- If there are various representations for these different architectures, then it will create chaos and ambiguity.
- So there should be a common agreement between all the processor vendors for handling the data.
- This common standard is the IEEE754 standard.

Data type representation

- Two types of Representation
  - **<u>Fixed Point</u> - A fixed-point number means that there are a fixed number of digits after the decimal point**
  - **<u>Floating Point</u>- A floating point number does not reserve a specific number of bits for the integer part or the fractional part.**

Fixed Point Number

- A fixed point number has a specific number of bits (or digits) reserved for the integer part and a specific number of bits reserved for the fractional part.

Fixed Point Numbers

- This representation has fixed number of bits for integer part and for fractional part.
- For example, if given fixed-point representation is IIII.FFFF, then you can store minimum value is 0000.0001 and maximum value is 9999.9999. There are three parts of a fixed-point number representation: the sign field, integer field, and fraction

Unsigned fixed point

Integer	Fraction

Signed fixed point

Sign	Integer	Fraction

Examples of Fixed Point representations

- Compute 0.75 + (-0.625) using Fixed point numbers
- Step 1 Rep. 0.75 in Binary (4 bits for Integer and 4 bits for fraction)
  - **0.75 ---- 0000.1100**

0.75 x 2 = 1.5 - - 1
0.5 x 2 = 1.0 - - 1
0.0 x 2 = 0 - - 0

- Step 2 -Rep 0.625 in binary
  - **0.625 ---0000.1010**
  - **Since it is (-0.625) we have to find the 2's complement**

0.625 x 2 = 1.250 - 1
0.250 x 2 = 0.500 - 0
0.500 x 2 =1.000 - 1
0.000 x 2 =0.000 -0

- Step 3 – Since the second no is a negative number, find the 2's complement of it
  - **(-0.625) --- 1111.0110**
- Step 4 – Now Add both the numbers

0000.1100 +
1111.0110
1)0000.0010

- $0.5+0.250$
- $-0.75+0.375$
- $1.125+-0.625$
- $-2.5+1.75$
- $3.0-1.5$

32 bit representation of Fixed point numbers

- **Assume the number using 32 bit format which reserves 1 bit for the sign, 15 bits for integer part and 16 bits for fraction part. The number is (-43.625)**

Sign(1 bit)	Integer (15 bits)	Fraction (16 bits)
1	000 0000 0010 1011	1010 0000 0000 0000

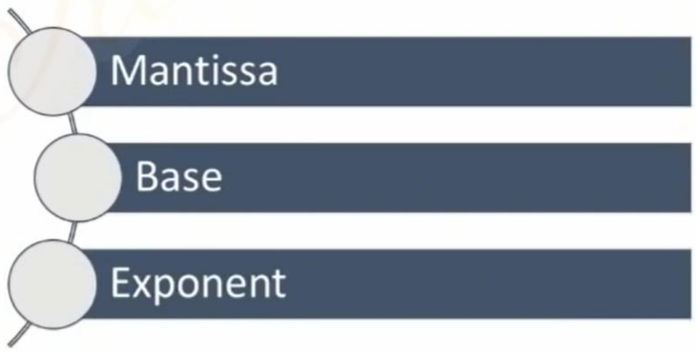
- **43 = 101011**
- **.625 =1010**
- **Since the given no is a negative no.the sign bit is 1.**
- **43.625 – 000 0000 0010 1011. 1010 0000 0000 0000**

Fixed-Point (Practice Questions)

- **Decimal → Fixed-Point Representation**
- Represent **25.75**
- Represent +**25.75**
- Represent **−12.5**
- Represent **+0.125**
- Represent **+145.3125**

Floating Point Numbers

- In the decimal number system, Very large and very small numbers are expressed in scientific notations, By starting number (Mantissa) and an exponent of 10
- Example of Floating point numbers are $6.53\ X\ 10^{-27}$ & $1.58\ X\ 10^{21}$
- Binary numbers are also expressed in the same notations by starting number and exponent of 2
- So in general we can say that the floating point representation has 3 parts



Floating Point Numbers

- The scientific notation of floating point representation is

$$\pm M \; X \; B^{E}$$

Number	Mantissa	Base	Exponent
$9 \; X \; 10^{8}$			
$110 \; X \; 2^{7}$			
4364.784			

NUMBER	MANTISSA	BASE	EXPONENT
9×10^8	9	10	8
110×2^7	110	2	7
4364.784	4364784	10	-3
110.101010	110101010	2	-6

IEEE 754 floating point representation

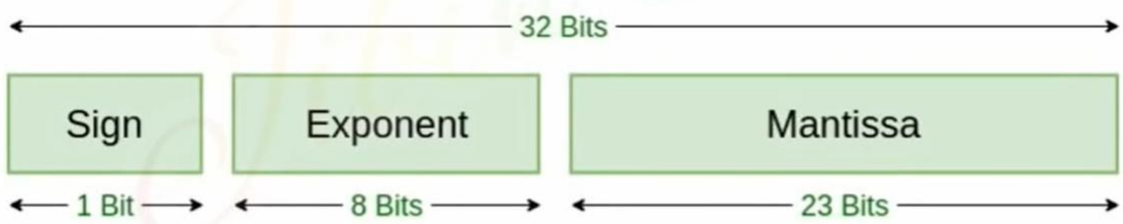
- IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.
- Two types
  - **Single precision**
  - **Double precision**

IEEE 754 Floating Point Representation

Single Precision Format

32 Bit number is Considered

Sign (1 bit)	Exponent 23-30 bits)	Mantissa (0-22 bits)



Single Precision
IEEE 754 Floating-Point Standard

Double Precision Format

64 Bit number is Considered

Sign (1 bit)	Exponent (52-62)	Mantissa (0-51)

Double Precision
IEEE 754 Floating-Point Standard

How to find single and double precision of a number?

IEEE 754 Steps

1. Convert Decimal to Binary
2. Normalize the Number
3. Apply single precision and double precision equations

Equations for Single and Double Precisions

Single Precision

$$(1.N)2^{E-127}$$

Double Precision

$$(1.N)2^{E-1023}$$

Represent $(1259.125)_{10}$ in single and double precision Format

Represent $(1259.125)_{10}$ in single and double precision Format

- (1259.125)10 = 100 1110 1011.001
- <u>Step 1 Convert to binary</u>
- 100 1110 1011.001
- <u>Step 2 Normalize the Number</u>
- $(1.N)2^{E-127}$ - Single Precision (127 means 128 bits)
- $(1.N)2^{E-1023}$ - Double Precision(1023 means 1024 bits)
- 1 00 1110 1011.001
- $1.\underbrace{00\ 1110\ 1011}_{N}001 \times 2^{10}$

(After the decimal point, there are 10 numbers , so E=10)

- Step 3 Apply Single precision format
- $(1.N)2^{E-127}$
- $1.\ 00\ 1110\ 1011001 \times 2^{10}$
- $(137)10 = (1000\ 1001)2 \rightarrow E$
- Single Precision Format :

We need to find Exponent
[E-127 =10]
E =127+10
= 137

31 Sign	30 Exponent 23	22 Mantissa 0
0	10001001	0011101011001.....000
1 bit (S)	8 bit (E)	23 bit (M)

DOUBLE PRECISION

- $(1259.125)_{10}$ = 100 1110 1011.001
- Step 1 Convert to binary
- 100 1110 1011.001
- Step 2 Normalize the Number
- $(1.N)2^{E-1023}$ - Double Precision(1023 means 1024 bits)
- 1 00 1110 1011.001
- $1.\underbrace{00\ 1110\ 1011}_{N}001 \times 2^{10}$

(After the sign bit, there are 10 numbers , so E=10)

- Step 3 Apply Single precision format
- $(1.N)2^{E-1023}$
- $1.\ 00\ 1110\ 1011001\ \times\ 2^{10}$

[E-1023 =10]
E =1023+10
= 1033

- $(1033)2 = (100\ 0000\ 1001)2 \rightarrow E$
- Double Precision Format : 1024 bits

63 Sign	62 Exponent 52	51 Mantissa 0
0	100 0000 1001	0011101011001.....000
1 bit (S)	11 bit (E)	52 bit (M)

- 1358.130
- 14.25

Booths Multiplication

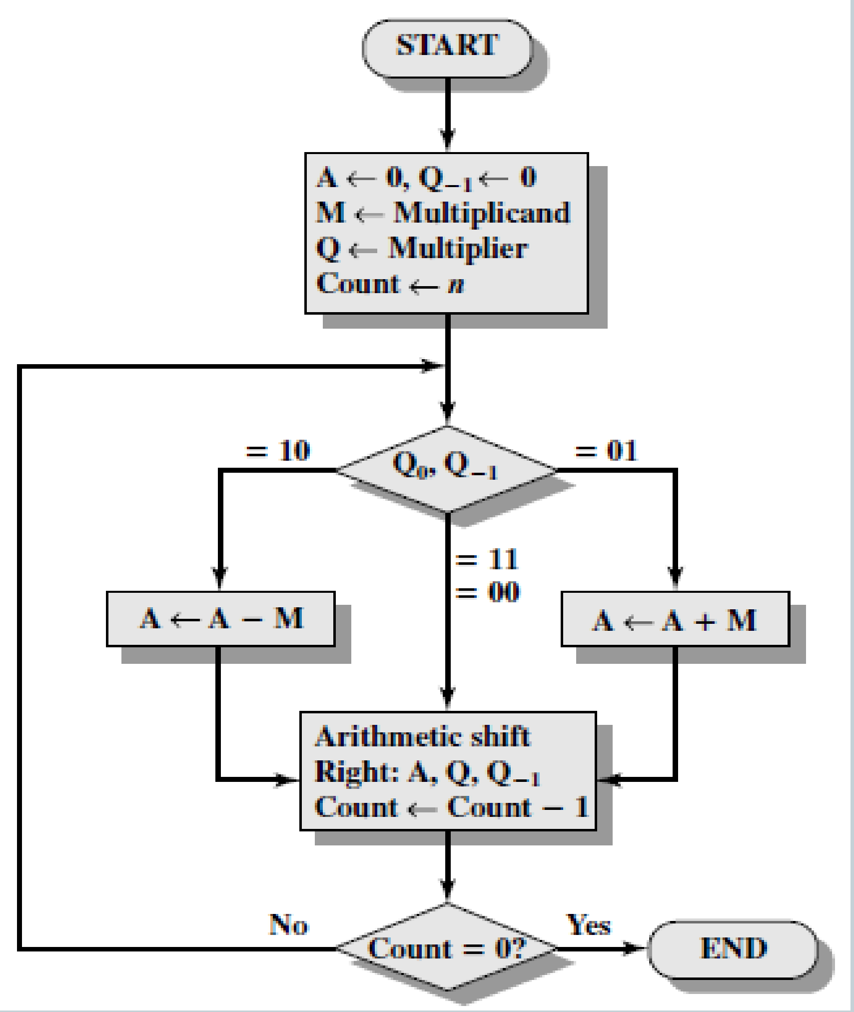
- For multiplying of negative numbers
- Method 1
  - Multiply the number as such
  - Negate the result if any one is negative.
- Method 2
  - Booths multiplication

- Booth's Multiplication Algorithm is a method used to perform signed binary multiplication efficiently.
- It reduces the number of arithmetic operations by encoding the multiplier in a way that minimizes the number of addition and subtraction steps.
- This algorithm is widely used in computer architecture for fast binary multiplication.

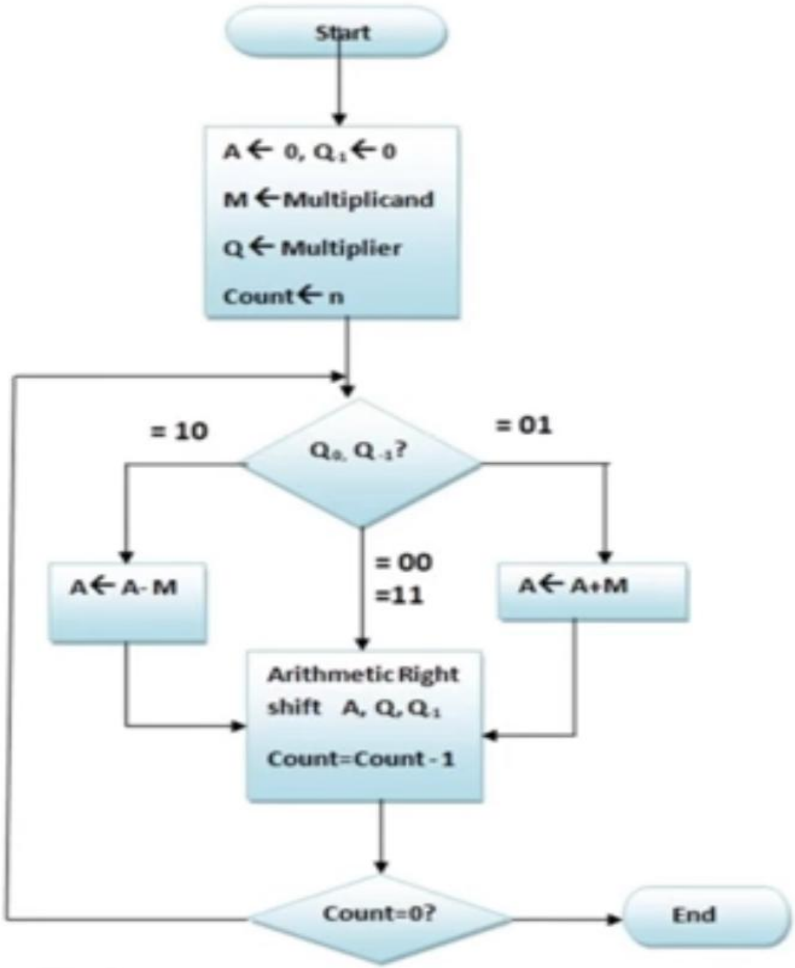
STEPS

- STEP 1: Initialize the multiplier, multiplicand, and product registers.
- STEP 2: Extend the multiplier with one extra bit (0) at the right.
- STEP 3: Based on the current and previous bits of the multiplier:
  - 10: Subtract the multiplicand from the product.
  - 01 : Add the multiplicand to the product.
  - 00 & 11 : No arithmetic operation.
- STEP 4 :Shift the product right by one bit.
- STEP 5: Repeat the process until all bits are processed.

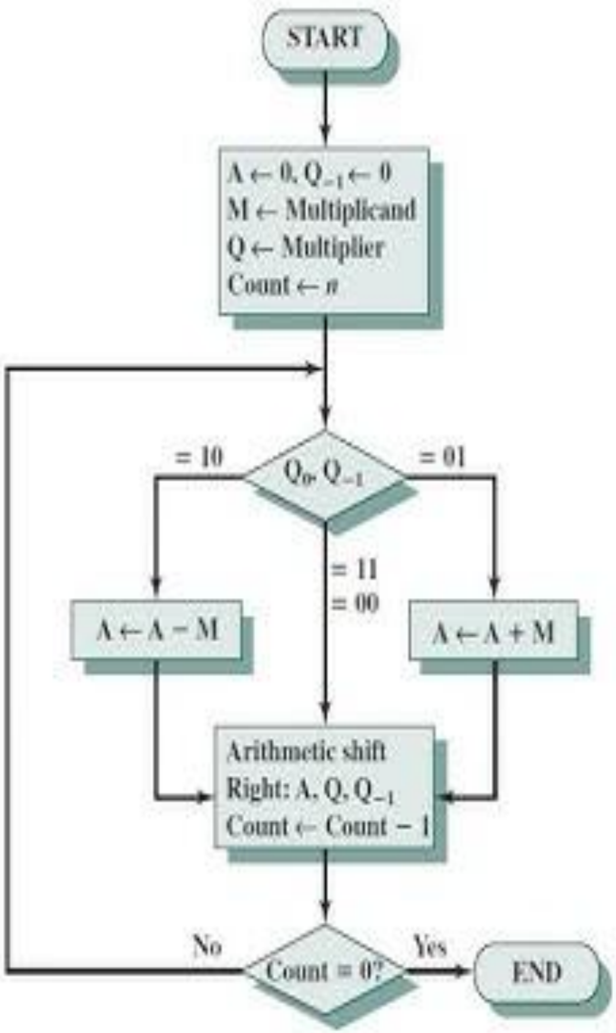
BOOTH'S ALGORITHM



Booths Multiplication



Count	A	Q	Q_{-1}	M	Operation
4	0000	0011	0	0111	Initialization
3	1001 1100	0011 1001	0 1	0111 0111	A ← A- M Arith. Right shift
2	1110	0100	1	0111	Arith. Right shift
1	0101 0010	0100 1010	1 0	0111 0111	A ← A+ M Arith. Right shift
0	0001	0101	0	0111	Arith. Right shift



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	
1110	0100	1	0111	Shift	Second cycle
0101	0100	1	0111	A←A + M	Third cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth cycle

Booth's Algorithm

Ref: "Computer Organization and Architecture Designing for Performance" By William Stallings

END