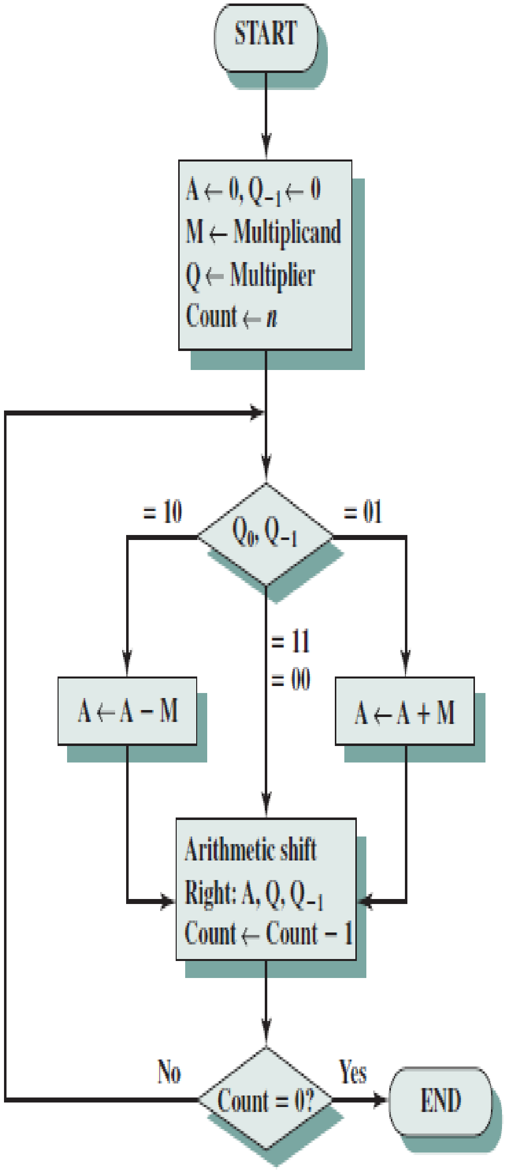
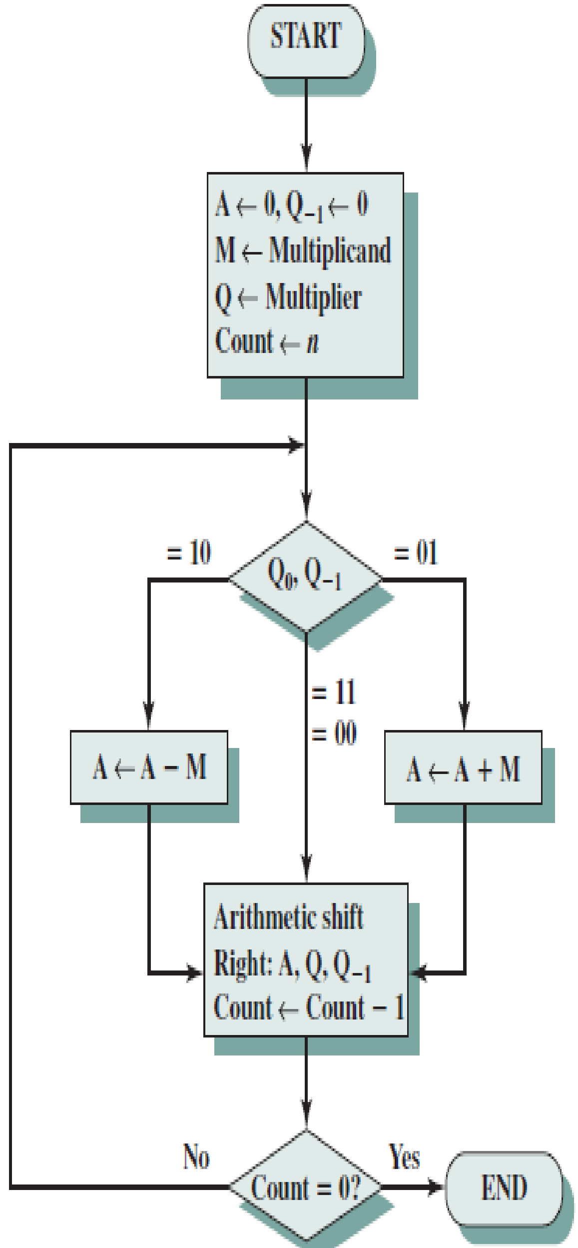
# Unit 2

# Booth’s Algorithm

- The booth algorithm is a multiplication algorithm that allows us to multiply the two signed binary integers in 2's complement, respectively.
- It is also used to speed up the performance of the multiplication process.
- It is very efficient too.
- It works on the string bits 0's in the multiplier that requires no additional bit only shift the right-most string bits and a string of 1's in a multiplier bit weight $2^k$ to weight $2^m$ that can be considered as $\mathbf{2^{k+1} - 2^m}$.

1. Multiplier and multiplicand are placed in the Q and M register respectively.
2. Result for this will be stored in the AC and Q registers.
3. Initially, AC and $Q_{-1}$ register will be 0.
4. Multiplication of a number is done in a cycle.
5. A 1-bit register $Q_{-1}$ is placed right of the least significant bit $Q_0$ of the register Q.
6. In each of the cycle, $Q_0$ and $Q_{-1}$ bits will be checked.
    1. If $Q_0$ and $Q_{-1}$ are 11 or 00 then the bits of AC, Q and $Q_{-1}$ are shifted to the right by 1 bit.
    2. If the value is shown 01 then multiplicand is added to AC. After addition, AC, $Q_0$, $Q_{-1}$ register are shifted to the right by 1 bit.
    3. If the value is shown 10 then multiplicand is subtracted from AC. After subtraction AC, $Q_0$, $Q_{-1}$ register is shifted to the right by 1 bit.





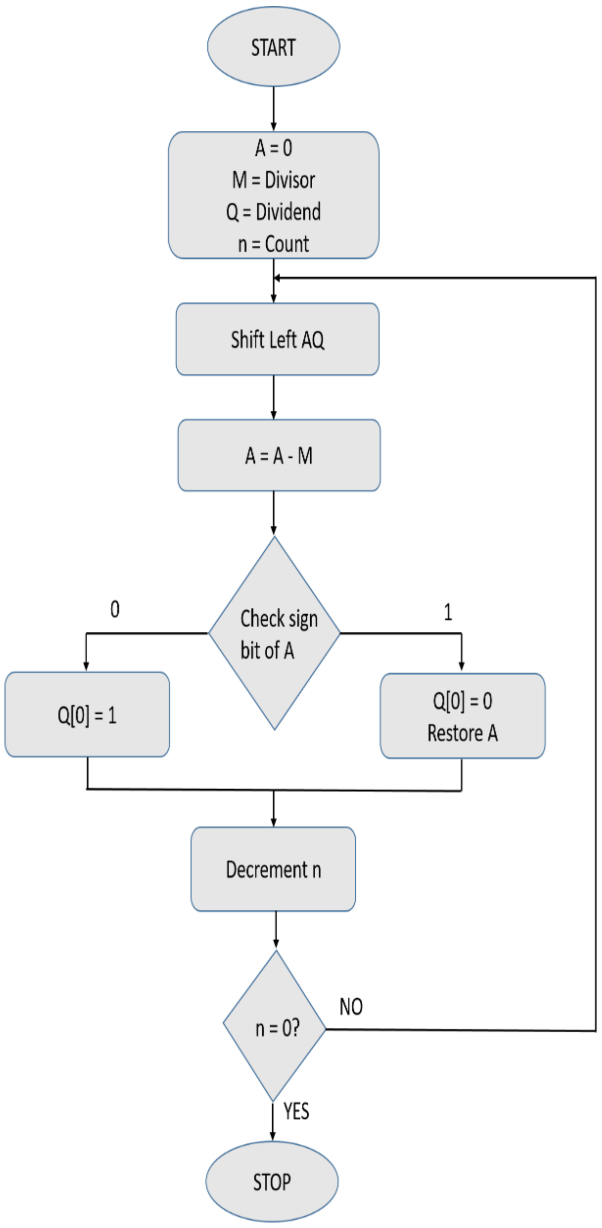

## Figure 10.13 Example of Booth's Algorithm (7 × 3)

| 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 |

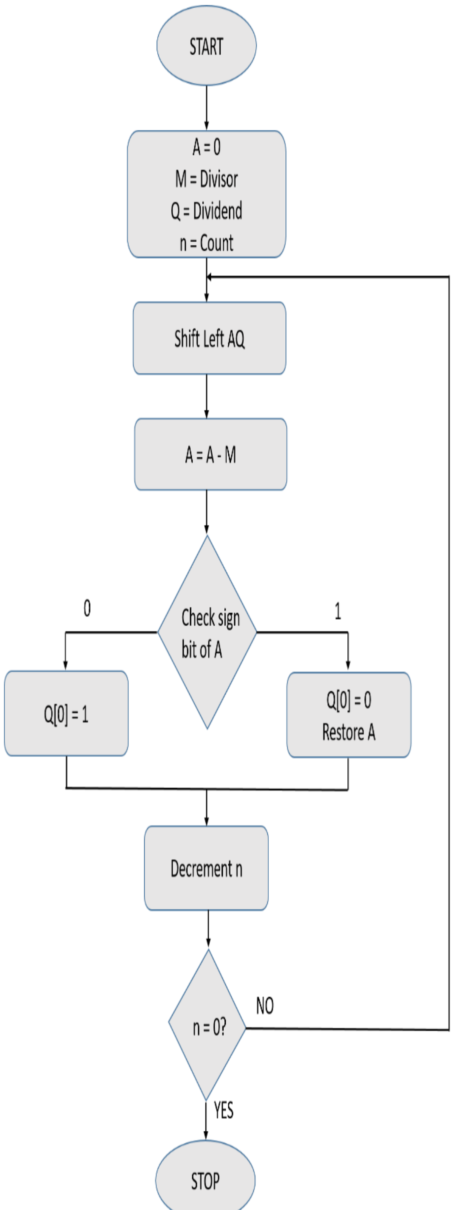
# Restoring Division Algorithm

- Restoring Division Algorithm is used to divide two unsigned integers.
- This algorithm is called restoring because it restores the value of **Accumulator(A)** after each or some iterations.

- **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
- **Step-2:** Then the content of register A and Q is shifted left as if they are a single unit
- **Step-3:** Then content of register M is subtracted from A and result is stored in A
- **Step-4:** Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M
- **Step-5:** The value of counter n is decremented
- **Step-6:** If the value of n becomes zero we get of the loop otherwise we repeat from step 2
- **Step-7:** Finally, the register Q contain the quotient and A contain remainder



# Restoring Division Algorithm For Unsigned Integer




11(Dividend)/3(Divisor)=3(Quotient) & 2 (Reminder)

-M=11101

| N | M=(Divisor) | A=initialize to 0 | Q=Dividend | Operation |
|---|---|---|---|---|
| 4 | 00011 | 00000 | 1011 | Initialize |
| | | 00001 | 011? | Shift left |
| | | 11110 | 011? | A=A-M<br>A=A+2's Complement of M |
| 3 | | 00001 | 0110 | A[n]=1 Q[0]=0 & Restore A |
| | | 00010 | 110? | Shift left |
| | | 11111 | 110? | A=A-M<br>A=A+2's Complement of M |
| 2 | | 00010 | 1100 | A[n]=1 Q[0]=0 & Restore A |
| | | 00101 | 100? | Shift Left |
| | | 00010 | 100? | A=A-M<br>A=A+2's Complement of M |
| 1 | | 00010 | 1001 | A[n]=1 Q[0]=1 No Restoration |
| | | 00101 | 001? | Shift left |
| | | 00010 | 001? | A=A-M<br>A=A+2's Complement of M |
| | | 00010 | 0011 | A[n]=1 Q[0]=1 No Restoration |

# Floating Point Representation

- It has three parts

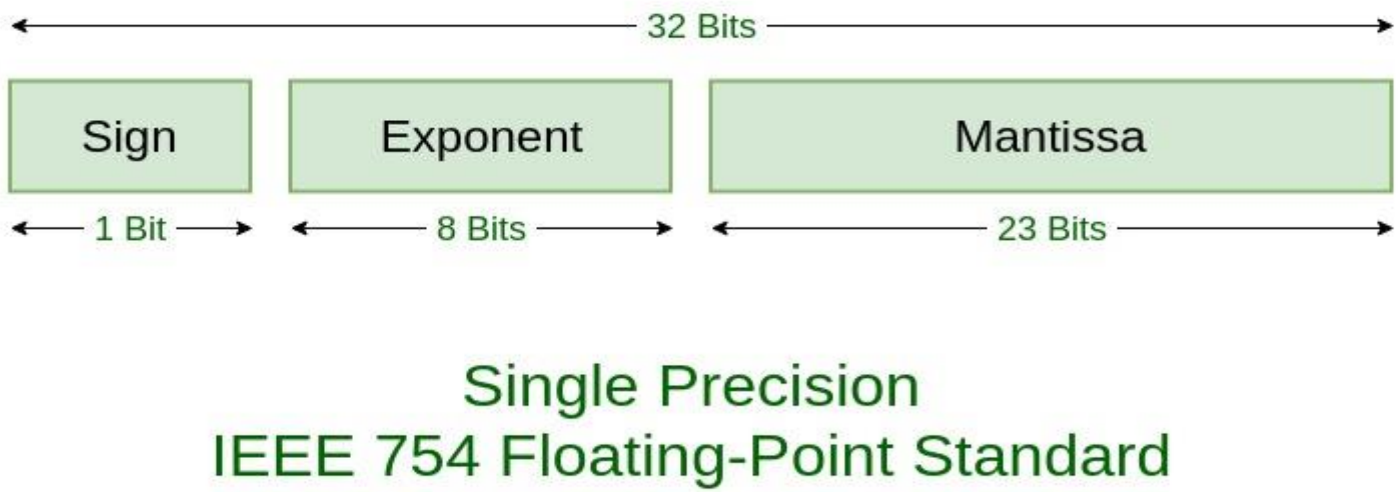
1. Mantissa
2. Base
3. Exponent

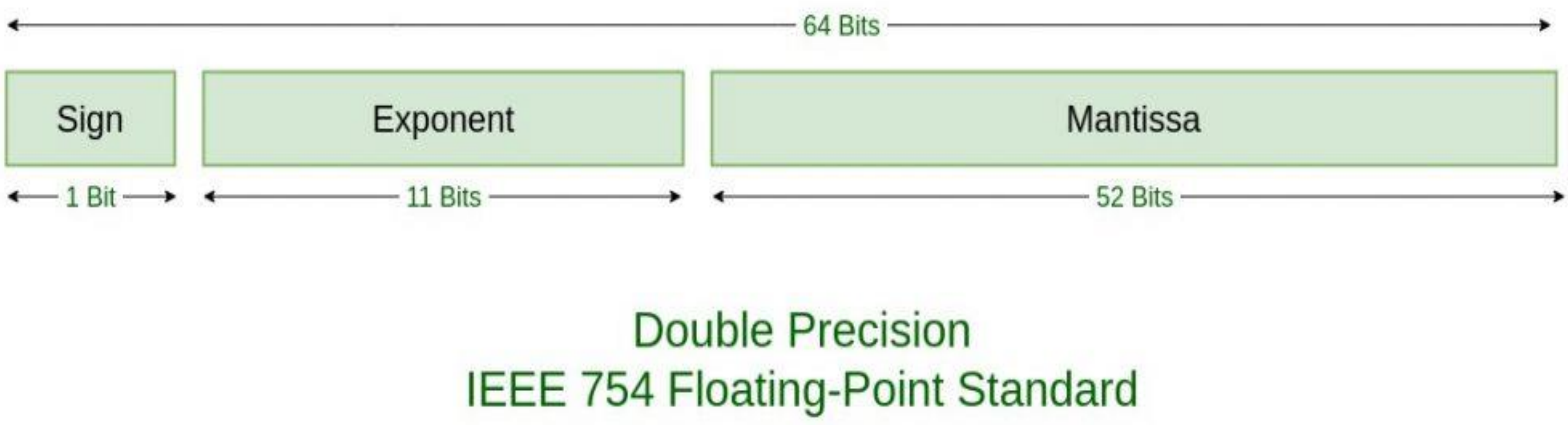
*Scientific Notation:*

$$\pm M \times B^{E}$$

| Numer | Mantissa | Base | Exponent |
|---|---|---|---|
| $9 \times 10^{8}$ | 9 | 10 | 8 |
| 4364.784 | 4364784 | 10 | -3 |

- **IEEE 754 numbers are divided into two based on the above three components: single precision and double precision.**






| TYPES | SIGN | BIASED EXPONENT | NORMALISED MANTISA | BIAS |
|---|---|---|---|---|
| Single precision | 1(31st bit) | 8(30-23) | 23(22-0) | 127 |
| Double precision | 1(63rd bit) | 11(62-52) | 52(51-0) | 1023 |

# Representation Format

Step 1 : Convert Decimal number into binary number.

Step 2 : Normalize the number

Step 3 : Single Precision format

$$[1.N]2^{E-127}$$

Step 4 : Double Precision format

$$[1.N]2^{E-1023}$$

# Eg.float num =10.75

- Step1. Covert Decimal number to binary number

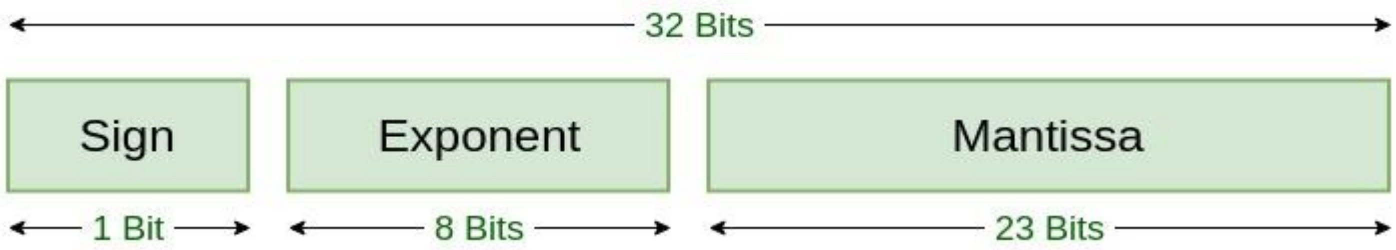
10 ->$(1010)_2$
0.75->$(11)_2$

(10.75)= 1010.11

- Step2: Normalize the number

1.Significat bit $* \ 2^{exponent}$
1.01011 x $2^3$
Here 01011 is significant bit
3 is a Exponent
Step3. Calculate Exponent
For Single precision
E-127=3
E=130
Convert 130 in binary form (10000010)

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |




Single Precision
IEEE 754 Floating-Point Standard

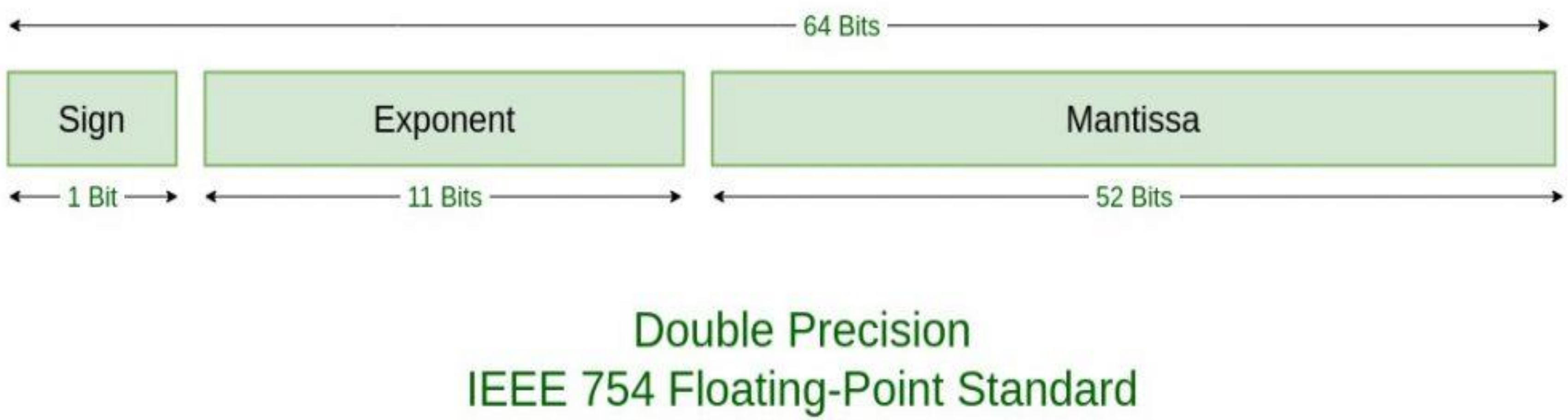
For Double precision

E-1023=3

E=1026

Convert 1026 In binary form

(10000000010)



# Example

1. 1259.125
2. 85.125
3. 17.125