# 3.1.3

# Real Numbers and Normalized Floating-Point Representation

Contents:

## Floating Point:

Floating point is a method of representing a real number in binary.

A real number is represented as:

$$-2.344 \times 2^3$$

Mantissa Exponent

The Mantissa holds the sign of the number.

The exponent holds the location of the decimal point is.

If the exponent is positive the decimal is moved to the right and vice versa.

## Representing Floating points in Binary:

**A binary floating point number has separate bits for the Mantissa and the Exponent.**

Both the mantissa and the exponent are in two's compliment form i.e. the value of the left most bit is negative.

| -1 | 1/2 | 1/4 | 1/8 | 1/16 | 1/32 | 1/64 | 1/128 | -8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |

Mantissa Exponent

## Binary Floating point to Decimal Conversion:

**Q: Convert 011000001010 to decimal.**

Step 1: Separately write mantissa and the exponent.

| -1 | 1/2 | 1/4 | 1/8 | 1/16 | 1/32 | 1/64 | 1/128 | -8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

Mantissa Exponent

The first 8 bits are always the Mantissa.

Step 2: Perform addition for the mantissa and the exponent.

- Mantissa: $^{1}/_{2}$ + $^{1}/_{4}$ = 0.5 + 0.25 = 0.75
- Exponent: -8 + 2 = -6

$0.5 = {}^{1}/_{2}$

$0.25 = {}^{1}/_{4}$

$0.125 = {}^{1}/_{8}$

Answer: **0.75 X $2^{-6}$**

## Decimal (>1 or improper fraction) to Binary Floating Point Conversion:

Q: Convert **+5.5** to its floating point equivalent.

Step 1: Write the decimal in fraction form.

- **5.5 = $^{55}/_{10}$ = $^{11}/_{2}$**

For conversion the:

1. The denominator must be base 2
2. The denominator must be 1 base 2 value more than the numerator.

Step 2: Divide the fraction into a fraction part and an exponent part.

- To create the exponent, change the value of the denominator so that it's one factor more than numerator (11). (You can only use power of 2 numbers like 1, 2, 4, 8, 16…) (The only value more than 11 is 16.)

  $^{11}/_{2}$ X $^{8}/_{8}$ = $^{11}/_{16}$ X 8

- $^{11}/_{16}$ is mantissa and **8** is the exponent.

Step 3: Form the fraction take the numerator and find its base 2 components.

- $^{11}/_{16}$ → 11 → 8 + 2 + 1

For 2's compliment, the left most bit is -128 and the rest are positive descending values.

For floating point, the left most bit of mantissa is -1 and the rest are ascending fractional values. Similarly, the left most bit of the exponent is -8 and the rest are positive descending values of base 2.

Step 4: Put the base 2 components in the fraction.

- $^{8}/_{16}$ , $^{2}/_{16}$ , $^{1}/_{16}$ → $^{1}/_{2}$ , $^{1}/_{8}$ , $^{1}/_{16}$

Step 5: Find the binary value for the exponent.

- **8 = $2^3$** i.e. value of exponent is **3**.
- Binary of **3** is **0011**

Step 6:

| -1 | 1/2 | 1/4 | 1/8 | 1/16 | 1/32 | 1/64 | 1/128 | -8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| Mantissa | | | | | | | | Exponent | | | |

## Decimal (<1 or proper fraction) to Binary Floating Point Conversion:

Q: Convert $^{11}/_{32}$ into binary floating point.

Step 1: Divide the fraction into a fraction part and an exponent part.

- To create the exponent, change the value of the denominator to so it's one factor more than the value of the numerator (11). (You can only use power of 2 numbers like 1, 2, 4, 8, 16…) (The only value more than 11 is 16.)

  $^{11}/_{32} = {}^{11}/_{16} \times {}^{1}/_{2}$

- $^{1}/_{2}$ can be written as $2^{-1}$ so, $^{11}/_{16}$ is the mantissa and **-1** is the exponent.

Perform **step 3 – 6** in the same way.

## Negative Decimal to Binary:

For negative decimal values after **step 2** we convert the fraction in such a way that one part is -1 and the other part is the fraction i.e.

$-11/16 = -1 + 5/16$

Then we perform **step 3** on $5/16$.

In **step 6** we write 1 in the most significant bit (-1)

Trick:

$-n/d = -1 + d - n/d$

e.g.: $-11/16 = -1 + 16 - 11/16$

## Normalization:

Normalization is the scientific notation equivalent for binary.

It provides the maximum precision with the bits available.

## Identifying Normalization:

**The first 2 bits of the mantissa of a normalized number are always different.**

For positive normalized number the first two bits should be: **01**

For negative normalized number the first two bits should be: **10**

## Normalizing a Floating Point Binary Number:

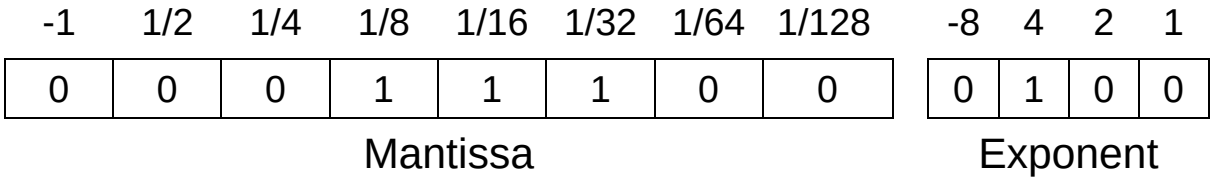
To normalize a number, we shift all the values of the mantissa except the value in the most significant bit (as that bit holds the sign of the number)

If the values of the mantissa are moved **right,** the overall value decreases, thus the value of the exponent is **increased by 1** (+1).

If the values of the mantissa are moved **left,** the overall value increases, thus the value of the exponent is **decreased by 1** (-1).

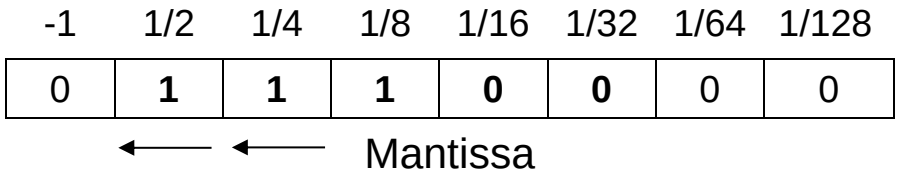
(NOTE: Always shift the mantissa to the left to prevent errors.)

Q: Normalize **000111000100**



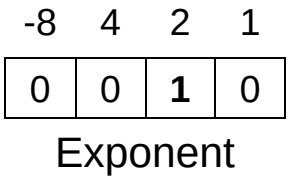

Step 1: Move the values of the mantissa

- In order to normalize the values of the mantissa should be moved left 2 bits.

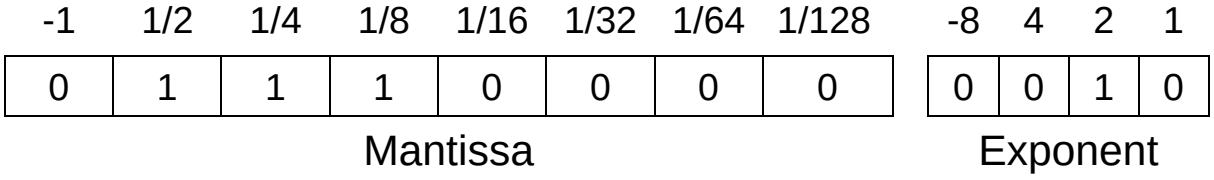



Step 2: Adjust the exponent.

- As the mantissa is moved 2 bits to the left, the overall value has increased, thus values of the exponent must be decreased by 2.
- **4 – 2 = 2**




Answer:




**The value in the most significant bit should not be shifted as it holds the sign of the number.**

## Effects of changing allocated bits to Mantissa and Exponent:

If the number of bits allocated to **the mantissa are increased**, the value becomes more **accuracy**.

If the number of bits allocated to the **exponent are increased**, the **range** of the value that can be stored increases.

## Overflow:

Overflow occurs when the byte (container) cannot hold all the values. This means that the value is greater than the largest value that can be represented.

## Underflow:

Underflow occurs when the number being stored is smaller than the smallest number that can be represented within the byte (container).

## Approximation:

Some values can not be stored in the mantissa, so the computers store the other closest approximation of the actual value.

## Rounding Error:

**The binary representation is only an approximation of the real number it represents (in certain cases). This approximation can lead to rounding errors.**

Suppose you wish to represent the number 1/7 in decimal. This number goes on infinitely repeating the sequence “0.142857”.

So, in order to store it we will round it off to $1.43 \times 10^{-1}$.

Applying complex and long mathematical functions on the approximated values will increase the rounding error.

For example, if you have a lot of digits, your rounding error might seem insignificant. But if you add up these rounded numbers repeatedly for a long period of time the rounding error will increase, and the calculated value will deviate from the correct final mathematical value.

## Maximum and Minimum representable for a given number of bits:

| | Mantissa | Exponent |
|---|---|---|
| Largest Positive | 01111111 | 01111111 |
| Largest Negative | 10000000 | 01111111 |
| Smallest Positive | 01000000 | 10000000 |
| Smallest Negative | 10111111 | 10000000 |