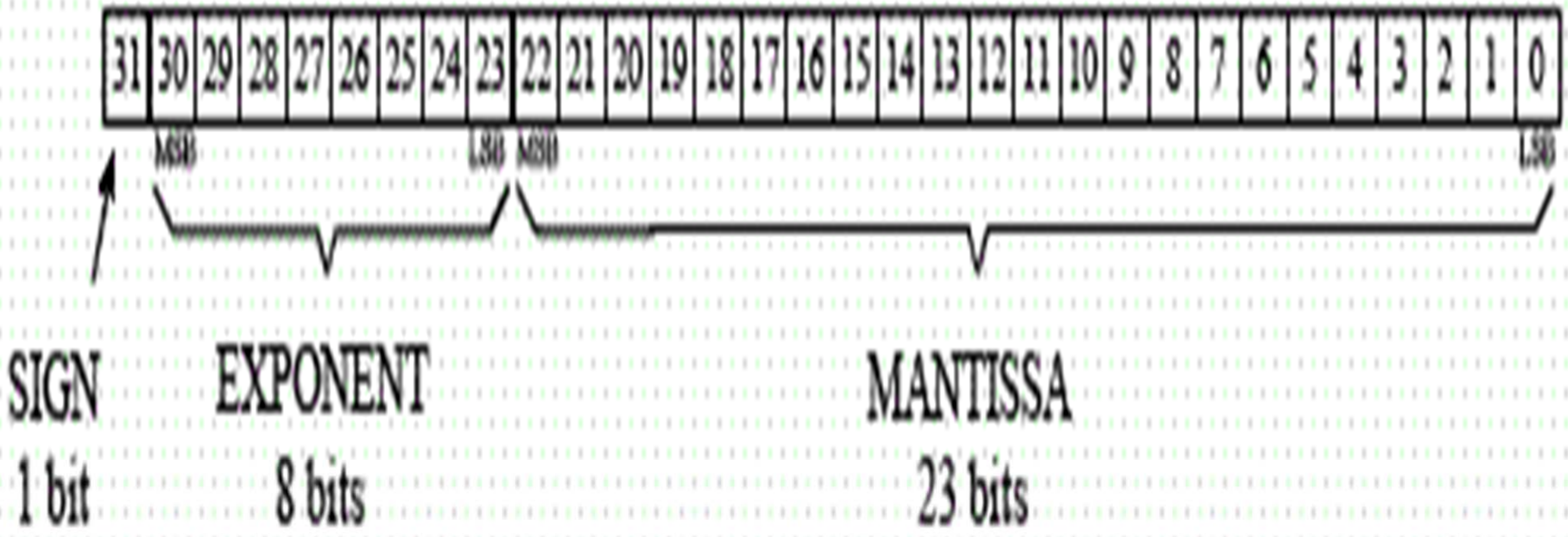
General representation of floating point



- 17 (decimal) = 10001 (binary)
- 10001 = 0.10001 x 2^5
- Then, we can now construct its representation

1bit	5 bits	8 bits
0	**00101**	**1000100**

sign field:

0 : positive value

1 : negative value

What if we want to store a negative exponent value?

- The previous example can't handle this problem, thus we could fix that by using **biased exponent**.
- For example, if we want to store 0.25, we will have 0.1 x 2^-1
- We can fix this by using excess-16 representation. So that we add 16 to the negative exponent (-1 + 16 = 15).

0	**01111**	**1000000**

Another problem using this method

0	10101	10001000
0	10110	01000100
0	10111	00100010
0	11000	00010001

= 17

We don’t have a unique representation for each number.

Remedy

- This problem can be fixed by **normalization.**
- Normalization is a convention that the leftmost bit of the significand must always be 1. So that we only have

0	01111	1000100

for decimal value 17.

Floating Point Arithmetic

- Addition

0	10010	11001000

0	10000	10011010

```
11.001000
  0.10011010
____________  +
11.10111010
```

0	10010	11101110

Multiplication

0	10010	11001000

t = 0.11001000 x 2^2

0	10000	10011010

= 0.10011010 x 2^0

0.11001000 x 0.10011010 = 0.0111100001010000

2^2 x 2^0 = 2^2

0	10001	11110000

Some other problems in floating point arithmetic

- Division by zero.
- Overflow, if the result is greater in magnitude than the given storage.
- Underflow, if the result is smaller in magnitude than the given storage.

The IEEE-754 Floating-Point Standard

- This was first introduced in 1985.
- This type of floating point includes two formats: single precision and double precision.

Single Precision IEEE-754

1bit	8 bits	23bits

- This representation uses an excess-127
- This representation assumes an implied 1 to the left of the radix point, for example we put 1 = 1.0 x 2^(0+127)

Floating Point Number	Single Precision Representation
1.0	0 01111111 00000000000000000000000
0.5	0 01111110 00000000000000000000000
19.5	0 10000011 00111000000000000000000
-3.75	1 10000000 11100000000000000000000

Double Precision IEEE-754

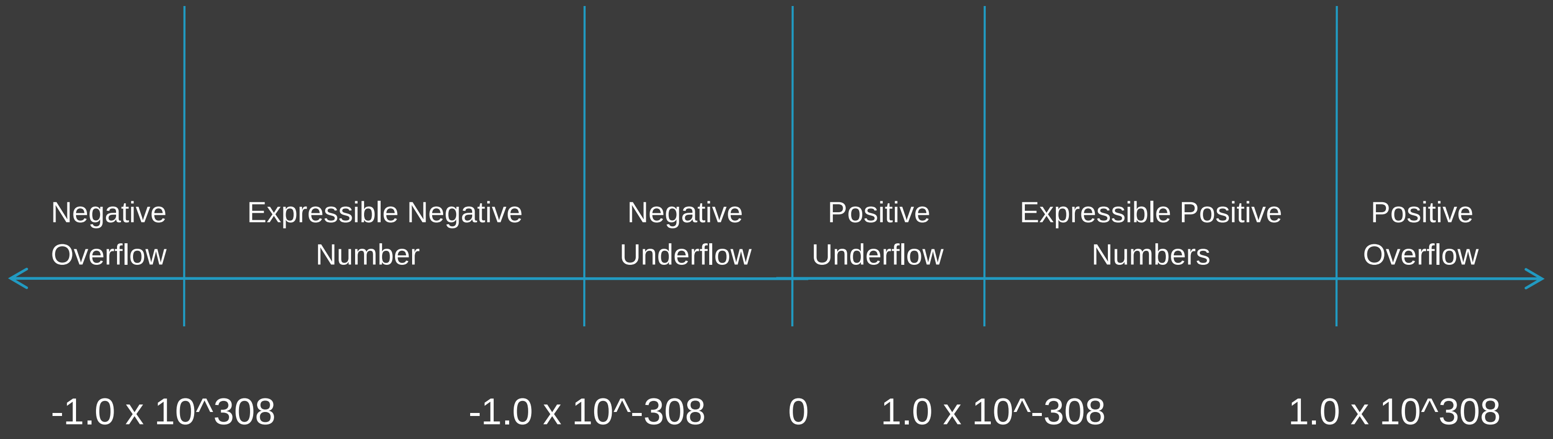
1bit	11 bits	52 bits

- This representation uses an excess-1023
- This representation assumes an implied 1 to the left of the radix point, for example we put 1 = 1.0 x 2^(0+1023). (same as the single precision)

Range, Precision, and Accuracy

- Range

  In double precision, for example, we have


Negative Overflow
Expressible Negative Number
Negative Underflow
Positive Underflow
Expressible Positive Numbers
Positive Overflow
-1.0 x 10^308
-1.0 x 10^-308
0
1.0 x 10^-308
1.0 x 10^308