

# Floating Point

Ref: 2.4

---

---

---

---

---

---

---

---

# Decimal Floating Point

3.141593

$6.02 \times 10^{23}$

33.33333...

$1.0 \times 10^{-9}$

*“decimal point”*

*Scientific Notation*

*Normalized Numbers*

---

---

---

---

---

---

---

---

# Binary Floating Point

100.0100

1.111111...

$.001 \times 2^5$

$1.001 \times 2^{17}$

*Binary Point*

*Positional Representation  
(negative powers of 2)*

*Normalized Numbers*

---

---

---

---

---

---

---

---

## Binary Normalization

Normalized: one *digit*  
to the left of the binary point.  
It must be a 1!

$$101.0111 \times 2^{13}$$

↓ normalize

$$1.010111 \times 2^{15}$$

We still use the term *digit*,  
although we mean "0" or "1".

$$1.010111 \times 2^{00001111}$$

↑  
exponents are binary !

CompOrg - Floating Point

4

---

---

---

---

---

---

---

---

## Representation

- For each binary floating point number we need:
  - sign
  - significand (mantissa).
  - exponent
    - need a signed exponent!

CompOrg - Floating Point

5

---

---

---

---

---

---

---

---

## Choices

- Suppose we want to store floating point numbers in 32 bits.
  - we need to decide how many bits should be used for the significand and how many for the exponent.
  - There is a tradeoff between *range* and *accuracy*.

CompOrg - Floating Point

6

---

---

---

---

---

---

---

---

## Desirable properties of a floating point format.

- Large Range – large and small exponents
- High Accuracy – make the most out of the significand.
- We want it to be *easy* to compare two numbers.

CompOrg - Floating Point

7

---

---

---

---

---

---

---

---

## IEEE 754 floating point standard

- Folks realized that it was silly to have different floating point formats on different computers:
  - sharing of data was a hassle.
  - an algorithm written to work with one format might need to be adjusted to work with other formats.
- Today, just about all computers support IEEE 754 format.

CompOrg - Floating Point

8

---

---

---

---

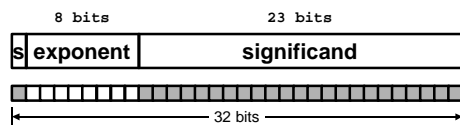
---

---

---

---

## 32 bit IEEE 754 format



CompOrg - Floating Point

9

---

---

---

---

---

---

---

---

## Sign and Magnitude

- Sign Bit:  
– 0 means positive, 1 means negative

Value of a number is:

$$(-1)^s \times F \times 2^E$$

← exponent  
↑ significand  
as we will see, IEEE 754 is more complex than this!

CompOrg - Floating Point

10

---

---

---

---

---

---

---

---

## Normalized Numbers and the significand

- Normalized binary numbers *always* start with a 1 (the leftmost bit of the significand value is a 1).
- Why store the 1 (it's always there)?
- IEEE 754 uses this, so the significand is really 24 bits (but only 23 need to be stored).
- All numbers must be normalized!

CompOrg - Floating Point

11

---

---

---

---

---

---

---

---

## A Tradeoff

- If  $-x$  is the smallest exponent (most negative), then the smallest number that can be represented as a normalized number:

$$1.000000000000000000000000 \times 2^{-x}$$

- If we don't require normalization we could represent

$$0.000000000000000000000001 \times 2^{-x-23}$$

CompOrg - Floating Point

12

---

---

---

---

---

---

---

---

## Denorms

- IEEE 754 actually supports *denormalized* numbers, but not all vendors support this part of the standard.
  - it adds a lot of complexity to the implementation of floating point arithmetic.
  - complexity means loss of speed (usually).

CompOrg - Floating Point

13

---

---

---

---

---

---

---

---

## Exponent Representation

- We need negative and positive exponents.
- Could use 2s complement notation
  - this would make comparison of floating point numbers a bit tricky.
    - exponent value 11111111 is smaller than 00000000.
- Instead they chose a *biased* representation.
  - exponent values are offset by a fixed bias.

CompOrg - Floating Point

14

---

---

---

---

---

---

---

---

## 32 bit IEEE 754 exponent

- The exponent uses 8 bits.
- The *bias* is 127.
  - treat the 8 bit exponent as a unsigned integer and subtract 127 from it.

**00000001** is the representation for -126

**10000000** is the representation for +1

**11111110** is the representation for +127

CompOrg - Floating Point

15

---

---

---

---

---

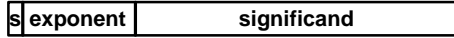
---

---

---



## Comparing Numbers



- Comparison of *normalized* floating point numbers:
  - check sign bits
  - check exponents.
    - unsigned integer comparison works. Larger exponents are represented by larger unsigned ints.
  - check significand.

CompOrg - Floating Point

19

---

---

---

---

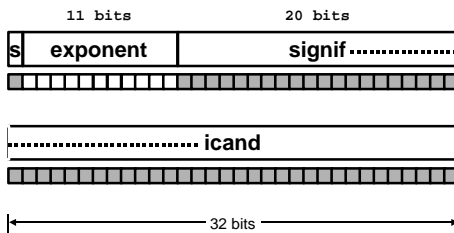
---

---

---

---

## Double Precision



CompOrg - Floating Point

20

---

---

---

---

---

---

---

---

## 64 bit IEEE 754

- exponent is 11 bits
  - bias is 1023
  - range is a *little* larger than the 32 bit format.
- Significand is 55 bits
  - plus the leading 1.
  - accuracy is much better than 32 bit format.

CompOrg - Floating Point

21

---

---

---

---

---

---

---

---

## Example Representations

$0.75_{10} \rightsquigarrow \frac{1}{2} + \frac{1}{4} \rightsquigarrow 0.11 \times 2^0 \rightsquigarrow 1.1 \times 2^{-1}$

0 01111110 100000000000000000000000

s exponent

significand

↑  
As unsigned int is 126.  
 $126 - 127 = -1$

↑  
Leading 1 is not stored!

CompOrg - Floating Point

22

---

---

---

---

---

---

---

---

## What number is this?

0 10000001 110000000000000000000000

s exponent

significand

You get 7 guesses.  
If you get it wrong we will do 7 more of these.

CompOrg - Floating Point

23

---

---

---

---

---

---

---

---

## Exercises

- What is the double precision (64 bit format) representation for the number 128?
- What is the single precision format for the number  $-8.125$ ?

CompOrg - Floating Point

24

---

---

---

---

---

---

---

---

## Floating Point Addition

What is the sum of  $1,234,823.333 + .0011$ ?

- Need to line up the decimal points first!
  - This is the same as shifting the significand while changing the exponents.

$$1,234,823.333 = 1.234823333 \times 10^6$$

$$.0011 = 1.1 \times 10^{-3} = 0.0000000011 \times 10^6$$

CompOrg - Floating Point

25

---

---

---

---

---

---

---

---

## Binary Floating Point Addition

Just like decimal:

- Line up the binary points
  - Shift one of the numbers
- Add significands (using integer addition)
- Normalize the result
- Might need to round the result or truncate.

CompOrg - Floating Point

26

---

---

---

---

---

---

---

---

## Floating Point Multiplication

- $1.3 \times 10^3 \text{ times } 3.0 \times 10^{-2} = 3.9 \times 10^1$

Add exponents

Multiply significands

Normalize result.

CompOrg - Floating Point

27

---

---

---

---

---

---

---

---

## Rounding

- Intermediate results (in the middle of multiplication or addition operations) might not fit.
- The internal representation of intermediate values uses 2 extra bits: *round* and *guard*.

CompOrg - Floating Point

28

---

---

---

---

---

---

---

---

## Decimal Rounding Example

- Add 2.56 to  $2.34 \times 10^2$
- Assume we have only 3 significant decimal digits.

$$\begin{array}{r} 2.3400 \\ + 0.0256 \\ \hline 2.3656 \end{array} \rightarrow 2.37$$

guard    round

$$\begin{array}{r} 2.34 \\ + 0.02 \\ \hline 2.36 \end{array}$$

without round and  
guard digits

CompOrg - Floating Point

29

---

---

---

---

---

---

---

---