# Floating point arithmetic (FPA)

* Most scientific computations are performed using real numbers

* Assessing accuracy of algorithms in NLA requires understanding of few topics in FPA.

* Science, Engg, ML/AI usually requires 7 to 8 significant digits precision

How do we calculate number of bits to encode 8 decimal digits? (Decimal digits are basically digits represented in base 10 numeral system)

* Recall base 10 :-

→ 1 digit .... 10 possibilities (0 .... 9)
→ Largest decimal digit [10 - 1]

→ 2 digits .... 100 possibilities (0 ... 99)
→ Largest decimal digit [100 - 1]

→ 3 digits .... 1000 possibilities (0 .... 999)
→ Largest decimal digit [1000 - 1]

⋮

→ n digits .... $10^n$ possibilities

→ Largest decimal digit $[10^n - 1]$

* Recall base 2

→ 1 digit ... 2 possibilities (0, 1)

→ Largest decimal digit 1 $(2^1 - 1)$

→ 2 digits ... 4 possibilities (01, 10, 11, 00)

→ Largest decimal digit $2^2 - 1 = 3$

⋮

n digits ... $2^n$ possibilities

→ Largest decimal digit $2^n - 1$

* In a base b, the largest decimal number you can represent in N digits is $b^N - 1$

Now

* No of bits to encode 8 decimal digits

Base 2 Base 10

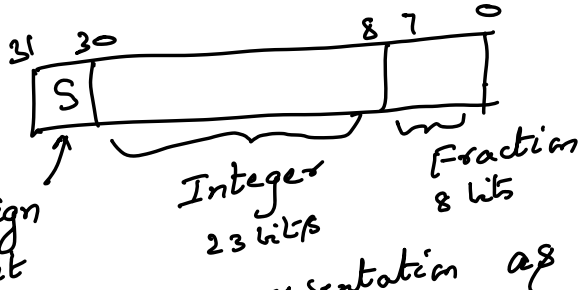
$2^N - 1 = 10^9 - 1$

$\Rightarrow N = 8 \log_2 10 \approx 26$ bits

* Computers store numbers as a sequence of 8 bit bytes. Thus 32 bits is a good size to use for storing real numbers i.e 8 decimal digits of precision!

* Given 32 bits to encode real numbers! Next how to break it up into integer & fractional part

* One method is to divide 32 bits into 2 parts, where one part to represent integer and another part to represent fraction.




Fixed point representation as we are fixing the demarcation between integer and fractional part b/w 7 and 8 bits

Largest number in above fixed point representation → $+\underbrace{111\ldots1}_{23\text{ bits}}.\underbrace{11111111}_{8\text{ bits}} = 1677215.99804687_5$

Smallest number → $+\underbrace{000\ldots0}_{23\text{ bits}}.\underbrace{00000001}_{8\text{ bits}}$

$= 0.00390625$

Drawback :-

Range of real numbers represented by the above largest and smallest may not be sufficient for many practical problems.

Binary floating point numbers :-

The same 32 bits are represented into two parts, a part called mantissa with its sign and other called exponent.

→ Mantissa :- binary fraction with non-zero leading bit

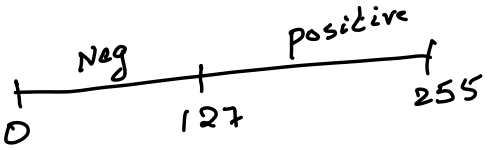
→ exponent :- binary integer

$$(\underbrace{\text{Sign}}_{1\text{ bit}}) \times \underbrace{\text{mantissa}}_{23\text{ bits}} \times 2^{\overbrace{\text{exponent}}^{8\text{ bits}}}$$

8 bits if all are 1's, the max exponent is 255 and min exponent is 0

We need to give flexibility to include negative exponents and biased representation is introduced.

The range 0 to 255 of an 8-bit exponent is divided into two parts




0 to 126 → Negative (-127 to -1)

127 → 0

128 to 255 → Positive (1 to 128)

} exp -127

Largest decimal number in the above representation

$$0.\underbrace{1111\ldots}_{23\text{ bits}} 1 \times 2^{\overbrace{11111111}^{8\text{ bits}}}$$

$$[1 \times 2^{-23} + 2^{-22} + 2^{-21} + \dots 2^{-1}] \times 2^{255}$$

But in our biased representation

$$[2^{-23} + 2^{-22} + \dots 2^{-1}) \times 2^{(255-127)}$$

$$\approx 3.4 \times 10^{38}$$

Smallest decimal number in above representation :-

$$0.\underbrace{100\cdots0}_{23 \text{ bits}} \times 2^{\overbrace{00000000}^{8 \text{ bits}}}$$

$$(1 \times 2^{-1}) \times 2^{0}$$

In our biased representation

$$(1 \times 2^{-1}) \times 2^{(0-127)}$$

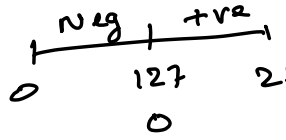
$$= \frac{2^{-127}}{2} \approx 0.293 \times 10^{-38}$$

Two's complement

0 : 00000000
1 : 00000001
2 : 00000010
126 : 01111110
127 : 01111111 ✓
-128 : 10000000
-127 : 10000001
-126 : 10000010
-2 : 11111110
-1 : 11111111

IEEE Standard (32 bit storage)

1. Sign → 1 bit allocated
2. Mantissa → 23 bits (Left of (virtual) decimal point is always 1) (Normalized significand)
3. Exponent → 8 bits




0 to 126 → negative
(-127 to -1)

127 → 0

128 to 255 → positive
(1 to 128)

| $b_0$ | $b_1\ b_2 \cdots b_8$ | $b_9\ b_{10}\ b_{11} \cdots b_{30}\ b_{31}$ |
|---|---|---|
| Sign | exponent | Significand |

$$(-1)^S \times (1.f)_2 \times 2^{\text{exponent}-127}$$

S → Signed bit (0 for +ve, 1 for negative)

f → represents bits in significand

This extra bit 1 in 1.f is automatically implied and increases the precision

Largest positive number $\approx 3.403 \times 10^{38}$

Smallest positive number $\approx 1.1755 \times 10^{-38}$

Eg:- Represent 52.21875 in IEEE single precision (32 bits)

$52.21875 = 110100.00111$

$= 1.10100000111 \times 2^5$

Biased representation

$exp - 127 = 5$

$exp = 132$

$= 1.1010000111 \times 2^{132}$

| 0 | 10000100 | 10100001110 0 ... 0 |
|---|---|---|
| sign | 8 bits | 23 bits |

**Double precision :-** 64 bits

Sign → 1 bit ✓

exponent → 11 bits ✓

Significand (mantissa) → 52 bits ✓

Largest number :- $1.79 \times 10^{308}$ ✓

Smallest number :- $2.23 \times 10^{-308}$ ✓

Larger range is possible

$(-1)^S \times (1.f) \times 2^{(exp-1023)}$

**Gaps between numbers :-**

Let us look at difference between 1 and next larger number that can be stored in the format we are interested in

Eg: **IEEE Single precision :-**

$1.\underbrace{00000\ldots0}_{23 \text{ bit}} \times 2^0 = 1.0$

Next largest number

$$1.\underbrace{000\cdots001}_{23\text{ bits}} \times 2^0 = 1 + 2^{-23}$$

Hence the interval $[1,2]$ is represented by the discrete subset

$$\{1,\ 1+2^{-23},\ 1+2\times2^{-23},\ \cdots\ 2\} \quad -(*)$$

The interval $[2,4]$ is represented by the discrete subset

$$2,\ 2+2\times2^{-23},\ 2+4\times2^{-23},\ \cdots\ 4$$

In general, the interval $[2^j, 2^{j+1}]$ is represented by multiplying eq(*) times $2^j$

Thus in IEEE single precision arithmetic the gaps between adjacent numbers in a relative sense are never larger than

$$2^{-23} \approx 1.19\times10^{-7}$$

For IEEE double precison gaps between numbers in a relative sense are never larger than $2^{-52} \approx 2.22\times10^{-16}$

## Machine Epsilon :-

Consider the discrete subset $\mathbb{F}$ of real numbers $\mathbb{R}$ to be our floating point number system depending on the precision we choose, then for all $x \in \mathbb{R}$ there exists $x' \in \mathbb{F}$ such that

$$\boxed{\frac{|x - x'|}{|x|} \leq \varepsilon_{mach}} \quad \text{— (**)}$$

This $\varepsilon_{mach}$ is called machine epsilon

Let $fl : \mathbb{R} \to \mathbb{F}$ is a function that takes a real number and round it off to nearest floating point number.

This equation (**) means for all $x \in \mathbb{R}$ there exists $\varepsilon$ with $|\varepsilon| \leq \varepsilon_{mach}$ such that $\boxed{fl(x) = x(1+\varepsilon)}$

i.e relative difference b/w a real number $x$ and its closest floating point number is always

smaller than $\varepsilon_{mach}$.

Usually this $\varepsilon_{mach}$ is defined as half the distance between 1 and next largest floating point number $1 \cdots$

In single precision :-

$$\varepsilon_{mach} \simeq \frac{2^{-23}}{2} \simeq 5.96 \times 10^{-8}$$

Double precision :-

$$\varepsilon_{mach} \simeq \frac{2^{-52}}{2} \simeq 1.11 \times 10^{-16}$$

Fundamental operations in floating point arithmetic :-

Classical arithmetic operations

$+, -, \times$ and $/$

are operations in $\mathbb{R}$

On a computer, we have analogous operations on IF, denote these operations as $\oplus, \ominus, \otimes$, and $\oslash$

Let $x, y \in \mathbb{F}$, $*$ denotes $+, -, \times$ and $/$ and $\circledast$ denotes $\oplus, \ominus, \otimes$ and $\oslash$, there exists $\varepsilon$ with $|\varepsilon| < \varepsilon_{Mach}$ such that

$$x \circledast y := fl(x * y) = x * y(1+\varepsilon)$$

i.e every operation of floating point arithmetic is exact upto a relative error of size atmost $\varepsilon_{mach}$