

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]
 - ⋮

$\rightarrow n$ digits $\dots 10^n$ possibilities
 \rightarrow Largest decimal digit $[10^n - 1]$

* Recall base 2

$\rightarrow 1$ digit $\dots 2$ possibilities $(0, 1)$
 \rightarrow Largest decimal digit $1 (2^1 - 1)$

$\rightarrow 2$ digits $\dots 4$ possibilities $(00, 01, 10, 11)$
 \rightarrow Largest decimal digit $2^2 - 1 = 3$

:

n digits $\dots 2^n$ possibilities
 \rightarrow Largest decimal digit $2^n - 1$

* In a base b , the largest decimal number you can represent in n digits

is $b^n - 1$

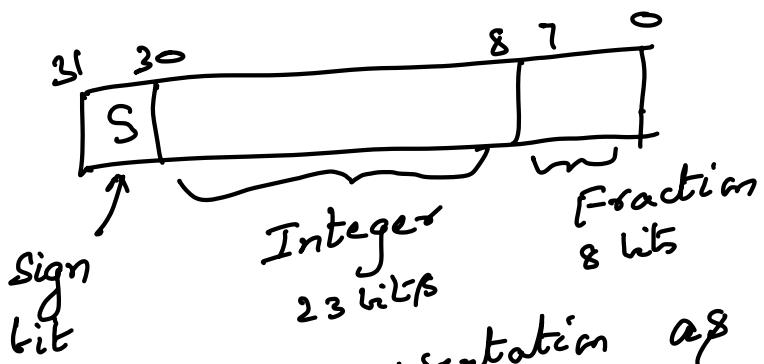
Now
* No of bits to encode 8 decimal digits

Base 2 Base 10

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

$$\Rightarrow n = 8 \log_2 10 \approx 26 \text{ bits}$$

- * 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 parts
- * One method is to divide 32 bits into 2 parts, where one part to represent integers 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

$$+ \underbrace{111 \dots 1}_{23 \text{ bits}} \cdot \underbrace{1111111}_{8 \text{ bits}} = 1677215.998046875$$

Smallest number

$$+ \underbrace{000 \dots 0}_{23 \text{ bits}} \cdot \underbrace{00000001}_{8 \text{ bits}} = 0.00390625$$

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

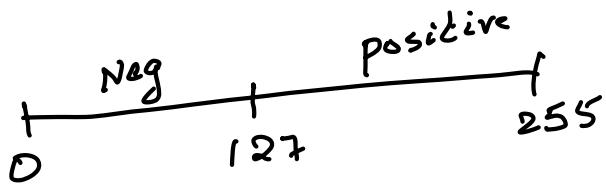
→ Exponent :- binary integer

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

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 \rightarrow negative (-127 to -1)

127 \rightarrow 0
128 to 255 \rightarrow positive (1 to 128)

Largest decimal number in the above representation

$$0.\underbrace{1111\dots1}_{23 \text{ bits}} \times 2^{\overbrace{111111}^{8 \text{ bits}}}$$

$$\xrightarrow{\quad} 10^6$$

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

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 :-

$$0.100\dots 0 \times 2^{\overbrace{00000000}^{8 \text{ bits}}}$$

$\underbrace{\quad}_{23 \text{ bits}}$

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

In our biased representation

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

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

(32 bit storage)

IEEE Standard

① Sign \rightarrow 1 bit allocated

② Mantissa \rightarrow 23 bits (left of decimal point)
(Normalized significand) (virtual)

③ Exponent \rightarrow 8 bits

Two's complement

0 : 00000000

1 : 00000001

2 : 00000010

126 :

01111110

127 :

01111111

-128 :

10000000

-127 :

10000000

-126 :

10000010

-2 :

11111110

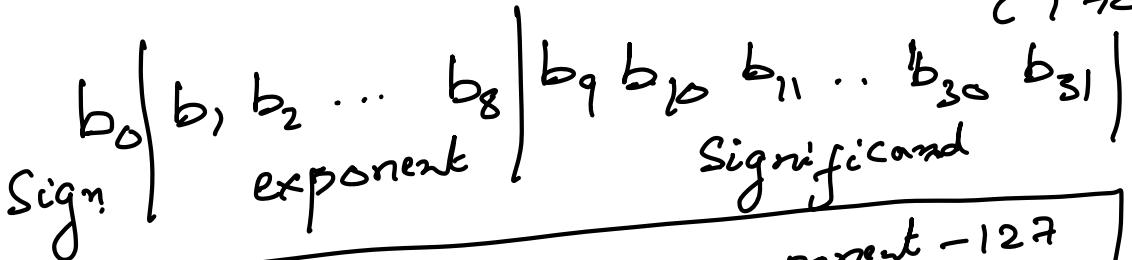
-1 :

11111111



$127 \rightarrow 0$

$128 \text{ to } 255 \rightarrow \text{positive}$
(1 to 128)



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

$s \rightarrow$ signed bit (0 for +ve, 1 for negative)

$f \rightarrow$ 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)

$$52.21875 = 110100.00111$$

$$= 1.1010000111 \times 2^5$$

biased representation

$$\text{exp} - 127 = 5$$

$$\text{exp} = 132$$

$$= \underbrace{1.1010000111}_{\text{sign}} \times 2^{132}$$

0	10000100	10100001100...0
sign	8 bits	23 bits

Double precision :- 64 bits

Sign \rightarrow 1 bit ✓

exponent \rightarrow 11 bits ✓

Significand
(mantissa) \rightarrow 52 bits ✓

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

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

Large range is possible
 $(-1)^f \times (1.f) \times 2^{\text{exp}-1023}$

Gaps between numbers:-

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

Eg: IEEE Single precision :-

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

Next largest number

$$\underbrace{1.000\cdots00}_\text{23 bits} \times 2^0 = 1 + 2^{-23}$$

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

$$\{1, 1 + 2^{-23}, 1 + 2 \times 2^{-23}, \dots, 2\} \rightarrow \text{---} \times$$

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

$$\underbrace{2, 2 + 2 \times 2^{-23}, 2 + 4 \times 2^{-23}, \dots, 4}_{\text{---}}$$

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

$$2^{-23} \approx 1.19 \times 10^{-7}$$

For IEEE double precision gaps between numbers in a relative sense are never larger than $2^{-52} \approx 2.22 \times 10^{-16}$

Machine Epsilon :-

Considers 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

$$\boxed{\frac{|x - x'|}{|x|} \leq \epsilon_{\text{mach}}} \quad - \text{**}$$

This ϵ_{mach} is called machine epsilon

Let $f_l : \mathbb{R} \rightarrow \mathbb{F}$ 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 ϵ with $|\epsilon| \leq \epsilon_{\text{mach}}$ such that

$$\boxed{f_l(x) = x(1 + \epsilon)}$$

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

smaller than ϵ_{mach} .

Usually this ϵ_{mach} is defined as half the distance between 1 and next largest floating point number

In single precision :-

$$\epsilon_{mach} \approx \frac{2^{-23}}{2} \approx 5.96 \times 10^{-8}$$

Double precision :-

$$\epsilon_{mach} \approx \frac{2^{-52}}{2} \approx 1.11 \times 10^{-16}$$

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, denotes these

operations as \oplus , \ominus , \otimes , and \oslash

Let $x, y \in \mathbb{F}$, $*$ denotes $+, -, \times$ and $/$
 and \circledast denotes $(+), (-), (\times)$ and (\div) , there
 exists ε with $|\varepsilon| < \varepsilon_{\text{mach}}$

such that

$$\boxed{x \circledast y := f[(x * y)] \\ = x * y (1 + \varepsilon)}$$

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