Floating Point Numbers Representation

Philippe Bonnet, bonnet@di.ku.dk
HPPS 2025 – 2a

(with slides from T.Henriksen)

Outline

- Numerical errors
- The fixed-point dilemma
- Floating-point representation
- IEEE754 and beyond

Numerical errors with floats

- Comparison errors
  - Never check for floating point numbers equality
  - Check that their difference is within a given bound
- Loop errors
  - Accumulation of errors in a loop
- Small number may get lost in relation to large number

See examples in 2a-code/

Fractional numbers

$$123.456 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2} + 6 \cdot 10^{-3}$$

Generally

$$a_{m-1} \cdots a_0 . a_{-1} \cdots a_{-n} = \sum_{i=-n}^{m-1} a_i \cdot 10^i$$

Even more generally, for radix r

$$a_{m-1} \cdots a_0 . a_{-1} \cdots a_{-n} = \sum_{i=-n}^{m-1} a_i \cdot r^i$$

Fractional binary numbers

Weight	2^{m-1}	2^{m-2}	$\cdots$	4	2	1	1/2	1/4	1/8	$\cdots$	2^{-n}
Digits	b_{m-1}	b_{m-2}	$\cdots$	b_2	b_1	b_0	b_{-1}	b_{-2}	b_{-3}	$\cdots$	b_{-n}

Representation

- Bits to the right of "binary point" represents fractional powers of 2.
- Represents rational number

$$\underbrace{b_{m-1} \cdots b_0}_{\text{integral part}} . \underbrace{b_{-1} \cdots b_{-n}}_{\text{fraction part}} = \sum_{i=-n}^{m-1} b_i \cdot 2^i$$

Fractional binary numbers

Value	Representation
$5\frac{3}{4}$	101.11_2
$2\frac{7}{8}$	10.111_2
$1\frac{7}{16}$	1.0111_2

Observations

- Divide by 2 by logical shifting right.
- Multiply by 2 by shifting left.
- Numbers of form 0.111 . . . are just below 1.0.
  - $1/2 + 1/4 + 1/8 + \cdots 1/2^n + \cdots \sim 1.0$.
  - Use notation $1.0 - \epsilon$.

Fractional binary numbers -- limitations

Problem for fixed size Types!

Limitation #1

- Can only represent fractional part of form $x/2^k$
- Other rational numbers have repeating binary representation

Value	Representation
$\frac{1}{3}$	$0.0101010101[01]\cdots_2$
$\frac{1}{5}$	$0.001100110011[0011]\cdots_2$
$\frac{1}{10}$	$0.0001100110011[0011]\cdots_2$

Limitation #2

- Just one setting of binary point within the w bits.
  - Limited range of numbers—very small values? Very large?

The fixed-point dilemna

Consider $w = 8$

1 **bit for fraction**

- Largest number: $1111111.1_2 = 127.5_{10}$
- Increment: $0000000.1_2 = 0.5_{10}$

7 **bits for fraction**

- Largest number: $1.1111111_2 = 1.9921875_{10}$
- Increment: $0.0000001_2 = 0.0078125_{10}$

4 **bits for fraction**

- Largest number: $1111.1111_2 = 15.9375_{10}$
- Increment: $0000.0001_2 = 0.0625_{10}$

Uniform increments (i.e., same precision) everywhere
=> Relative precision is worse close to 0!

Floating-point representation

Numerical form

$$(-1)^s \cdot m \cdot 2^e$$

Significand is also called Mantissa.

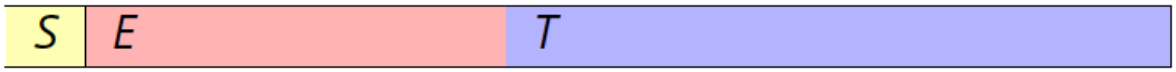
- **Sign bit** s determines whether number negative or positive.
- **Significand** m normally a fractional value in range $[1, 2)$.
- **Exponent** e weights value by power of two.

Encoding

- Most significant bit is sign bit.
- E field encodes e (but is not equal to e).
- T field encodes m (but is not equal to m).

Numerical errors!

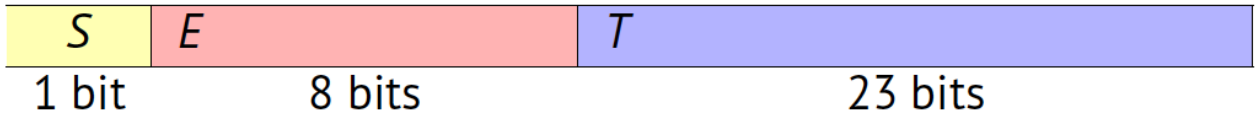
How are E and T encoded?



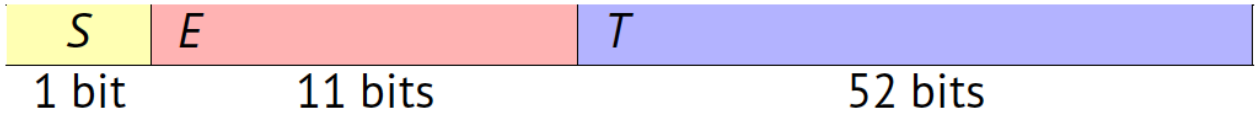
$(-1)^s \cdot m \cdot 2^e$

Precision (IEEE Standard 754)

32-bit single precision: `float`

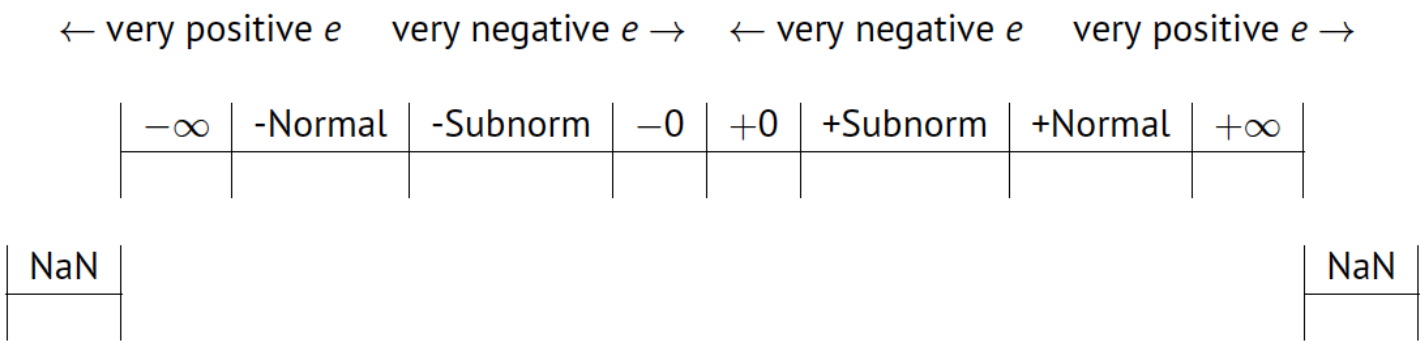


64-bit double precision: `double`



$(-1)^s \cdot m \cdot 2^e$

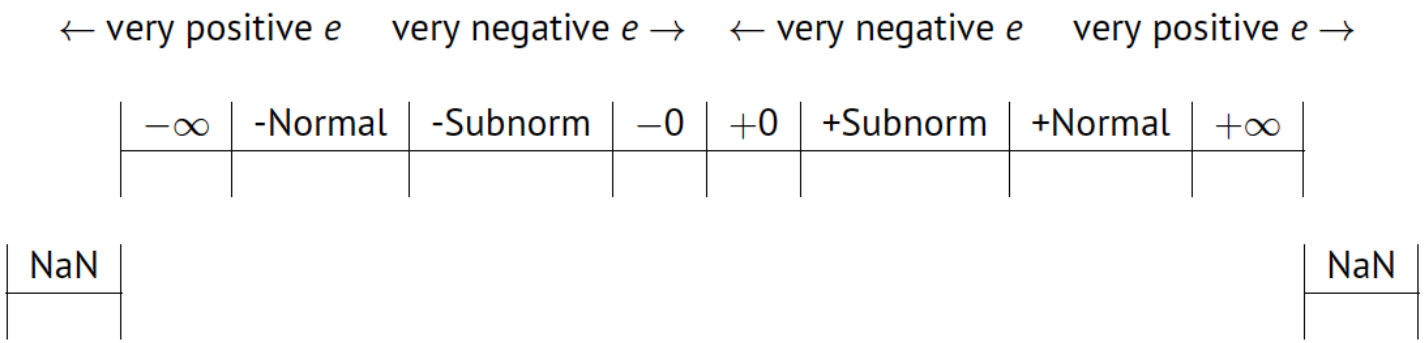
Floating-point representation - NaN



- Not a Number (NaN)
  - E = <111 … 1>
  - T ≠ <000… 0>
  - NaN is for the cases where no numerical value is defined, e.g., 0*∞, $\sqrt{-1}$
  - NaN is not greater than or lower than any number. Sign bit is irrelevant.

$(-1)^s \cdot m \cdot 2^e$

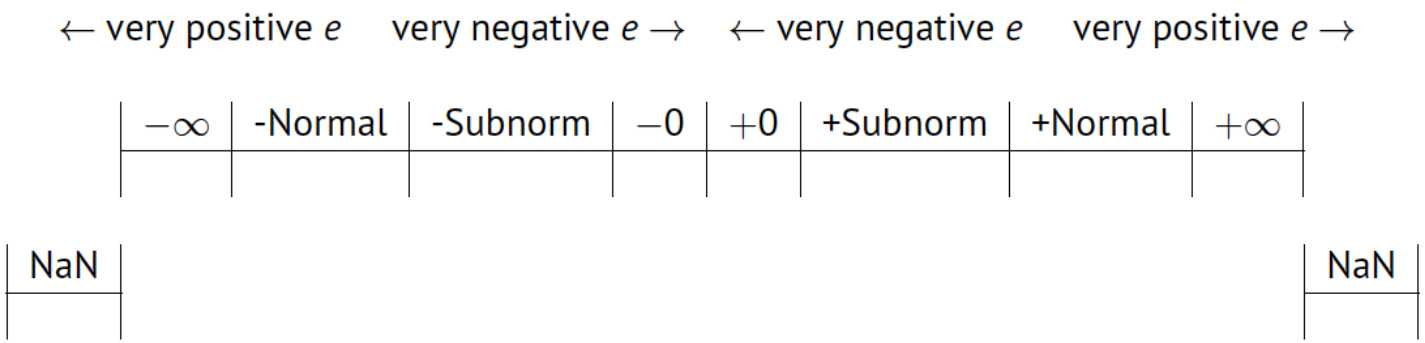
Floating-point representation - Infinity



- Infinity (+∞, -∞)
  - E = <111 … 1>
  - T = <000… 0>
  - $\frac{1}{0} = \infty$; $\frac{-1}{0} = -\infty$

$(-1)^s \cdot m \cdot 2^e$

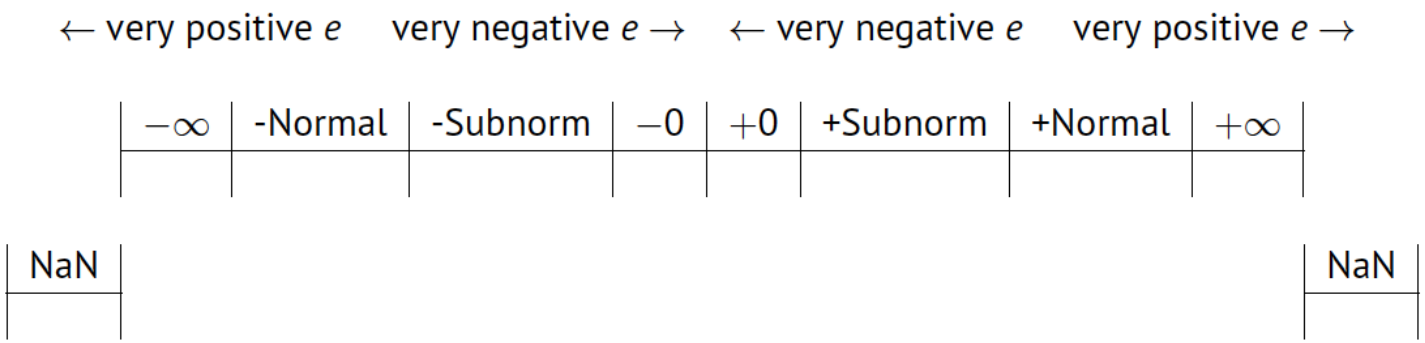
Floating-point representation - Zero



- Zero
  - E = < 000… 0 >
  - T = <000… 0>
  - Note that -0 and +0 have distinct encodings!
    At bit-level, -0 ≠ 0 but in C: 0 == -0

$(-1)^s \cdot m \cdot 2^e$

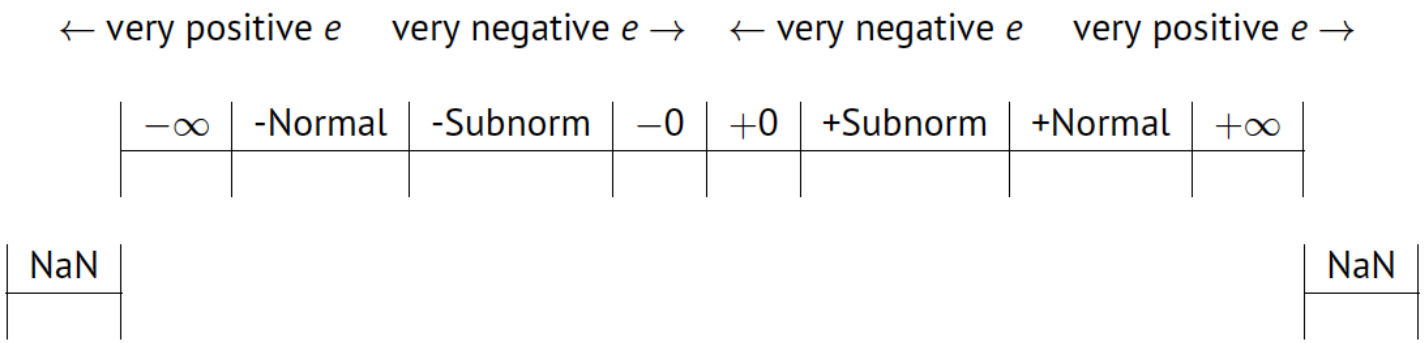
Floating-point representation – Normal values



- E encoded as biased value: *e = B2U(E) − b*
  - *See last week's slides*
  - *Single precision: b = 127 => E* $\in [-126; 127]$
  - *Double precision: b = 1023 => E* $\in [-1022; 1023]$
- *T encoded with extra leading 1: m = 1 + B2U(T) . 2^{1-p}*
  - *p = 1 + sizeof(T) - single precision p = 1 + 23 = 24; p = 1 + 52 = 53*
  - *T = <000…0> => m = 1; T = <111…1> => m = 2-ε*

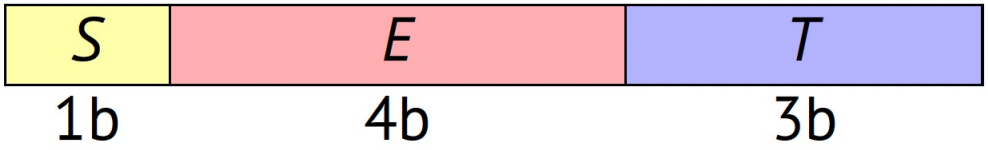
$(-1)^s \cdot m \cdot 2^e$

Floating-point representation – Subnorm



- E = <000..0>: e = 1 – b
- *T encoded with extra leading 0: m = B2U(T) . 2^{1-p}*
  - *T = <000..0> => 0*
  - *T ≠<000..0> => Numbers close to 0*

FP8 E4M3



8-bit floating-point representation

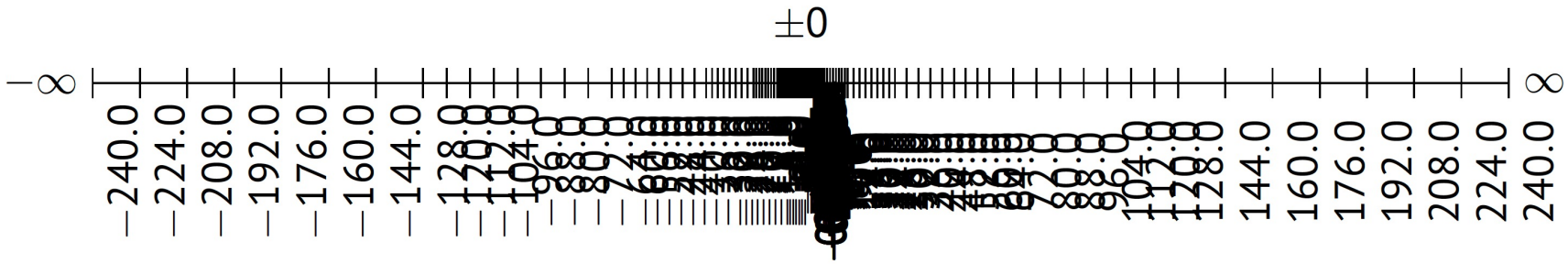
- Sign bit is the most significant bit (leftmost).
- The next four bits are *E* with a bias of 7.
- The last three bits are *T*.

Alternative: E5M2 – E encoded on 5 bits, T on 2 bits.

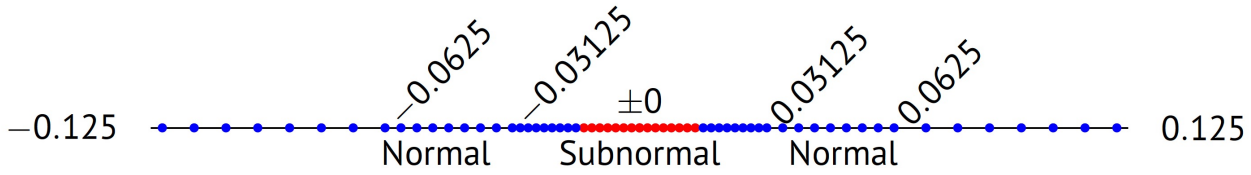
FP8 values

	S	E	T	e	Value	
Denormalised	0	0000	000	−6	0	
	0	0000	001	−6	$1/8 \cdot 1/64 = 1/512$	closest to zero
	0	0000	010	−6	$2/8 \cdot 1/64 = 2/512$	
	...					
	0	0000	111	−6	$7/8 \cdot 1/64 = 7/512$	largest denorm
Normalised	0	0001	000	−6	$8/8 \cdot 1/64 = 8/512$	smallest norm
	0	0001	001	−6	$9/8 \cdot 1/64 = 9/512$	
	...					
	0	0110	110	−1	$14/8 \cdot 1/2 = 14/16$	
	0	0110	111	−1	$15/8 \cdot 1/2 = 15/16$	Closest to 1
	0	0111	000	0	$8/8 \cdot 1 = 1$	
	0	0111	001	0	$9/8 \cdot 9/8 = 1$	Closest to 1
	0	0111	010	0	$10/8 \cdot 10/8 = 1$	
	...					
	0	1110	110	7	$14/8 \cdot 128 = 224$	
	0	1110	111	7	$15/8 \cdot 128 = 240$	
	0	1111	000	N/A	∞	

FP8 – non-uniform distribution of values



Example:
150 represented as 144
151 represented as 144
152 represented as 144
153 represented as 160



Take-aways

- The fixed representation of decimal number leads to numerical errors
- Floating-points are implemented in most languages
- Floating-points can represent infinity and NaN
- Floating points have a dynamic range
- Small floating-point types used for machine learning
  - Compact representation vs. loss of precision