

Floating point representation

(Unsigned) Fixed-point representation

The numbers are stored with a fixed number of bits for the integer part and a fixed number of bits for the fractional part.

Suppose we have 8 bits to store a real number, where 5 bits store the integer part and 3 bits store the fractional part:

$$\left(\underset{2^4}{1} \underset{2^3}{0} \underset{2^2}{1} \underset{2^1}{1} \underset{2^0}{1} . \underset{2^{-1}}{0} \underset{2^{-2}}{1} \underset{2^{-3}}{1} \right)_2$$

Smallest number: $(00000.001)_2 = 0.125$

Largest number: $(11111.111)_2 = 31.875$

(Unsigned) Fixed-point representation

Suppose we have 64 bits to store a real number, where 32 bits store the integer part and 32 bits store the fractional part:

$$(a_{31} \dots a_2 a_1 a_0 . b_1 b_2 b_3 \dots b_{32})_2 = \sum_{k=0}^{31} a_k 2^k + \sum_{k=1}^{32} b_k 2^{-k}$$
$$= a_{31} \times 2^{31} + a_{30} \times 2^{30} + \dots + a_0 \times 2^0 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \dots + b_{32} \times 2^{-32}$$

Smallest number:

$$a_i = 0 \quad \forall i \text{ and } b_1, b_2, \dots, b_{31} = 0 \text{ and } b_{32} = 1 \rightarrow 2^{-32} \approx 10^{-10}$$

Largest number:

$$a_i = 1 \quad \forall i \text{ and } b_i = 1 \quad \forall i \rightarrow 2^{31} + \dots + 2^0 + 2^{-1} + \dots + 2^{-32} \approx 10^9$$

(Unsigned) Fixed-point representation

Suppose we have 64 bits to store a real number, where 32 bits store the integer part and 32 bits store the fractional part:

$$(a_{31} \dots a_2 a_1 a_0 . b_1 b_2 b_3 \dots b_{32})_2 = \sum_{k=0}^{31} a_k 2^k + \sum_{k=1}^{32} b_k 2^{-k}$$

Smallest number $\rightarrow \approx 10^{-10}$

Largest number $\rightarrow \approx 10^9$

0

∞



(Unsigned) Fixed-point representation

Range: difference between the largest and smallest numbers possible.

More bits for the integer part \rightarrow increase range

Precision: smallest possible difference between any two numbers

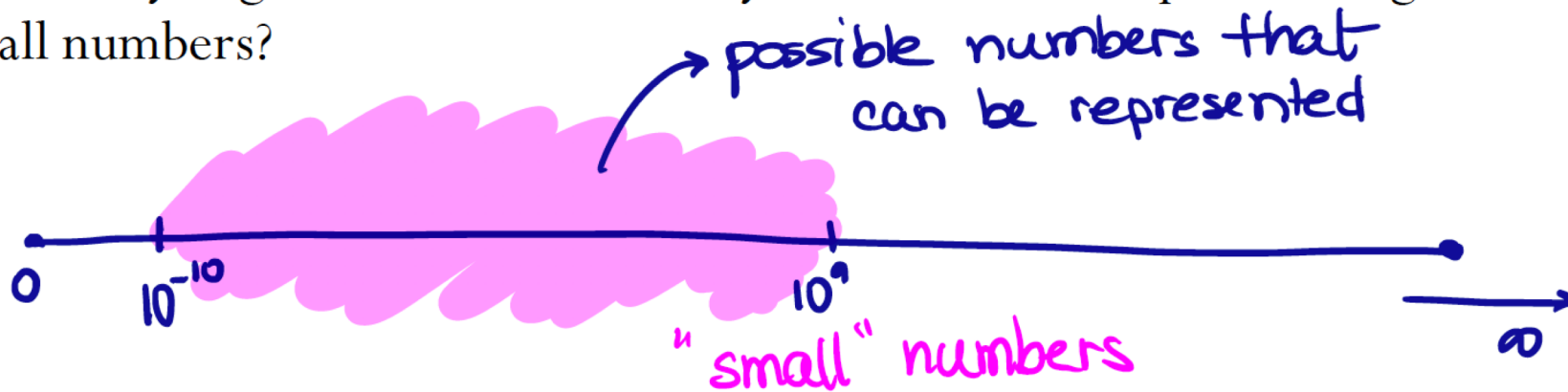
More bits for the fractional part \rightarrow increase precision

$$(a_2 a_1 a_0 . b_1 b_2 b_3)_2 \quad \text{OR} \quad (a_1 a_0 . b_1 b_2 b_3 b_4)_2$$

Wherever we put the binary point, there is a trade-off between the amount of range and precision. **It can be hard to decide how much you need of each!**

Fix: Let the binary point “float”

How many 'digits' of relative accuracy are available to represent large and small numbers?



$$(a_{31} a_{30} \dots a_{14} a_{13} a_{12} a_{11} \dots a_2 a_1 a_0 \dots)_2$$

about 19 digits

12 bits $\rightarrow (4095)_{10}$

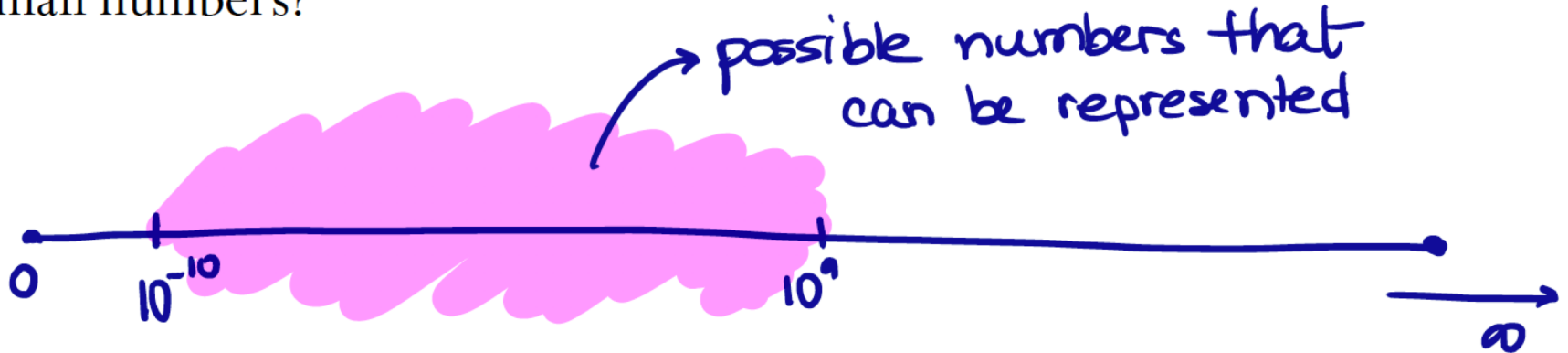
13 bits $\rightarrow (8191)_{10}$

14 bits $\rightarrow (16383)_{10}$

"large" numbers $\geq 10^4$

when the digits corresponding to a_0 all the way to 12, 13 and 14 bits are set to 1.

How many 'digits' of relative accuracy are available to represent large and small numbers?



($\dots 00000\dots 010\dots 0$)₂

$$2^{-10} \rightarrow \sim 10^{-3}$$

$$\vdots$$
$$2^{-20} \rightarrow \sim 10^{-6}$$

$$2^{-25} \rightarrow \sim 10^{-8}$$

$$2^{-32} \rightarrow \sim 10^{-10}$$

} very few accurate digits to represent "small" numbers (~8?)

Floating-point numbers

A floating-point number can represent numbers of different order of magnitude (very large and very small) with the same number of fixed digits.

In general, in the binary system, a floating number can be expressed as

$$x = \pm q \times 2^m$$

q is the significand, normally a fractional value in the range [1.0,2.0)

m is the exponent

Floating-point numbers

Numerical Form:

$$x = \pm q \times 2^m = \pm b_0 \cdot \underbrace{b_1 b_2 b_3 \dots b_n}_{\text{Fractional part of significand (n digits)}} \times 2^m$$

Fractional part of significand
(n digits)

$$b_i \in \{0,1\}$$

$$\text{Exponent range: } m \in [L, U]$$

$$\text{Precision: } p = n + 1$$

“Floating” the binary point

$$(1011.1)_2 = 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 + 1 \times \frac{1}{2} = (11.5)_{10}$$

$$\begin{aligned}(10111)_2 &= 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = (23)_{10} \\ &= (1011.1)_2 \times 2^1 = (23)_{10}\end{aligned}$$

$$\begin{aligned}(101.11)_2 &= 1 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times \frac{1}{2} + 1 \times \frac{1}{4} = (5.75)_{10} \\ &= (1011.1)_2 \times 2^{-1} = (5.75)_{10}\end{aligned}$$

Move “binary point” to the left by one bit position: Divide the decimal number by 2

Move “binary point” to the right by one bit position: Multiply the decimal number by 2

Converting floating points

Convert $(39.6875)_{10} = (100111.1011)_2$ into floating point representation

$$(39.6875)_{10} = (100111.1011)_2 = (1.001111011)_2 \times 2^5$$

Normalized floating-point numbers

Normalized floating point numbers are expressed as

$$x = \pm 1.b_1b_2b_3 \dots b_n \times 2^m = \pm 1.f \times 2^m$$

where f is the fractional part of the significand, m is the exponent and $b_i \in \{0,1\}$.

Hidden bit representation:

The first bit to the left of the binary point $b_0 = 1$ does not need to be stored, since its value is fixed.

This representation "adds" 1-bit of precision (we will show some exceptions later, including the representation of number **zero**).

Clicker question

Determine the normalized floating point representation

1. $f \times 2^m$ of the decimal number $x = 47.125$ (f in binary representation and m in decimal)

A) $(1.01110001)_2 \times 2^5$

B) $(1.01110001)_2 \times 2^4$

C) $(1.01111001)_2 \times 2^5$

D) $(1.01111001)_2 \times 2^4$

Normalized floating-point numbers

$$x = \pm q \times 2^m = \pm 1.b_1b_2b_3 \dots b_n \times 2^m = \pm 1.f \times 2^m$$

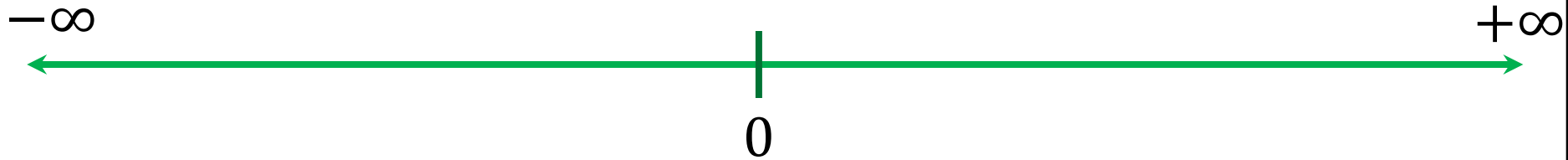
- **Exponent range:** $[L, U]$
- **Precision:** $p = n + 1$
- **Smallest positive normalized FP number:**

$$\text{UFL} = 2^L$$

- **Largest positive normalized FP number:**

$$\text{OFL} = 2^{U+1}(1 - 2^{-p})$$

Normalized floating point number scale



Floating-point numbers: Simple example

A "toy" number system can be represented as $x = \pm 1.b_1b_2 \times 2^m$
for $m \in [-4,4]$ and $b_i \in \{0,1\}$.

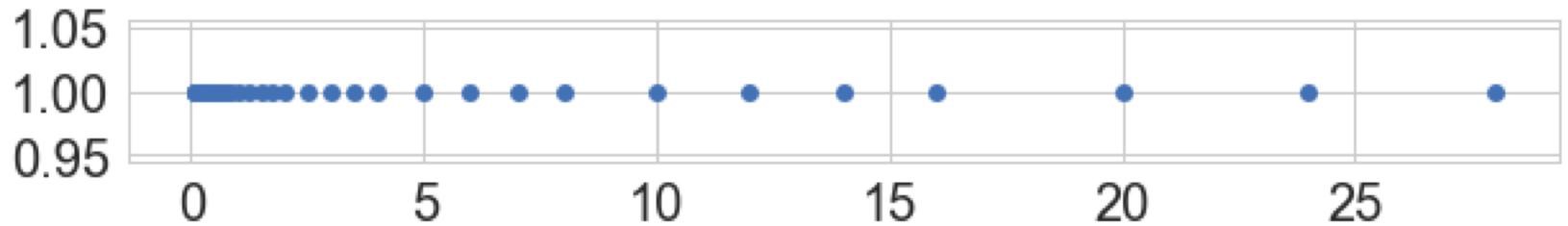
$(1.00)_2 \times 2^0 = 1$	$(1.00)_2 \times 2^1 = 2$	$(1.00)_2 \times 2^2 = 4.0$
$(1.01)_2 \times 2^0 = 1.25$	$(1.01)_2 \times 2^1 = 2.5$	$(1.01)_2 \times 2^2 = 5.0$
$(1.10)_2 \times 2^0 = 1.5$	$(1.10)_2 \times 2^1 = 3.0$	$(1.10)_2 \times 2^2 = 6.0$
$(1.11)_2 \times 2^0 = 1.75$	$(1.11)_2 \times 2^1 = 3.5$	$(1.11)_2 \times 2^2 = 7.0$

$(1.00)_2 \times 2^3 = 8.0$	$(1.00)_2 \times 2^4 = 16.0$	$(1.00)_2 \times 2^{-1} = 0.5$
$(1.01)_2 \times 2^3 = 10.0$	$(1.01)_2 \times 2^4 = 20.0$	$(1.01)_2 \times 2^{-1} = 0.625$
$(1.10)_2 \times 2^3 = 12.0$	$(1.10)_2 \times 2^4 = 24.0$	$(1.10)_2 \times 2^{-1} = 0.75$
$(1.11)_2 \times 2^3 = 14.0$	$(1.11)_2 \times 2^4 = 28.0$	$(1.11)_2 \times 2^{-1} = 0.875$

$(1.00)_2 \times 2^{-2} = 0.25$	$(1.00)_2 \times 2^{-3} = 0.125$	$(1.00)_2 \times 2^{-4} = 0.0625$
$(1.01)_2 \times 2^{-2} = 0.3125$	$(1.01)_2 \times 2^{-3} = 0.15625$	$(1.01)_2 \times 2^{-4} = 0.078125$
$(1.10)_2 \times 2^{-2} = 0.375$	$(1.10)_2 \times 2^{-3} = 0.1875$	$(1.10)_2 \times 2^{-4} = 0.09375$
$(1.11)_2 \times 2^{-2} = 0.4375$	$(1.11)_2 \times 2^{-3} = 0.21875$	$(1.11)_2 \times 2^{-4} = 0.109375$

Same steps are performed to obtain the negative numbers. For simplicity, we will show only the positive numbers in this example.

$$x = \pm 1.b_1b_2 \times 2^m \text{ for } m \in [-4,4] \text{ and } b_i \in \{0,1\}$$



- Smallest normalized positive number:

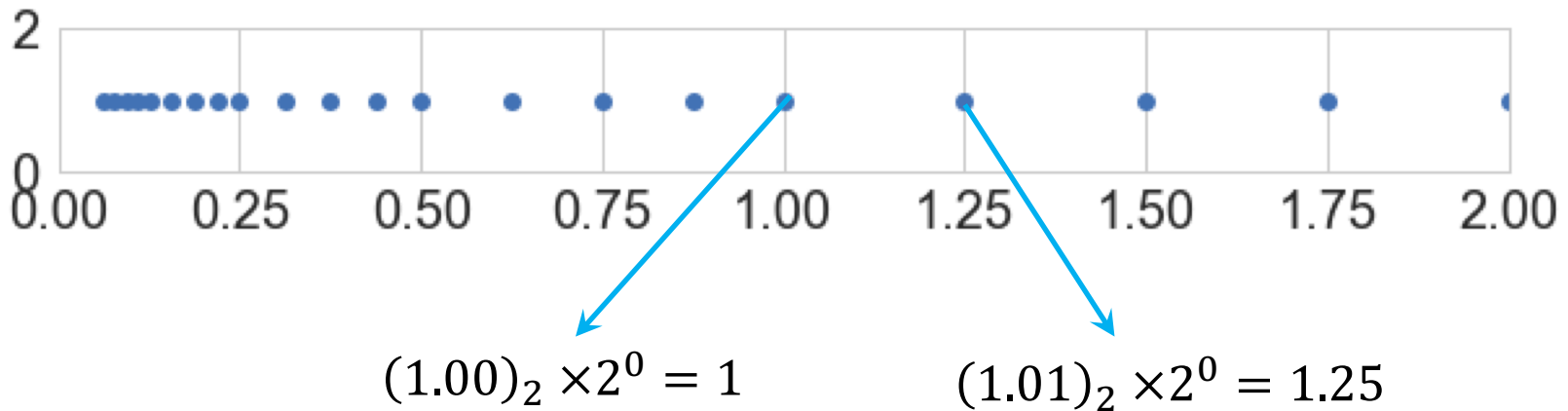
$$(1.00)_2 \times 2^{-4} = 0.0625$$
- Largest normalized positive number:

$$(1.11)_2 \times 2^4 = 28.0$$
- Any number x closer to zero than **0.0625** would UNDERFLOW to zero.
- Any number x outside the range **-28.0** and **+28.0** would OVERFLOW to infinity.

Machine epsilon

- **Machine epsilon** (ϵ_m): is defined as the distance (gap) between 1 and the next largest floating point number.

$$x = \pm 1.b_1b_2 \times 2^m \text{ for } m \in [-4,4] \text{ and } b_i \in \{0,1\}$$



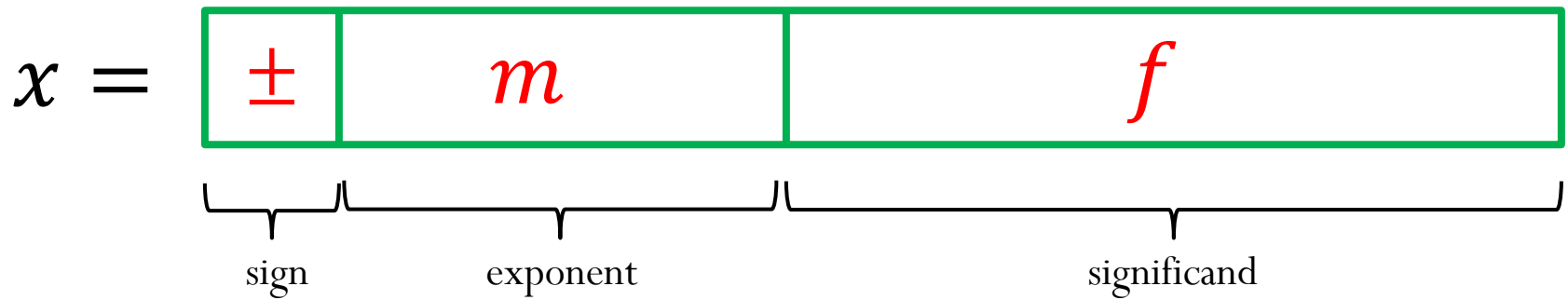
$$\epsilon_m = (0.01)_2 \times 2^0 = \mathbf{0.25}$$

Machine numbers: how floating point numbers are stored?

Floating-point number representation

What do we need to store when representing floating point numbers in a computer?

$$x = \pm 1.f \times 2^m$$



Initially, different floating-point representations were used in computers, generating inconsistent program behavior across different machines.

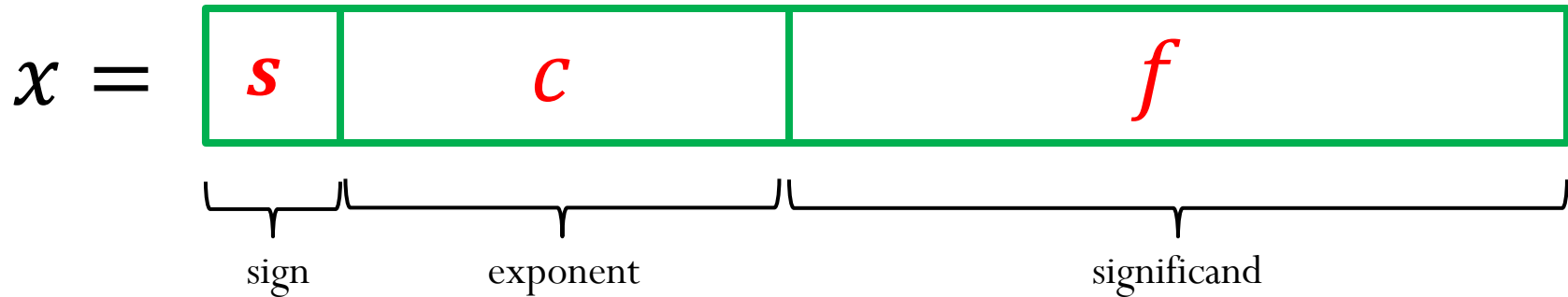
Around 1980s, computer manufacturers started adopting a standard representation for floating-point number: IEEE (Institute of Electrical and Electronics Engineers) 754 Standard.

Floating-point number representation

Numerical form:

$$x = \pm 1.f \times 2^m$$

Representation in memory:

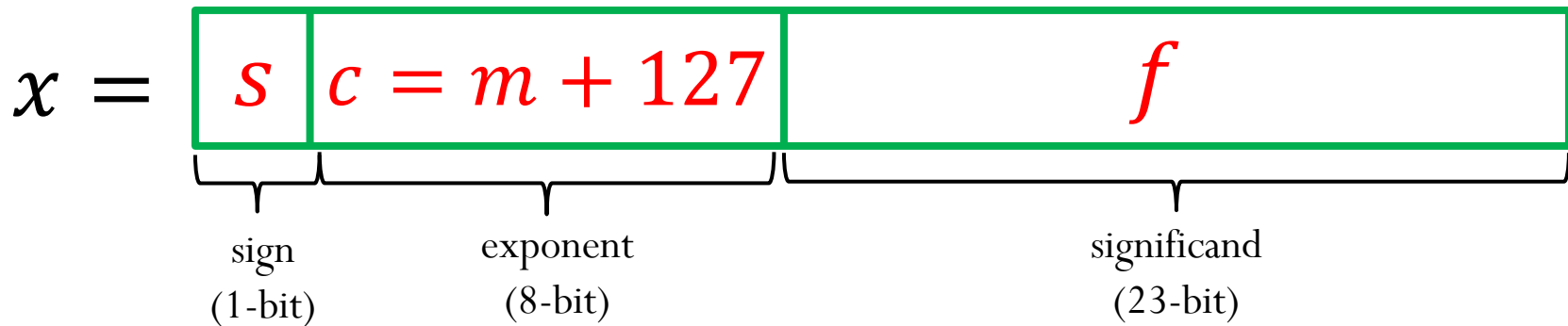


$$x = (-1)^s 1.f \times 2^{c - \text{shift}} \quad m = c - \text{shift}$$

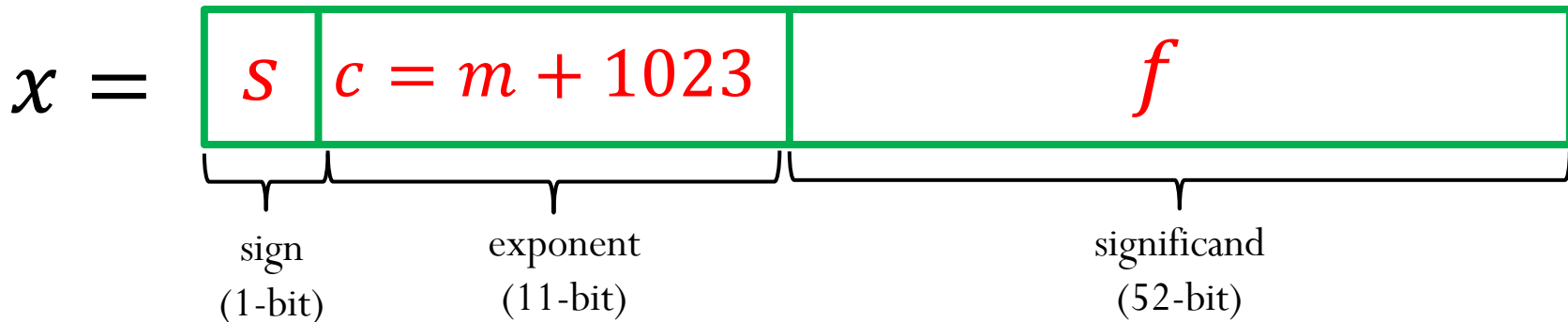
Precisions:

Finite representation: not all numbers can be represented exactly!

IEEE-754 Single precision (32 bits):



IEEE-754 Double precision (64 bits):



Special Values:

$$x = (-1)^s 1.f \times 2^m = \boxed{\begin{array}{|c|c|c|} \hline s & c & f \\ \hline \end{array}}$$

1) Zero:

$$x = \boxed{\begin{array}{|c|c|c|} \hline s & 000 \dots 000 & 0000 \dots 0000 \\ \hline \end{array}}$$

2) Infinity: $+\infty$ ($s = 0$) and $-\infty$ ($s = 1$)

$$x = \boxed{\begin{array}{|c|c|c|} \hline s & 111 \dots 111 & 0000 \dots 0000 \\ \hline \end{array}}$$

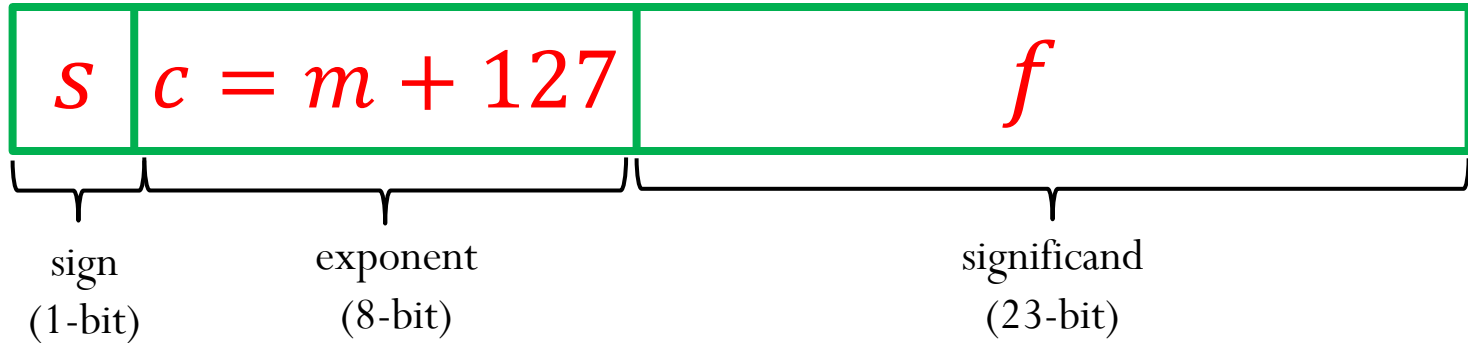
3) NaN: (results from operations with undefined results)

$$x = \boxed{\begin{array}{|c|c|c|} \hline s & 111 \dots 111 & \textit{anything} \neq 00 \dots 00 \\ \hline \end{array}}$$

Note that the exponent $c = (000 \dots 000)$ and $c = (111 \dots 111)$ are reserved for these special cases, which limits the exponent range for the other numbers.

IEEE-754 Single Precision (32-bit)

$$x = (-1)^s 1.f \times 2^m$$



$s = 0$: positive sign, $s = 1$: negative sign

Reserved exponent number for special cases:

$$c = (11111111)_2 = 255 \quad \text{and} \quad c = (00000000)_2 = 0$$

Therefore $0 < c < 255$

The largest exponent is $U = 254 - 127 = 127$

The smallest exponent is $L = 1 - 127 = -126$

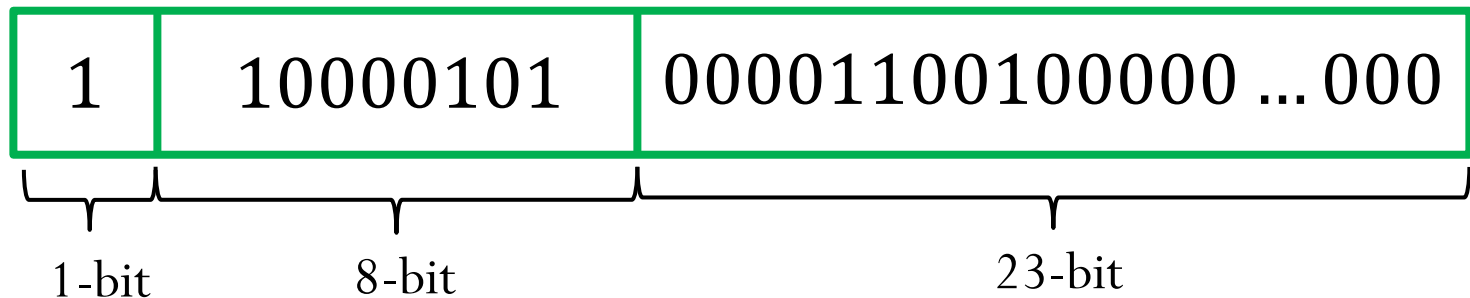
IEEE-754 Single Precision (32-bit)

$$x = (-1)^s 1.f \times 2^m$$

Example: Represent the number $x = -67.125$ using IEEE Single-Precision Standard

$$67.125 = (1000011.001)_2 = (1.000011001)_2 \times 2^6$$

$$c = 6 + 127 = 133 = (10000101)_2$$



IEEE-754 Single Precision (32-bit)

$$x = (-1)^s 1.f \times 2^m = \boxed{s \quad c \quad f} \quad c = m + 127$$

- **Machine epsilon** (ϵ_m): is defined as the distance (gap) between 1 and the next largest floating point number.

$$(1)_{10} = \boxed{0 \quad 01111111 \quad 000000000000000000000000}$$

$$(1)_{10} + \epsilon_m = \boxed{0 \quad 01111111 \quad 000000000000000000000001}$$

$$\epsilon_m = 2^{-23} \approx 1.2 \times 10^{-7}$$

- **Smallest positive normalized FP number:**

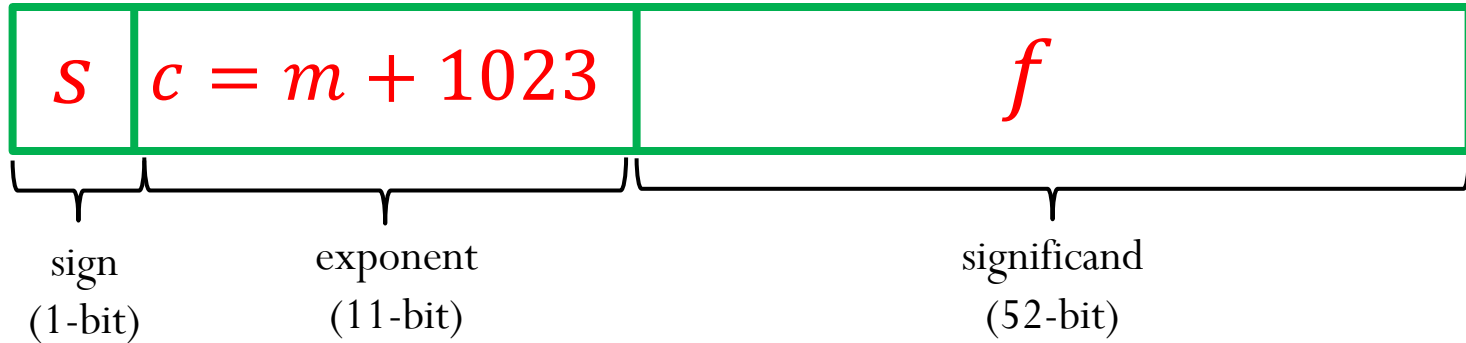
$$\text{UFL} = 2^L = 2^{-126} \approx 1.2 \times 10^{-38}$$

- **Largest positive normalized FP number:**

$$\text{OFL} = 2^{U+1}(1 - 2^{-p}) = 2^{128}(1 - 2^{-24}) \approx 3.4 \times 10^{38}$$

IEEE-754 Double Precision (64-bit)

$$x = (-1)^s 1.f \times 2^m$$



$s = 0$: positive sign, $s = 1$: negative sign

Reserved exponent number for special cases:

$$c = (11111111111)_2 = 2047 \text{ and } c = (00000000000)_2 = 0$$

Therefore $0 < c < 2047$

The largest exponent is $U = 2046 - 1023 = 1023$

The smallest exponent is $L = 1 - 1023 = -1022$

IEEE-754 Double Precision (64-bit)

$$x = (-1)^s 1.f \times 2^m = \boxed{s \quad c \quad f} \quad c = m + 1023$$

- **Machine epsilon** (ϵ_m): is defined as the distance (gap) between 1 and the next largest floating point number.

$$(1)_{10} = \boxed{0 \quad 0111 \dots 111 \quad 000000000000 \dots 0000000000}$$

$$(1)_{10} + \epsilon_m = \boxed{0 \quad 0111 \dots 111 \quad 000000000000 \dots 0000000001}$$

$$\epsilon_m = 2^{-52} \approx 2.2 \times 10^{-16}$$

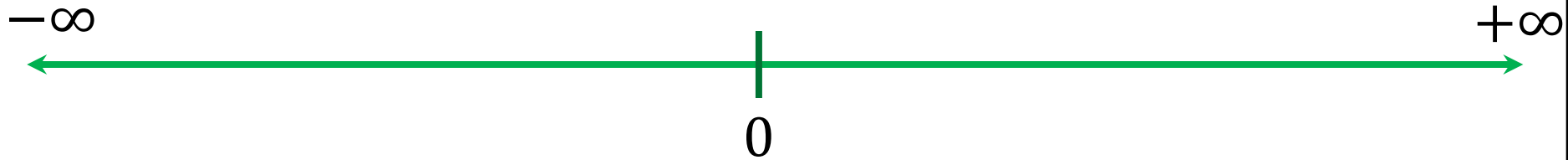
- **Smallest positive normalized FP number:**

$$\text{UFL} = 2^L = 2^{-1022} \approx 2.2 \times 10^{-308}$$

- **Largest positive normalized FP number:**

$$\text{OFL} = 2^{U+1}(1 - 2^{-p}) = 2^{1024}(1 - 2^{-53}) \approx 1.8 \times 10^{308}$$

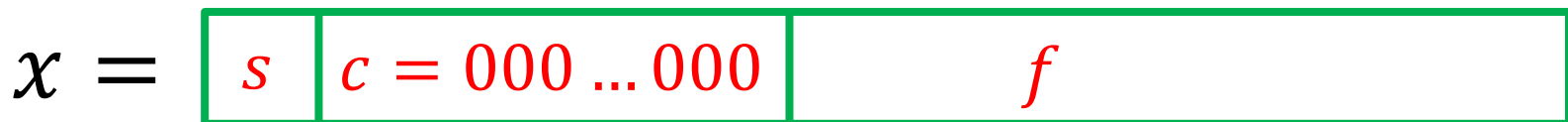
Normalized floating point number scale (double precision)



Subnormal (or denormalized) numbers

- Noticeable gap around zero, present in any floating system, due to normalization
 - ✓ The smallest possible significand is 1.00
 - ✓ The smallest possible exponent is L
- Relax the requirement of normalization, and allow the leading digit to be zero, only when the exponent is at its minimum ($m = L$)
- Computations with subnormal numbers are often slow.

Representation in memory (another special case):



Numerical value:

$$x = (-1)^s 0.f \times 2^L$$

Note that this is a special case, and the exponent m is **not** evaluated as $m = c - \text{shift} = -\text{shift}$. Instead, the exponent is set to the lower bound, $m = L$

Subnormal (or denormalized) numbers

IEEE-754 Single precision (32 bits):

$$c = (00000000)_2 = 0$$

Exponent set to $m = -126$

Smallest positive subnormal FP number: $2^{-23} \times 2^{-126} \approx 1.4 \times 10^{-45}$

IEEE-754 Double precision (64 bits):

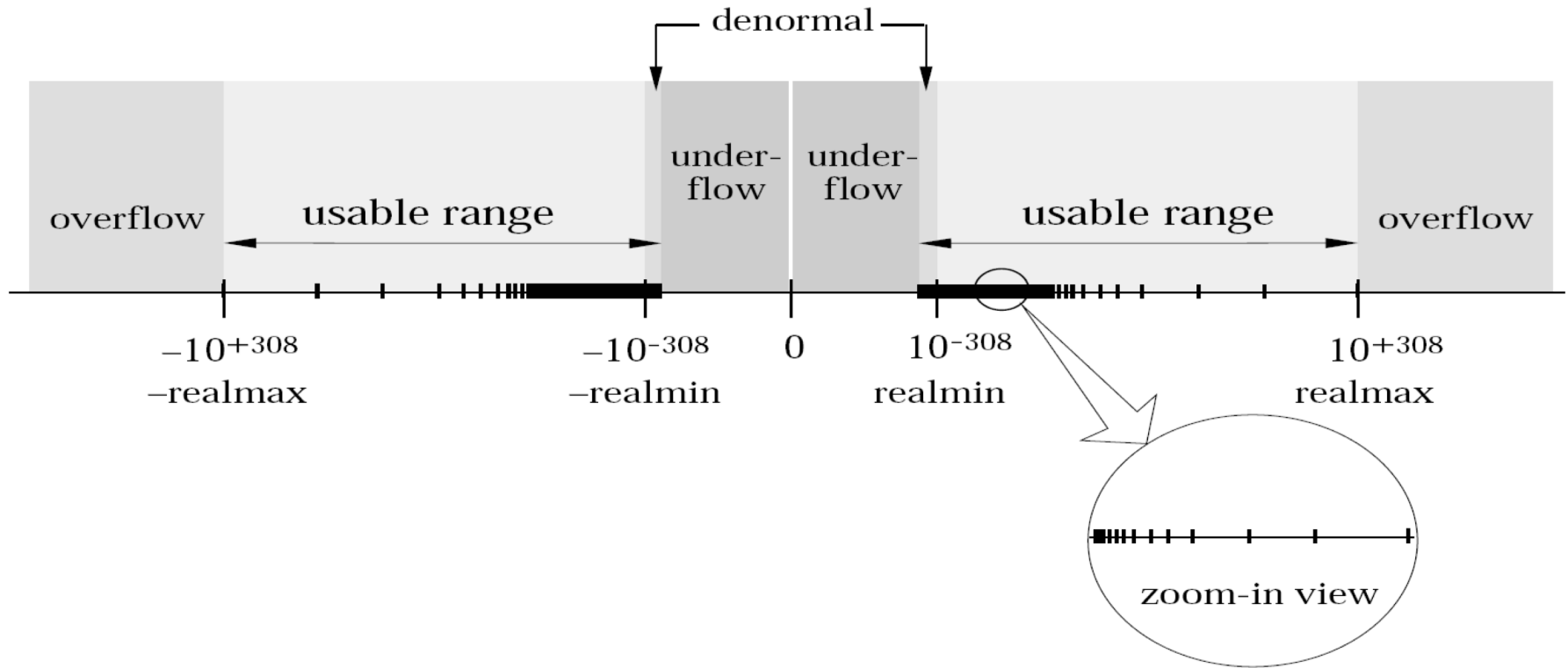
$$c = (000000000000)_2 = 0$$

Exponent set to $m = -1022$

Smallest positive subnormal FP number: $2^{-52} \times 2^{-1022} \approx 4.9 \times 10^{-324}$

Allows for more gradual underflow to zero (however subnormal numbers don't have as many accurate digits as normalized numbers)

IEEE-754 Double Precision



Summary for Single Precision

$$x = (-1)^s 1.f \times 2^m = \boxed{s \quad c \quad f} \quad m = c - 127$$

Stored binary exponent (c)	Significand fraction (f)	value
00000000	0000...0000	zero
00000000	<i>any</i> $f \neq 0$	$(-1)^s 0.f \times 2^{-126}$
00000001	<i>any</i> f	$(-1)^s 1.f \times 2^{-126}$
⋮	⋮	⋮
11111110	<i>any</i> f	$(-1)^s 1.f \times 2^{127}$
11111111	<i>any</i> $f \neq 0$	NaN
11111111	0000...0000	infinity

Example

Determine the single-precision representation of the decimal number $x = 37.625$

- Convert the decimal number to binary: $(37.625)_{10} = (100101.101)_2$

	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
	32	16	8	4	2	1	0.5	0.25	0.125
#	1	0	0	1	0	1	1	0	1
37.625	5.625	5.625	5.625	1.625	1.625	0.625	0.125	0.125	0

- Convert the binary number to the normalized FP representation $1.f \times 2^m$

$$(100101.101)_2 = (1.00101101)_2 \times 2^5$$

$$s = 0 \quad f = 00101101 \dots 00 \quad m = 5$$

$$c = m + 127 = 132 = (10000100)_2$$

0 10000100 001011010000000000000000

What is the equivalent decimal number?

0 00000000 000000000000000000000000

1 11111111 000000000000000000000000

0 11111111 1111111111000011111111

0 00000000 111100000000000000000000

0 01111111 000000000000000000000000

Clicker question

A number system can be represented as $x = \pm 1.b_1b_2b_3 \times 2^m$
for $m \in [-5,5]$ and $b_i \in \{0,1\}$.

- 1) **What is the smallest positive normalized FP number:**
a) 0.0625 b) 0.09375 c) 0.03125 d) 0.046875 e) 0.125
- 2) **What is the largest positive normalized FP number:**
a) 28 b) 60 c) 56 d) 32
- 3) **How many additional numbers (positive and negative) can be represented when using subnormal representation?**
a) 7 b) 14 c) 3 d) 6 e) 16
- 4) **What is the smallest positive subnormal number?**
a) 0.00390625 b) 0.00195313 c) 0.03125 d) 0.0136719
- 5) **Determine machine epsilon**
a) 0.0625 b) 0.00390625 c) 0.0117188 d) 0.125

Clicker question


Determine the double-precision machine representation of the decimal number $x = -37.625$

A) 1 10000100000 00101101000000 ... 0

B) 1 10000000100 00101101000000 ... 0

C) 0 10000100000 00101101000000 ... 0

D) 0 10000000100 00101101000000 ... 0



(52-bit)