

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:

$$(1\ 0\ 1\ 1\ 1.0\ 1\ 1)_2$$

$2^4\ 2^3\ 2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}$

Smallest number:

Largest number:

(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:

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

$$= a_{31} \times 2^{31} + a_{30} \times 2^{30} + \dots + a_0 \times 2^0 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \dots + b_{32} \times 2^{-32}$$



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!**

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 bits.

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:

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

Fractional part of significand
(n bits)

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

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

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

“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

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

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

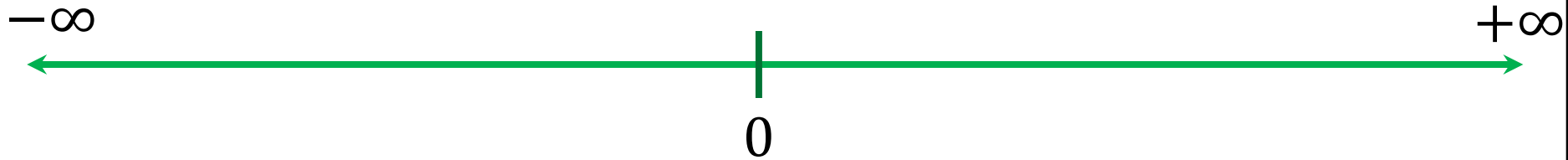
D) $(1.01111001)_2 \times 2^4$

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:**
- **Precision:**
- **Smallest positive normalized FP number:**
- **Largest positive normalized FP number:**

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

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

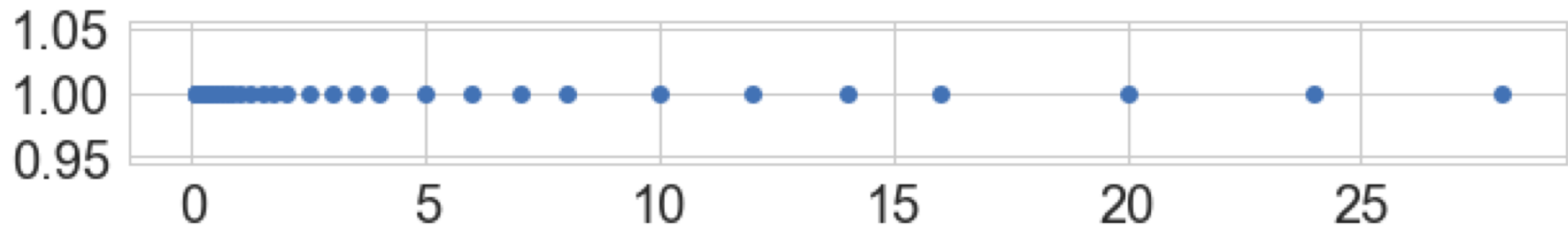
$(1.00)_2 \times 2^0 = 1$	$(1.00)_2 \times 2^1 = 2$	$(1.00)_2 \times 2^2 = 4.0$
$(1.01)_2 \times 2^0 = 1.25$	$(1.01)_2 \times 2^1 = 2.5$	$(1.01)_2 \times 2^2 = 5.0$
$(1.10)_2 \times 2^0 = 1.5$	$(1.10)_2 \times 2^1 = 3.0$	$(1.10)_2 \times 2^2 = 6.0$
$(1.11)_2 \times 2^0 = 1.75$	$(1.11)_2 \times 2^1 =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

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

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

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

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

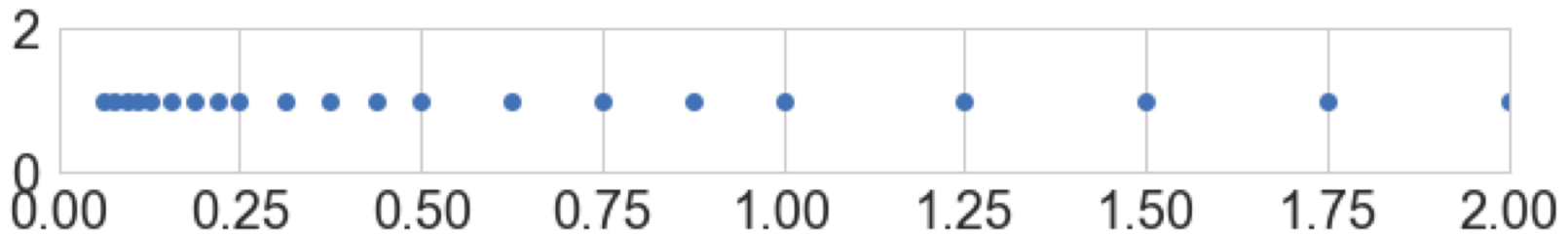


- Smallest normalized positive number:
- Largest normalized positive number:

Machine epsilon

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

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

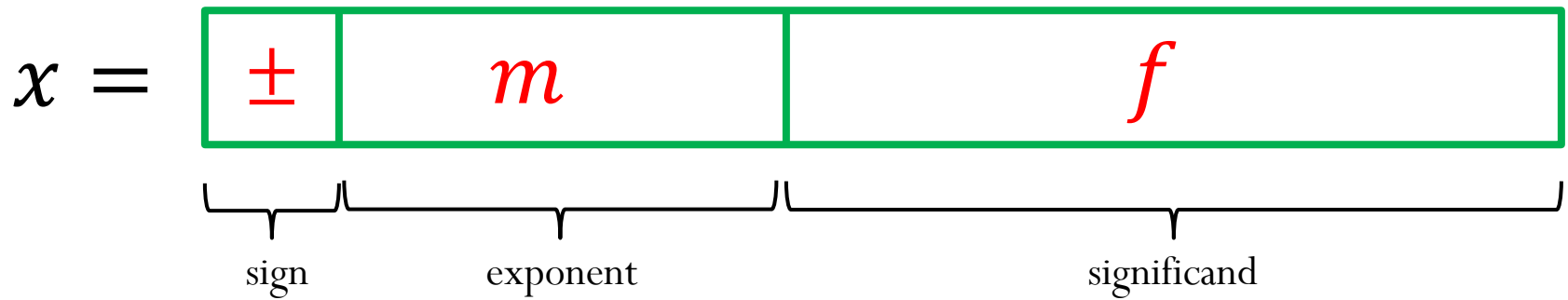


Machine numbers: how floating point numbers are stored?

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.

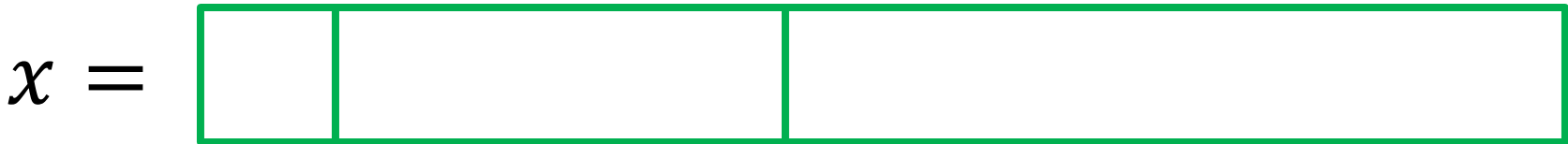
Around 1980s, computer manufacturers started adopting a standard representation for floating-point number: IEEE (Institute of Electrical and Electronics Engineers) 754 Standard.

Floating-point number representation

Numerical form:

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

Representation in memory:



Precisions:

IEEE-754 Single precision (32 bits):

$x =$



IEEE-754 Double precision (64 bits):

$x =$



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:

$$x = \boxed{\begin{array}{|c|c|c|} \hline & & \\ \hline \end{array}}$$

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

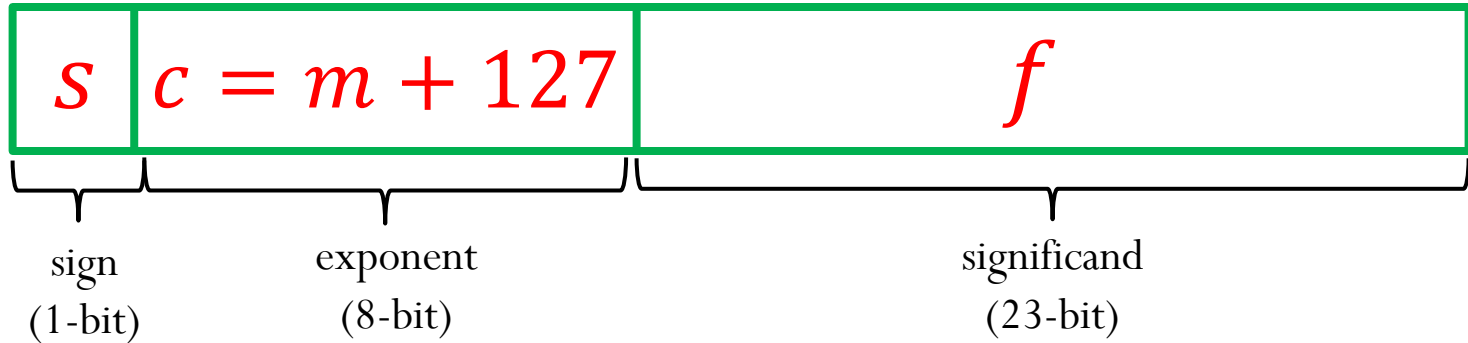
$$x = \boxed{\begin{array}{|c|c|c|} \hline & & \\ \hline \end{array}}$$

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

$$x = \boxed{\begin{array}{|c|c|c|} \hline & & \\ \hline \end{array}}$$

IEEE-754 Single Precision (32-bit)

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



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

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

IEEE-754 Single Precision (32-bit)

$$x = (-1)^s 1.f \times 2^m = \boxed{\begin{array}{|c|c|c|} \hline s & c & f \\ \hline \end{array}} \quad c = m + 127$$

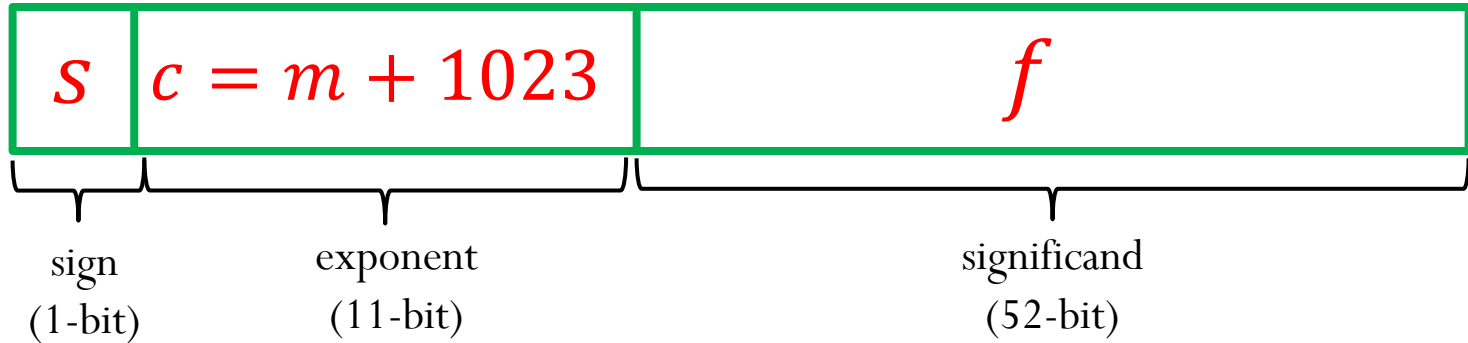
- **Machine epsilon** (ϵ_m): is defined as the distance (gap) between 1 and the next largest floating point number.

- **Smallest positive normalized FP number:**

- **Largest positive normalized FP number:**

IEEE-754 Double Precision (64-bit)

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



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.

- **Smallest positive normalized FP number:**

- **Largest positive normalized FP number:**

Subnormal (or denormalized) numbers

- Noticeable gap around zero, present in any floating system, due to normalization
- 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:

Subnormal (or denormalized) numbers

IEEE-754 Single precision (32 bits):

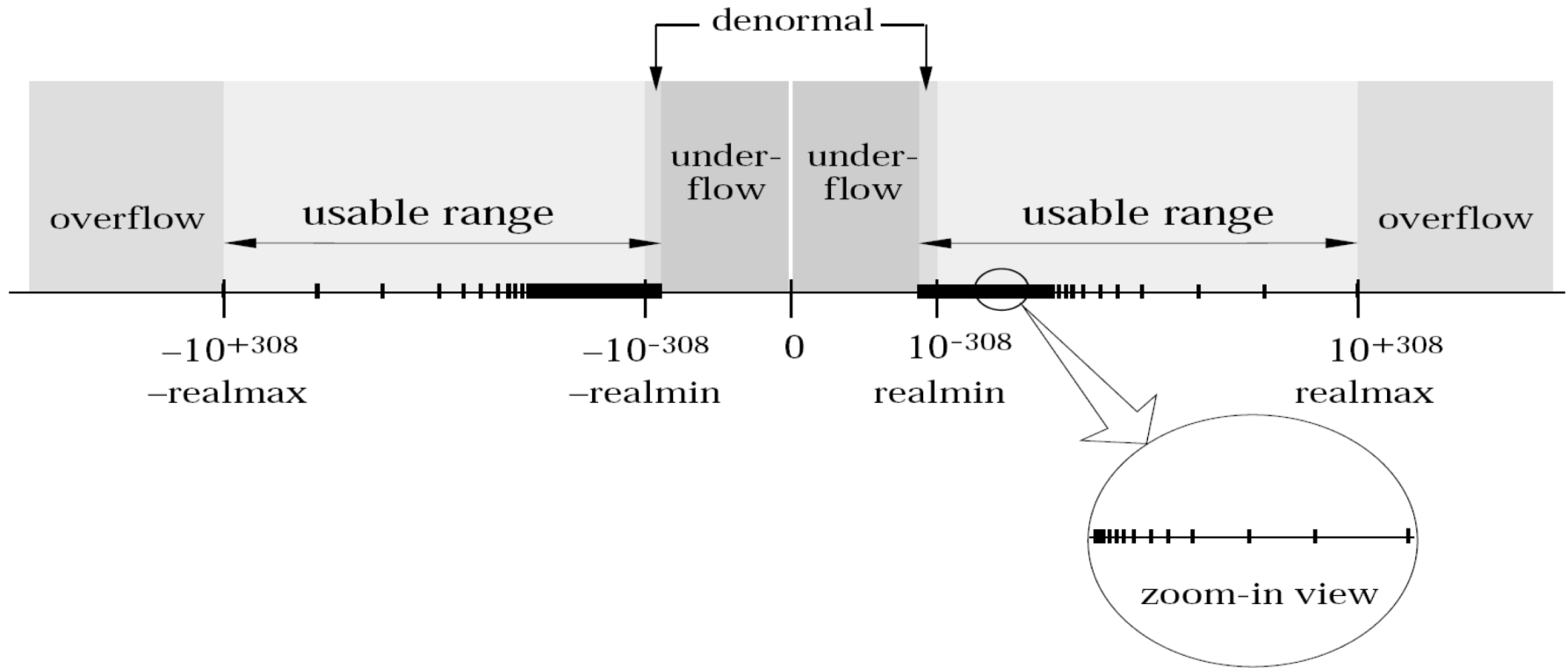
$$c = (00000000)_2 = 0$$

IEEE-754 Double precision (64 bits):

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

Subnormal (or denormalized) 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	$any\ f \neq 0$	$(-1)^s 0.f \times 2^{-126}$
00000001	$any\ f$	$(-1)^s 1.f \times 2^{-126}$
⋮	⋮	⋮
11111110	$any\ f$	$(-1)^s 1.f \times 2^{127}$
11111111	$any\ f \neq 0$	NaN
11111111	0000...0000	infinity

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

