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:

\[(1 0 1 1 1 . 0 1 1)_{2}\]

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:

\[
(a_{31} \ldots a_2 a_1 a_0 \cdot b_1 b_2 b_3 \ldots b_{32})_2 = \sum_{k=0}^{31} a_k 2^k + \sum_{k=1}^{32} b_k 2^{-k}
\]

\[
= a_{31} \times 2^{31} + a_{30} \times 2^{30} + \ldots + a_0 \times 2^0 + b_1 \times 2^{-1} + b_2 \times 2^2 + \ldots + b_{32} \times 2^{-32}
\]

**Smallest number:**
\[
a_i = 0 \ \forall i \text{ and } b_1, b_2, \ldots, b_{31} = 0 \text{ and } b_{32} = 1 \rightarrow 2^{-32} \approx 10^{-10}
\]

**Largest number:**
\[
a_i = 1 \ \forall i \text{ and } b_i = 1 \ \forall i \rightarrow 2^{31} + \ldots + 2^0 + 2^{-1} + \ldots + 2^{-32} \approx 10^9
\]
(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:

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

Smallest number \( \rightarrow \approx 10^{-10} \)
Largest number \( \rightarrow \approx 10^9 \)
(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?

possible numbers that can be represented

“small” numbers

\[(a_2, a_1, a_0, \ldots, a_{14}, a_{13})_2\]

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?

possible numbers that can be represented

\[
\left( \ldots 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \right)_2
\]

\[
\begin{align*}
2^{-10} & \rightarrow \sim 10^{-3} \\
2^{-20} & \rightarrow \sim 10^{-6} \\
2^{-25} & \rightarrow \sim 10^{-8} \\
2^{-32} & \rightarrow \sim 10^{-10}
\end{align*}
\]

\{ \text{very few accurate digits to represent small numbers (\sim 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:

\[ x = \pm q \times 2^m = \pm b_0.b_1b_2b_3 \ldots b_n \times 2^m \]

Fractional part of significand
(n digits)

\[ b_i \in \{0,1\} \]

Exponent range: \( m \in [L,U] \)

Precision: \( p = n + 1 \)
“Floating” the binary point

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

\[(10111)_2 = 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 = (23)_{10}\]
\[= (1011.1)_2 \times 2^1 = (23)_{10}\]

\[(101.11)_2 = 1 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times \frac{1}{2} + 1 \times \frac{1}{4} = (5.75)_{10}\]
\[= (1011.1)_2 \times 2^{-1} = (5.75)_{10}\]

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

\[ x = \pm 1. b_1 b_2 b_3 \ldots b_n \times 2^m = \pm 1. f \times 2^m \]

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**).
Iclicker 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

\[ x = \pm q \times 2^m = \pm 1. b_1 b_2 b_3 \ldots b_n \times 2^m = \pm 1. f \times 2^m \]

- 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
A "toy" number system can be represented as $x = \pm 1. b_1 b_2 \times 2^m$ for $m \in [-4,4]$ and $b_i \in \{0,1\}$.

<table>
<thead>
<tr>
<th>$b_1 b_2$</th>
<th>$x$ (in decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>1.25</td>
</tr>
<tr>
<td>110</td>
<td>1.5</td>
</tr>
<tr>
<td>111</td>
<td>1.75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$b_1 b_2$</th>
<th>$x$ (in decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>101</td>
<td>2.5</td>
</tr>
<tr>
<td>110</td>
<td>3.0</td>
</tr>
<tr>
<td>111</td>
<td>3.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$b_1 b_2$</th>
<th>$x$ (in decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>4.0</td>
</tr>
<tr>
<td>101</td>
<td>5.0</td>
</tr>
<tr>
<td>110</td>
<td>6.0</td>
</tr>
<tr>
<td>111</td>
<td>7.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$b_1 b_2$</th>
<th>$x$ (in decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>8.0</td>
</tr>
<tr>
<td>101</td>
<td>10.0</td>
</tr>
<tr>
<td>110</td>
<td>12.0</td>
</tr>
<tr>
<td>111</td>
<td>14.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$b_1 b_2$</th>
<th>$x$ (in decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2^3</td>
</tr>
<tr>
<td>101</td>
<td>2^4</td>
</tr>
<tr>
<td>110</td>
<td>2^5</td>
</tr>
<tr>
<td>111</td>
<td>2^6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$b_1 b_2$</th>
<th>$x$ (in decimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>2^-2</td>
</tr>
<tr>
<td>101</td>
<td>2^-3</td>
</tr>
<tr>
<td>110</td>
<td>2^-4</td>
</tr>
<tr>
<td>111</td>
<td>2^-5</td>
</tr>
</tbody>
</table>

Same steps are performed to obtain the negative numbers. For simplicity, we will show only the positive numbers in this example.
\[ x = \pm 1. b_1 b_2 \times 2^m \quad \text{for} \quad m \in [-4,4] \quad \text{and} \quad b_i \in \{0,1\} \]

- 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** ($\epsilon_m$): is defined as the distance (gap) between 1 and the next larger floating point number.

$$x = \pm 1. b_1 b_2 \times 2^m \text{ for } m \in [-4,4] \text{ and } b_i \in \{0,1\}$$

$\epsilon_m = (0.01)_2 \times 2^0 = 0.25$
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:

\[ x = s.c \times f \]

\[ s = \text{sign} \quad c = \text{exponent} \quad f = \text{significand} \]

\[ x = (-1)^s 1.f \times 2^{c - \text{shift}} \]

\[ m = c - \text{shift} \]
Precisions:

IEEE-754 Single precision (32 bits):

\[ x = \begin{array}{ccc}
S & c = m + 127 & f \\
sign & exponent & significand \\
(1-bit) & (8-bit) & (23-bit)
\end{array} \]

Finite representation: not all numbers can be represented exactly!

IEEE-754 Double precision (64 bits):

\[ x = \begin{array}{ccc}
S & c = m + 1023 & f \\
sign & exponent & significand \\
(1-bit) & (11-bit) & (52-bit)
\end{array} \]
Special Values:

\[ x = (-1)^s \cdot f \times 2^m = \begin{array}{c|c|c} s & c & f \end{array} \]

1) **Zero**:  
\[ x = \begin{array}{c|c|c} s & 000 \ldots 000 & 0000 \ldots 0000 \end{array} \]

2) **Infinity**:  
\[ x = \begin{array}{c|c|c} s & 111 \ldots 111 & 0000 \ldots 0000 \end{array} \]

3) **NaN**: (results from operations with undefined results)  
\[ x = \begin{array}{c|c|c} s & 111 \ldots 111 & \text{anything} \neq 00 \ldots 00 \end{array} \]

Note that the exponent \( c = (000 \ldots 000) \) and \( c = (111 \ldots 111) \) are reserved for these special cases, which limits the exponent range for the other numbers.
IEEE-754 Single Precision (32-bit)

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

- **S** sign (1-bit)
- **c = m + 127** exponent (8-bit)
- **f** significand (23-bit)

\( s = 0 \): positive sign, \( s = 1 \): negative sign

Reserved exponent number for special cases:
\( c = (11111111)_2 = 255 \) and \( c = (00000000)_2 = 0 \)

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)

\[ x = (-1)^s 1. f \times 2^m = \begin{array}{c} s \end{array} \begin{array}{c} c \end{array} \begin{array}{c} f \end{array} \quad c = m + 127 \]

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

\[ (1)_{10} = \begin{array}{c} 0 \end{array} \begin{array}{c} 01111111 \end{array} \begin{array}{c} 00000000000000000000000000000000001 \end{array} \]

\[ (1)_{10} + \varepsilon_m = \begin{array}{c} 0 \end{array} \begin{array}{c} 01111111 \end{array} \begin{array}{c} 00000000000000000000000000000000101 \end{array} \]

\[ \varepsilon_m = 2^{-23} \approx 1.2 \times 10^{-7} \]

- 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)

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

<table>
<thead>
<tr>
<th>S</th>
<th>c = m + 1023</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>sign (1-bit)</td>
<td>exponent (11-bit)</td>
<td>significand (52-bit)</td>
</tr>
</tbody>
</table>

\( s = 0 \): positive sign, \( s = 1 \): negative sign

Reserved exponent number for special cases:
\( c = (11111111111)_2 = 2047 \) and \( c = (000000000000)_2 = 0 \)

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)

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

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

\[
(1)_{10} = \begin{array}{c}
0 \\
0111 \ldots 111 \\
000000000000 \ldots 000000000
\end{array}
\]

\[
(1)_{10} + \epsilon_m = \begin{array}{c}
0 \\
0111 \ldots 111 \\
000000000000 \ldots 000000001
\end{array}
\]

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

- **Smallest positive normalized FP number:**

\[
\text{UFL} = 2^0 = 2^{-1022} \approx 2.2 \times 10^{-308}
\]

- **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):

\[ x = s \begin{array}{c} \scriptstyle c = 000 \ldots 000 \end{array} f \]

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

\[ x = (−1)^s \, 1.f \times 2^m = \begin{array}{c} s \, c \, f \\ m = c - 127 \end{array} \]

<table>
<thead>
<tr>
<th>Stored binary exponent (c)</th>
<th>Significand fraction (f)</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000</td>
<td>0000…0000</td>
<td>zero</td>
</tr>
<tr>
<td>000000000</td>
<td>any f ≠ 0</td>
<td>((-1)^s , 0.f \times 2^{-126})</td>
</tr>
<tr>
<td>000000001</td>
<td>any f</td>
<td>((-1)^s , 1.f \times 2^{-126})</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>111111110</td>
<td>any f</td>
<td>((-1)^s , 1.f \times 2^{127})</td>
</tr>
<tr>
<td>111111111</td>
<td>any f ≠ 0</td>
<td>NaN</td>
</tr>
<tr>
<td>111111111</td>
<td>0000…0000</td>
<td>infinity</td>
</tr>
</tbody>
</table>
Example

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

- Convert the decimal number to binary: $(37.625)_{10} = (100101.101)_{2}$

<table>
<thead>
<tr>
<th>$2^5$</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th>$2^{-1}$</th>
<th>$2^{-2}$</th>
<th>$2^{-3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0.5</td>
<td>0.25</td>
<td>0.125</td>
</tr>
<tr>
<td>#</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>37.625</td>
<td>5.625</td>
<td>5.625</td>
<td>5.625</td>
<td>1.625</td>
<td>1.625</td>
<td>0.625</td>
<td>0.125</td>
<td>0</td>
</tr>
</tbody>
</table>

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

$(100101.101)_{2} = (1.00101101)_{2} \times 2^5$

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

$c = m + 127 = 132 = (10000100)_{2}$

0 10000100 0010110100000000000000000
What is the equivalent decimal number?

<table>
<thead>
<tr>
<th>0</th>
<th>00000000</th>
<th>00000000000000000000000000000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11111111</td>
<td>00000000000000000000000000000000</td>
</tr>
<tr>
<td>0</td>
<td>11111111</td>
<td>11111111110000111111111111</td>
</tr>
<tr>
<td>0</td>
<td>00000000</td>
<td>11110000000000000000000000000000</td>
</tr>
<tr>
<td>0</td>
<td>01111111</td>
<td>00000000000000000000000000000000</td>
</tr>
</tbody>
</table>
Iclicker question

A number system can be represented as \( x = \pm 1. b_1 b_2 b_3 \times 2^m \)
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
A number system can be represented as $x = \pm 1. b_1 b_2 b_3 b_4 \times 2^m$
for $m \in [-6, 6]$ and $b_i \in \{0, 1\}$.

1) Let’s say you want to represent the decimal number 19.625 using the binary number system above. Can you represent this number exactly?

2) What is the range of integer numbers that you can represent exactly using this binary system?
Iclicker question

Determine the decimal number corresponding to the following single-precision machine number:

1 10011001 000000000000000000000000001

A) 67,108,872
B) −67,108,872
C) 67,108,864
D) −67,108,864
Determine the double-precision machine representation of the decimal number $x = -37.625$

A) $1 \ 10000100000 \ 001011010000000 \ ... \ 0$

B) $1 \ 10000000100 \ 001011010000000 \ ... \ 0$

C) $0 \ 10000100000 \ 001011010000000 \ ... \ 0$

D) $0 \ 10000000100 \ 001011010000000 \ ... \ 0$

(52-bit)