IEEE Floating Point Representation
The 60’s & 70’s had different floating point systems. e.g. IBM 360/370 had numbers $\pm m \times 16^E$, i.e. hexadecimal base.

Through the efforts of W. Kahan & others, a binary floating point standard was developed, and followed e.g. by Intel and Motorola: IEEE floating point standard (ANSI/IEEE Std 754-1985).
We write IEEE FPS or FPA.

Three important requirements:

- **consistent** representation of floating point numbers across machines

- **correctly rounded** arithmetic

- **consistent** and **sensible** treatment of exceptional situations (e.g. division by 0).
Normalized Numbers
We normalized nonzero numbers thus:
\[ x = m \times 2^E, \text{ where } 1 \leq m < 2, \text{ i.e.} \]
\[ m = (b_0.b_1b_2b_3\ldots)_2, \text{ with } b_0 = 1. \]
Since we know this bit is 1, it need not be stored. So use 23 bits of the significand field to store \( b_1, b_2, \ldots, b_{23} \) instead of \( b_0, b_1, \ldots, b_{22} \), changing precision from \( \epsilon = 2^{-22} \) to \( \epsilon = 2^{-23} \). The stored bitstring is now the fractional part of the significand, referred to as the fraction field. Given a string of bits in the fraction field, it is necessary to imagine that the symbols “1.” appear in front, called hidden bit normalization.

Note: A pattern of all zeros in the fraction field of a normalized number represents the significand 1.0, not 0.0, so:

zero will have to be a special number
Special Numbers:
Another **special number** is the number $\infty$. This allows e.g. $1.0/0.0 \rightarrow \infty$, instead of terminating with an **overflow** message.

What about $-\infty$?
Have $-\infty$ as well as $\infty$ and $-0$ as well as $0$, where $-0$ and $0$ are **two different representations for the same value**, while $-\infty$ and $\infty$ represent **two very different numbers**.

Another **special number** is NaN, or “Not a Number”, and is an **error pattern**.

All **special numbers**, including **subnormal numbers**, are represented by a **special bit pattern** in the **exponent field**.

There are **3 standard types** in IEEE FPA: single, double, and extended precision. **Single precision** numbers use **32-bit words**
Table 1: IEEE Single Precision

<table>
<thead>
<tr>
<th>If exponent $a_1 \ldots a_8$ is</th>
<th>Then value is</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(00000000)<em>2 = (0)</em>{10}$</td>
<td>$\pm(0.b_1\ldots b_{23})_2 \times 2^{-126}$</td>
</tr>
<tr>
<td>$(00000001)<em>2 = (1)</em>{10}$</td>
<td>$\pm(1.b_1\ldots b_{23})_2 \times 2^{-126}$</td>
</tr>
<tr>
<td>$(00000010)<em>2 = (2)</em>{10}$</td>
<td>$\pm(1.b_1\ldots b_{23})_2 \times 2^{-125}$</td>
</tr>
<tr>
<td>$(00000011)<em>2 = (3)</em>{10}$</td>
<td>$\pm(1.b_1\ldots b_{23})_2 \times 2^{-124}$</td>
</tr>
<tr>
<td>$(01111111)<em>2 = (127)</em>{10}$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$(10000000)<em>2 = (128)</em>{10}$</td>
<td>$\pm(1.b_1\ldots b_{23})_2 \times 2^0$</td>
</tr>
<tr>
<td>$(11111100)<em>2 = (252)</em>{10}$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$(11111101)<em>2 = (253)</em>{10}$</td>
<td>$\pm(1.b_1\ldots b_{23})_2 \times 2^1$</td>
</tr>
<tr>
<td>$(11111110)<em>2 = (254)</em>{10}$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>$(11111111)<em>2 = (255)</em>{10}$</td>
<td>$\pm(1.b_1\ldots b_{23})_2 \times 2^{125}$</td>
</tr>
<tr>
<td>$(11111111)<em>2 = (255)</em>{10}$</td>
<td>$\pm\infty$ if $b_1, b_{23} = 0$; NaN otherwise.</td>
</tr>
</tbody>
</table>

The $\pm$ refers to the sign, $0$ for positive. Line 1 shows zero requires a zero bitstring for the exponent field as well as for the fraction:

| 0 | 00000000 | 00000000000000000000000000000000 |

Initial unstored bit is 0, not 1, in line 1.
IEEE Single Precision, ctd.

\[ \pm \ a_1a_2a_3 \ldots a_8 \ b_1b_2b_3 \ldots b_{23} \]

If exponent is zero, but fraction is nonzero, the number represented is subnormal.

All lines except the first and the last refer to the normalized numbers, i.e. all the numbers which are not special.

The exponent representation \( a_1a_2 \ldots a_8 \) uses biased representation: this bitstring is the binary representation of \( E + 127 \).

(127 is the exponent bias).

\( 1 = (1.000 \ldots 0)_2 \times 2^0 \) is stored as

\[
\begin{array}{c|c|c}
0 & 01111111 & \underbrace{00000000000000000000000000000000} \\
\end{array}
\]

The exponent bitstring is (binary) 0+127 and the fraction is 0 (the fractional part of 1.0).

Q: How is \( 32. = (1.0)_2 \times 2^5 \) stored ??

\[
\begin{array}{c|c|c}
0 & 10000100 & \underbrace{00000000000000000000000000000000} \\
\end{array}
\]
\[ \pm a_1 a_2 a_3 \ldots a_8 \mid b_1 b_2 b_3 \ldots b_{23} \] ctd.

**Exponent range** for normalized numbers is 00000001 to 11111110 (1 to 254), representing **actual exponents**

\[ E_{\min} = -126 \text{ to } E_{\max} = 127 \]

The **smallest normalized number** which can be stored is \((1.000 \ldots 0)_2 \times 2^{-126}:\)

| 0 | 00000001 | 00000000000000000000000000000000 |

approximately \(1.2 \times 10^{-38}\). The **largest normalized number** is \((1.111 \ldots 1)_2 \times 2^{127},\)

| 0 | 11111110 | 11111111111111111111111111 |

approximately \(3.4 \times 10^{38}\).

| If exponent \(a_1 \ldots a_8\) is \((11111111)_2 = (255)_{10}\) | Then value is \(\pm \infty\) if \(b_1, \ldots, b_{23} = 0;\) \(\text{NaN}\) otherwise |

This last line shows an **exponent bitstring of all ones** is a special pattern for \(\pm \infty\) and \(\text{NaN}\), depending on the value of the fraction.
**IEEE Single Precision, ctd.**

\[ \pm \begin{array}{c|cc} \pm & a_1a_2a_3\ldots a_8 & b_1b_2b_3\ldots b_{23} \\
\hline 
\end{array} \]

<table>
<thead>
<tr>
<th>If exponent (a_1..a_8) is</th>
<th>Then value is</th>
</tr>
</thead>
<tbody>
<tr>
<td>((00..00)<em>2 = (0)</em>{10})</td>
<td>(\pm(0.b_1..b_{23})_2 \times 2^{-126})</td>
</tr>
</tbody>
</table>

The idea of the 1st line is although \(2^{-126}\) is the smallest **normalized** number, we can represent **smaller** numbers called **subnormal** numbers. e.g. \(2^{-127} = (0.1)_2 \times 2^{-126}\), as

\[
\begin{array}{c|c}
0 & 00000000 \quad 1000000000000000000000000000000000000000 \\
\end{array}
\]

and \(2^{-149} = (0.0000\ldots .01)_2 \times 2^{-126}\)

(with 22 zero bits after the binary point) as

\[
\begin{array}{c|c}
0 & 00000000 \quad 00000000000000000000000000000000000000001 \\
\end{array}
\]

the smallest nonzero number we can store.

The \(2^{-126}\) in the first line allows us numbers below the smallest **normalized** number. (Subnormal numbers cannot be normalized, as this gives exponents which do not fit).
**The Toy System, ctd.**
Significand $b_0.b_1b_2$, exponent $E \in \{-1, 0, 1\}$:

\[
\begin{array}{c}
\cdots\ 0 & 1 & 2 & 3 \\
\end{array}
\]

The Toy System, with **Subnormal** Numbers.

We get three extra numbers: $(0.11)_2 \times 2^{-1}$, $(0.10)_2 \times 2^{-1}$ and $(0.01)_2 \times 2^{-1}$. **The gap between 0 and the smallest positive normalized number is filled in by the subnormal numbers**, (same spacing as for normalized numbers with exponent $-1$)

**IEEE Single Precision, ctd.**
Subnormal numbers are **less accurate**, (less room for nonzero bits in the fraction). e.g. $(1/10) \times 2^{-133} = (0.11001100\ldots)_2 \times 2^{-136}$ is

\[
\begin{array}{c|c|c}
0 & 00000000 & 000000000001100110011001
\end{array}
\]
IEEE Double Precision

2nd Format: Each **double precision** floating point number is stored in a **64-bit double word**. Ideas the same; field widths (11 & 52) and exponent bias (1023) different. $b_1, \ldots, b_{52}$ can be stored instead of $b_1, \ldots, b_{23}$.

**Table 2: IEEE Double Precision**

<table>
<thead>
<tr>
<th>±</th>
<th>$a_1a_2a_3\ldots a_{11}$</th>
<th>$b_1b_2b_3\ldots b_{52}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>If exponent is $a_{1..11}$</td>
<td>Then value is</td>
<td></td>
</tr>
<tr>
<td>(000..0000)$<em>2$ = (0)$</em>{10}$</td>
<td>$\pm(0.b_1..b_{52})_2 \times 2^{-1022}$</td>
<td></td>
</tr>
<tr>
<td>(000..0001)$<em>2$ = (1)$</em>{10}$</td>
<td>$\pm(1.b_1..b_{52})_2 \times 2^{-1022}$</td>
<td></td>
</tr>
<tr>
<td>(000..0010)$<em>2$ = (2)$</em>{10}$</td>
<td>$\pm(1.b_1..b_{52})_2 \times 2^{-1021}$</td>
<td></td>
</tr>
<tr>
<td>(000..0011)$<em>2$ = (3)$</em>{10}$</td>
<td>$\pm(1.b_1..b_{52})_2 \times 2^{-1020}$</td>
<td></td>
</tr>
<tr>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td></td>
</tr>
<tr>
<td>(01..111)$<em>2$ = (1023)$</em>{10}$</td>
<td>$\pm(1.b_1..b_{52})_2 \times 2^0$</td>
<td></td>
</tr>
<tr>
<td>(10..000)$<em>2$ = (1024)$</em>{10}$</td>
<td>$\pm(1.b_1..b_{52})_2 \times 2^1$</td>
<td></td>
</tr>
<tr>
<td>$\downarrow$</td>
<td>$\downarrow$</td>
<td></td>
</tr>
<tr>
<td>(11..100)$<em>2$ = (2044)$</em>{10}$</td>
<td>$\pm(1.b_1..b_{52})_2 \times 2^{1021}$</td>
<td></td>
</tr>
<tr>
<td>(11..101)$<em>2$ = (2045)$</em>{10}$</td>
<td>$\pm(1.b_1..b_{52})_2 \times 2^{1022}$</td>
<td></td>
</tr>
<tr>
<td>(11..110)$<em>2$ = (2046)$</em>{10}$</td>
<td>$\pm(1.b_1..b_{52})_2 \times 2^{1023}$</td>
<td></td>
</tr>
<tr>
<td>(11..111)$<em>2$ = (2047)$</em>{10}$</td>
<td>$\pm\infty$ if $b_1, \ldots, b_{52} = 0$; NaN otherwise</td>
<td></td>
</tr>
</tbody>
</table>
**Extended precision**, the third format, is usually an **80-bit word**, with **1 bit sign**, **15 bit exponent** and **64 bit significand**, with leading bit of a normalized number **not** hidden.

We see that the first **single precision** number larger than 1 is $1 + 2^{-23}$, while the first **double precision** number larger than 1 is $1 + 2^{-52}$.

**Table 3: What is that Precision?**

<table>
<thead>
<tr>
<th>Format</th>
<th>Error $\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE Single</td>
<td>$2^{-23} \approx 1.2 \times 10^{-7}$</td>
</tr>
<tr>
<td>IEEE Double</td>
<td>$2^{-52} \approx 1.1 \times 10^{-16}$</td>
</tr>
<tr>
<td>IEEE Extended</td>
<td>$2^{-63} \approx 1.1 \times 10^{-19}$</td>
</tr>
</tbody>
</table>

The fraction of a single precision normalized number has **exactly 23 bits of accuracy**. This corresponds to **approximately 7 decimal digits of accuracy**. In double precision, the fraction has **exactly 52 bits of accuracy**. This corresponds to **approximately 16 decimal digits of accuracy**.
Q: IEEE single precision:

(i) How is 2. represented ??

| 0 | 10000000 | 00000000000000000000000000000000 |

(ii) What is the next biggest IEEE single precision number larger than 2 ??

| 0 | 10000000 | 00000000000000000000000000000001 |

(iii) What is the gap between 2 and the first IEEE single precision number larger than 2?

\[ 2^{-23} \times 2 = 2^{-22}. \]

General Result: Let \( x = m \times 2^E \) be a normalized single precision number, with \( 1 \leq m < 2 \). The gap between \( x \) and the next largest single precision number is

\[ \epsilon \times 2^E. \]
Rounding
We use “IEEE Computer Floating Point Numbers” (IEEE CFPNs, or just Floating Point Numbers) to include ±0, subnormal & normalized CFPNs, & ±∞ in a given format, e.g. single. These form a finite set.

If \( N_{\text{min}} \) & \( N_{\text{max}} \) are the minimum & maximum positive normalized such numbers, we say the binary representation \( x = m \times 2^E \) of a general real number is “normalized” if

\[
N_{\text{min}} \leq |x| \leq N_{\text{max}}, \text{ and } 1 \leq m < 2.
\]

Q: For any number \( x \) which is not a floating point number, what are two obvious choices for the floating point approximation to \( x \) ??

\( x_– \) closest floating point number less than \( x \),

\( x_+ \) the closest greater than \( x \).
Rounding, ctd.
Significand $b_0.b_1b_2$, exponent $E \in \{-1, 0, 1\}$:

\[
\begin{array}{cccc}
\ldots & 0 & 1 & x_2 & 3 \\
\hline
\end{array}
\]

Rounding in the Toy System.

e.g. if $x = 1.7$, then $x_- = 1.5$ and $x_+ = 1.75$.

Using IEEE single precision, if

\[x = (b_0.b_1b_2 \ldots b_{23}b_{24}b_{25} \ldots)_2 \times 2^E,\]

is positive and normalized, then

\[x_- = (b_0.b_1b_2 \ldots b_{23})_2 \times 2^E,\]

is obtained by discarding $b_{24}$, $b_{25}$, etc.

An algorithm for $x_+$ is more complicated since it may involve some bit “carries”. If $x$ is negative, the situation is reversed: $x_+$ is obtained by dropping bits $b_{24}$, $b_{25}$, etc.
Correctly Rounded Arithmetic
The IEEE standard defines the correctly rounded value of \( x \), \( \text{round}(x) \).
If \( x \) is a floating point number, \( \text{round}(x) = x \).
Otherwise \( \text{round}(x) \) depends on the rounding mode in effect:

- **Round down:** \( \text{round}(x) = x_- \).

- **Round up:** \( \text{round}(x) = x_+ \).

- **Round towards zero:**
  \( \text{round}(x) \) is either \( x_- \) or \( x_+ \), whichever is between zero and \( x \).

- **Round to nearest:** \( \text{round}(x) \) is either \( x_- \) or \( x_+ \), whichever is nearer to \( x \). In the case of a tie, the one with its least significant bit equal to zero is chosen.
Correctly Rounded Arithmetic, ctd.
If $x$ is positive, then $x_-$ is between zero and $x$, so round down and round towards zero have the same effect. Round towards zero simply requires truncating the binary expansion, i.e. discarding bits.
The most useful mode is round to nearest.

Significand $b_0.b_1b_2$, exponent $E \in \{-1,0,1\}$:

```
......0 1 x2 3
```

Rounding in the Toy System

e.g. if $x = 1.7$, then $x_- = 1.5$ and $x_+ = 1.75$, so with $x = 1.7$, this gives a “rounded” value of $x$ equal to 1.75

“Round” with no qualification means “round to nearest”.
Absolute Rounding Error

\[ |\text{round}(x) - x| \] is called the absolute rounding error associated with \( x \), (value depends on mode). For all modes the absolute rounding error is less than the gap between \( x_- \) and \( x_+ \).

For general normalized \( x > 0 \),

\[ x = (b_0.b_1b_2 \ldots b_{23}b_{24}b_{25} \ldots)_2 \times 2^E, \quad b_0 = 1. \]

**IEEE single** \( x_- = (b_0.b_1b_2 \ldots b_{23})_2 \times 2^E \).

So for any mode \( |\text{round}(x) - x| < 2^{-23} \times 2^E \).

In general for any rounding mode:

\[ |\text{round}(x) - x| < \epsilon \times 2^E, \quad (1) \]

**Q:** (i) For round towards zero, could the absolute rounding error equal \( \epsilon \times 2^E \) ??

(ii) Does (1) hold if \( x \) is subnormal, i.e. \( E = -126 \) and \( b_0 = 0 \) ??
Relative Rounding Error

The **relative rounding error** is defined to be

\[ \delta \equiv \frac{\text{round}(x)}{x} - 1 = \frac{\text{round}(x) - x}{x}. \]

Since \(|\text{round}(x) - x| < \epsilon \times 2^E\), & for **normalized** numbers

\[ x = \pm m \times 2^E, \quad \text{where } m \geq 1, \]

we have, for **all** rounding modes,

\[ |\delta| < \frac{\epsilon \times 2^E}{2^E} = \epsilon. \]  \hspace{1cm} (2)

**Q:** Does (2) hold if \( x \) is **subnormal**, i.e. \( E = -126 \) and \( b_0 = 0 \) ?? Why ??

Note for any **normalized** \( x \)

\[ \text{round}(x) = x(1 + \delta), \quad |\delta| < \epsilon. \]
An Important Idea

From the definition of $\delta$ we see

$$\text{round}(x) = x(1 + \delta),$$

so the rounded value of an arbitrary normalized number $x$ is equal to $x(1 + \delta)$, where, regardless of the rounding mode,

$$|\delta| < \epsilon.$$ 

This is very important, because you can think of the stored value of $x$ as not exact, but as exact within a factor of $1 + \epsilon$. IEEE single precision numbers are good to a factor of about $1 + 10^{-7}$, which means that they have about 7 accurate decimal digits.
Special Case of Round to Nearest

For **round to nearest**, the **absolute** rounding error can be **no more than half** the gap **between** $x_-$ **and** $x_+$. This means in IEEE single, for all $|x| \leq N_{\text{max}}$:

$$|\text{round}(x) - x| \leq 2^{-24} \times 2^E,$$

and in general

$$|\text{round}(x) - x| \leq \frac{1}{2} \epsilon \times 2^E.$$

The previous analysis for **round to nearest** then gives for **normalized** $x$:

$$\text{round}(x) = x(1 + \delta),$$

$$|\delta| \leq \frac{1}{2} \epsilon \times 2^E = \frac{1}{2} \epsilon.$$