Quick survey of floating-point numbers

Background: fractional binary numbers
IEEE floating-point standard
Floating-point operations and rounding
Lessons for the programmer

There are many more details that we will skip.
It's a 58-page standard...

Optional reading: Patterson & Hennessy 3.5

Fractional Binary Numbers

\[ \sum_{k=-j}^{i} b_k \cdot 2^k \]

Fixed-Point Representation

Implied binary point. Example:
\[ b_7 b_6 b_5 b_4 \[ b_1 b_0 \]
Same hardware as for integer arithmetic.
\[ b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \]

Fixed point = fixed range and fixed precision
range: difference between largest and smallest representable numbers
precision: smallest difference between any two representable numbers

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 and 3/4</td>
<td>101.111_2</td>
</tr>
<tr>
<td>2 and 7/8</td>
<td>10.111_2</td>
</tr>
<tr>
<td>47/64</td>
<td>0.1011111_2</td>
</tr>
</tbody>
</table>

Observations
Shift left =
Shift right =
Numbers of the form 0.111111..._2 are...?

Limitations:
Exact representation possible only for numbers of the form \( x \times 2^y \), where \( x \) and \( y \) are integers.
Other rational numbers have repeating bit representations
\( 1/3 = 0.333333..._2 = 0.0101010[01]..._2 \)
IEEE Floating Point

Analogous to scientific notation

\[ \begin{align*}
12000000 & = 1.2 \times 10^7 \\
0.0000012 & = 1.2 \times 10^{-6}
\end{align*} \]

IEEE Standard 754 used by all major CPUs today

IEEE = Institute of Electrical and Electronics Engineers

Driven by numerical concerns
- Rounding, overflow, underflow
- Numerically well-behaved, but hard to make fast in hardware

Floating Point Representation

Numerical form:

\[ V = (-1)^s \times M \times 2^E \]

Sign bit \( s \) determines whether number is negative or positive
Significant (mantissa) \( M \) normally a fractional value in range \([1.0,2.0)\]
Exponent \( E \) weights value by a (possibly negative) power of two

Representation:
- MSB \( s \) = sign bit
- \( \exp \) field encodes \( E \) (but is not equal to \( E \))
- \( \frac{\text{frac}}{\text{field}} \) encodes \( M \) (but is not equal to \( M \))

Precisions

Single precision (float): 32 bits

- 1 bit
- 8 bits
- 23 bits

Double precision (double): 64 bits

- 1 bit
- 11 bits
- 52 bits

Finite representation of infinite range:
- Not all values can be represented exactly.
- Some are approximated.

Normalization and Special Values

\[ V = (-1)^s \times M \times 2^E \]

"Normalized" = \( M \) has the form 1.xxxxx
- As in scientific notation
- \( 0.011 \times 2^3 = 1.1 \times 2^1 \), latter is more compact
- Do not store the (guaranteed) leading 1.

Special values:
- (How do we represent 0.0? 1.0/0.0?)
  - zero:
    - \( s = 0 \)
    - \( \exp = \infty \)
    - \( \frac{\text{frac}}{\text{field}} = 00...0 \)
  - +\( \infty \), -\( \infty \):
    - \( \exp = \text{max} \)
    - \( \frac{\text{frac}}{\text{field}} = 00...0 \)
  - 1/0.0 = -1.0/0.0 = +\( \infty \), 1.0/0.0 = -1.0/0.0 = -\( \infty \)

-NaN ("Not a Number"):
  - \( \exp = \text{max} \)
  - \( \frac{\text{frac}}{\text{field}} \neq 00...0 \)
  - Undefined results: sqrt(-1), \( \infty \), etc.

Denormalized/subnormal values (near 0.0) not covered here.
Floating Point Operations

\[ V = (-1)^s M \times 2^E \]

1. Compute the exact result.
2. Round the result to make it fit into desired precision:
   - Overflow if exponent too large.
   - Drop LSBs of significand if result is too precise to fit into frac.
   - Underflow if nearest representable value is 0 (for non-zero result).

\[ x + y = \text{Round}(x + y) \]
\[ x \times y = \text{Round}(x \times y) \]

Lessons for the programmer

float ≠ real number ≠ double
Rounding breaks associativity and other properties.

double a = ..., b = ...;
... if (a == b) ...
... if (closeEnough(a, b)) ...