I refuses
to recognise teh existence of teh monday
• Floating point standard: machine not compiler
  – GPUs

• Practice problem:
  – 2.34

(Shift practice)
Decimal Numbers

• 1234.5678
Fractional Binary Numbers

- **Representation**
  - Bits to right of “binary point” represent fractional powers of 2
  - Represents rational number: \[ \sum_{k=-j}^{i} b_k \cdot 2^k \]
## Fractional Binary Numbers: Examples

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-3/4</td>
<td>101.11₂</td>
</tr>
<tr>
<td>2-7/8</td>
<td>10.111₂</td>
</tr>
<tr>
<td>63/64</td>
<td>0.111111₁₂</td>
</tr>
</tbody>
</table>

### Observations

- Divide by 2 by shifting right
- Multiply by 2 by shifting left
- Numbers of form $0.\overline{111111}_2$ are just below 1.0
  - $1/2 + 1/4 + 1/8 + ... + 1/2^i + ... \rightarrow 1.0$
  - Use notation $1.0 - \varepsilon$

• Limitations?
Representable Numbers

- **Limitation**
  - Can only exactly represent numbers of the form $x/2^k$
  - Other rational numbers have repeating bit representations

<table>
<thead>
<tr>
<th>Value</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1/3$</td>
<td>$0.010101010101[01]..._2$</td>
</tr>
<tr>
<td>$1/5$</td>
<td>$0.001100110011[0011]..._2$</td>
</tr>
<tr>
<td>$1/10$</td>
<td>$0.0001100110011[0011]..._2$</td>
</tr>
</tbody>
</table>

Example: Patriot Missile in 1st Gulf War
Heard of them?

- 6-bit IEEE-like format
IEEE Floating Point

- **IEEE Standard 754**
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
    - Supported by all major CPUs

- **Driven by numerical concerns**
  - Nice standards for rounding, overflow, underflow
  - Hard to make fast in hardware
    - Numerical analysts predominated over hardware designers in defining standard
Floating Point Representation

■ Numerical Form:

\[(−1)^s \ M \ 2^E\]

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

■ Encoding

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

- Single precision: 32 bits
  - s: 1 bit, exp: 8 bits, frac: 23 bits

- Double precision: 64 bits
  - s: 1 bit, exp: 11 bits, frac: 52 bits

- Extended precision: 80 bits (Intel only)
  - s: 1 bit, exp: 15 bits, frac: 63 or 64 bits
Normalized Values

- **Condition:** $\text{exp} \neq 000...0$ and $\text{exp} \neq 111...1$

- **Exponent coded as biased value:** $E = \text{Exp} - \text{Bias}$
  - $\text{Exp}$: unsigned value $\text{exp}$
  - $\text{Bias} = 2^{e-1} - 1$, where $e$ is number of exponent bits
    - Single precision: $127$ ($\text{Exp: 1...254, E: -126...127}$)
    - Double precision: $1023$ ($\text{Exp: 1...2046, E: -1022...1023}$)

- **Significand coded with implied leading 1:** $M = 1.\text{xxx...x}_2$
  - $\text{xxx...x}$: bits of $\text{frac}$
  - Minimum when $000...0$ ($M = 1.0$)
  - Maximum when $111...1$ ($M = 2.0 - \varepsilon$)
  - Get extra leading bit for “free”
Normalized Encoding Example

- Value: \( \text{Float } F = 15213.0; \)
  - \( 15213_{10} = 11101101101101_2 \)
    \[ \begin{array}{c}
    0 \quad \text{sign bit} \\
    11101101101101_2 \\
    \end{array} \]

- Significand
  - \( M = \)
  - \( \text{frac} = \)

- Exponent
  - \( E = \)
  - \( \text{Bias} = \)
  - \( \text{Exp} = \)

- Result:
  \[ \begin{array}{c}
  0 \quad \text{sign bit} \\
  \text{exp} \\
  \text{frac} \\
  \end{array} \]
Tiny Floating Point Example

8-bit Floating Point Representation
- the sign bit is in the most significant bit.
- the next four bits are the exponent, with a bias of 7.
- the last three bits are the frac

Same general form as IEEE Format
- normalized, denormalized
- representation of 0, NaN, infinity