IEEE standard: basics I

- IEEE 754 during 80s, now standard everywhere
- Two IEEE standards:
  - 754: specify $\beta = 2$, $p = 24$ for single, $\beta = 2$, $p = 53$ for double
  - 854 ($\beta = 2$ or 10, does not specify how floating-point numbers are encoded into bits)
- Why IEEE 854 allows $\beta = 2$ or 10 but not other numbers? Some reasons:
  1. 10 is the base we use
  2. smaller $\beta$ causes smaller relative error
Why smaller $\beta$ causes smaller relative error? Consider

$\beta = 16, p = 1$ versus $\beta = 2, p = 4$

Both use 4 bits for significand, but their $\epsilon$ values are different

$$\epsilon = \frac{16}{2} 16^{-1} = 1/2, \quad \epsilon = \frac{2}{2} 2^{-4} = 1/16$$

We can see that $\epsilon$ of $\beta = 2, p = 4$ is smaller
However, IBM/370 uses $\beta = 16$. Why? Two possible reasons:

First, assume $4$ bytes = $32$ bits are allocated for a number. Let $\beta = 16$, $p = 6$.

significand:

$4 \times 6 = 24$ bits,

exponents:
32 − 24 − 1 = 7 bits (1 bit for sign)

range of exponent:

\[16^{-2^6} \text{ to } 16^{2^6} = 2^{28}\]

If instead \(\beta = 2\) is used and range of exponents is the same, then

9 bits \((-2^8 \text{ to } 2^8 = 2^9)\) for exponents

and

\[32 − 9 − 1 = 22\] for significand

Same exponents, less significand for \(\beta = 2\) (24 vs. 22)
Second reason: cost of shifting.
If $\beta = 16$, less frequently to adjust exponents when adding or subtracting two numbers
For modern computers, this saving is not important
Single precision: $\beta = 2, p = 24$ (23 bits as normalized), exponent 8, 1 bit for sign. Thus

$$32 = 23 + 8 + 1$$

An example: $176.625 = 1.0101100101 \times 2^7$

$$0 \quad 10000110 \quad 0101100101000000000000000$$

1 of 1. $\cdots$ is not stored (normalized)

Biased exponent (described later in detail)
10000110 = 128 + 4 + 2 = 134, 134 − 127 = 7
Note that exponent may be negative, but here we don’t use a sign bit for exponents

- Use rounding even

<table>
<thead>
<tr>
<th>Binary</th>
<th>Rounded</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.000111</td>
<td>10.00</td>
<td>(&lt; 1/2, down)</td>
</tr>
<tr>
<td>10.001110</td>
<td>10.01</td>
<td>(&gt; 1/2, up)</td>
</tr>
<tr>
<td>10.11100</td>
<td>11.00</td>
<td>(1/2, up)</td>
</tr>
<tr>
<td>10.10100</td>
<td>10.10</td>
<td>(1/2, down)</td>
</tr>
</tbody>
</table>
IEEE standard: significands and exponents

This example is from http://www.cs.cmu.edu/afs/cs/academic/class/15213-s12/www/lectures/04-float-4up.pdf

A summary

<table>
<thead>
<tr>
<th></th>
<th>Fortran</th>
<th>C</th>
<th>Bits</th>
<th>Exp.</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>REAL*4</td>
<td>float</td>
<td>32</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>Single-extended</td>
<td></td>
<td></td>
<td>44</td>
<td>≤ 11</td>
<td>32</td>
</tr>
<tr>
<td>Double</td>
<td>REAL*8</td>
<td>double</td>
<td>64</td>
<td>11</td>
<td>53</td>
</tr>
<tr>
<td>Double-extended</td>
<td></td>
<td></td>
<td>≥80</td>
<td>≥ 15</td>
<td>≥64</td>
</tr>
</tbody>
</table>
IEEE standard: significands and exponents

- Extended precision: we will give some brief discussion later
- In the above table, for single

\[ 32 = 8 + (24 - 1) + 1 = 8 + 24 \]

but for single-extended

\[ 44 \neq 11 + 32 \]

- Why \( 44 \neq 11 + 32 \)?
Hardware implementation of extended precision normal don’t use a hidden bit
(Refer to we normalized each number so 1 is not stored)

- Minimal normalized positive number

\[ 1 \times 2^{-126} \approx 1.17 \times 10^{-38} \]

\[ e_{\text{min}} = -126 \]
IEEE standard: significands and exponents

- 8 bits for exponent: 0 to 255. IEEE uses a biased approach for exponents

\[(0 \text{ to } 255) - 127 = -127 \text{ to } 128\]

- Then, $-127$ for 0 and denormalized numbers (discussed later), $-126$ to 127 for exponents, 128 for special quantity

- Thus

\[e_{\text{min}} = -126 \text{ and } e_{\text{max}} = 127\]
Why not $e_{\text{min}} = -127$ and $e_{\text{max}} = 126$

reasons: $1/2^{e_{\text{min}}}$ not overflow, $1/2^{e_{\text{max}}}$ underflow, but less serious