frac vs. frak

M=1.xxxxxxxxxxx

http://www.youtube.com/watch?v=r7KcpgQKo2I
Normalized Values

- Condition: $\exp \neq 000...0$ and $\exp \neq 111...1$

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

- Significand coded with implied leading 1: $M = 1 \ . \ xxx...x_2$
  - $xxx...x$: bits of $\frac{\text{frac}}{}$
  - Minimum when $000...0$ ($M = 1.0$)
  - Maximum when $111...1$ ($M = 2.0 - \varepsilon$)
  - Get extra leading bit for “free”
Denormalized Values

- **Condition:** $\text{exp} = 000...0$
- **Exponent value:** $E = -\text{Bias} + 1$ (instead of $E = 0 - \text{Bias}$)
- **Significand coded with implied leading 0:** $M = 0 . \text{xxx...x}_2$
  - $\text{xxx...x}$: bits of $\text{frac}$
- **Cases**
  - $\text{exp} = 000...0, \ \text{frac} = 000...0$
    - Represents value 0
    - Note distinct values: $+0$ and $-0$ (why?)
  - $\text{exp} = 000...0, \ \text{frac} \neq 000...0$
    - Numbers very close to 0.0
    - Lose precision as get smaller
    - Equispaced
Special Values

- **Condition:** exp = 111...1
  - **Case:** exp = 111...1, frac = 000...0
    - Represents value $\infty$ (infinity)
    - Operation that overflows
    - Both positive and negative
    - E.g., $1.0/0.0 = -1.0/-0.0 = +\infty$, $1.0/-0.0 = -\infty$
  - **Case:** exp = 111...1, frac $\neq$ 000...0
    - Not-a-Number (NaN)
    - Represents case when no numeric value can be determined
    - E.g., $\sqrt{-1}$, $\infty - \infty$, $\infty \times 0$
## Rounding Binary Numbers

### Binary Fractional Numbers
- “Even” when least significant bit is 0
- “Half way” when bits to right of rounding position = 100...₂

### Examples
- Round to nearest 1/4 (2 bits right of binary point)

<table>
<thead>
<tr>
<th>Value</th>
<th>Binary</th>
<th>Rounded</th>
<th>Action</th>
<th>Rounded Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 3/32</td>
<td>10.00011₂</td>
<td>10.00₂</td>
<td>(&lt;1/2—down)</td>
<td>2</td>
</tr>
<tr>
<td>2 3/16</td>
<td>10.00110₂</td>
<td>10.01₂</td>
<td>(&gt;1/2—up)</td>
<td>2 1/4</td>
</tr>
<tr>
<td>2 7/8</td>
<td>10.11100₂</td>
<td>11.00₂</td>
<td>( 1/2—up)</td>
<td>3</td>
</tr>
<tr>
<td>2 5/8</td>
<td>10.10100₂</td>
<td>10.10₂</td>
<td>( 1/2—down)</td>
<td>2 1/2</td>
</tr>
</tbody>
</table>
Creating Floating Point Number

- **Steps**
  - Normalize to have leading 1
  - Round to fit within fraction
  - Postnormalize to deal with effects of rounding

- **Case Study**
  - Convert 8-bit unsigned numbers to tiny floating point format
  - Example Numbers
    - 128 10000000
    - 15 00001101
    - 33 00010001
    - 35 00010011
    - 138 10001010
    - 63 00111111
Creating Floating Point Number

Steps
- Normalize to have leading 1
- Round to fit within fraction
- Postnormalize to deal with effects of rounding

Case Study
- Convert 8-bit unsigned numbers to tiny floating point format
- Example Numbers
  - 128 10000000
  - 15 00001101
  - 33 00010001
  - 35 00010011
  - 138 10001010
  - 63 00111111
Creating Floating Point Number

- **Steps**
  - Normalize to have leading 1
  - Round to fit within fraction
  - Postnormalize to deal with effects of rounding

- **Case Study**
  - Convert 8-bit unsigned numbers to tiny floating point format
  - **Example Numbers**
    | Integer | Binary | Float |
    |---------|--------|-------|
    | 128     | 10000000 | 0 1110 000 |
    | 15      | 00001101 | 0 1010 101 |
    | 33      | 00010001 | 0 1011 000 |
    | 35      | 00010011 | 0 1011 010 |
    | 138     | 10001010 | 0 1111 000 |
    | 63      | 00111111 | 0 1100 1111 → 0 1101 000 |
• 2.84, a/b