frac vs. *frak*

M=1.xxxxxxxxxx

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

Condition 1

Normalized Values

- **Condition: `exp` ≠ `000…0` and `exp` ≠ `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`**
  - Minimum when **`000…0`** (*M* = 1.0)
  - Maximum when **`111…1`** (*M* = 2.0 – ε)
  - Get extra leading bit for "free"

Denormalized Values

Condition 2

- **Condition: `exp` = `000…0`**

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

Special Values

Condition 3

- **Condition: `exp` = `111…1`**

- **Case: `exp` = `111…1`, `frac` = `000…0`**
  - Represents value ∞ (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` ≠ `000…0`**
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - E.g., sqrt(–1), $\infty - \infty$, $\infty * 0$

Rounding Binary Numbers

- **Binary Fractional Numbers**
  - "Even" when least significant bit is 0
  - "Half way" when bits to right of rounding position = $100\ldots_2$

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

Value	Binary	Rounded	Action	Rounded Value
2 3/32	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up)	3
2 5/8	10.10100_2	10.10_2	(1/2—down)	2 1/2

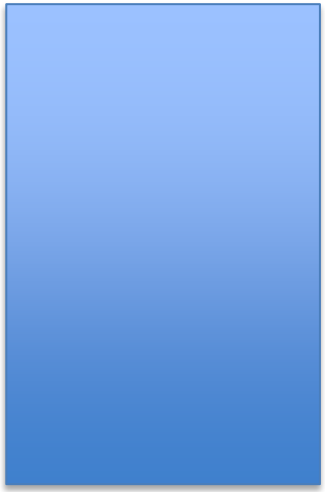
Creating Floating Point Number

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

7	6 … 3	2 … 0
s	exp	frac

- **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

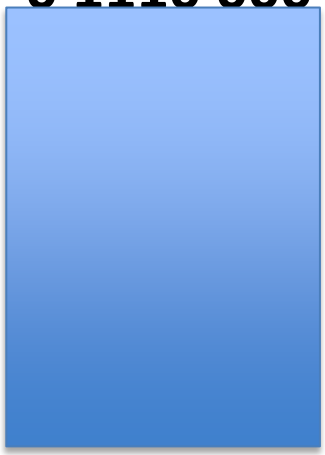
- **Steps**
  - Normalize to have leading 1
  - Round to fit within fraction
  - Postnormalize to deal with effects of rounding

7	6 … 3	2 … 0
s	exp	frac

- **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

0 1110 000



Creating Floating Point Number

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

7	6 … 3	2 … 0
s	exp	frac

- **Case Study**
  - Convert 8-bit unsigned numbers to tiny floating point format
  - Example Numbers

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 11111 → 0 1101 000**

- 2.84, a/b