Section 2: Integer & Floating Point Numbers

- Representation of integers: unsigned and signed
- Unsigned and signed integers in C
- Arithmetic and shifting
- Sign extension

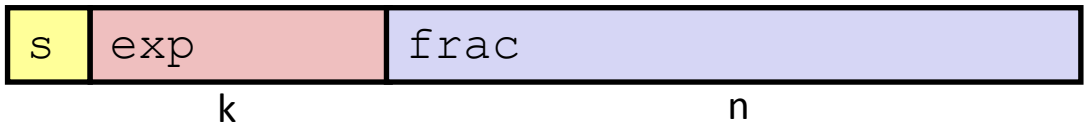
- Background: fractional binary numbers
- IEEE floating-point standard
- Floating-point operations and rounding
- Floating-point in C

How do we do operations?

- **Unlike the representation for integers, the representation for floating-point numbers is not *exact***

Floating Point Operations: Basic Idea

$V = (-1)^S * M * 2^E$



- $x +_f y = Round(x + y)$
- $x *_f y = Round(x * y)$
- **Basic idea for floating point operations:**
  - First, compute the exact result
  - Then, *round* the result to make it fit into desired precision:
    - Possibly overflow if exponent too large
    - Possibly drop least-significant bits of significand to fit into **`frac`**

Rounding modes

- **Possible rounding modes (illustrated with dollar rounding):**

	$1.40	$1.60	$1.50	$2.50	–$1.50
Round-toward-zero	$1	$1	$1	$2	–$1
Round-down (-∞)	$1	$1	$1	$2	–$2
Round-up (+∞)	$2	$2	$2	$3	–$1
Round-to-nearest	$1	$2	??	??	??
Round-to-even	$1	$2	$2	$2	–$2

- **What could happen if we're repeatedly rounding the results of our operations?**
  - If we always round in the same direction, we could introduce a statistical bias into our set of values!
- **Round-to-even avoids this bias by rounding up about half the time, and rounding down about half the time**
  - Default rounding mode for IEEE floating-point

Mathematical Properties of FP Operations

- **If overflow of the exponent occurs, result will be ∞ or -∞**
- **Floats with value ∞, -∞, and NaN can be used in operations**
  - Result is usually still ∞, -∞, or NaN; sometimes intuitive, sometimes not

- **Floating point operations are not always associative or distributive, due to rounding!**
  - (3.14 + 1e10) - 1e10 != 3.14 + (1e10 - 1e10)
  - 1e20 * (1e20 - 1e20) != (1e20 * 1e20) - (1e20 * 1e20)