- **Floating Point Representation** handles decimals, similar to scientific notation
  - **Fractional binary numbers** 0010.110 have a signed (or unsigned) representation left of decimal and fractional (e.g. 3/4) representation right of the decimal: number out of power of 2
  - Fixed in range and precision (cannot represent 1/3 precisely, for example)
  - **IEEE Floating point** is analogous to scientific notation
    - Value: \[ (-1)^s \times M \times 2^E \]
    - **Sign** bit determines negative, positive (MSB)
    - **Mantissa or Fraction** determines the fractional number in the range \([1.0, 2.0)\)
    - **Exponent** weights the value by a power of two (possibly negative or very small weight)
  - Normalized Form starts with 1.xxxx and we don’t bother storing the 1
    - Special values for +/-infinity, not-a-number, and zero
  - Floating point math is complicated: yields overflow, rounded/imprecise numbers, and underflow
    - compute the result, then round to desired precision, overflow as necessary, or drop part of fraction to fit
    - numbers lose precision: order of operations is important
  - C offers **float (4 bytes) and double (8 bytes)**
    - Casting between integers and floats may lose precision or overflow
    - Never test for equality: is not intuitive!
- **ISA (Instruction Set Architecture)** includes:
  - Set of Possible **instructions**
  - **System State** includes:
    - PC, which points to the currently executing instruction
    - Registers, which temporarily store data we're working with
    - Memory, which has longer-term storage
  - Defines the impact each instruction has on system state
  - ISAs are dominated by x86 (IA32) and x86-64 (an evolution of IA32)
- **Architecture**: everything you need to know to write assembly code
- **Microarchitecture**: the hardware implementation of the Architecture