Section 2: Integer & Floating Point Numbers

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

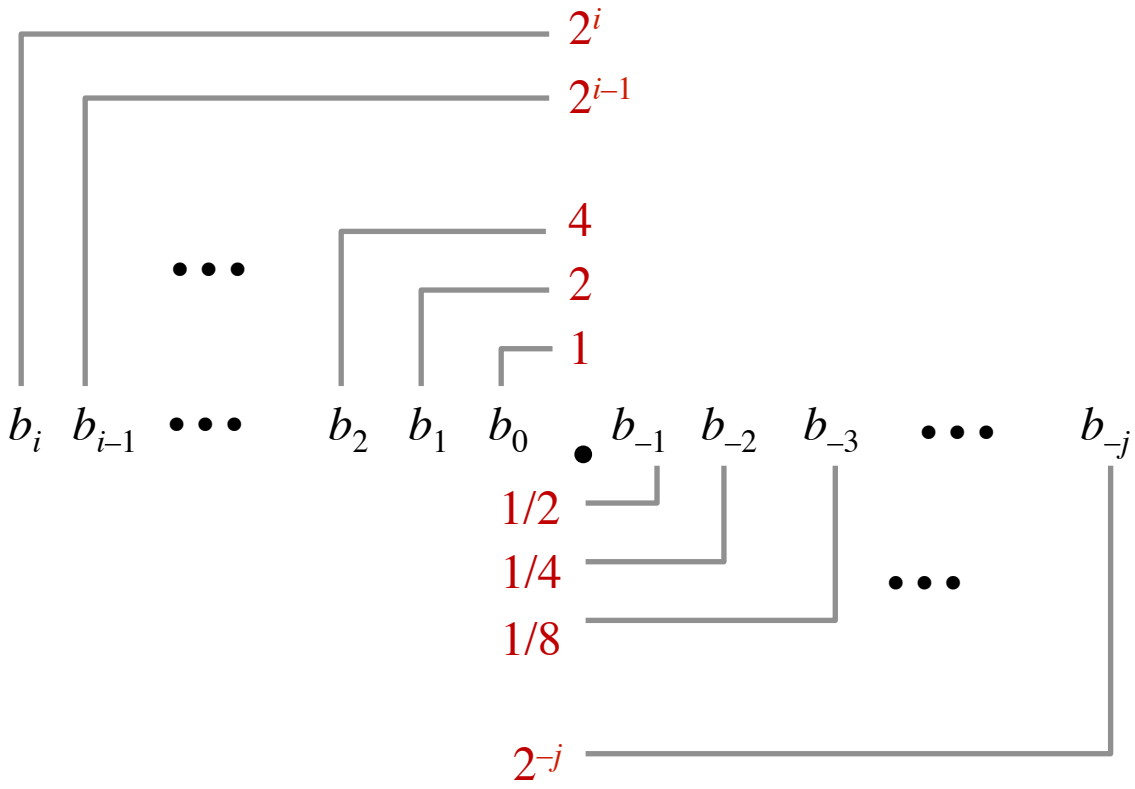
- Background: fractional binary numbers
- IEEE floating-point standard
- Floating-point operations and rounding
- Floating-point in C

Fractional Binary Numbers

- **What is 1011.101_2?**

- **How do we interpret fractional *decimal* numbers?**
  - e.g. 107.95_{10}
  - Can we interpret fractional binary numbers in an analogous way?

Fractional Binary Numbers



- **Representation**
  - Bits to right of "binary point" represent fractional powers of 2
  - Represents rational number: $\sum_{k=-j}^{i} b_k \cdot 2^k$

Fractional Binary Numbers: Examples

- **Value** **Representation**
  - 5 and 3/4 101.11_2
  - 2 and 7/8 10.111_2
  - 63/64 0.111111_2

- **Observations**
  - Divide by 2 by shifting right
  - Multiply by 2 by shifting left
  - Numbers of the form **0 . 111111**…$_2$ are just below 1.0
    - $1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$
    - Shorthand notation for all **1** bits to the right of binary point: $1.0 - \varepsilon$

Representable Values

- **Limitations of fractional binary numbers:**
  - Can only exactly represent numbers that can be written as $x * 2^y$
  - Other rational numbers have repeating bit representations

- **Value** **Representation**
  - 1/3 $0.0101010101[01]\ldots_2$
  - 1/5 $0.001100110011[0011]\ldots_2$
  - 1/10 $0.0001100110011[0011]\ldots_2$

Fixed Point Representation

- **We might try representing fractional binary numbers by picking a fixed place for an implied binary point**
  - "fixed point binary numbers"
- **Let's do that, using 8-bit fixed point numbers as an example**
  - #1: the binary point is between bits 2 and 3
    $b_7\ b_6\ b_5\ b_4\ b_3\ [.]\ b_2\ b_1\ b_0$
  - #2: the binary point is between bits 4 and 5
    $b_7\ b_6\ b_5\ [.]\ b_4\ b_3\ b_2\ b_1\ b_0$
- **The position of the binary point affects the *range* and *precision* of the representation**
  - range: difference between largest and smallest numbers possible
  - precision: smallest possible difference between any two numbers

Fixed Point Pros and Cons

- **Pros**
  - It's simple. The same hardware that does integer arithmetic can do fixed point arithmetic
    - In fact, the programmer can use ints with an implicit fixed point
    - ints are just fixed point numbers with the binary point to the right of b_0
- **Cons**
  - There is no good way to pick where the fixed point should be
    - Sometimes you need range, sometimes you need precision – the more you have of one, the less of the other.