Floating-point numbers

Fractional binary numbers
IEEE floating-point standard
Floating-point operations and rounding
Lessons for programmers

Many more details we will skip (it’s a 58-page standard...)
See CSAPP 2.4 for a little more

Fractional Binary Numbers

Value | Representation
--- | ---
5 and 3/4 | 101.11₂
2 and 7/8 | 10.11₁₂
47/64 | 0.10111₁₂

Observations
Shift left =
Shift right =
Numbers of the form 0.11111...₂ are...?

Limitations:
Exact representation possible only for numbers of the form x * 2^y,
where x and y are integers.
Other rationals have repeating bit representations
1/3 = 0.33333...₁₀ = 0.01010101[01]₁₀

Fixed-Point Representation

Implied binary point. Example:
b₁ b₂ b₃ b₄ b₅ [.] b₆ b₇ b₈

Same hardware as for integer arithmetic.
b₁ b₂ b₃ b₄ b₅ [ ]

Fixed point = fixed range and fixed precision
range: difference between largest and smallest representable numbers
precision: smallest difference between any two representable numbers
IEEE Floating Point

Analogous to scientific notation
12 000 000 1.2 x 10^7 1.2e7
0.000 001 2 1.2 x 10^-6 1.2e-6

IEEE Standard 754 used by all major CPUs today
IEEE = Institute of Electrical and Electronics Engineers

Driven by numerical concerns
Rounding, overflow, underflow
Numerically well-behaved, but hard to make fast in hardware

Floating Point Representation

Numerical form:
\[ V_{10} = (-1)^s \times M \times 2^E \]

Sign bit \( s \) determines whether number is negative or positive
Significand (mantissa) \( M \) normally a fractional value in range [1.0,2.0)
Exponent \( E \) weights value by a (possibly negative) power of two

Representation:
MSB \( s \) = sign bit \( s \)
exp field encodes \( E \) (but is not equal to \( E \))
frac field encodes \( M \) (but is not equal to \( M \))

Normalization and Special Values

\[ V = (-1)^s \times M \times 2^E \]

“Normalized” = \( M \) has the form 1.xxxxx
As in scientific notation
0.011 x 2^3 = 1.1 x 2^1, latter is more compact
Do not store the (guaranteed) leading 1.

Special values: (How do we represent 0.0? 0/0?)
zero: \( s = 0 \), exp == 00...0, frac == 00...0
+inf, -inf: exp == 11...1, frac == 00...0
1.0/0.0 = +inf, -1.0/0.0 = -inf, 1.0/-0.0 = -1.0, 0.0/0.0 = 0.0

NaN (“Not a Number”): exp == 11...1, frac != 00...0
sqrt(-1), \( \infty - \infty \), etc.

Denormalized/subnormal values (near 0.0) not covered here.
Floating Point Arithmetic

\[ V = (-1)^s \cdot M \cdot 2^E \]

```c
double x = ..., y = ...;
double z = x + y;
```

1. Compute exact result.
2. Round, to fit:
   - Overflow exponent if it is too wide for `exp`.
   - Drop LSBs of significand if it is too wide for `frac`.
   - Underflow if nearest representable value is 0.

Lessons for programmers

```c
V = (-1)^s \cdot M \cdot 2^E
```

- `float` is a real number ≠ `double`
- Rounding breaks associativity and other properties.

```c
double a = ..., b = ...;
...
if (a == b) ...
```

- `if (abs(a - b) < epsilon) ...`

More shortly...