MATH50003
Numerical Analysis

II.1 Reals

Dr Sheehan Olver

Part II: Representing Numbers

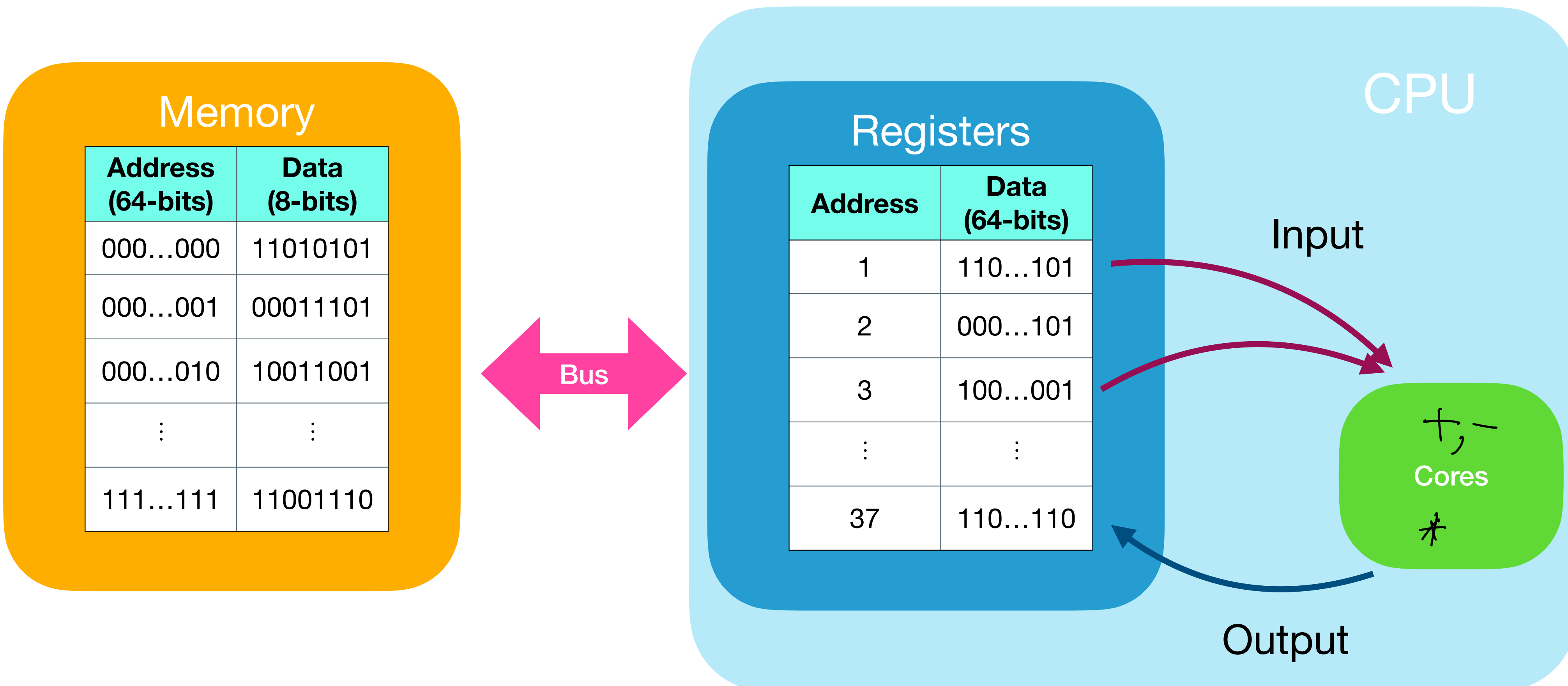
How do computers compute with numbers?

Why are there errors, eg. in divided differences?

Can we understand and bound these errors?

Simplified Model of a Computer

How do computers compute?



Mathematical model

CPUs work on *p*-bits at a time

Cores take (1x or 2x) *p*-bits
and return $\underbrace{p}_{=64}$-bits.

Operations are

$$\overbrace{f}^{-(x)} : \mathbb{Z}_{2^p} \to \mathbb{Z}_{2^p} \qquad \text{or}$$

$$\underbrace{f}_{x+y} : \mathbb{Z}_{2^p} \times \mathbb{Z}_{2^p} \to \mathbb{Z}_{2^p}$$

for $\mathbb{Z}_m := \{0, 1, \ldots, \underbrace{m}_{2^p = 2^{64}} - 1\}$

But how to handle ∞-cardinality sets integers/reals?

Limitations

- Memory is finite
- All operations work on *p*-bits at a time
- No such thing as throwing an error
- Any operation that manipulates more than *p*-bits must be a composition of simpler functions

Part II

Representing Numbers

1. Reals via floating point
2. Floating point arithmetic and bounding errors
3. Interval arithmetic for rigorous computations

Appendix B Integers

Ariane 5 rocket explosion

Learn floating point, or else…

II.1.1 Real numbers in binary

We can represent any real number using binary digits

bits

Definition 3 (binary format)**.** For $B_0, \ldots, B_p \in \{0, 1\}$ denote an integer in *binary format* by:

base -2

$$\pm(B_p \ldots B_1 B_0)_2 := \pm \sum_{k=0}^{p} B_k 2^k$$

Examples

B_0

$(1)_2 = 1 \cdot 2^0 = 1$

B_0

$(10)_2 = 1 \cdot 2^1 + 0 \cdot 2^0 = 2$

$-(11)_2 = -(1 \cdot 2^1 + 1 \cdot 2^0) = -3$

$(10001)_2 = \underbrace{2^4}_{16} + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + \underbrace{1 \cdot 2^0}_{1} = 17$

Definition 4 (real binary format)**.** For $b_1, b_2, \ldots \in \{0, 1\}$, Denote a non-negative real number in *binary format* by:

$$(B_p \ldots B_0.b_1 b_2 b_3 \ldots)_2 := (B_p \ldots B_0)_2 + \sum_{k=1}^{\infty} \frac{b_k}{2^k}.$$

$$(101.101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$$

$$= 5.625.$$

Example 3 (rational in binary)**.**

Consider

$$\frac{1}{3} = 0.333\_\_\_\_\_\_ = \sum_{k=1}^{\infty} \frac{3}{10^k}$$

Claim:

$$\frac{1}{3} = (0.010101\_\_\_\_\_\_)_2 = \sum_{k=1}^{\infty} \frac{1}{2^{2k}}$$

$1/4$ $1/16$ $1/64$

Use <u>Geometric Series</u>:

$$\sum_{k=0}^{\infty} z^k = \frac{1}{1-z} \qquad \text{for} \qquad |z|<1$$

Write $z = \frac{1}{4}$, so that

$$-1 + 1 + \sum_{k=1}^{\infty} \frac{1}{(2^2)^k} = \sum_{k=0}^{\infty} z^k - 1 = \frac{1}{1-\underset{1/4}{z}} - 1$$

$$= \frac{1}{3}$$

II.1.2 Floating-point numbers

How do we represent an uncountable set with only *p*-bits?

Bit Format: $\underbrace{s}_{\text{sign}}\ \underbrace{q_{Q-1}\dots q_0}_{\text{exponent bits}}\ \underbrace{b_1\dots b_S}_{\text{significand}}$

bits: $p = 1 + Q + S$

Definition 5 (floating-point numbers)**.** Given integers σ (the *exponential shift*), Q (the number of *exponent bits*) and S (the *precision*), define the set of *Floating-point numbers* by dividing into *normal*, *sub-normal*, and *special number* subsets:

$$F_{\sigma,Q,S} := \underbrace{F^{\text{normal}}_{\sigma,Q,S}}_{\subset \mathbb{R}\setminus\{0\}} \cup \underbrace{F^{\text{sub}}_{\sigma,Q,S}}_{\subset \mathbb{R}\cup\{-0\}} \cup \underbrace{F^{\text{special}}}_{\{-\infty,\infty,\text{NaN}\}}.$$

How do bits dictate whether its normal/sub/special?

Look at exponent. 3 examples:

Normal

0 10000 1010000000

3.125

Sub

$1\ \underbrace{00000}_{=0}\ 1100000000$

-3×2^{-16}

Special

$1\ \underbrace{11111}_{=1}\ 0000000000$

$-\infty$

for $F_{15,5,10}$.

II.1.3 IEEE float-point numbers

What exponent shift/number of bits/precision is used in practice?

Half (ML) $F_{16} := F_{15,5,10}$

Single (GPUs) $F_{32} := F_{127,8,23}$

Double (Standard) $F_{64} := F_{1023,11,52}$

DON'T MEMORISE

Half-precision
$F_{16} := F_{15,5,10}$

$$F^{\text{normal}}_{\sigma,Q,S} := \{\pm 2^{q-\sigma} \times (1.b_1 b_2 b_3 \ldots b_S)_2 : 1 \le q < 2^Q - 1\}.$$

is a 1 by default.

Example 4 (interpreting 16-bits as a float). Consider the number with bits

$$\underbrace{0}_{\substack{0 \text{ implies} \\ +}}\ \underbrace{\overset{q_4 - q_0}{10000}}_{\text{encode}}\ \underbrace{1010000000}_{b_1 b_2 - b_S}$$

$q = (10000)_2 = 2^4 = 16$

$\Rightarrow$ encodes

$+\ 2^{16-15} \times (1.1010000000)_2$

$= 2 \times \left(1 + \frac{1}{2} + \frac{1}{8}\right) = 3.25$

Example 5 (rational to 16-bits)**.** How is the number $1/3$ stored in F_{16}?

$$\frac{1}{3} = (0.0101\_\_\_\_\_\_)_2$$

$$= 2^{-2} \times (1.0101\_\_\_\_\_\_)_2$$

we want $= 1$

$$= 2^{13-15} \times (1.0101\_\_\_\_\_\_)_2 \quad \notin \quad F_{16}$$

$$\approx \quad + \; 2^{13-15} \times (1.0101010101)_2$$

round (more detail later)

Bits: $\quad 0 \quad \underbrace{01101} \quad 0101010101$

$(01101)_2 = 13$

II.1.4 Sub-normal and special numbers

Sub-normal have exponent bits all 0, special have all 1

If $q = (00000)_2 = 0$, this becomes 1, not q

$$F^{\text{sub}}_{\sigma,Q,S} := \{\pm 2^{1-\sigma} \times (0.b_1 b_2 b_3 \ldots b_S)_2\}.$$

0, not 1 anymore

Example 6 (subnormal in 16-bits)**.** Consider the number with bits

$$\underset{\text{means } -}{1}\ \overbrace{00000}^{\text{all } 0}\ \underbrace{1100000000}_{b_1 b_2 \text{ — } b_{10}}$$

encodes

$$-2^{1-15} \times (0.11)_2 = -2^{-14}\left(\frac{1}{2} + \frac{1}{4}\right)$$

$$= -3 \times 2^{-16}$$

$F^{\text{special}} := \{\infty, -\infty, \text{NaN}\}$

When $q = (11\cdots1)_2$ number is special

If all $b_k = 0$, $\pm\infty$

If any $b_k \neq 0$ NaN (LOTS OF NaNs!)

Example 7 (special in 16-bits)**.** The number with bits

$$\underbrace{1}_{-}\ \underbrace{11111}_{\text{all } 1}\ \underbrace{0000000000}_{\text{all } 0}$$

$-\infty$

On the other hand, the number with bits

$$1\ \underbrace{11111}_{\text{all } 1}\ \underbrace{0000000001}_{b_{10} \neq 0}$$

NaN

Time for code.