MATH50003
Numerical Analysis

II.1 Reals

Dr Sheehan Olver

Chapter II: Representing Numbers

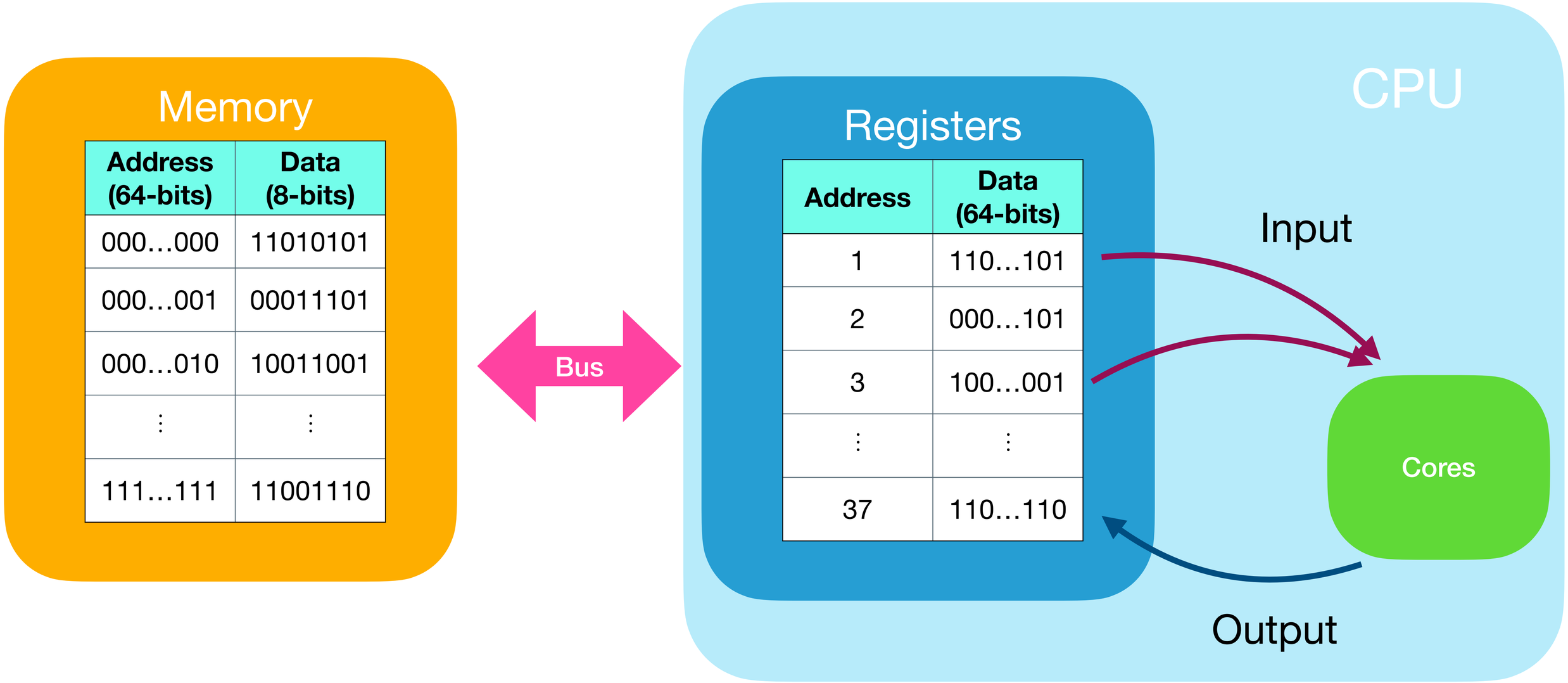
How do computers compute with numbers?

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 p-bits.

Operations are

$$f : \mathbb{Z}_{2^p} \to \mathbb{Z}_{2^p} \quad \text{or}$$
$$f : \mathbb{Z}_{2^p} \times \mathbb{Z}_{2^p} \to \mathbb{Z}_{2^p}$$

for $\mathbb{Z}_m := \{0,1,\ldots,m-1\}$

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

II.1.1 Real numbers in binary

We can represent any real number using binary digits

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

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

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

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

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

II.1.2 Floating-point numbers

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

Bit Format: $s \; q_{Q-1} \ldots q_0 \; b_1 \ldots b_S$

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:

$$F_{\sigma,Q,S} := F^{\text{normal}}_{\sigma,Q,S} \cup F^{\text{sub}}_{\sigma,Q,S} \cup F^{\text{special}}.$$

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

Look at exponent. 3 examples:

0 10000 1010000000

1 00000 1100000000

1 11111 0000000000

II.1.3 IEEE float-point numbers

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

$$F_{16} := F_{15,5,10}$$

$$F_{32} := F_{127,8,23}$$

$$F_{64} := F_{1023,11,52}$$

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

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

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

0 10000 1010000000

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

II.1.4 Sub-normal and special numbers

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

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

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

1 00000 1100000000

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

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

1 11111 0000000000

On the other hand, the number with bits

1 11111 0000000001

Time for code.