Floating Point: Implementation, Normalization

Previously: Consider *mathematical* view of FP. (via example: $(1.101)_2$)

Next: Consider *implementation* of FP in hardware.

Do you notice a source of inefficiency in our number representation?

$$(1101)_2 = \underbrace{(\underbrace{1}_{\text{leading 1: not stored}}.\underbrace{101}_{\text{fraction}})_2}_{\text{significand}} \cdot 2^{\overbrace{3}^{\text{exponent}}}$$

$$3 = -1023 + \underbrace{1026}_{\text{stored bit pattern for the exponent}}$$

In exam, go to:
relate.cs.illinois.edu

Unrepresentable numbers?

Can you think of a somewhat central number that we cannot represent as

$$x = (1.\_\_\_\_\_\_\_\_\_\_\_)_2 \cdot 2^{-p}?$$

zero

"=" 2^{-1075}

use special exponent (at bottom of range by convention) to mean:

- implicit leading 1 turned off
- "intended" exponent stays at lowest +1

Demo: Picking apart a floating point number [cleared]

Subnormal Numbers

What is the smallest representable number in an FP system with 4 stored bits in the significand and a stored exponent range of $[-7, 8]$?

Smallest repr. <u>normal</u> number:

$$(1.\underline{0000}) \cdot 2^{-7}$$

Subnormal Numbers, Attempt 2

What is the smallest representable number in an FP system with 4 stored bits in the significand and a (stored) exponent range of $[-7, 8]$?

−7 : special exponent

↳ means −6

$(\underline{0}.0001 \cdot 2^{-6})$

Why learn about subnormals?

− subnormals done w/ software fallback

→ slow

Underflow

- FP systems without subnormals will *underflow* (return 0) as soon as the exponent range is exhausted.
- This smallest representable *normal* number is called the *underflow level*, or *UFL*.
- Beyond the underflow level, subnormals provide for *gradual underflow* by 'keeping going' as long as there are bits in the significand, but it is important to note that subnormals don't have as many accurate digits as normal numbers.
  Read a story on the epic battle about gradual underflow
- Analogously (but much more simply–no 'supernormals'): the overflow level, *OFL*.

Rounding Modes

How is rounding performed? (Imagine trying to represent π.)

$$(\underbrace{1.1101010}_{\text{representable}}11)_2$$

- chop ("r. to 0")
- round to $\pm\infty$
- round to nearest

Chop: $(1.1101010)_2$

RtN: $(1.1101011)_2$

What is done in case of a tie? $0.5 = (0.1)_2$ ("Nearest"?)

Round to even:

$0.5 \to 0$ $1.5 \to 2$

Demo: Density of Floating Point Numbers [cleared]

Demo: Floating Point vs Program Logic [cleared]

Smallest Numbers Above...

- What is smallest FP number > 1? Assume 4 bits in the significand.

$$(1.0001)_2 = x_i(1+\varepsilon)$$

What's the smallest FP number > 1024 in that same system?

$$((1.0001)_2 \cdot 2^{10} = x_i(1+\varepsilon)$$

Can we give that number a name?

Unit Roundoff

Unit roundoff or *machine precision* or *machine epsilon* or $\varepsilon_{\text{mach}}$ is the smallest number such that

$$\text{float}(1+\varepsilon) > 1.$$

- Technically that makes $\varepsilon_{\text{mach}}$ depend on the rounding rule.

  Assuming round-towards-infinity, in the above system, $\varepsilon_{\text{mach}} = (0.00001)_2$.
- Note the extra zero.
- Another, related, quantity is *ULP*, or *unit in the last place*. ($\varepsilon_{\text{mach}} = 0.5\,\text{ULP}$)

FP: Relative Rounding Error

What does this say about the relative error incurred in floating point calculations?