(IEEE) Floating-point representation

MATH 6610 Lecture 05

September 11, 2020

Finite representation of numbers

Numbers in decimal are represented as

$$34.1503,$$

Finite representation of numbers

Numbers in decimal are represented as

$$34.1503,$$

This actually means the more complicated expression

$$3 \times 10^{1} \;+\; 4 \times 10^{0} \;+\; 1 \times 10^{-1} \;+\; 5 \times 10^{-2} \;+\; 0 \times 10^{-3} \;+\; 3 \times 10^{-4}$$

Finite representation of numbers

Numbers in decimal are represented as

$$34.1503,$$

This actually means the more complicated expression

$$3 \times 10^{1} \ + \ 4 \times 10^{0} \ + \ 1 \times 10^{-1} \ + \ 5 \times 10^{-2} \ + \ 0 \times 10^{-3} \ + \ 3 \times 10^{-4}$$

The portion 34 is the "integer" part, and 1503 is the "fractional" part.

They are separated by the radix (point).

The base 10 is to be understood (implied) by context.

Finite representation of numbers

Numbers in decimal are represented as

$$34.1503,$$

This actually means the more complicated expression

$$3 \times 10^{1} \; + \; 4 \times 10^{0} \; + \; 1 \times 10^{-1} \; + \; 5 \times 10^{-2} \; + \; 0 \times 10^{-3} \; + \; 3 \times 10^{-4}$$

The portion 34 is the "integer" part, and 1503 is the "fractional" part.

They are separated by the radix (point).

The base 10 is to be understood (implied) by context.

Without context, the base b can be other positive integers:

$$b = 6 \quad \Rightarrow \quad 3 \times 6^{1} \; + \; 4 \times 6^{0} \; + \; 1 \times 6^{-1} \; + \; 5 \times 6^{-2} \; + \; 0 \times 6^{-3} \; + \; 3 \times 6^{-4}$$

Computer representations

Circuits typically detect the presence (1) or absence (0) of an electrical signal.

For this reason, base $b = 2$ is the standard computer format, e.g.,:

$$100010.0010011001111$$

is approximately equal to the decimal 34.1503.

Computer representations

Circuits typically detect the presence (1) or absence (0) of an electrical signal.

For this reason, base $b = 2$ is the standard computer format, e.g.,:

$$100010.0010011001111$$

is approximately equal to the decimal 34.1503.

Computers store this *binary* representation of these numbers, and each digit is a "bit".

8 bits = 1 "byte"

Computer representations

Circuits typically detect the presence (1) or absence (0) of an electrical signal.

For this reason, base $b = 2$ is the standard computer format, e.g.,:

$$100010.0010011001111$$

is approximately equal to the decimal 34.1503.

Computers store this *binary* representation of these numbers, and each digit is a "bit".

8 bits = 1 "byte"

Binary representations must store the integer part, the fractional part, and typically also a sign.

Fixed point vs floating point

Fixed-point representations have a fixed radix point, with the size of the fractional part predetermined:

$$100010.\quad \underbrace{0010011001111}_{\text{fixed number of entries}}$$

Then the fractional precision of this representation is fixed for any number.

This *truncation* of finite representations is one of the main challenges to address in numerical computations.

Fixed point vs floating point

Fixed-point representations have a fixed radix point, with the size of the fractional part predetermined:

$$100010.\quad \underbrace{0010011001111}_{\text{fixed number of entries}}$$

Then the fractional precision of this representation is fixed for any number.

This *truncation* of finite representations is one of the main challenges to address in numerical computations.

Fixed-point representation has a restriced range of precision (pre)defined by the size of the fractional part.

Fixed point vs floating point

Fixed-point representations have a fixed radix point, with the size of the fractional part predetermined:

$$100010.\quad \underbrace{0010011001111}_{\text{fixed number of entries}}$$

Then the fractional precision of this representation is fixed for any number.

This *truncation* of finite representations is one of the main challenges to address in numerical computations.

Fixed-point representation has a restriced range of precision (pre)defined by the size of the fractional part.

Floating-point representations allow the radix to float:

$$1.\underbrace{0001000100110011111}_{\text{fixed number of entries}} \quad + \quad \text{exponent}$$

The exponent encodes which exponent the radix is aligned with. (Above: 5)

Floating-point representations

Generally speaking, floating-point representations store:

$$\text{significand} + \text{exponent} + \text{sign}$$

The significand combines the integer and fractional parts of the number.

The exponent encodes the location of the radix.

Floating-point representations allows for a (much) larger operating range than fixed-point representations.

Floating-point representations

Generally speaking, floating-point representations store:

$$\text{significand} + \text{exponent} + \text{sign}$$

The significand combines the integer and fractional parts of the number.

The exponent encodes the location of the radix.

Floating-point representations allows for a (much) larger operating range than fixed-point representations.

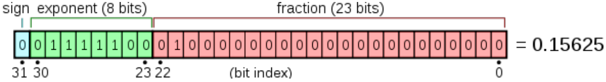
The most popular representation is the IEEE 754 standard, defining various formats:

- Binary 16
- Binary 32 ("single precision")
- Binary 64 ("double precision")
- $\vdots$

Most high-level scientific computing languages use double precision as default.

Floating-point details

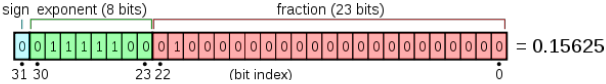
These standards define bit allocation:



Source: `https://en.wikipedia.org/wiki/File:Float_example.svg`

Floating-point details

These standards define bit allocation:



Source: `https://en.wikipedia.org/wiki/File:Float_example.svg` The number of bits allocated to the exponent indicates the rounding precision of the format.

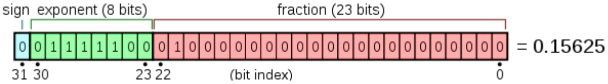
Machine precision or machine epsilon is the maximum relative rounding error due to finite representation,

$$\left| \frac{x - \mathrm{fl}(x)}{x} \right|.$$

Roughly speaking, machine epsilon is also the largest ϵ such that $1 + \epsilon$ is rounded to 1.

Floating-point details

These standards define bit allocation:



Source: `https://en.wikipedia.org/wiki/File:Float_example.svg` The number of bits allocated to the exponent indicates the rounding precision of the format.

Machine precision or machine epsilon is the maximum relative rounding error due to finite representation,

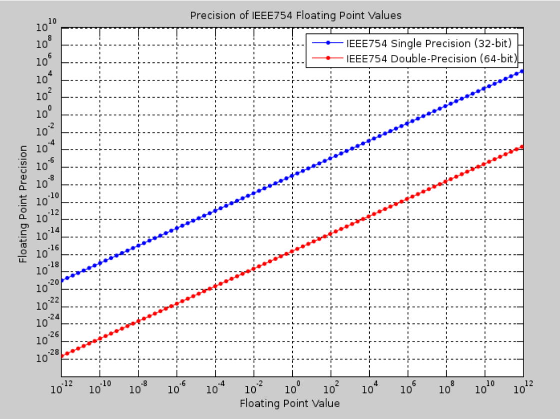
$$\left| \frac{x - \mathrm{fl}(x)}{x} \right|.$$

Roughly speaking, machine epsilon is also the largest ϵ such that $1 + \epsilon$ is rounded to 1.

The roundoff or truncation error associated with a floating-point format is essentially equal to $\mathrm{base}^{-\#(\text{exponent digits})}$.

Machine precision

Floating-point representation ensures that there is not really an absolute error committed by computer representations, there is only a relative error.



Source: `https://en.wikipedia.org/wiki/File:IEEE754.svg`

Roundoff and computing

The rounding truncation imparted by finite representations requires attention to how algorithms are implemented. E.g.,

- Implementation of the formula $\sqrt{1+x^4}-1$ for small positive x.
- Evaluation of e^x for $x < 0$.
- Evaluation of $\frac{f(x+h)-f(x)}{h}$ for small h.