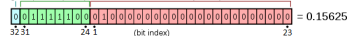
Algorithms and Data Structures for DS

Encoding
Machine numbers for $\mathbb{R}$



Learning goals

- IEEE 745
- Types in `C`
- Floating point numbers in $\mathbb{R}$
- Distance between numbers and machine epsilon
- Associative and distributive properties

REALS ON A MACHINE

- Floating point numbers on a machine are approximations, not exact representations of $\mathbb{R}$
- Because there is only a finite number of machine numbers, there are no arbitrarily small or large numbers and also no arbitrarily close numbers
- A finite subset of the real numbers cannot be closed w.r.t. rational operations $(+, -, *, /)$
- We seek operations similar to those in $\mathbb{R}$, with fast and easy implementation on digital computers

IEEE 754

- IEEE (**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers) 754 defines standard representations for floating point numbers in computers
- Implemented pretty much everywhere
- Inspired by scientific notation 1.57E-6
- Uses a sign, mantissa and exponent
- Base b, we only discuss $b = 2$, nearly always used

IEEE 754 - CONTENTS

- Number format, including special cases like NaN
- Interchange formats: Encodings (bit strings) that may be used to exchange floating-point data in efficient and compact form
- Rounding rules during arithmetic and conversions
- Operations: arithmetic and other operations (such as trigonometric functions) on arithmetic formats
- Exception handling: indications of exceptional conditions (such as division by zero, overflow, etc.)

ARIANE FLIGHT V88 – FROM WIKIPEDIA

- *Ariane flight V88 was the failed maiden flight of the Arianespace Ariane 5 rocket, vehicle no. 501, on 4 June 1996. It carried the Cluster spacecraft, a constellation of four European Space Agency research satellites. The launch ended in failure due to multiple errors in the software design: dead code, intended only for Ariane 4, with inadequate protection against integer overflow led to an exception handled inappropriately, halting the whole otherwise unaffected inertial navigation system. This caused the rocket to veer off its flight path 37 seconds after launch, beginning to disintegrate under high aerodynamic forces, and finally self-destructing via its automated flight termination system. The failure has become known as one of the most infamous and expensive software bugs in history. The failure resulted in a loss of more than USD 370 million.*

- *The greater values of BH caused a data conversion from a 64-bit floating point number to a 16-bit signed integer value to overflow and cause a hardware exception. The programmers had protected only four out of seven critical variables against overflow*

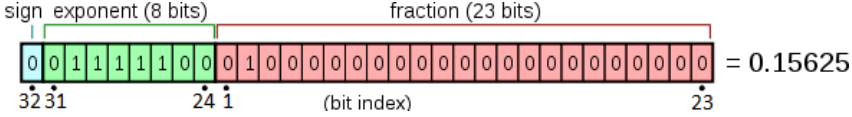
- *As noted by Kahan, the unhandled trap consecutive to a floating-point to 16-bit integer conversion overflow that caused the loss of an Ariane 5 rocket would not have happened under the default IEEE 754 floating-point policy.*

FORMAT FOR SINGLE / DOUBLE PRECISION

- Sign bit $S \in \{-1, +1\}$
- Mantissa of length m, the significant bits / digits
- Exponent is bias-shifted for negative numbers
- Representation:

$$x = S \cdot b^{e-bias} \cdot (1 + \sum_{i=1}^{m} u_i)$$

- NB: Can save one bit / have it implicit, because in binary numbers 1.001001×2^e first digit before dot must be a 1
- Single precision = 32 bit with 8 bit exponent and bias 127, m=23 mantissa bits

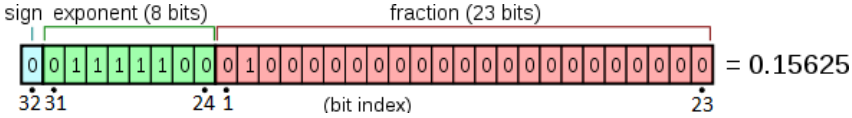


Source: ▸ Wikipedia 2024

FORMAT FOR SINGLE / DOUBLE PRECISION

- Representation:

$$x = S \cdot b^{e-bias} \cdot (1 + \sum_{i=1}^{m} u_i)$$



Source: ▸ Wikipedia 2024

- Sign bit $u_{32} = 0 \Rightarrow S = +1$
- $e = \sum_{i=1}^{8} u_{i+23} 2^{i-1} - 127 = 2^2 + 2^3 + 2^4 + 2^5 + 2^6 - 127 = -3$
- $x = S \cdot b^e \cdot (1 + \sum_{i=1}^{23} u_i b^{-i}) = 1 \cdot 2^{-3} \cdot (1 + 2^{-2}) = 0.15625$
- Double precision uses 64bit, 52 bit mantissa and 11 bit exponent with bias 1023

WHY EXCESS CODING IN EXPONENT?

- Optimized for fast and simple comparisons
- Addition is complicated anyway
- Usually compare exponents first, before fraction
- Exponents should be ordered correctly, even for negative values
- Would be more complicated two's complement
- Then binary comparison wouldn't match numerical ordering, because of the most-significant bit

IEEE 754 - SPECIAL CASES

- **0**: If all mantissa bits and all exponent bits are 0, then $x = \pm 0$; pos. and neg. zeros are used when rounding to 0
- ∞: If all mantissa bits are 0 and all exponent bits are 1, then $x = \pm\infty$; results from division by 0, or if the result is too large or too small
- **NaN**: If all exponent bits are 1 and at least one mantissa bit is 1, then $x =$ NaN ("Not a Number"). results from $0/0$ or $\infty - \infty$

IEEE 754 - DENORMALIZED NUMBERS

- Remember our formula with the hidden bit

$$x = S \cdot b^e \cdot (1 + \sum_{i=1}^{m} u_i b^{-i})$$

- This usage of the hidden bit is called a “normalized number”
- A number is considered normalized if at least one exponent bit is 1
- If all exponent bits are 0, the number is considered **denormalized**

$$x = S \cdot b^{e_{\min}} \cdot \left(\sum_{i=1}^{m} u_i b^{-i} \right)$$

- This allows even smaller numbers close to 0, as we have a 0 before the dot and the option to put many 0s after it

IEEE 754 RANGES OVERVIEW

Format	Significand (bits)	Exponent bits	Exponent range (bias)
Half (16-bit)	11 (10+1)	5	$[-14, +15]$ (bias 15)
Single (32-bit)	24 (23+1)	8	$[-126, +127]$ (bias 127)
Double (64-bit)	53 (52+1)	11	$[-1022, +1023]$ (bias 1023)
Quad (128-bit)	113 (112+1)	15	$[-16382, +16383]$ (bias 16383)

Format	Min positive (subnorm)	Min positive (norm)	Max finite
Half (16-bit)	5.96×10^{-8}	6.10×10^{-5}	6.55×10^{4}
Single (32-bit)	1.40×10^{-45}	1.18×10^{-38}	3.40×10^{38}
Double (64-bit)	4.94×10^{-324}	2.23×10^{-308}	1.80×10^{308}
Quad (128-bit)	6.48×10^{-4966}	3.36×10^{-4932}	1.19×10^{4932}

FLOATS IN C

- Type table:

Type	Explanation
float	Single-precision floating-point type. Usually IEEE 754 (32 bits)
double	Double-precision floating-point type. Usually IEEE 754 (64 bits)
long double	Extended precision floating-point type. Can be 80, 96, 128, or 256 bits depending on the system

- Compiler translates them for CPU
- Standard machines provide hardware support for single and double precision floating-point arithmetic

FLOATS IN R

- Data type `numeric` for vectors of reals, no scalar reals
- Always uses in double precision, 64 bit
- Caution: C code in libraries could do something different
- `.Machine` contains info about the encoding

```
.Machine$double.base      # base
## [1] 2

.Machine$double.digits    # number of mantissa bits
## [1] 53

.Machine$double.exponent  # number of exponent bits
## [1] 11

.Machine$double.xmin      # smallest float
## [1] 2.225074e-308

.Machine$double.xmax      # largest float
## [1] 1.797693e+308
```

FLOATS IN R

- Let's check this again

```
0.1 + 0.2 == 0.3
## [1] FALSE
```

- Neither of the numbers can be represented exactly:

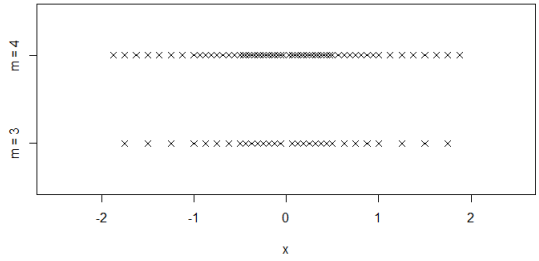
```
sprintf("%.20f", 0.1)    # decimal notation (20 digits)
## [1] "0.10000000000000000555"

sprintf("%.20f", 0.2)
## [1] "0.20000000000000001110"

sprintf("%.20f", 0.1 + 0.2)
## [1] "0.30000000000000004441"
```

- Can avoid problem by using comparison **with tolerance** `all.equal`

```
all.equal(0.1 + 0.2, 0.3)
## [1] TRUE
```

DISTANCES BETWEEN MACHINE FLOATS

- Machine floats $\mathcal{M}$ are **not uniformly distributed**
- Interval $[b^{i-1}, b^i]$ contains the same number of values as $[b^i, b^{i+1}]$
- Despite the latter being b times larger



- The smallest numbers around 0 are $\pm b^{e_{\min}-m}$ – denormalized
- For numbers greater than b^{m+1}, the distance between numbers is > 1 and even integer parts are no longer exact

MACHINE EPSILON

- Define machine eps as the smallest ϵ so: float$(1 + \epsilon) \neq 1$
- The smallest number greater than 1 is $1 + b^{-m}$
- So we define $\epsilon = b^{-m}$
- For double precision this is $\epsilon = 2^{-52}$
- **Relative distance** between machine numbers is approximately ϵ, and **absolute distance** is $|x| \cdot \epsilon$
- This machine epsilon is our minimal accuracy, and an upper bound on the relative rounding error we make, when numbers are handled on a machine

MACHINE EPSILON

- For double prec, our precision is about 16 decimal digits

```
options(digits = 20)
1 + 1 / (2^53)
## [1] 1

1 + 1 / (2^52)
## [1] 1.0000000000000002

1 / (2^52)
## [1] 2.2204460492503131e-16

.Machine$double.eps
## [1] 2.2204460492503131e-16
```

PECULIARITIES OF MACHINE ARITHMETIC

- Common arithmetic properties are no longer fulfilled.
- For simplicity, we use rounded decimal representation with $m = 4$
- **Associative property:**

$$a = 4, b = 5003, c = 5000 \quad \Rightarrow$$
$$a = 0.4 \cdot 10^1, b = 0.5003 \cdot 10^4, c = 0.5 \cdot 10^4$$

$$\begin{aligned}
(\tilde{a} + \tilde{b}) &= 0.4 \cdot 10^1 + 0.5003 \cdot 10^4 = 0.5007 \cdot 10^4 \\
(\tilde{a} + \tilde{b}) + \tilde{c} &= 0.5007 \cdot 10^4 + 0.5 \cdot 10^4 = 1.0007 \cdot 10^4 \\
&\approx 0.1001 \cdot 10^5 = 10010
\end{aligned}$$

$$\begin{aligned}
(\tilde{b} + \tilde{c}) &= 0.5003 \cdot 10^4 + 0.5 \cdot 10^4 = 1.0003 \cdot 10^4 \\
&\approx 0.1000 \cdot 10^5 \\
(\tilde{b} + \tilde{c}) + \tilde{a} &= 0.1000 \cdot 10^5 + 0.4 \cdot 10^1 = 0.10004 \cdot 10^5 \\
&\approx 0.1000 \cdot 10^5 = 10000
\end{aligned}$$

PECULIARITIES OF MACHINE ARITHMETIC

- **Distributive property:**

$$\begin{aligned} 2 \cdot (\tilde{b} - \tilde{c}) &= 2 \cdot (0.5003 \cdot 10^4 - 0.5 \cdot 10^4) \\ &= 0.0006 \cdot 10^4 = 6 \end{aligned}$$

$$\begin{aligned} (2 \cdot \tilde{b} - 2 \cdot \tilde{c}) &= 2 \cdot 0.5003 \cdot 10^4 - 2 \cdot 0.5 \cdot 10^4 \\ &= 1.0006 \cdot 10^4 - 1 \cdot 10^4 \\ &\approx 0.1001 \cdot 10^5 - 0.1 \cdot 10^5 = 0.0001 \cdot 10^5 = 10 \end{aligned}$$

Problem in the second example: catastrophic cancellation.

EXAMPLES

```
1e16 - 1e16
## [1] 0
```

```
(1e16 + 1) - 1e16
## [1] 0
```

```
(1e16 + 2) - 1e16
## [1] 2
```

$1e16$ cannot be represented exactly since it is larger than 2^{53}, hence the distance is greater than 1.

EXAMPLES

```
x = seq(1, 2e16, length = 100000)
s1 = sum(x)
s2 = sum(rev(x))
s1
## [1] 1e+21
```

```
s2
## [1] 1e+21
## [1] 1e+21
```

```
s1 - s2
## [1] -262144
```