Computing Unit 2: Numbers

Kurt Hornik

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION OF MBAS ACCREDITED

Outline

- Integers
- Doubles

Integers

How can we store integers using bits?

Integers

How can we store integers using bits?

E.g., using $k = 3$ bits, can do bit sequences

$$\texttt{000}, \texttt{001}, \ldots, \texttt{111}$$

Integers

How can we store integers using bits?

E.g., using $k = 3$ bits, can do bit sequences

$\texttt{000}, \texttt{001}, \ldots, \texttt{111}$

There are $2 \cdot 2 \cdot 2 = 2^3 = 8$ different such sequences.

Integers

How can we store integers using bits?

E.g., using $k = 3$ bits, can do bit sequences

`000`, `001`, . . . , `111`

There are $2 \cdot 2 \cdot 2 = 2^3 = 8$ different such sequences.

For general k, there are 2^k such sequences.

In R, $k = 32$ bits (4 bytes) are used for integers.

Integers

Which numbers should these bit sequences correspond to?

Integers

Which numbers should these bit sequences correspond to?

Obvious idea: the numbers with binary representation given by the respective bit sequences. I.e.,

$$\begin{array}{lrcl} \texttt{000}: & 0 * 2^2 + 0 * 2^1 + 0 * 2^0 & = & 0 \\ \texttt{001}: & 0 * 2^2 + 0 * 2^1 + 1 * 2^0 & = & 1 \\ \vdots & \vdots & & \\ \texttt{111}: & 1 * 2^2 + 1 * 2^1 + 1 * 2^0 & = & 7 \end{array}$$

Integers

Which numbers should these bit sequences correspond to?

Obvious idea: the numbers with binary representation given by the respective bit sequences. I.e.,

$$
\begin{array}{lcl}
\texttt{000}: & 0 * 2^2 + 0 * 2^1 + 0 * 2^0 & = & 0 \\
\texttt{001}: & 0 * 2^2 + 0 * 2^1 + 1 * 2^0 & = & 1 \\
\vdots & \vdots & & \\
\texttt{111}: & 1 * 2^2 + 1 * 2^1 + 1 * 2^0 & = & 7
\end{array}
$$

This would give the 8 numbers from 0 to 7.

Integers

Which numbers should these bit sequences correspond to?

Obvious idea: the numbers with binary representation given by the respective bit sequences. I.e.,

$$\begin{aligned}
\texttt{000}: &\quad 0 * 2^2 + 0 * 2^1 + 0 * 2^0 &= &\quad 0 \\
\texttt{001}: &\quad 0 * 2^2 + 0 * 2^1 + 1 * 2^0 &= &\quad 1 \\
\vdots &\qquad\qquad\quad \vdots & & \\
\texttt{111}: &\quad 1 * 2^2 + 1 * 2^1 + 1 * 2^0 &= &\quad 7
\end{aligned}$$

This would give the 8 numbers from 0 to 7.

But what about *negative* integers?

Integers

Which numbers should these bit sequences correspond to?

Obvious idea: the numbers with binary representation given by the respective bit sequences. I.e.,

$$\begin{array}{lcl} \texttt{000}: & 0 * 2^2 + 0 * 2^1 + 0 * 2^0 & = & 0 \\ \texttt{001}: & 0 * 2^2 + 0 * 2^1 + 1 * 2^0 & = & 1 \\ \vdots & \vdots & & \\ \texttt{111}: & 1 * 2^2 + 1 * 2^1 + 1 * 2^0 & = & 7 \end{array}$$

This would give the 8 numbers from 0 to 7.

But what about *negative* integers?

Three possibilities: (a) sign and magnitude, (b) bias, (c) two’s complement.

Sign and magnitude

See also https://en.wikipedia.org/wiki/Signed_number_representations#Signed_magnitude_representation.

Take one bit for the sign, and the rest as before.

Sign and magnitude

See also https://en.wikipedia.org/wiki/Signed_number_representations#Signed_magnitude_representation.

Take one bit for the sign, and the rest as before.

E.g., using 3 bits:

$$\sigma\beta_1\beta_0 \longleftrightarrow \pm(\beta_1 * 2^1 + \beta_0 * 2^0).$$

This would give the numbers

$$-3, -2, -1, -0, 0, 1, 2, 3$$

Sign and magnitude

See also `https://en.wikipedia.org/wiki/Signed_number_representations#Signed_magnitude_representation`.

Take one bit for the sign, and the rest as before.

E.g., using 3 bits:

$$\sigma\beta_1\beta_0 \longleftrightarrow \pm(\beta_1 * 2^1 + \beta_0 * 2^0).$$

This would give the numbers

$$-3, -2, -1, -0, 0, 1, 2, 3$$

Note: there are two ways to represent 0!

Sign and magnitude

For general k:

$$\sigma\beta_{k-2}\cdots\beta_0 \longleftrightarrow \pm\sum_{i=0}^{k-2}\beta_i 2^i.$$

Why $k-2$?

Sign and magnitude

For general k:

$$\sigma\beta_{k-2}\cdots\beta_0 \longleftrightarrow \pm\sum_{i=0}^{k-2}\beta_i 2^i.$$

Why $k-2$?

Can do 2^k-1 different numbers: $2^{k-1}-1$ positive and negative ones each, and zero in two different ways. Check:

$$(2^{k-1}-1)+(2^{k-1}-1)+1 = 2(2^{k-1}-1)+1 = 2^k-2+1 = 2^k-1.$$

Sign and magnitude

For general k:

$$\sigma\beta_{k-2}\cdots\beta_0 \longleftrightarrow \pm\sum_{i=0}^{k-2}\beta_i 2^i.$$

Why $k-2$?

Can do 2^k-1 different numbers: $2^{k-1}-1$ positive and negative ones each, and zero in two different ways. Check:

$$(2^{k-1}-1)+(2^{k-1}-1)+1 = 2(2^{k-1}-1)+1 = 2^k-2+1 = 2^k-1.$$

Simple, but not used “in practice”.

Biased scheme

See also `https://en.wikipedia.org/wiki/Offset_binary`.

Idea: “put zero in the middle”.

But how? An even number of numbers has no middle!

Biased scheme

See also `https://en.wikipedia.org/wiki/Offset_binary`.

Idea: "put zero in the middle".

But how? An even number of numbers has no middle!

In our case with $k = 3$, biasing by 3 gives

$$(0 - 3) = -3, (1 - 3) = -2, \ldots, (7 - 3) = 4.$$

Biased scheme

See also `https://en.wikipedia.org/wiki/Offset_binary`.

Idea: “put zero in the middle”.

But how? An even number of numbers has no middle!

In our case with $k = 3$, biasing by 3 gives

$$(0-3) = -3, (1-3) = -2, \ldots, (7-3) = 4.$$

Numbers are taken as

$$\beta_2\beta_1\beta_0 \longleftrightarrow \sum_{i=0}^{2} \beta_i 2^i - 3,$$

where $3 = 2^2 - 1 = 2^{k-1} - 1$.

Biased scheme

For general k: bias by $2^{k-1}-1$, and take numbers as

$$\beta_{k-1}\cdots\beta_0 \longleftrightarrow \sum_{i=0}^{k-1}\beta_i 2^i - (2^{k-1}-1).$$

Biased scheme

For general k: bias by $2^{k-1} - 1$, and take numbers as

$$\beta_{k-1} \cdots \beta_0 \longleftrightarrow \sum_{i=0}^{k-1} \beta_i 2^i - (2^{k-1} - 1).$$

The smallest such number is

$$0 - (2^{k-1} - 1) = -(2^{k-1} - 1).$$

Biased scheme

For general k: bias by $2^{k-1} - 1$, and take numbers as

$$\beta_{k-1} \cdots \beta_0 \longleftrightarrow \sum_{i=0}^{k-1} \beta_i 2^i - (2^{k-1} - 1).$$

The smallest such number is

$$0 - (2^{k-1} - 1) = -(2^{k-1} - 1).$$

The largest such number is

$$(2^k - 1) - (2^{k-1} - 1) = 2^{k-1}(2 - 1) = 2^{k-1}.$$

Biased scheme

We need this later for specifically for $k = 11$.

Biased scheme

We need this later for specifically for $k = 11$.

There are $2^{11} = 2048$ different bit sequences, “initially” corresponding to $0, \ldots, 2047$.

Biased scheme

We need this later for specifically for $k = 11$.

There are $2^{11} = 2048$ different bit sequences, “initially” corresponding to $0, \ldots, 2047$.

Biasing by $2^{10} - 1 = 1023$ these become

$$(0 - 1023) = -1023, \ldots, (2047 - 1023) = 1024.$$

Two’s complement scheme

See alse https://en.wikipedia.org/wiki/Two%27s_complement.

Interpret bit sequences of length k as remainder classes modulo 2^k.

Two's complement scheme

See alse `https://en.wikipedia.org/wiki/Two%27s_complement`.

Interpret bit sequences of length k as remainder classes modulo 2^k.

In our case with $k = 3$ so that $2^3 = 8$: when doing integer division by 8, a remainder of 7 is equivalent to a remainer of -1. (If we add 1 to 7, we get 8, and no remainder.)

Two's complement scheme

See alse `https://en.wikipedia.org/wiki/Two%27s_complement`.

Interpret bit sequences of length k as remainder classes modulo 2^k.

In our case with $k = 3$ so that $2^3 = 8$: when doing integer division by 8, a remainder of 7 is equivalent to a remainer of -1. (If we add 1 to 7, we get 8, and no remainder.)

So we can do

$$0, 1, 2, 3, 4 \leftrightarrow -4, 5 \leftrightarrow -3, 6 \leftrightarrow -2, 7 \leftrightarrow -1.$$

Two’s complement scheme

See alse `https://en.wikipedia.org/wiki/Two%27s_complement`.

Interpret bit sequences of length k as remainder classes modulo 2^k.

In our case with $k = 3$ so that $2^3 = 8$: when doing integer division by 8, a remainder of 7 is equivalent to a remainer of -1. (If we add 1 to 7, we get 8, and no remainder.)

So we can do

$$0, 1, 2, 3, 4 \leftrightarrow -4, 5 \leftrightarrow -3, 6 \leftrightarrow -2, 7 \leftrightarrow -1.$$

The corresponding bit sequences are:

`000`, `001`, `010`, `011`, `100`, `101`, `110`, `111`.

So all sequences with the *highest bit on* are taken as the negative of their “two’s complement”.

Two's complement scheme

Note that for the bit sequences with the highest bit on, the remaining bits correspond to the numbers 0, 1, 2 and 3, which we take as -4, -3, -2, and -1: i.e., from which we subtract 4!

Two's complement scheme

Note that for the bit sequences with the highest bit on, the remaining bits correspond to the numbers 0, 1, 2 and 3, which we take as -4, -3, -2, and -1: i.e., from which we subtract 4!

So in our case:

$$\beta_2\beta_1\beta_0 \longleftrightarrow \sum_{i=0}^{1} \beta_i 2^i - \beta_2 \cdot 4$$

Two's complement scheme

In general, using k bits:

$$\beta_{k-1} \cdots \beta_0 \longleftrightarrow \sum_{i=0}^{k-2} \beta_i 2^i - \beta_{k-1} \cdot 2^{k-1}.$$

Two's complement scheme

In general, using k bits:

$$\beta_{k-1} \cdots \beta_0 \longleftrightarrow \sum_{i=0}^{k-2} \beta_i 2^i - \beta_{k-1} \cdot 2^{k-1}.$$

The smallest such number is

$$10 \cdots 0 \longleftrightarrow \sum_{i=0}^{k-2} 0 \cdot 2^i - 1 \cdot 2^{k-1} = -2^{k-1}.$$

Two's complement scheme

In general, using k bits:

$$\beta_{k-1}\cdots\beta_0 \longleftrightarrow \sum_{i=0}^{k-2}\beta_i 2^i - \beta_{k-1}\cdot 2^{k-1}.$$

The smallest such number is

$$10\cdots 0 \longleftrightarrow \sum_{i=0}^{k-2} 0\cdot 2^i - 1\cdot 2^{k-1} = -2^{k-1}.$$

The largest such number is

$$01\ldots 1 \longleftrightarrow \sum_{i=0}^{k-2} 1\cdot 2^i - 0\cdot 2^{k-1} = 2^{k-1} - 1.$$

Two's complement scheme

How can we see that $\sum_{i=0}^{k-2} 1 \cdot 2^i = 2^{k-1} - 1$?

Two's complement scheme

How can we see that $\sum_{i=0}^{k-2} 1 \cdot 2^i = 2^{k-1} - 1$?

1. Elegant: this is the largest binary number one can do using $k - 1$ bits, which is one less than 2^{k-1}.

Two's complement scheme

How can we see that $\sum_{i=0}^{k-2} 1 \cdot 2^i = 2^{k-1} - 1$?

1. Elegant: this is the largest binary number one can do using $k-1$ bits, which is one less than 2^{k-1}.
2. Brute force using geometric sum: if $q \neq 1$ we have

$$\sum_{i=0}^{n-1} q^i = \frac{q^n - 1}{q - 1},$$

hence with $q = 2$ and $n = k - 1$

$$\sum_{i=0}^{k-2} 1 \cdot 2^i = \frac{2^{k-1} - 1}{2 - 1} = 2^{k-1} - 1.$$

Two’s complement scheme

Two’s complement is what digital computers actually use for integer arithmetic. See the Wikipedia article for reasons why.

Two's complement scheme

Two's complement is what digital computers actually use for integer arithmetic. See the Wikipedia article for reasons why.

R uses $k = 32$ bits and two's complement with one modification:

$\texttt{10}\cdots\texttt{0} \longleftrightarrow$ `NA_integer_` *(the integer missing value).*

Two's complement scheme

Two's complement is what digital computers actually use for integer arithmetic. See the Wikipedia article for reasons why.

R uses $k = 32$ bits and two's complement with one modification:

$$\texttt{10}\cdots\texttt{0} \longleftrightarrow \texttt{NA\_integer\_}$$ *(the integer missing value).*

So the $2^{32} = 4294967296$ bit sequences have one zero, one NA, and $(2^{32} - 2)/2 = 2^{31} - 1 = 2147483647$ positive and negative integers each.

Two's complement scheme

Two's complement is what digital computers actually use for integer arithmetic. See the Wikipedia article for reasons why.

R uses $k = 32$ bits and two's complement with one modification:

$$10\cdots0 \longleftrightarrow \texttt{NA\_integer\_} \text{ (the integer missing value).}$$

So the $2^{32} = 4294967296$ bit sequences have one zero, one NA, and $(2^{32} - 2)/2 = 2^{31} - 1 = 2147483647$ positive and negative integers each.

The smallest such integer is $-(2^{31} - 1)$, the largest is $2^{31} - 1$.

Two's complement scheme

Trying to add one to the largest integer in integer arithmetic is not possible:

```
R> (imax <- .Machine$integer.max)
[1] 2147483647
R> imax + 1L
[1] NA
```

Two's complement scheme

Trying to add one to the largest integer in integer arithmetic is not possible:

```
R> (imax <- .Machine$integer.max)
[1] 2147483647
R> imax + 1L
[1] NA
```

Similarly,

```
R> as.integer(c(2^31 - 1, 2^31))
[1] 2147483647         NA
```

Outline

- Integers
- Doubles

Doubles

R uses *double precision floating point numbers* (“doubles”) for its numeric computations.

This is what is commonly used as a fixed precision model for the real numbers.

This is a standardized model: IEEE 754 (e.g., `https://en.wikipedia.org/wiki/IEEE_754`); equivalently, ISO/IEC/IEEE 60559 (but 754 is easier to remember).

Floating point numbers

E.g., 123.45 is a decimal floating point number everyone understands to be the same as

$$123.45 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}.$$

Floating point numbers

E.g., 123.45 is a decimal floating point number everyone understands to be the same as

$$123.45 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}.$$

One can also write this as

$$123.45 = 12345 \cdot 10^{-2} = 1.2345 \cdot 10^2.$$

The last is the *normalized form*.

Floating point numbers

E.g., 123.45 is a decimal floating point number everyone understands to be the same as

$$123.45 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}.$$

One can also write this as

$$123.45 = 12345 \cdot 10^{-2} = 1.2345 \cdot 10^2.$$

The last is the *normalized form*.

The sequence of (here, decimal) digits 12345 is called the *significand* (or *mantissa*), the 2 is the *exponent* (or *characteristic*) of the number.

Floating point number systems

A *floating point number system* is characterized by four integers: b (base or radix), p (precision), and $e_{\min}$ and $e_{\max}$ (minimal and maximal exponents).

Floating point number systems

WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

A *floating point number system* is characterized by four integers: b (base or radix), p (precision), and $e_{\min}$ and $e_{\max}$ (minimal and maximal exponents).

It consists of numbers of the form

$$x = \pm\left(\delta_0 + \frac{\delta_1}{b^1} + \cdots + \frac{\delta_{p-1}}{b^{p-1}}\right) b^e,$$

where $e_{\min} \le e \le e_{\max}$ and for $0 \le i \le p-1$,

$$\delta_i \in \{0, \ldots, b-1\}.$$

The number is normalized if $\delta_0 \neq 0$.

Floating point number systems

In decimal, the base is $b = 10$, and the digits go from 0 to 9.

Floating point number systems

In decimal, the base is $b = 10$, and the digits go from 0 to 9.

In octal?

Floating point number systems

In decimal, the base is $b = 10$, and the digits go from 0 to 9.

In octal? Base is $b = 8$, digits go from 0 to 7.

Floating point number systems

In decimal, the base is $b = 10$, and the digits go from 0 to 9.

In octal? Base is $b = 8$, digits go from 0 to 7.

In hexadecimal?

Floating point number systems

In decimal, the base is $b = 10$, and the digits go from 0 to 9.

In octal? Base is $b = 8$, digits go from 0 to 7.

In hexadecimal? Base is $b = 16$, digits are $0, \ldots, 9$, a, . . . f. Or $0, \ldots, 9$, A, . . . , F.

Floating point number systems

In decimal, the base is $b = 10$, and the digits go from 0 to 9.

In octal? Base is $b = 8$, digits go from 0 to 7.

In hexadecimal? Base is $b = 16$, digits are $0, \dots, 9$, a, ... f. Or $0, \dots, 9$, A, ..., F.

In binary?

Floating point number systems

In decimal, the base is $b = 10$, and the digits go from 0 to 9.

In octal? Base is $b = 8$, digits go from 0 to 7.

In hexadecimal? Base is $b = 16$, digits are $0, \ldots, 9$, a, ... f. Or $0, \ldots, 9$, A, ..., F.

In binary? Base is $b = 2$, digits are 0 or 1 (bits again).

Floating point number systems

In decimal, the base is $b = 10$, and the digits go from 0 to 9.

In octal? Base is $b = 8$, digits go from 0 to 7.

In hexadecimal? Base is $b = 16$, digits are 0, . . . , 9, a, . . . f. Or 0, . . . , 9, A, . . . , F.

In binary? Base is $b = 2$, digits are 0 or 1 (bits again).

Note that in binary, if the number is normalized, we must have $\delta_0 = 1$. So if we know it is normalized, we do not have to store δ_0!

IEEE 754

Clearly, all floating point numbers can be represented by the triple

$$(\text{sign}, \text{exponent}, \text{significand}).$$

IEEE 754 is a standard for base 2 which says: for *double precision*, use 64 bits (8 bytes) overall, split as

sign: 1 bit, exponent: 11 bits, significand: 52 bits.

IEEE 754

Clearly, all floating point numbers can be represented by the triple

$$(\text{sign}, \text{exponent}, \text{significand}).$$

IEEE 754 is a standard for base 2 which says: for *double precision*, use 64 bits (8 bytes) overall, split as

sign: 1 bit, *exponent: 11 bits,* *significand: 52 bits.*

In principle, the exponent is represented using the biased scheme (see before). So the exponent range would be

$$-1023, -1022, \ldots, 1023, 1024$$

but the smallest (all 0 bits) and the largest (all 1 bits) exponents are special!

IEEE 754

Representing binary floating point numbers in IEEE 754 works as follows:

(a) Exponent neither all 0 bits or all 1 bits: this is the normalized number

$$\sigma\left(1+\frac{\delta_1}{2}+\cdots+\frac{\delta_{52}}{2^{52}}\right)2^e.$$

IEEE 754

Representing binary floating point numbers in IEEE 754 works as follows:

(a) Exponent neither all 0 bits or all 1 bits: this is the normalized number

$$\sigma\left(1 + \frac{\delta_1}{2} + \cdots + \frac{\delta_{52}}{2^{52}}\right)2^e.$$

(b) Exponents all 0 bits: this is the de-normalized number

$$\sigma\left(0 + \frac{\delta_1}{2} + \cdots + \frac{\delta_{52}}{2^{52}}\right)2^{-1022}.$$

IEEE 754

Representing binary floating point numbers in IEEE 754 works as follows:

(a) Exponent neither all 0 bits or all 1 bits: this is the normalized number

$$\sigma\left(1+\frac{\delta_1}{2}+\cdots+\frac{\delta_{52}}{2^{52}}\right)2^e.$$

(b) Exponents all 0 bits: this is the de-normalized number

$$\sigma\left(0+\frac{\delta_1}{2}+\cdots+\frac{\delta_{52}}{2^{52}}\right)2^{-1022}.$$

(c) Exponent all 1 bits: if all bits in the significand are 0, this is $\pm\infty$; otherwise, it is a NaN.

IEEE 754

Note that for both normalized and de-normalized numbers, δ_0 never gets stored: so the signficand is represented by the bit sequence $\delta_1 \cdots \delta_{52}$.

The standard layout for the double precision representation is

σ	$\epsilon_{10} \cdots \epsilon_0$	$\delta_1 \cdots \delta_{52}$

Let's try some examples.

IEEE 754

Question: which IEEE 754 floating point number does

σ	$1 \cdots 1$	$0 \cdots 0$

correspond to?

IEEE 754

Question: which IEEE 754 floating point number does

σ	$1 \cdots 1$	$0 \cdots 0$

correspond to?

Answer: this is easy. Exponent has all 1 bits, significand has all 0 bits, so by rule (c), $\sigma\infty$ (i.e., $\pm\infty$).

IEEE 754

Question: which IEEE 754 floating point number does

σ	$1\cdots1$	$0\cdots0$

correspond to?

Answer: this is easy. Exponent has all 1 bits, significand has all 0 bits, so by rule (c), $\sigma\,\infty$ (i.e., $\pm\infty$).

Note that this is how get *two* infinities!

IEEE 754

Question: which IEEE 754 floating point number does

σ	$1\cdots1$	$0\cdots0$

correspond to?

Answer: this is easy. Exponent has all 1 bits, significand has all 0 bits, so by rule (c), $\sigma\infty$ (i.e., $\pm\infty$).

Note that this is how get *two* infinities!

For connaisseurs: two-point compactification of the real numbers.

IEEE 754

Question: which IEEE 754 floating point number does

σ	$0 \cdots 0$	$0 \cdots 0$

correspond to?

IEEE 754

Question: which IEEE 754 floating point number does

σ	$0 \cdots 0$	$0 \cdots 0$

correspond to?

Answer: this is easy. Exponent has all 0 bits, so by rule (b), this is a denormalized number, which has $\delta_0 = 0$ and for general $\delta_1, \ldots, \delta_{52}$ is given by

$$\sigma \left(\sum_{i=1}^{52} \frac{\delta_i}{2^i} \right) 2^{-1022}.$$

Here, all δ_i are 0, hence is the sum, and we get $\sigma\, 0$ (i.e., ± 0).

IEEE 754

Question: which IEEE 754 floating point number does

σ	$0 \cdots 0$	$0 \cdots 0$

correspond to?

Answer: this is easy. Exponent has all 0 bits, so by rule (b), this is a denormalized number, which has $\delta_0 = 0$ and for general $\delta_1, \ldots, \delta_{52}$ is given by

$$\sigma \left(\sum_{i=1}^{52} \frac{\delta_i}{2^i} \right) 2^{-1022}.$$

Here, all δ_i are 0, hence is the sum, and we get $\sigma\, 0$ (i.e., ± 0).

Note that this is how get *two* zeroes! (Remember Unit 1!)

IEEE 754

Question: what is the smallest positive de-normalized number we can do?

IEEE 754

Question: what is the smallest positive de-normalized number we can do?

Answer: this is easy. By rule (b), all bits in the exponent must be 0, and the smallest significand we can get is $0\ldots01$. The number is thus represented as

1	$0\cdots0$	$0\cdots01$

and its value is

$$\left(\sum_{i=1}^{52}\frac{\delta_i}{2^i}\right)2^{-1022} = 2^{-52}2^{-1022} = 2^{-1074}.$$

IEEE 754

Question: what is the smallest positive de-normalized number we can do?

Answer: this is easy. By rule (b), all bits in the exponent must be 0, and the smallest significand we can get is $0 \dots 01$. The number is thus represented as

1	$0 \cdots 0$	$0 \cdots 01$

and its value is

$$\left(\sum_{i=1}^{52} \frac{\delta_i}{2^i}\right) 2^{-1022} = 2^{-52} 2^{-1022} = 2^{-1074}.$$

In decimal:

```
R> 2^(-1074)
[1] 4.940656e-324
```

IEEE 754

Question: what is the largest positive de-normalized number we can do?

IEEE 754

Question: what is the largest positive de-normalized number we can do?

Answer: this is easy. By rule (b), all bits in the exponent must be 0, and the smallest significand we can get is $1 \ldots 1$. The number is thus represented as

1	$0 \cdots 0$	$1 \cdots 1$

and its value is

$$\left(\sum_{i=1}^{52} \frac{\delta_i}{2^i}\right) 2^{-1022} = 2^{-1022} \sum_{i=1}^{52} 2^{-i} = \cdots = 2^{-1022}(1 - 2^{-52}),$$

as $\sum_{i=1}^{52} 2^{-i} = 2^{-52} \sum_{i=0}^{51} 2^i = 2^{-52}(2^{52} - 1) = 1 - 2^{-52}$ (brute force, can also go elegant).

IEEE 754

Question: what is the smallest positive normalized number we can do?

IEEE 754

Question: what is the smallest positive normalized number we can do?

Answer: this is easy. We must make

- the exponent as small as possible, i.e., $0 \ldots 01$ which will correspond to -1022 (all 0 would not be normalized!)

IEEE 754

Question: what is the smallest positive normalized number we can do?

Answer: this is easy. We must make

- the exponent as small as possible, i.e., $0 \ldots 01$ which will correspond to -1022 (all 0 would not be normalized!)
- the significand as small as possible, i.e., $0 \ldots 0$.

IEEE 754

Question: what is the smallest positive normalized number we can do?

Answer: this is easy. We must make

- the exponent as small as possible, i.e., $0 \ldots 01$ which will correspond to -1022 (all 0 would not be normalized!)
- the significand as small as possible, i.e., $0 \ldots 0$.

The number is thus represented as

1	$0 \cdots 01$	$0 \cdots 0$

IEEE 754

The value of this number is

$$\left(1 + \sum_{i=1}^{52} \frac{0}{2^i}\right) 2^{-1022} = 2^{-1022}.$$

IEEE 754

The value of this number is

$$\left(1 + \sum_{i=1}^{52} \frac{0}{2^i}\right) 2^{-1022} = 2^{-1022}.$$

Note that this nicely continues above the de-normalized numbers, for which we already determined the positive ones to lie in the range from 2^{-1074} to $2^{-1022}(1-2^{-52})$!

IEEE 754

The value of this number is

$$\left(1 + \sum_{i=1}^{52} \frac{0}{2^i}\right) 2^{-1022} = 2^{-1022}.$$

Note that this nicely continues above the de-normalized numbers, for which we already determined the positive ones to lie in the range from 2^{-1074} to $2^{-1022}(1 - 2^{-52})$!

In R:

```
R> c(2^(-1022), .Machine$double.xmin)
[1] 2.225074e-308 2.225074e-308
```

IEEE 754

Question: what is the largest positive number we can do?

IEEE 754

Question: what is the largest positive number we can do?

Answer: this is easy. It must be a normalized number, and we must make

- the exponent as large as possible, i.e., $1 \ldots 10$ which will correspond to 1023 (all 1 would not be normalized!)

IEEE 754

Question: what is the largest positive number we can do?

Answer: this is easy. It must be a normalized number, and we must make

- the exponent as large as possible, i.e., $1 \dots 10$ which will correspond to 1023 (all 1 would not be normalized!)
- the significand as large as possible, i.e., $1 \dots 1$.

IEEE 754

Question: what is the largest positive number we can do?

Answer: this is easy. It must be a normalized number, and we must make

- the exponent as large as possible, i.e., $1 \dots 10$ which will correspond to 1023 (all 1 would not be normalized!)
- the significand as large as possible, i.e., $1 \dots 1$.

The number is thus represented as

1	$1 \cdots 10$	$1 \cdots 1$

IEEE 754

The value of this number is

$$\left(1+\sum_{i=1}^{52}\frac{1}{2^i}\right)2^{1023} = (1+1-2^{-52})2^{1023} = 2^{1024}(1-2^{-53})$$

IEEE 754

The value of this number is

$$\left(1+\sum_{i=1}^{52}\frac{1}{2^i}\right)2^{1023} = (1+1-2^{-52})2^{1023} = 2^{1024}(1-2^{-53})$$

In R,

```
R> c(2^1023 * (2 - 2^(-52)), .Machine$double.xmax)
[1] 1.797693e+308 1.797693e+308
```

IEEE 754

However,

```
R> 2^1024 * (1 - 2^(-53))
[1] Inf
```

Why?

IEEE 754

Question: how can we represent 1?

IEEE 754

Question: how can we represent 1?

Answer. This is . . . hmm, easy again.

IEEE 754

Question: how can we represent 1?

Answer. This is . . . hmm, easy again.

This must be a normalized number for which

$$1 = \left(1 + \sum_{i=1}^{52} \frac{\delta_i}{2^i}\right) 2^e.$$

So we must have $\delta_1 = \cdots = \delta_{52} = 0$ and $e = 0$, with exponent bits giving 1023 before biasing.

IEEE 754

Question: how can we represent 1?

Answer. This is . . . hmm, easy again.

This must be a normalized number for which

$$1 = \left(1 + \sum_{i=1}^{52} \frac{\delta_i}{2^i}\right) 2^e.$$

So we must have $\delta_1 = \cdots = \delta_{52} = 0$ and $e = 0$, with exponent bits giving 1023 before biasing.

Thus, the representation must be

1	$01 \cdots 1$	$0 \cdots 0$

IEEE 754

Question: what is the smallest positive number greater than 1?

IEEE 754

Question: what is the smallest positive number greater than 1?

Answer: this is easy again. This must be like 1, but with δ_{52} flipped from 0 to 1.

IEEE 754

Question: what is the smallest positive number greater than 1?

Answer: this is easy again. This must be like 1, but with δ_{52} flipped from 0 to 1.

This has representation

1	$01 \cdots 1$	$0 \cdots 01$

and value

$$1 + 2^{-52}$$

IEEE 754

What we have just shown is: modulo rounding effects,

$\epsilon = 2^{-52}$ is the smallest positive floating-point number x such that $1 + x \neq 1$!

IEEE 754

What we have just shown is: modulo rounding effects,

$\epsilon = 2^{-52}$ is the smallest positive floating-point number x such that $1 + x \neq 1$!

In R,

```
R> c(2^(-52), .Machine$double.eps)
[1] 2.220446e-16 2.220446e-16
```

IEEE 754

What we have just shown is: modulo rounding effects,

$\epsilon = 2^{-52}$ is the smallest positive floating-point number x such that $1 + x \neq 1$!

In R,

```
R> c(2^(-52), .Machine$double.eps)
[1] 2.220446e-16 2.220446e-16
```

So

the maximal precision we can expect for floating point computations is 16 decimal digits after the comma (52 binary digits).

IEEE 754

To illustrate:

```
R> (1 + 2^(-52)) == 1
[1] FALSE
R> (1 + 2^(-53)) == 1
[1] TRUE
```

IEEE 754

To illustrate:

```
R> (1 + 2^(-52)) == 1
[1] FALSE
R> (1 + 2^(-53)) == 1
[1] TRUE
```

So the basic rule

$$1 + x = 1 \quad \Rightarrow \quad x = 0$$

does not hold in floating point arithmetic!

IEEE 754

Similarly,

```
R> x <- 1
R> y <- 2^(-53)
R> (x + y) + y == x + (y + y)

[1] FALSE
```

IEEE 754

Similarly,

```
R> x <- 1
R> y <- 2^(-53)
R> (x + y) + y == x + (y + y)
```

```
[1] FALSE
```

So the basic rule $(x + y) + z = x + (y + z)$ (law of associativity) does not hold in floating point arithmetic!

IEEE 754

Similarly,

```
R> x <- 1
R> y <- 2^(-53)
R> (x + y) + y == x + (y + y)
```

```
[1] FALSE
```

So the basic rule $(x + y) + z = x + (y + z)$ (law of associativity) does not hold in floating point arithmetic!

Why?

IEEE 754

To illustrate the rounding effects:

```
R> 1 + 2^(-53) == 1
[1] TRUE
R> 1 + (2^(-53) + 2^(-54)) == 1
[1] FALSE
R> 1 + (2^(-53) + 2^(-105)) == 1
[1] FALSE
R> 1 + (2^(-53) + 2^(-106)) == 1
[1] TRUE
```

Why?

IEEE 754

Question: what is the largest positive number less than 1?

IEEE 754

Question: what is the largest positive number less than 1?

Answer: this is . . .

IEEE 754

Question: what is the largest positive number less than 1?

Answer: this is . . . hmm, not quite so easy.

It must be a normalized number.

1 obviously is the smallest number we can do with exponent 0.

So we are looking for the largest number with exponent -1, i.e., 1022 before biasing.

IEEE 754

Question: what is the largest positive number less than 1?

Answer: this is . . . hmm, not quite so easy.

It must be a normalized number.

1 obviously is the smallest number we can do with exponent 0.

So we are looking for the largest number with exponent -1, i.e., 1022 before biasing.

Thus, the representation must be

1	$01 \cdots 10$	$1 \cdots 1$

IEEE 754

The value of this number is

$$\left(1+\sum_{i=1}^{52}\frac{1}{2^i}\right)2^{-1}=(1+1-2^{-52})2^{-1}=1-2^{-53}.$$

IEEE 754

The value of this number is

$$\left(1 + \sum_{i=1}^{52} \frac{1}{2^i}\right) 2^{-1} = (1 + 1 - 2^{-52}) 2^{-1} = 1 - 2^{-53}.$$

What we have just shown is: modulo rounding effects,

$\epsilon = 2^{-53}$ *is the smallest positive floating-point number* x *such that* $1 - x \neq 1$*!*

IEEE 754

The value of this number is

$$\left(1 + \sum_{i=1}^{52} \frac{1}{2^i}\right) 2^{-1} = (1 + 1 - 2^{-52}) 2^{-1} = 1 - 2^{-53}.$$

What we have just shown is: modulo rounding effects,

$\epsilon = 2^{-53}$ *is the smallest positive floating-point number* x *such that* $1 - x \neq 1$*!*

In R,

```
R> c(2^(-53), .Machine$double.neg.eps)
```

```
[1] 1.110223e-16 1.110223e-16
```