P6 – Scientific Programming

Marcus Mohr
Jens Oeser

Geophysics Section
Department of Earth and Environmental Sciences
Ludwig-Maximilians-Universität München

SoSe 2021

Part #10

Representation of (Numerical) Data

representing integral and real numbers,
IEEE Floating Point standard

Elementary Numerical Data

The basic data numerical simulations deal with are numbers. Mathematicians know quite different sets of numbers, e.g.

$\mathbb{N}$	natural numbers	$\{1, 2, 3, 4, \ldots\}$
$\mathbb{Z}$	integral numbers	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{Q}$	rational numbers	$\frac{n}{d}$ with $n \in \mathbb{Z}$, $d \in \mathbb{N}$
$\mathbb{R}$	real numbers	e.g. 0, $\frac{1}{2}$, $\sqrt{2}$, π
$\mathbb{C}$	complex numbers	$u + iv$ with $u, v \in \mathbb{R}$, $i = \sqrt{-1}$

Numerical algorithms are executed by computers, so how are numbers represented there?

Machine Numbers

- Computers are finite systems $\longrightarrow$ not all x in the infinite sets $\mathbb{N}$ and $\mathbb{R}$ are (directly) representable (in hardware).
- Numbers that can be exactly represented on a computer are called machine numbers.
- Because their representation is significantly different one distinguishes
  - integral machine numbers
  - real-valued machine numbers

Machine Numbers (cont.)

Integers

Integral machine numbers can straightforwardly be represented in binary fashion and represent a subset of $\mathbb{Z}$.

Examples: -1, 12, 5543

Fixed-Point Numbers

are numbers representable by a fixed number of digits after (and sometimes also in front of) the decimal point. They represent a subset of $\mathbb{Q}$.

Examples: 0.12, -345.63, 2.98

Floating-Point Numbers

Computers represent a finite subset of $\mathbb{R}$ using Floating Point format.

Representing Numbers

In its history humankind developed different ways to represent numbers. The following three examples all represent the same number from $\mathbb{N}$:

MCMLXXI
roman


egyptian

1971
modern

roman: $\mathsf{I} \equiv 1$, $\mathsf{X} \equiv 10$, $\mathsf{L} \equiv 50$, $\mathsf{C} \equiv 100$, $\mathsf{M} \equiv 1000$

egyptian:

$\mathsf{I} \equiv 1$	$\cap \equiv 10$	$\equiv 100$	$\equiv 1000$
(single stroke)	(heel bone)	(coil of rope)	(water lily)

modern: what precisely does 1971 symbolise?

Decimal System

In everyday life we are accustomed to represent whole numbers using the decimal system, i.e.

$$1971 = 1 \cdot 1000 + 9 \cdot 100 + 7 \cdot 10 + 1 \cdot 1$$

We represent a number $n \in \mathbb{N}$ by a set of digits d_i specifying the factor in front of the associated power of basis 10

$$n = d_k d_{k-1} \ldots d_2 d_1 d_0 = d_k \cdot 10^k + \ldots d_2 \cdot 10^2 + d_1 \cdot 10^1 + d_0 \cdot 10^0$$

with

$$d_i \in \{0, 1, 2, \ldots, 8, 9\}$$

Positional Notation

- The decimal system is one example for a numeral system using positional notation (German: Stellenwertsystem).
- Positional notation may be based on any natural number $p \in \mathbb{N} \setminus \{1\}$. We then obtain a p-based (p-adische) form

$$n = d_k p^k + d_{k-1} p^{k-1} + \ldots + d_1 p + d_0 = \sum_{j=0}^{k} d_k p^k$$

with digits $d_i \in \{0, 1, \ldots, p-1\}$

- One of the first known uses of positional notation dates back to the Babylonians which employed a 60-based system.

Informatics

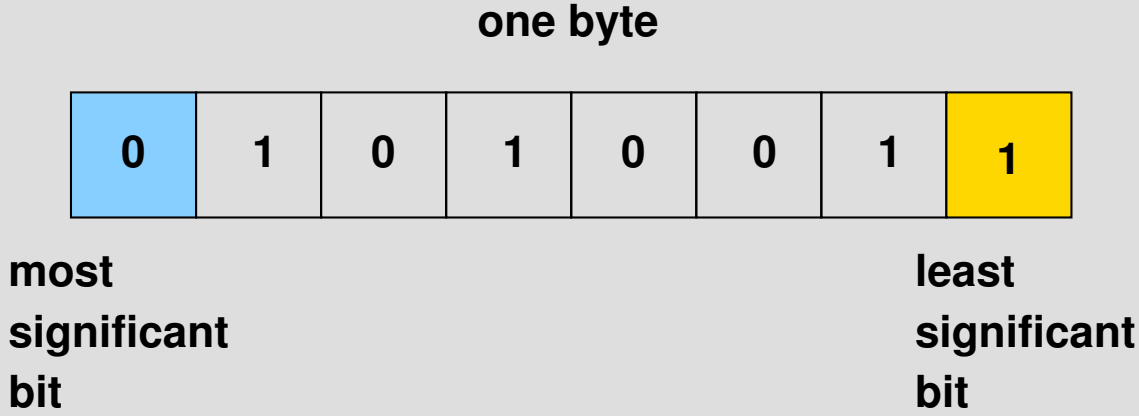
- In informatics the following numeral systems using positional notation are of importance
  - 2-based (binary numbers)
  - 8-based (octal numbers)
  - 16-based (hexadecimal numbers)
- Several languages offer possibilities to enter numbers / constants directly in (some of) these systems
  e.g. C: 125 (decimal) = 0175 (octal) = 0x7D (hexadecimal)

Unsigned Integers

- Unsigned integers can be used to store non-negative numbers $x \in \mathbb{N}_0$.
- Any such x can be expressed in the binary system as

$$83 = 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

- Representation on the computer is straightforward (apart from endianess)



Unsigned Integers (cont.)

The range of representable values is determined by the number of bits available.

Unsigned Integer Range

Given k bits we can represent any integer $n \in \mathbb{N}_0$ with

$$0 \leqslant n \leqslant 2^k - 1$$

- 0 might be represented by all bits not set (= O)
- all bits set (= L) might represent $2^k - 1$

Computations with Integers

- Computations with integers in binary representation works as in decimal representation. Example: (Addition of 4-bit integers)

$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	
O	L	O	L	$(5)_{10}$
O	L	L	O	$(6)_{10}$
L	O	L	L	$(11)_{10}$

- It can efficiently be implemented in hardware (e.g. addition using an XOR gate (exclusive or) and an AND gate for the carry over)
- Computations with integers are always exact, if the result falls into the representable range. If not, an overflow occurs.

Representing Signed Integers

- There are multiple ways to represent signed integers.
- Obvious and/or important
  - straightforward is the sign-and-magnitude approach
  - most computers use the two's complement approach
  - the excess-N or biased approach is mostly used for storing the exponent in floating-point representations

Representing Negative Integers (cont.)

Sign-and-Magnitude approach

- reserve one bit for the sign of $x \in \mathbb{Z}$
- and the remaining $(k-1)$ bits for the magnitude of x.

Properties:

- data range for k bits is $[-(2^{k-1}-1), 2^{k-1}-1]$
- two representations of zero:
  - (L)00...00 $= -0$ and
  - (0)00...00 $= +0$
- arithmetic ops require special logic (we need to consult sign bit)

Two's Complement (1/5)

- non-negative integers are represented directly
- negative integers are represented by a bit pattern obtained by negating all bits of $|x|$ and adding 1.

Properties:

- k bits give the range $[-2^{k-1}, 2^{k-1} - 1]$
- unique representation of 0 by `00...0`
- $x \geqslant 0$, if most significant bit (MSB) is `0`, and $x < 0$, if MSB is `1`.
- bit pattern of $x < 0$ is that of $\tilde{x} = 2^k - |x|$ stored as unsigned

Two's Complement (2/5)

As an example consider representing $(-6)_{10}$ with 4 bits

1. determine bit pattern for the magnitude of $(-6)_{10}$

O	L	L	O

2. negate this bit pattern

L	O	O	L

3. add a $(+1)_{10}$

L	O	L	O

Two's Complement (3/5)

Let us check the two's complement of $(-6)_{10}$ for $k = 4$ bits

1. bit pattern for $2^4 = (16)_{10}$

L	O	O	O	O

2. bit pattern of $|(-6)_{10}| = (6)_{10}$

O	O	L	L	O

3. resulting two's complement

O	L	O	L	O

Two's Complement (4/5)

- Given a bit patterns we can determine the value it represents by interpreting it as follows:

-8	4	2	1	
O	L	O	L	$\rightarrow (+5)_{10}$
O	L	L	O	$\rightarrow (+6)_{10}$
L	O	L	O	$\rightarrow (-6)_{10}$

- sign is encoded implicitely
- arithmetic ops in this representation require no specialised logic

Two's Complement (5/5)

Arithmetic operations can operate directly on bit patterns; e.g.

	-8	4	2	1	
	O	L	O	L	$\rightarrow (+5)_{10}$
$-$	O	L	L	O	$\rightarrow (+6)_{10}$
$=$	L	L	L	L	$\rightarrow (-1)_{10}$

or

	O	O	L	L	$\rightarrow (+3)_{10}$
$+$	L	O	L	L	$\rightarrow (-5)_{10}$
$=$	L	L	L	O	$\rightarrow (-2)_{10}$

Excess-N

- The biased or excess-N approach is another way to represent signed integers.
- It works by
  - a priori choosing a fixed bias $N \in \mathbb{N}$.
  - representing a number $x \in \mathbb{Z}$ by $\tilde{x} = x + N$.
  - typically we use (for a k-bit field) $N = (2^{k-1} - 1)$
  - then $\tilde{x} \in [0, 2N + 1 = 2^k - 1]$ for all $x \in [-N, N + 1]$.
- The approach today is primarily used in the floating point context.

Comparison

bit field	unsigned integer	sign & magnitude	two's complement	excess-3
OOO	0	+0	0	-3
OOL	1	1	1	-2
OLO	2	2	2	-1
OLL	3	3	3	0
LOO	4	-0	-4	1
LOL	5	-1	-3	2
LLO	6	-2	-2	3
LLL	7	-3	-1	4

Example: Data Ranges

The only limitations in representing $\mathbb{N}$ resp. $\mathbb{Z}$ stem from the finiteness of computer systems (examples from C/C++ language on my i686 machine)

data-type	size in bytes	smallest value	largest value
signed char	1 byte	-128	127
signed short int	2 bytes	-32,768	32,767
signed int	4 bytes	-2,147,483,648	2,147,483,647
unsigned char	1 byte	0	255
unsigned short int	2 bytes	0	65,535
unsigned int	4 bytes	0	4,294,967,295

An unsigned 64-bit type can address $2^{64} - 1 = 1.8447 \cdot 10^{19} = 16$ exa-bytes.

Hexadecimal System

- The hexadecimal system is a positional system with base 16.
- We need digits to represent $0, 1, \ldots, 14, 15$

Hexadecimal Digits

The 16 digits of the hexadecimal system are $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$.

- Since $2^4 = 16$ we can represent 1 byte $= 2$ nibbles (or half-byte) by 2 hexadecimal numbers.
- It is often employed in informatics to represent bit sequences in a more compact form than binary representation (e.g. machine code)

Example: Hexadecimal Numbers

binary	hexadecimal	decimal
LLLL	F	15
L,LLLL	1F	31
LL,OLLL,LLOO,OLOL	37C5	14,277
LOLO,LLOO,LLOL,LLOO	ACDC	44,252
L,OOOO,OOOO,OOOO,OOOO	1,0000	65,536
LOLO,LLLL,LLLL,LLLO,OOOO,LOOO,OOOL,OLOL	AFFE,0815	2,952,661,013

IPv6 address of `lin-m-ssh.geophysik.uni-muenchen.de`

2001:4ca0:4a02:2000::2:120

Representing $\mathbb{R}$

Challenge: The set $\mathbb{R}$ of real numbers is infinite and without gaps:

- for each $y \in \mathbb{R}$ there exist $x, z \in \mathbb{R}$ with $x < y < z$
- for each pair $x, z \in \mathbb{R}$ exists $y \in \mathbb{R}$ with $x < y < z$

On computer systems $\mathbb{R}$ is typically approximated by

- fixed-point numbers
- floating-point numbers (more precisely a finite subset of these)

For scientific computing floating-point numbers are the important ones.

Floating-Point Numbers

Every real number $x \in \mathbb{R}$ can be written in floating-point format

$$x = M \cdot B^E \quad \text{with} \quad \begin{cases} M & \text{significand/mantissa} \\ B & \text{base} \\ E & \text{exponent} \end{cases}$$

Example:

base 10		base 2		
24	$= 24 \cdot 10^0$	24	$= 6 \cdot 2^2$	
	$= 2.4 \cdot 10^1$		$= 3 \cdot 2^3$	
	$= 0.24 \cdot 10^2$		$= 1.5 \cdot 2^4$	
	$= 240 \cdot 10^{-1}$		$= 48 \cdot 2^{-1}$	

Floating-Point Numbers (cont.)

To obtain a unique representation (of $\mathbb{R}\backslash\{0\}$) one normalises and requires

$$1 \leqslant |M| = d_0 + d_1 \cdot B^{-1} + \ldots + d_{t-1} \cdot B^{-(t-1)} < B$$

where $t \in \mathbb{N} \cup \infty$ specifies the number of digits with respect to base B.

Example: $t = 2$

base 10	base 2
$24 = 2.4 \cdot 10^1$	$24 = 1.5 \cdot 2^4$
$1 \leqslant \|2.4\| < 10$	$1 \leqslant \|1.5\| < 2$
$\|2.4\| = 2 \cdot 10^0 + 4 \cdot 10^{-1}$	$\|1.5\| = 1 \cdot 2^0 + 1 \cdot 2^{-1}$

Machine Numbers

On a computer the significand and exponent of a floating-point number are represented by finite integers.

The set of real-valued machine numbers is

$$\mathbb{M}_{B,t,\alpha,\beta} = \left\{ M \cdot B^E \right\} = \left\{ \tilde{M} \cdot B^{1-t} \cdot B^E \right\} \subsetneq \mathbb{R}$$

with

- significand M resp. $\tilde{M}$
  - $M = 0$ or $1 \leqslant |M| < B$
  - $\tilde{M} \in \mathbb{Z}$ and $\tilde{M} = 0$ or $B^{t-1} \leqslant |\tilde{M}| < B^t$
- $E \in \mathbb{Z}$ and $\alpha \leqslant E \leqslant \beta$
- fixed $t \in \mathbb{N}$

Example: $\mathbb{M}_{2,4,-3,1}$

The set of machine numbers $\mathbb{M}_{2,4,-3,1}$ is characterised by

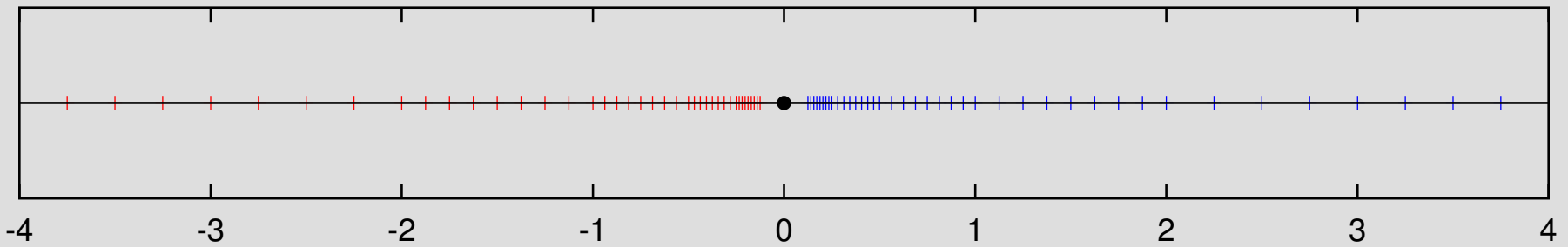
- base $B = 2$
- exponents E in $\{-3, -2, -1, 0, +1\}$
- and a 4–digit significand

The latter yields $8 = 2^3 \leqslant |\tilde{M}| < 2^4 = 16$ and, thus,

	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
$\lvert\tilde{M}\rvert =$	L	?	?	?

Example: $\mathbb{M}_{2,4,-3,1}$ (cont.)

	significand M							
$\tilde{M}$:	*LOOO*	*LOOL*	*LOLO*	*LOLL*	*LLOO*	*LLOL*	*LLLO*	*LLLL*
M :	1.000	1.125	1.250	1.375	1.500	1.625	1.750	1.875
2^{-3}	0.125000	0.140625	0.156250	0.171875	0.187500	0.203125	0.218750	0.234375
2^{-2}	0.250000	0.281250	0.312500	0.343750	0.375000	0.406250	0.437500	0.468750
2^{-1}	0.500000	0.562500	0.625000	0.687500	0.750000	0.812500	0.875000	0.937500
2^{+0}	1.000000	1.125000	1.250000	1.375000	1.500000	1.625000	1.750000	1.875000
2^{+1}	2.000000	2.250000	2.500000	2.750000	3.000000	3.250000	3.500000	3.750000



Implementation

The technical implementation of real-valued machine numbers is typically based on three approaches:

- normalisation
- implicit assumption on first bit of significand
- a 'biased exponent' to allow for negative values

Most computer systems realise the IEEE standard for floating-point numbers.

IEEE?

This abbreviation stands for the Institute of Electrical and Electronics Engineers

- world-wide professional organisation of engineers from the fields of electrics, electronics, information technology and computer science
- organises conferences, publishes journals, ...
- forms committees to provide industrial standards for hard- and software (e.g. IEEE standards for ethernet, WLAN, ...)

IEEE Floating Point Standard(s)

The IEEE standards

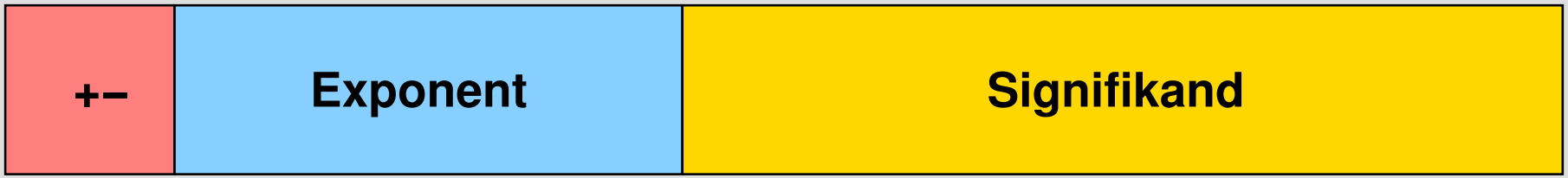
- IEEE Standard 754 for Binary Floating-Point Arithmetic (1985)
- IEEE Standard 854 for Radix-Independent Floating-Point Arithmetic (1987)
- IEEE 754-2008 (combined and updated version, published 2008)

deal with

- internal representation of floating-point numbers
- available levels of precision
- exception handling
- rounding

IEEE Standard: Storage

IEEE 754 defines the following format for FP representation



- 1 bit for the sign
- K bits for the (biased) exponent
- N bits for the significand (+ 'hidden bit')

IEEE Standard: Precision

IEEE 754 (1985) defines four levels of precision

precision	bits for significand	bits for exponent	total size
single	23	8	4 bytes
single extended	$\geqslant 31$	$\geqslant 11$	> 5 bytes
double	52	11	8 bytes
double extended	$\geqslant 63$	$\geqslant 15$	> 9 bytes

of these double is by far the most commonly used.

Update: IEEE 754-2008

The IEEE 754-2008 standard defines four levels of precision for base-2 representations

- binary16 (half-precision)
- binary32 (single precision)
- binary64 (double precision)
- binary128 (quadruple precision)

Basic format is the same as before; number gives bits in representation.

IEEE Standard: Exponent Range

Example: single precision (excess-127)

- 8 bits $\longrightarrow$ range for biased exponent (get's stored):

$$0 \leqslant \tilde{E} \leqslant \left(2^8 - 1\right) = 255$$

- bias 127 $\longrightarrow$ range for unbiased exponent

$$-127 \leqslant E \leqslant 128$$

- minimal & maximal exponent are reserved for special values

$$-126 \leqslant E \leqslant 127$$

Machine Precision

Rounding:

Finite precision floating-point arithmetic requires to round the result of each operation (e.g. $a + b$) to the nearest machine number.

Machine Precision:

We denote the largest relative rounding error

$$\text{eps}(x) \quad = \quad \left| \frac{x - \text{rd}(x)}{x} \right|$$

as machine precision eps_{mach}.

Machine Precision (cont.)

For $x \in \mathbb{R}$ with $x_{\min} \leqslant x \leqslant x_{\max}$ we get for $\mathrm{rd}(x) = \bar{x}$

$$\begin{aligned} x &= \sigma \cdot B^E \quad \text{with } 1 \leqslant |\sigma| < B \\ \bar{x} &= \bar{\sigma} \cdot B^E \quad \text{with } 1 \leqslant |\bar{\sigma}| \leqslant B \end{aligned}$$

the estimate

$$\mathrm{eps}(x) = \left| \frac{\sigma \cdot B^E - \bar{\sigma} \cdot B^E}{\sigma \cdot B^E} \right| \leqslant |\sigma - \bar{\sigma}| \leqslant \frac{1}{2} B^{(1-t)}$$

Thus, $\mathrm{eps}_{\mathrm{mach}} = 2^{-t}$ for $B = 2$.

IEEE Standard: Value Ranges

	single precision	double precision
E_{min}	-126	-1022
E_{max}	127	1023
x_{min}	$\approx 1.2 \cdot 10^{-38}$	$\approx 2.2 \cdot 10^{-308}$
x_{max}	$\approx 3.4 \cdot 10^{+38}$	$\approx 1.8 \cdot 10^{+308}$
eps_{mach}	$\approx 6.0 \cdot 10^{-8}$	$\approx 1.1 \cdot 10^{-16}$
significant decimals	$6 - 9$	$15 - 17$

IEEE Standard: Ranges (cont.)

We can compare IEEE ranges to the values from `float.h`

(on x86_64 GNU/Linux)

macro	float	double	long double ‡
{FLT,DBL,LDBL}_RADIX	2	2	2
{FLT,DBL,LDBL}_MANT_DIG	24	53	64
{FLT,DBL,LDBL}_MIN_EXP†	-125	-1021	-16381
{FLT,DBL,LDBL}_MAX_EXP	128	1024	16384
{FLT,DBL,LDBL}_MIN	1.176e-38	2.226e-308	3.362e-4932
{FLT,DBL,LDBL}_MAX	3.403e+38	1.798e+308	1.190e+4932

† smallest n such that r^{n-1} is representable as normalised fp-value with radix r.

‡ not `binary128`, would have LDBL_MANT_DIG $= 113$.

IEEE Standard: Ranges (cont.)

We can check on these by using intrinsic Fortran90 routines

x =	REAL	DOUBLE PRECISION
kind(x)	4	4
huge(x)	3.4028235E+38	1.7976931348623 16E+308
tiny(x)	1.1754944E-38	2.2250738585072 01E-308
minexponent(x)	-125	-1021
maxexponent(x)	128	1024
range(x)	37	307
precision(x)	6	15
radix(x)	2	2

Sample results for linterm01 (Coredump) and linterm02.

IEEE Standard: Special Values I

NaN	NaN = Not a Number, result of undefined operations such as e.g. $\sqrt{-1}$, $0 \times \infty$, $0/0$ or operations with NaN as operand
$\pm\infty$	infinity results from an overflow, i.e. the result is outside the representable range, or from operations such as $\pm 10/0 = \pm\infty$ and $\infty + 3 = \infty$
± 0	internally $+0$ and -0 are distinguished; important e.g. in case of underflow or to check results for being $\pm\infty$
subnormals	subnormals are non-normalised numbers used to reduce the 'gap' around 0

IEEE Standard: Special Values II

- Exceptions:
  The IEEE standard distinguishes five types of exceptions: overflow, underflow, division by zero, illegal operation and inexact.
- Status Flag:
  In case an exception occurs a status flag is set, that can be queried.
- Quiet and Signaling NaNs:
  Default behaviour is to proceed after an exception, if possible (quiet NaNs); the standard (theoretically) also provides the opportunity to halt on an exception (signaling NaN), as is typical for non-IEEE systems.

IEEE Standard: Summary

exponent	significand	meaning
$E = e_{\min} - 1$	$s = 0$	± 0
$E = e_{\min} - 1$	$s \neq 0$	$(0.s)_2 \times 2^{e_{\min}}$
$e_{\min} \leqslant E \leqslant e_{\max}$	s	$\pm(1.s)_2 \times 2^E$
$E = e_{\max} + 1$	$s = 0$	$\pm\infty$
$E = e_{\max} + 1$	$s \neq 0$	NaN

overview on the meaning of values of a floating-point number in the IEEE standard

Outlook

What other possibilities exist?

- support for computations with finite, but adjustable precision (e.g. GNU Multi-Precision Library or also computer algebra systems such as Maple or Mathematica)
- symbolic computations (computer algebra)
- interval arithmetics

References

- Kristina Reiss, Gerald Schmieder: Basiswissen Zahlentheorie, Springer, or any other book on elementary number theory
- Wikipedia article on Integer (computer science) and further links
- David Goldberg: What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM Computing Surveys, vol. 23, 1991