WIKIPEDIA
The Free Encyclopedia

# IEEE 754

The **IEEE Standard for Floating-Point Arithmetic** (**IEEE 754**) is a technical standard for floating-point arithmetic established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). The standard addressed many problems found in the diverse floating-point implementations that made them difficult to use reliably and portably. Many hardware floating-point units use the IEEE 754 standard.

The standard defines:

- *arithmetic formats:* sets of binary and decimal floating-point data, which consist of finite numbers (including signed zeros and subnormal numbers), infinities, and special "not a number" values (NaNs)
- *interchange formats:* encodings (bit strings) that may be used to exchange floating-point data in an efficient and compact form
- *rounding rules:* properties to be satisfied when rounding numbers 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.*)

IEEE 754-2008, published in August 2008, includes nearly all of the original IEEE 754-1985 standard, plus the IEEE 854-1987 Standard for Radix-Independent Floating-Point Arithmetic. The current version, IEEE 754-2019, was published in July 2019.[1] It is a minor revision of the previous version, incorporating mainly clarifications, defect fixes and new recommended operations.

## History

The first standard for floating-point arithmetic, IEEE 754-1985, was published in 1985. It covered only binary floating-point arithmetic.

A new version, IEEE 754-2008, was published in August 2008, following a seven-year revision process, chaired by Dan Zuras and edited by Mike Cowlishaw. It replaced both IEEE 754-1985 (binary floating-point arithmetic) and IEEE 854-1987 Standard for Radix-Independent Floating-Point Arithmetic. The binary formats in the original standard are included in this new standard along with three new basic formats, one binary and two decimal. To conform to the current standard, an implementation must implement at least one of the basic formats as both an arithmetic format and an interchange format.

The international standard **ISO/IEC/IEEE 60559:2011** (with content identical to IEEE 754-2008) has been approved for adoption through ISO/IEC JTC 1/SC 25 under the ISO/IEEE PSDO Agreement[2][3] and published.[4]

The current version, IEEE 754-2019 published in July 2019, is derived from and replaces IEEE 754-2008, following a revision process started in September 2015, chaired by David G. Hough and edited by Mike Cowlishaw. It incorporates mainly clarifications (e.g. *totalOrder*) and defect fixes (e.g. *minNum*), but also includes some new recommended operations (e.g. *augmentedAddition*).[5][6]

The international standard **ISO/IEC 60559:2020** (with content identical to IEEE 754-2019) has been approved for adoption through ISO/IEC JTC 1/SC 25 and published.[7]

The next projected revision of the standard is in 2028.[8]

## Formats

An IEEE 754 *format* is a "set of representations of numerical values and symbols". A format may also include how the set is encoded.[9]

A floating-point format is specified by

- a base (also called *radix*) $b$, which is either 2 (binary) or 10 (decimal) in IEEE 754;
- a precision $p$;
- an exponent range from *emin* to *emax*, with *emin* = 1 − *emax* for all IEEE 754 formats.

A format comprises

- Finite numbers, which can be described by three integers: $s$ = a *sign* (zero or one), $c$ = a *significand* (or *coefficient*) having no more than $p$ digits when written in base $b$ (i.e., an integer in the range through 0 to $b^p - 1$), and $q$ = an *exponent* such that $emin \le q + p - 1 \le emax$. The numerical value of such a finite number is $(-1)^s \times c \times b^q$.[a] Moreover, there are two zero values, called signed zeros: the sign bit specifies whether a zero is +0 (positive zero) or −0 (negative zero).
- Two infinities: +∞ and −∞.
- Two kinds of NaN (not-a-number): a quiet NaN (qNaN) and a signaling NaN (sNaN).

For example, if $b = 10$, $p = 7$, and $emax = 96$, then $emin = -95$, the significand satisfies $0 \le c \le 9\,999\,999$, and the exponent satisfies $-101 \le q \le 90$. Consequently, the smallest non-zero positive number that can be represented is $1\times10^{-101}$, and the largest is $9999999\times10^{90}$ ($9.999999\times10^{96}$), so the full range of numbers is $-9.999999\times10^{96}$ through $9.999999\times10^{96}$. The numbers $-b^{1-emax}$ and $b^{1-emax}$ (here, $-1\times10^{-95}$ and $1\times10^{-95}$) are the smallest (in magnitude) *normal numbers*; non-zero numbers between these smallest numbers are called subnormal numbers.

### Representation and encoding in memory

Some numbers may have several possible exponential format representations. For instance, if $b = 10$, and $p = 7$, then −12.345 can be represented by $−12345×10^{−3}$, $−123450×10^{−4}$, and $−1234500×10^{−5}$. However, for most operations, such as arithmetic operations, the result (value) does not depend on the representation of the inputs.

For the decimal formats, any representation is valid, and the set of these representations is called a *cohort*. When a result can have several representations, the standard specifies which member of the cohort is chosen.

For the binary formats, the representation is made unique by choosing the smallest representable exponent allowing the value to be represented exactly. Further, the exponent is not represented directly, but a bias is added so that the smallest representable exponent is represented as 1, with 0 used for subnormal numbers. For numbers with an exponent in the normal range (the exponent field being neither all ones nor all zeros), the leading bit of the significand will always be 1. Consequently, a leading 1 can be implied rather than explicitly present in the memory encoding, and under the standard the explicitly represented part of the significand will lie between 0 and 1. This rule is called *leading bit convention*, *implicit bit convention*, or *hidden bit convention*. This rule allows the binary format to have an extra bit of precision. The leading bit convention cannot be used for the subnormal numbers as they have an exponent outside the normal exponent range and scale by the smallest represented exponent as used for the smallest normal numbers.

Due to the possibility of multiple encodings (at least in formats called *interchange formats*), a NaN may carry other information: a sign bit (which has no meaning, but may be used by some operations) and a *payload*, which is intended for diagnostic information indicating the source of the NaN (but the payload may have other uses, such as *NaN-boxing*[10][11][12]).

## Basic and interchange formats

The standard defines five basic formats that are named for their numeric base and the number of bits used in their interchange encoding. There are three binary floating-point basic formats (encoded with 32, 64 or 128 bits) and two decimal floating-point basic formats (encoded with 64 or 128 bits). The binary32 and binary64 formats are the *single* and *double* formats of IEEE 754-1985 respectively. A conforming implementation must fully implement at least one of the basic formats.

The standard also defines *interchange formats*, which generalize these basic formats.[13] For the binary formats, the leading bit convention is required. The following table summarizes some of the possible interchange formats (including the basic formats).

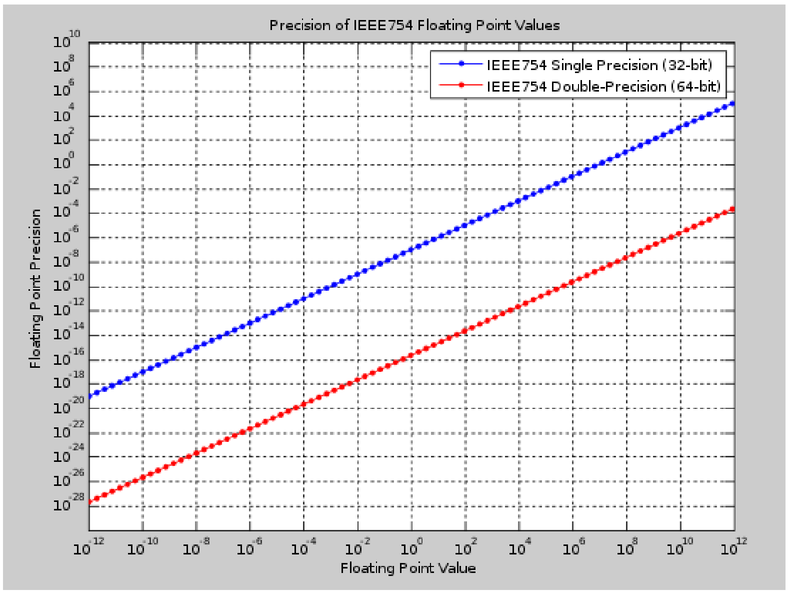
| | | | **Significand** | | **Exponent** | | **Properties[b]** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Name** | **Common name** | **Radix** | **Digits[c]** | **Decimal digits[d]** | **Min** | **Max** | ***MAXVAL*** | **$\log_{10}$ *MAXVAL*** | ***MINVAL*>0 (normal)** | ***MINVAL*>0 (subnorm)** | **Notes** |
| binary16 | Half precision | 2 | 11 | 3.31 | −14 | 15 | 65504 | 4.816 | $6.10·10^{−5}$ | $5.96·10^{−8}$ | Interchange |
| binary32 | Single precision | 2 | 24 | 7.22 | −126 | +127 | $3.40·10^{38}$ | 38.532 | $1.18·10^{−38}$ | $1.40·10^{−45}$ | Basic |
| binary64 | Double precision | 2 | 53 | 15.95 | −1022 | +1023 | $1.80·10^{308}$ | 308.255 | $2.23·10^{−308}$ | $4.94·10^{−324}$ | Basic |
| binary128 | Quadruple precision | 2 | 113 | 34.02 | −16382 | +16383 | $1.19·10^{4932}$ | 4932.075 | $3.36·10^{−4932}$ | $6.48·10^{−4966}$ | Basic |
| binary256 | Octuple precision | 2 | 237 | 71.34 | −262142 | +262143 | $1.61·10^{78193}$ | 78913.207 | $2.48·10^{−78913}$ | $2.25·10^{−78984}$ | Interchange |
| decimal32 | | 10 | 7 | 7 | −95 | +96 | $1.0·10^{97}$ | $97 − 2.2·10^{−15}$ | $1·10^{−95}$ | $1·10^{−101}$ | Interchange |
| decimal64 | | 10 | 16 | 16 | −191 | +192 | $1.0·10^{193}$ | $193 − 2.2·10^{−33}$ | $1·10^{−191}$ | $1·10^{−206}$ | Basic |
| decimal128 | | 10 | 34 | 34 | −6143 | +6144 | $1.0·10^{6145}$ | $6145 − 2.2·10^{−69}$ | $1·10^{−6143}$ | $1·10^{−6176}$ | Basic |

In the table above, integer values are exact where as values in decimal notation (e.g. 1.0) are rounded values. The minimum exponents listed are for normal numbers; the special subnormal number representation allows even smaller (in magnitude) numbers to be represented with some loss of precision. For example, the smallest positive number that can be represented in binary64 is $2^{−1074}$; contributions to the −1074 figure include the *emin* value −1022 and all but one of the 53 significand bits ($2^{−1022 − (53 − 1)} = 2^{−1074}$). The decimal representation does not define subnormal numbers as such, but numbers with a mantissa with leading zero(s) can be interpreted as subnormal as they offer fewer digits available to express precision.

Decimal digits is the precision of the format expressed in terms of an equivalent number of decimal digits. It is computed as *digits* × $\log_{10}$ *base*. E.g. binary128 has approximately the same precision as a 34 digit decimal number.

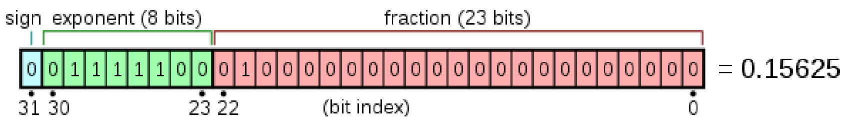
$\log_{10}$ *MAXVAL* is a measure of the range of the encoding. Its integer part is the largest exponent shown on the output of a value in scientific notation with one leading digit in the significand before the decimal point (e.g. $1.698·10^{38}$ is near the largest value in binary32, $9.999999·10^{96}$ is the largest value in decimal32).

The binary32 (single) and binary64 (double) formats are two of the most common formats used today. The figure below shows the absolute precision for both formats over a range of values. This figure can be used to select an appropriate format given the expected value of a number and the required precision.



Precision of binary32 and binary64 in the range $10^{-12}$ to $10^{12}$

An example of a layout for 32-bit floating point is



and the 64 bit layout is similar.

## Extended and extendable precision formats

The standard specifies optional extended and extendable precision formats, which provide greater precision than the basic formats.[14] An extended precision format extends a basic format by using more precision and more exponent range. An extendable precision format allows the user to specify the precision and exponent range. An implementation may use whatever internal representation it chooses for such formats; all that needs to be defined are its parameters ($b$, $p$, and $emax$). These parameters uniquely describe the set of finite numbers (combinations of sign, significand, and exponent for the given radix) that it can represent.

The standard recommends that language standards provide a method of specifying $p$ and $emax$ for each supported base $b$.[15] The standard recommends that language standards and implementations support an extended format which has a greater precision than the largest basic format supported for each radix $b$.[16] For an extended format with a precision between two basic formats the exponent range must be as great as that of the next wider basic format. So for instance a 64-bit extended precision binary number must have an 'emax' of at least 16383. The x87 80-bit extended format meets this requirement.

## Interchange formats

Interchange formats are intended for the exchange of floating-point data using a bit string of fixed length for a given format.

### Binary

For the exchange of binary floating-point numbers, interchange formats of length 16 bits, 32 bits, 64 bits, and any multiple of 32 bits $\geq 128$[e] are defined. The 16-bit format is intended for the exchange or storage of small numbers (e.g., for graphics).

The encoding scheme for these binary interchange formats is the same as that of IEEE 754-1985: a sign bit, followed by $w$ exponent bits that describe the exponent offset by a *bias*, and $p - 1$ bits that describe the significand. The width of the exponent field for a $k$-bit format is computed as $w = \text{round}(4 \log_2(k)) - 13$. The existing 64- and 128-bit formats follow this rule, but the 16- and 32-bit formats have more exponent bits (5 and 8 respectively) than this formula would provide (3 and 7 respectively).

As with IEEE 754-1985, the biased-exponent field is filled with all 1 bits to indicate either infinity (trailing significand field = 0) or a NaN (trailing significand field ≠ 0). For NaNs, quiet NaNs and signaling NaNs are distinguished by using the most significant bit of the trailing significand field exclusively,[f] and the payload is carried in the remaining bits.

### Decimal

For the exchange of decimal floating-point numbers, interchange formats of any multiple of 32 bits are defined. As with binary interchange, the encoding scheme for the decimal interchange formats encodes the sign, exponent, and significand. Two different bit-level encodings are defined, and interchange is complicated by the fact that some external indicator of the encoding in use may be required.

The two options allow the significand to be encoded as a compressed sequence of decimal digits using densely packed decimal or, alternatively, as a binary integer. The former is more convenient for direct hardware implementation of the standard, while the latter is more suited to software emulation on a binary computer. In either case, the set of numbers (combinations of sign, significand, and exponent) that may be encoded is identical, and special values (±zero with the minimum exponent, ±infinity, quiet NaNs, and signaling NaNs) have identical encodings.

## Rounding rules

The standard defines five rounding rules. The first two rules round to a nearest value; the others are called *directed roundings*:

### Roundings to nearest

- **Round to nearest, ties to even** – rounds to the nearest value; if the number falls midway, it is rounded to the nearest value with an even least significant digit.
- **Round to nearest, ties away from zero** (or **ties to away**) – rounds to the nearest value; if the number falls midway, it is rounded to the nearest value above (for positive numbers) or below (for negative numbers).

At the extremes, a value with a magnitude strictly less than $k = b^{\text{emax}}\left(b - \frac{1}{2}b^{1-p}\right)$ will be rounded to the minimum or maximum finite number (depending on the value's sign). Any numbers with exactly this magnitude are considered ties; this choice of tie may be conceptualized as the midpoint between $\pm b^{\text{emax}}(b - b^{1-p})$ and $\pm b^{\text{emax}+1}$, which, were the exponent not limited, would be the next representable floating-point numbers larger in magnitude. Numbers with a magnitude strictly larger than $k$ are rounded to the corresponding infinity.[17]

"Round to nearest, ties to even" is the default for binary floating point and the recommended default for decimal. "Round to nearest, ties to away" is only required for decimal implementations.[18]

### Directed roundings

- **Round toward 0** – directed rounding towards zero (also known as *truncation*).
- **Round toward +∞** – directed rounding towards positive infinity (also known as *rounding up* or *ceiling*).
- **Round toward −∞** – directed rounding towards negative infinity (also known as *rounding down* or *floor*).

Example of rounding to integers using the IEEE 754 rules

| Mode | Example value | | | |
|---|---|---|---|---|
| | **+11.5** | **+12.5** | **−11.5** | **−12.5** |
| to nearest, ties to even | +12.0 | +12.0 | −12.0 | −12.0 |
| to nearest, ties away from zero | +12.0 | +13.0 | −12.0 | −13.0 |
| toward 0 | +11.0 | +12.0 | −11.0 | −12.0 |
| toward +∞ | +12.0 | +13.0 | −11.0 | −12.0 |
| toward −∞ | +11.0 | +12.0 | −12.0 | −13.0 |

Unless specified otherwise, the floating-point result of an operation is determined by applying the rounding function on the infinitely precise (mathematical) result. Such an operation is said to be *correctly rounded*. This requirement is called *correct rounding*.[19]

## Required operations

Required operations for a supported arithmetic format (including the basic formats) include:

- Conversions to and from integer[20][21]
- Previous and next consecutive values[20]
- Arithmetic operations (add, subtract, multiply, divide, square root, fused multiply–add, remainder, minimum, maximum)[20][21]
- Conversions (between formats, to and from strings, *etc.*)[22][23]
- Scaling and (for decimal) quantizing[24][25]
- Copying and manipulating the sign (abs, negate, *etc.*)[26]
- Comparisons and total ordering[27][28]
- Classification of numbers (subnormal, finite, *etc.*) and testing for NaNs[29]
- Testing and setting status flags[30]

### Comparison predicates

The standard provides comparison predicates to compare one floating-point datum to another in the supported arithmetic format.[31] Any comparison with a NaN is treated as unordered. −0 and +0 compare as equal.

### Total-ordering predicate

The standard provides a predicate *totalOrder*, which defines a total ordering on canonical members of the supported arithmetic format.[32] The predicate agrees with the comparison predicates when one floating-point number is less than the other. The *totalOrder* predicate does not impose a total ordering on all encodings in a format. In particular, it does not distinguish among different encodings of the same floating-point representation, as when one or both encodings are non-canonical.[32] IEEE 754-2019 incorporates clarifications of *totalOrder*.

For the binary interchange formats whose encoding follows the IEEE 754-2008 recommendation on placement of the NaN signaling bit, the comparison is identical to one that type puns the floating-point numbers to a sign–magnitude integer (assuming a payload ordering consistent with this comparison), an old trick for FP comparison without an FPU.[33]

## Exception handling

The standard defines five exceptions, each of which returns a default value and has a corresponding status flag that is raised when the exception occurs.[g] No other exception handling is required, but additional non-default alternatives are recommended (see § Alternate exception handling).

The five possible exceptions are

- Invalid operation: mathematically undefined, *e.g.*, the square root of a negative number. By default, returns qNaN.
- Division by zero: an operation on finite operands gives an exact infinite result, *e.g.*, 1/0 or log(0). By default, returns ±infinity.
- Overflow: a finite result is too large to be represented accurately (*i.e.*, its exponent with an unbounded exponent range would be larger than *emax*). By default, returns ±infinity for the round-to-nearest modes (and follows the rounding rules for the directed rounding modes).
- Underflow: a result is very small (outside the normal range). By default, returns a number less than or equal to the minimum positive normal number in magnitude (following the rounding rules); a subnormal number always implies an underflow exception, but by default, if it is exact, no flag is raised.
- Inexact: the exact (*i.e.*, unrounded) result is not representable exactly. By default, returns the correctly rounded result.

These are the same five exceptions as were defined in IEEE 754-1985, but the *division by zero* exception has been extended to operations other than the division.

Some decimal floating-point implementations define additional exceptions,[34][35] which are not part of IEEE 754:

- Clamped: a result's exponent is too large for the destination format. By default, trailing zeros will be added to the coefficient to reduce the exponent to the largest usable value. If this is not possible (because this would cause the number of digits needed to be more than the destination format) then an overflow exception occurs.
- Rounded: a result's coefficient requires more digits than the destination format provides. An inexact exception is signaled if any non-zero digits are discarded.

Additionally, operations like quantize when either operand is infinite, or when the result does not fit the destination format, will also signal invalid operation exception.[36]

## Special values

### Signed zero

In the IEEE 754 standard, zero is signed, meaning that there exist both a "positive zero" (+0) and a "negative zero" (−0). In most run-time environments, positive zero is usually printed as "`0`" and the negative zero as "`-0`". The two values behave as equal in numerical comparisons, but some operations return different results for +0 and −0. For instance, 1/(−0) returns negative infinity, while 1/(+0) returns positive infinity (so that the identity $1/(1/\pm\infty) = \pm\infty$ is maintained). Other common functions with a discontinuity at *x*=0 which might treat +0 and −0 differently include log(*x*), signum(*x*), and the principal square root of $y + xi$ for any negative number *y*. As with any approximation scheme, operations involving "negative zero" can occasionally cause confusion. For example, in IEEE 754, $x = y$ does not always imply $1/x = 1/y$, as 0 = −0 but 1/0 ≠ 1/(−0).[37]

### Subnormal numbers

Subnormal values fill the underflow gap with values where the absolute distance between them is the same as for adjacent values just outside the underflow gap. This is an improvement over the older practice to just have zero in the underflow gap, and where underflowing results were replaced by zero (flush to zero).[38]

Modern floating-point hardware usually handles subnormal values (as well as normal values), and does not require software emulation for subnormals.

### Infinities

The infinities of the extended real number line can be represented in IEEE floating-point datatypes, just like ordinary floating-point values like 1, 1.5, etc. They are not error values in any way, though they are often (depends on the rounding) used as replacement values when there is an overflow. Upon a divide-by-zero exception, a positive or negative infinity is returned as an exact result. An infinity can also be introduced as a numeral (like C's "INFINITY" macro, or "∞" if the programming language allows that syntax).

IEEE 754 requires infinities to be handled in a reasonable way, such as

- $(+\infty) + (+7) = (+\infty)$

- $(+\infty) \times (-2) = (-\infty)$
- $(+\infty) \times 0$ = NaN – there is no meaningful thing to do

### NaNs

IEEE 754 specifies a special value called "Not a Number" (NaN) to be returned as the result of certain "invalid" operations, such as 0/0, $\infty \times 0$, or sqrt(−1). In general, NaNs will be propagated, i.e. most operations involving a NaN will result in a NaN, although functions that would give some defined result for any given floating-point value will do so for NaNs as well, e.g. NaN ^ 0 = 1. There are two kinds of NaNs: the default *quiet* NaNs and, optionally, *signaling* NaNs. A signaling NaN in any arithmetic operation (including numerical comparisons) will cause an "invalid operation" exception to be signaled.

The representation of NaNs specified by the standard has some unspecified bits that could be used to encode the type or source of error; but there is no standard for that encoding. In theory, signaling NaNs could be used by a runtime system to flag uninitialized variables, or extend the floating-point numbers with other special values without slowing down the computations with ordinary values, although such extensions are not common.

## Design rationale

It is a common misconception that the more esoteric features of the IEEE 754 standard discussed here, such as extended formats, NaN, infinities, subnormals etc., are only of interest to numerical analysts, or for advanced numerical applications. In fact the opposite is true: these features are designed to give safe robust defaults for numerically unsophisticated programmers, in addition to supporting sophisticated numerical libraries by experts. The key designer of IEEE 754, William Kahan notes that it is incorrect to "... [deem] features of IEEE Standard 754 for Binary Floating-Point Arithmetic that ...[are] not appreciated to be features usable by none but numerical experts. The facts are quite the opposite. In 1977 those features were designed into the Intel 8087 to serve the widest possible market... Error-analysis tells us how to design floating-point arithmetic, like IEEE Standard 754, moderately tolerant of well-meaning ignorance among programmers".[39]

William Kahan. A primary architect of the Intel 80x87 floating-point coprocessor and IEEE 754 floating-point standard.

- The special values such as infinity and NaN ensure that the floating-point arithmetic is algebraically complete: every floating-point operation produces a well-defined result and will not —by default—throw a machine interrupt or trap. Moreover, the choices of special values returned in exceptional cases were designed to give the correct answer in many cases. For instance, under IEEE 754 arithmetic, continued fractions such as R(z) := 7 − 3/[z − 2 − 1/(z − 7 + 10/[z − 2 − 2/(z − 3)])] will give the correct answer on all inputs, as the potential divide by zero, e.g. for z = 3, is correctly handled by giving +infinity, and so such exceptions can be safely ignored.[40] 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.[39]
- Subnormal numbers ensure that for *finite* floating-point numbers x and y, x − y = 0 if and only if x = y, as expected, but which did not hold under earlier floating-point representations.[41]
- On the design rationale of the x87 80-bit format, Kahan notes: "This Extended format is designed to be used, with negligible loss of speed, for all but the simplest arithmetic with float and double operands. For example, it should be used for scratch variables in loops that implement recurrences like polynomial evaluation, scalar products, partial and continued fractions. It often averts premature Over/Underflow or severe local cancellation that can spoil simple algorithms".[42] Computing intermediate results in an extended format with high precision and extended exponent has precedents in the historical practice of scientific calculation and in the design of scientific calculators e.g. Hewlett-Packard's financial calculators performed arithmetic and financial functions to three more significant decimals than they stored or displayed.[42] The implementation of extended precision enabled standard elementary function libraries to be readily developed that normally gave double precision results within one unit in the last place (ULP) at high speed.
- Correct rounding of values to the nearest representable value avoids systematic biases in calculations and slows the growth of errors. Rounding ties to even removes the statistical bias that can occur in adding similar figures.
- Directed rounding was intended as an aid with checking error bounds, for instance in interval arithmetic. It is also used in the implementation of some functions.
- The mathematical basis of the operations, in particular correct rounding, allows one to prove mathematical properties and design floating-point algorithms such as 2Sum, Fast2Sum and Kahan summation algorithm, e.g. to improve accuracy or implement multiple-precision arithmetic subroutines relatively easily.

A property of the single- and double-precision formats is that their encoding allows one to easily sort them without using floating-point hardware. Their bits interpreted as a two's-complement integer already sort the positives correctly, with the negatives reversed. With an xor to flip the sign bit for positive values and all bits for negative values, all the values become sortable as unsigned integers (with −0 < +0).[33] It is unclear whether this property is intended.

## Recommendations

### Alternate exception handling

The standard recommends optional exception handling in various forms, including presubstitution of user-defined default values, and traps (exceptions that change the flow of control in some way) and other exception handling models that interrupt the flow, such as try/catch. The traps and other exception mechanisms remain optional, as they were in IEEE 754-1985.

### Recommended operations

Clause 9 in the standard recommends additional mathematical operations[43] that language standards should define.[44] None are required in order to conform to the standard.

The following are recommended arithmetic operations, which must round correctly:[45]

- $e^x$, $2^x$, $10^x$
- $e^x - 1$, $2^x - 1$, $10^x - 1$
- $\ln x$, $\log_2 x$, $\log_{10} x$
- $\ln(1+x)$, $\log_2(1+x)$, $\log_{10}(1+x)$
- $\sqrt{x^2 + y^2}$
- $\sqrt{x}$
- $(1+x)^n$
- $x^{\frac{1}{n}}$
- $x^n$, $x^y$
- $\sin x$, $\cos x$, $\tan x$
- $\arcsin x$, $\arccos x$, $\arctan x$, $\operatorname{atan2}(y,x)$
- $\operatorname{sinPi} x = \sin \pi x$, $\operatorname{cosPi} x = \cos \pi x$, $\operatorname{tanPi} x = \tan \pi x$ (see also: Multiples of π)
- $\operatorname{asinPi} x = \frac{\arcsin x}{\pi}$, $\operatorname{acosPi} x = \frac{\arccos x}{\pi}$, $\operatorname{atanPi} x = \frac{\arctan x}{\pi}$, $\operatorname{atan2Pi}(y,x) = \frac{\operatorname{atan2}(y,x)}{\pi}$ (see also: Multiples of π)
- $\sinh x$, $\cosh x$, $\tanh x$
- $\operatorname{arsinh} x$, $\operatorname{arcosh} x$, $\operatorname{artanh} x$

The $\operatorname{asinPi}$, $\operatorname{acosPi}$ and $\operatorname{tanPi}$ functions were not part of the IEEE 754-2008 standard because they were deemed less necessary.[46] $\operatorname{asinPi}$, $\operatorname{acosPi}$ were mentioned, but this was regarded as an error.[5] All three were added in the 2019 revision.

The recommended operations also include setting and accessing dynamic mode rounding direction,[47] and implementation-defined vector reduction operations such as sum, scaled product, and dot product, whose accuracy is unspecified by the standard.[48]

As of 2019, *augmented arithmetic operations*[49] for the binary formats are also recommended. These operations, specified for addition, subtraction and multiplication, produce a pair of values consisting of a result correctly rounded to nearest in the format and the error term, which is representable exactly in the format. At the time of publication of the standard, no hardware implementations are known, but very similar operations were already implemented in software using well-known algorithms. The history and motivation for their standardization are explained in a background document.[50][51]

As of 2019, the formerly required *minNum*, *maxNum*, *minNumMag*, and *maxNumMag* in IEEE 754-2008 are now deprecated due to their non-associativity. Instead, two sets of new minimum and maximum operations are recommended.[52] The first set contains *minimum*, *minimumNumber*, *maximum* and *maximumNumber*. The second set contains *minimumMagnitude*, *minimumMagnitudeNumber*, *maximumMagnitude* and *maximumMagnitudeNumber*. The history and motivation for this change are explained in a background document.[53]

### Expression evaluation

The standard recommends how language standards should specify the semantics of sequences of operations, and points out the subtleties of literal meanings and optimizations that change the value of a result. By contrast, the previous 1985 version of the standard left aspects of the language interface unspecified, which led to inconsistent behavior between compilers, or different optimization levels in an optimizing compiler.

Programming languages should allow a user to specify a minimum precision for intermediate calculations of expressions for each radix. This is referred to as *preferredWidth* in the standard, and it should be possible to set this on a per-block basis. Intermediate calculations within expressions should be calculated, and any temporaries saved, using the maximum of the width of the operands and the preferred width if set. Thus, for instance, a compiler targeting x87 floating-point hardware should have a means of specifying that intermediate calculations must use the double-extended format. The stored value of a variable must always be used when evaluating subsequent expressions, rather than any precursor from before rounding and assigning to the variable.

### Reproducibility

The IEEE 754-1985 version of the standard allowed many variations in implementations (such as the encoding of some values and the detection of certain exceptions). IEEE 754-2008 has reduced these allowances, but a few variations still remain (especially for binary formats). The reproducibility clause recommends that language standards should provide a means to write reproducible programs (i.e., programs that will produce the same result in all implementations of a language) and describes what needs to be done to achieve reproducible results.

## Character representation

The standard requires operations to convert between basic formats and *external character sequence* formats.[54] Conversions to and from a decimal character format are required for all formats. Conversion to an external character sequence must be such that conversion back using round to nearest, ties to even will recover the original number. There is no requirement to preserve the payload of a quiet NaN or signaling NaN, and conversion from the external character sequence may turn a signaling NaN into a quiet NaN.

The original binary value will be preserved by converting to decimal and back again using:[55]

- 5 decimal digits for binary16,
- 9 decimal digits for binary32,
- 17 decimal digits for binary64,
- 36 decimal digits for binary128.

For other binary formats, the required number of decimal digits is[h]

$$1 + \lceil p \log_{10}(2) \rceil,$$

where *p* is the number of significant bits in the binary format, e.g. 237 bits for binary256.

When using a decimal floating-point format, the decimal representation will be preserved using:

- 7 decimal digits for decimal32,
- 16 decimal digits for decimal64,
- 34 decimal digits for decimal128.

Algorithms, with code, for correctly rounded conversion from binary to decimal and decimal to binary are discussed by Gay,[56] and for testing – by Paxson and Kahan.[57]

### Hexadecimal literals

The standard recommends providing conversions to and from *external hexadecimal-significand character sequences*, based on C99's hexadecimal floating point literals. Such a literal consists of an optional sign (`+` or `-`), the indicator "0x", a hexadecimal number with or without a period, an exponent indicator "p", and a decimal exponent with optional sign. The syntax is not case-sensitive.[58] The decimal exponent scales by powers of 2, so for example `0x0.1p-4` is 1/256.[59]

## See also

- bfloat16 floating-point format
- Coprocessor
- C99 for code examples demonstrating access and use of IEEE 754 features.
- Floating-point arithmetic, for history, design rationale and example usage of IEEE 754 features.
- Fixed-point arithmetic, for an alternative approach at computation with rational numbers (especially beneficial when the exponent range is known, fixed, or bound at compile time).
- IBM System z9, the first CPU to implement IEEE 754-2008 decimal arithmetic (using hardware microcode).
- IBM z10, IBM z196, IBM zEC12, and IBM z13, CPUs that implement IEEE 754-2008 decimal arithmetic fully in hardware.
- ISO/IEC 10967, language-independent arithmetic (LIA).
- Minifloat, low-precision binary floating-point formats following IEEE 754 principles.
- POWER6, POWER7, and POWER8 CPUs that implement IEEE 754-2008 decimal arithmetic fully in hardware.
- strictfp, an obsolete keyword in the Java programming language that previously restricted arithmetic to IEEE 754 single and double precision to ensure reproducibility across common hardware platforms (as of Java 17, this behavior is required)
- Table-maker's dilemma for more about the correct rounding of functions.
- Standard Apple Numerics Environment
- Tapered floating point
- Posit, an alternative number format

## Notes

a. For example, if the base is 10, the sign is 1 (indicating negative), the significand is 12345, and the exponent is −3, then the value of the number is $(-1)^1 \times 12345 \times 10^{-3} = -1 \times 12345 \times 0.001 = -12.345$.

b. Approximative values. For exact values see each format's individual Wikipedia entry

c. Number of digits in the radix used, including any implicit digit, but not counting the sign bit.

d. Corresponding number of decimal digits, see text for more details.

e. Contrary to decimal, there is no binary interchange format of 96-bit length. Such a format is still allowed as a non-interchange format, though.

f. The standard recommends 0 for signaling NaNs, 1 for quiet NaNs, so that a signaling NaNs can be quieted by changing only this bit to 1, while the reverse could yield the encoding of an infinity.

g. No flag is raised in certain cases of underflow.

h. As an implementation limit, correct rounding is only guaranteed for the number of decimal digits required plus 3 for the largest supported binary format. For instance, if binary32 is the largest supported binary format, then a conversion from a decimal external sequence with 12 decimal digits is guaranteed to be correctly rounded when converted to binary32; but conversion of a sequence of 13 decimal digits is not; however, the standard recommends that implementations impose no such limit.

## References

1. IEEE 754 2019
2. Haasz, Jodi. "FW: ISO/IEC/IEEE 60559 (IEEE Std 754-2008)" (https://web.archive.org/web/20171027190846/http://grouper.ieee.org/groups/754/email/msg04167.html). *grouper.ieee.org*. Archived from the original (http://grouper.ieee.org/groups/754/email/msg04167.html) on 2017-10-27. Retrieved 2018-04-04.
3. "ISO/IEEE Partner Standards Development Organization (PSDO) Cooperation Agreement" (https://grouper.ieee.org/groups/802/minutes/jul2008/opening_reports/psdo1.pdf) (PDF). ISO. 2007-12-19. Retrieved 2021-12-27.
4. ISO/IEC JTC 1/SC 25 2011.
5. Cowlishaw, Mike (2013-11-13). "IEEE 754-2008 errata" (http://speleotrove.com/misc/IEEE754-errata.html). *speleotrove.com*. Retrieved 2020-01-24.
6. "Revising ANSI/IEEE Std 754-2008" (http://754r.ucbtest.org/). *ucbtest.org*. Retrieved 2018-04-04.
7. ISO/IEC JTC 1/SC 25 2020.
8. Riedy, E. Jason (2018-06-26), "Plans for IEEE Standard 754 – 2028" (http://www.ecs.umass.edu/arith-2018/Presentations/ieee754-2028-ejr.pdf) (PDF), *25th IEEE Symposium on Computer Arithmetic*, Amherst, MA: IEEE
9. IEEE 754 2008, §2.1.27.
10. "SpiderMonkey Internals" (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals). *developer.mozilla.org*. Retrieved 2018-03-11.
11. Klemens, Ben (September 2014). *21st Century C: C Tips from the New School* (https://books.google.com/books?id=ASuiBAAAQBAJ). O'Reilly Media, Incorporated. p. 160. ISBN 9781491904442. Retrieved 2018-03-11.
12. "zuiderkwast/nanbox: NaN-boxing in C" (https://github.com/zuiderkwast/nanbox). *GitHub*. Retrieved 2018-03-11.
13. IEEE 754 2008, §3.6.
14. IEEE 754 2008, §3.7.
15. IEEE 754 2008, §3.7 states: "Language standards should define mechanisms supporting extendable precision for each supported radix."
16. IEEE 754 2008, §3.7 states: "Language standards or implementations should support an extended precision format that extends the widest basic format that is supported in that radix."
17. IEEE 754 2008, §4.3.1. "In the following two rounding-direction attributes, an infinitely precise result with magnitude at least $b^{\mathrm{emax}}(b - \frac{1}{2}b^{1-p})$ shall round to $\infty$ with no change in sign."
18. IEEE 754 2008, §4.3.3
19. IEEE 754 2019, §2.1
20. IEEE 754 2008, §5.3.1
21. IEEE 754 2008, §5.4.1
22. IEEE 754 2008, §5.4.2
23. IEEE 754 2008, §5.4.3
24. IEEE 754 2008, §5.3.2
25. IEEE 754 2008, §5.3.3
26. IEEE 754 2008, §5.5.1
27. IEEE 754 2008, §5.10
28. IEEE 754 2008, §5.11
29. IEEE 754 2008, §5.7.2
30. IEEE 754 2008, §5.7.4
31. IEEE 754 2019, §5.11
32. IEEE 754 2019, §5.10
33. Herf, Michael (December 2001). "radix tricks" (http://stereopsis.com/radix.html). *stereopsis: graphics*.
34. "9.4. decimal — Decimal fixed point and floating point arithmetic — Python 3.6.5 documentation" (https://docs.python.org/library/decimal.html#signals). *docs.python.org*. Retrieved 2018-04-04.
35. "Decimal Arithmetic - Exceptional conditions" (http://speleotrove.com/decimal/daexcep.html). *speleotrove.com*. Retrieved 2018-04-04.
36. IEEE 754 2008, §7.2(h)
37. Goldberg 1991.
38. Muller, Jean-Michel; Brisebarre, Nicolas; de Dinechin, Florent; Jeannerod, Claude-Pierre; Lefèvre, Vincent; Melquiond, Guillaume; Revol, Nathalie; Stehlé, Damien; Torres, Serge (2010). *Handbook of Floating-Point Arithmetic* (https://books.google.com/books?id=baFvrIOPvncC&pg=PA16) (1 ed.). Birkhäuser. doi:10.1007/978-0-8176-4705-6 (https://doi.org/10.1007%2F978-0-8176-4705-6). ISBN 978-0-8176-4704-9. LCCN 2009939668 (https://lccn.loc.gov/2009939668).
39. Kahan, William Morton; Darcy, Joseph (2001) [1998-03-01]. "How Java's floating-point hurts everyone everywhere" (http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf) (PDF). Archived (https://web.archive.org/web/20000816043653/http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf) (PDF) from the original on 2000-08-16. Retrieved 2003-09-05.
40. Kahan, William Morton (1981-02-12). "Why do we need a floating-point arithmetic standard?" (http://www.cs.berkeley.edu/~wkahan/ieee754status/why-ieee.pdf) (PDF). p. 26. Archived (https://web.archive.org/web/20041204070746/http://www.cs.berkeley.edu/~wkahan/ieee754status/why-ieee.pdf) (PDF) from the original on 2004-12-04.
41. Severance, Charles (1998-02-20). "An Interview with the Old Man of Floating-Point" (http://www.eecs.berkeley.edu/~wkahan/ieee754status/754story.html).

42. Kahan, William Morton (1996-06-11). "The Baleful Effect of Computer Benchmarks upon Applied Mathematics, Physics and Chemistry" (http://www.cs.berkeley.edu/~wkahan/ieee754status/baleful.pdf) (PDF). Archived (https://web.archive.org/web/20131013011212/http://www.cs.berkeley.edu/~wkahan/ieee754status/baleful.pdf) (PDF) from the original on 2013-10-13.
43. IEEE 754 2019, §9.2
44. IEEE 754 2008, Clause 9
45. IEEE 754 2019, §9.2.
46. "Re: Missing functions tanPi, asinPi and acosPi" (https://web.archive.org/web/20170706053605/http://grouper.ieee.org/groups/754/email/msg03842.html). *grouper.ieee.org*. Archived from the original (http://grouper.ieee.org/groups/754/email/msg03842.html) on 2017-07-06. Retrieved 2018-04-04.
47. IEEE 754 2008, §9.3.
48. IEEE 754 2008, §9.4.
49. IEEE 754 2019, §9.5
50. Riedy, Jason; Demmel, James. "Augmented Arithmetic Operations Proposed for IEEE-754 2018" (http://www.ecs.umass.edu/arith-2018/pdf/arith25_34.pdf) (PDF). 25th IEEE Symbosium on Computer Arithmetic (ARITH 2018). pp. 49–56. Archived (https://web.archive.org/web/20190723172615/http://www.ecs.umass.edu/arith-2018/pdf/arith25_34.pdf) (PDF) from the original on 2019-07-23. Retrieved 2019-07-23.
51. "754 Revision targeted for 2019" (http://754r.ucbtest.org/background/). *754r.ucbtest.org*. Retrieved 2019-07-23.
52. IEEE 754 2019, §9.6.
53. Chen, David. "The Removal of MinNum and MaxNum Operations from IEEE 754-2019" (http://grouper.ieee.org/groups/msc/ANSI_IEEE-Std-754-2019/background/minNum_maxNum_Removal_Demotion_v3.pdf) (PDF). *grouper.ieee.org*. Retrieved 2020-02-05.
54. IEEE 754 2008, §5.12.
55. IEEE 754 2008, §5.12.2.
56. Gay, David M. (1990-11-30), *Correctly rounded binary-decimal and decimal-binary conversions* (http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.4049), Numerical Analysis Manuscript, Murry Hill, NJ, US: AT&T Laboratories, 90-10
57. Paxson, Vern; Kahan, William (1991-05-22), *A Program for Testing IEEE Decimal–Binary Conversion*, Manuscript, CiteSeerX 10.1.1.144.5889 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.144.5889)
58. IEEE 754 2008, §5.12.3
59. "6.9.3. Hexadecimal floating point literals — Glasgow Haskell Compiler 9.3.20220129 User's Guide" (https://ghc.gitlab.haskell.org/ghc/doc/users_guide/exts/hex_float_literals.html). *ghc.gitlab.haskell.org*. Retrieved 2022-01-29.

### Standards

- *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE STD 754-1985. IEEE. 1985-10-12. pp. 1–20. doi:10.1109/IEEESTD.1985.82928 (https://doi.org/10.1109%2FIEEESTD.1985.82928). ISBN 0-7381-1165-1.
- IEEE Computer Society (2008-08-29). *IEEE Standard for Floating-Point Arithmetic*. IEEE STD 754-2008. IEEE. pp. 1–70. doi:10.1109/IEEESTD.2008.4610935 (https://doi.org/10.1109%2FIEEESTD.2008.4610935). ISBN 978-0-7381-5753-5. IEEE Std 754-2008.
- IEEE Computer Society (2019-07-22). *IEEE Standard for Floating-Point Arithmetic*. IEEE STD 754-2019. IEEE. pp. 1–84. doi:10.1109/IEEESTD.2019.8766229 (https://doi.org/10.1109%2FIEEESTD.2019.8766229). ISBN 978-1-5044-5924-2. IEEE Std 754-2019.
- ISO/IEC JTC 1/SC 25 (June 2011). *ISO/IEC/IEEE 60559:2011 — Information technology — Microprocessor Systems — Floating-Point arithmetic* (https://www.iso.org/standard/57469.html). ISO. pp. 1–58.
- ISO/IEC JTC 1/SC 25 (May 2020). *ISO/IEC 60559:2020 — Information technology — Microprocessor Systems — Floating-Point arithmetic* (https://www.iso.org/standard/80985.html). ISO. pp. 1–74.

### Secondary references

- Decimal floating-point (http://speleotrove.com/decimal) arithmetic, FAQs, bibliography, and links
- Comparing binary floats (http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm)
- IEEE 754 Reference Material (http://babbage.cs.qc.cuny.edu/IEEE-754.old/References.xhtml)
- IEEE 854-1987 (http://speleotrove.com/decimal/854mins.html) – History and minutes
- Supplementary readings for IEEE 754 (https://web.archive.org/web/20171230124220/http://grouper.ieee.org/groups/754/reading.html). Includes historical perspectives.

## Further reading

- Goldberg, David (March 1991). "What Every Computer Scientist Should Know About Floating-Point Arithmetic" (http://perso.ens-lyon.fr/jean-michel.muller/goldberg.pdf) (PDF). *ACM Computing Surveys*. **23** (1): 5–48. doi:10.1145/103162.103163 (https://doi.org/10.1145%2F103162.103163). S2CID 222008826 (https://api.semanticscholar.org/CorpusID:222008826). Archived (https://web.archive.org/web/20060720140912/http://perso.ens-lyon.fr/jean-michel.muller/goldberg.pdf) (PDF) from the original on 2006-07-20. Retrieved 2016-01-20. ([1] (http://www.validlab.com/goldberg/paper.pdf), [2] (http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html), [3] (http://www.cse.msu.edu/~cse320/Documents/FloatingPoint.pdf))
- Hecker, Chris (February 1996). "Let's Get To The (Floating) Point" (http://chrishecker.com/images/f/fb/Gdmfp.pdf) (PDF). *Game Developer Magazine*: 19–24. ISSN 1073-922X (https://www.worldcat.org/issn/1073-922X).
- Severance, Charles (March 1998). "IEEE 754: An Interview with William Kahan" (http://www.dr-chuck.com/dr-chuck/papers/columns/r3114.pdf) (PDF). *IEEE Computer*. **31** (3): 114–115. doi:10.1109/MC.1998.660194 (https://doi.org/10.1109%2FMC.1998.660194). S2CID 33291145 (https://api.semanticscholar.org/CorpusID:33291145). Retrieved 2019-03-08.

- Cowlishaw, Mike (June 2003). "Decimal floating-point: Algorism for computers". *16th IEEE Symposium on Computer Arithmetic, 2003. Proceedings* (http://speleotrove.com/decimal/IEEE-cowlishaw-arith16.pdf) (PDF). Los Alamitos, Calif.: IEEE Computer Society. pp. 104–111. doi:10.1109/ARITH.2003.1207666 (https://doi.org/10.1109%2FARITH.2003.1207666). ISBN 978-0-7695-1894-7. S2CID 18713046 (https://api.semanticscholar.org/CorpusID:18713046). Retrieved 2014-11-14.. (Note: *Algorism* is not a misspelling of the title; see also algorism.)
- Monniaux, David (May 2008). "The pitfalls of verifying floating-point computations" (https://hal.science/hal-00128124/en/). *ACM Transactions on Programming Languages and Systems*. **30** (3): 1–41. arXiv:cs/0701192 (https://arxiv.org/abs/cs/0701192). doi:10.1145/1353445.1353446 (https://doi.org/10.1145%2F1353445.1353446). ISSN 0164-0925 (https://www.worldcat.org/issn/0164-0925). S2CID 218578808 (https://api.semanticscholar.org/CorpusID:218578808).: A compendium of non-intuitive behaviours of floating-point on popular architectures, with implications for program verification and testing.
- Muller, Jean-Michel; Brunie, Nicolas; de Dinechin, Florent; Jeannerod, Claude-Pierre; Joldes, Mioara; Lefèvre, Vincent; Melquiond, Guillaume; Revol, Nathalie; Torres, Serge (2018) [2010]. *Handbook of Floating-Point Arithmetic* (https://cds.cern.ch/record/1315760) (2 ed.). Birkhäuser. doi:10.1007/978-3-319-76526-6 (https://doi.org/10.1007%2F978-3-319-76526-6). ISBN 978-3-319-76525-9.
- Overton, Michael L. (2001). Written at Courant Institute of Mathematical Sciences, New York University, New York, US. *Numerical Computing with IEEE Floating Point Arithmetic* (1 ed.). Philadelphia, US: SIAM. doi:10.1137/1.9780898718072 (https://doi.org/10.1137%2F1.9780898718072). ISBN 978-0-89871-482-1. 978-0-89871-571-2, 0-89871-571-7.
- Cleve Moler on Floating Point numbers (http://blogs.mathworks.com/cleve/2014/07/07/floating-point-numbers/)
- Beebe, Nelson H. F. (2017-08-22). *The Mathematical-Function Computation Handbook - Programming Using the MathCW Portable Software Library* (1 ed.). Salt Lake City, UT, US: Springer International Publishing AG. doi:10.1007/978-3-319-64110-2 (https://doi.org/10.1007%2F978-3-319-64110-2). ISBN 978-3-319-64109-6. LCCN 2017947446 (https://lccn.loc.gov/2017947446). S2CID 30244721 (https://api.semanticscholar.org/CorpusID:30244721).
- Hough, David G. (December 2019). "The IEEE Standard 754: One for the History Books" (https://www.computer.org/csdl/magazine/co/2019/12/08909942/1f8KFWxbTCU). *Computer*. IEEE. **52** (12): 109–112. doi:10.1109/MC.2019.2926614 (https://doi.org/10.1109%2FMC.2019.2926614). S2CID 208281213 (https://api.semanticscholar.org/CorpusID:208281213).

## External links

- *Kahan on creating IEEE Standard Floating Point* (https://www.youtube.com/watch?v=ATCpecsyPE8). *Turing Awardee Clips*. 2020-11-16. Archived (https://ghostarchive.org/varchive/youtube/20211108/ATCpecsyPE8) from the original on 2021-11-08.
- Online IEEE 754 binary calculators (https://babbage.cs.qc.cuny.edu/IEEE-754/)