WIKIPEDIA

# 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.*)

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

## Contents

**External links**

# Standard development

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 JTC1/SC 25 under the ISO/IEEE PSDO Agreement[2] and published.[3]

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 and errata, but also includes some new recommended operations.[4][5]

# 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.[6]

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 $-12345\times10^{-3}$, $-123450\times10^{-4}$, and $-1234500\times10^{-5}$. 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 not all ones or 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*[7][8][9]).

## 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.[10] For the binary formats, the leading bit convention is required. The following table summarizes the smallest interchange formats (including the basic ones).

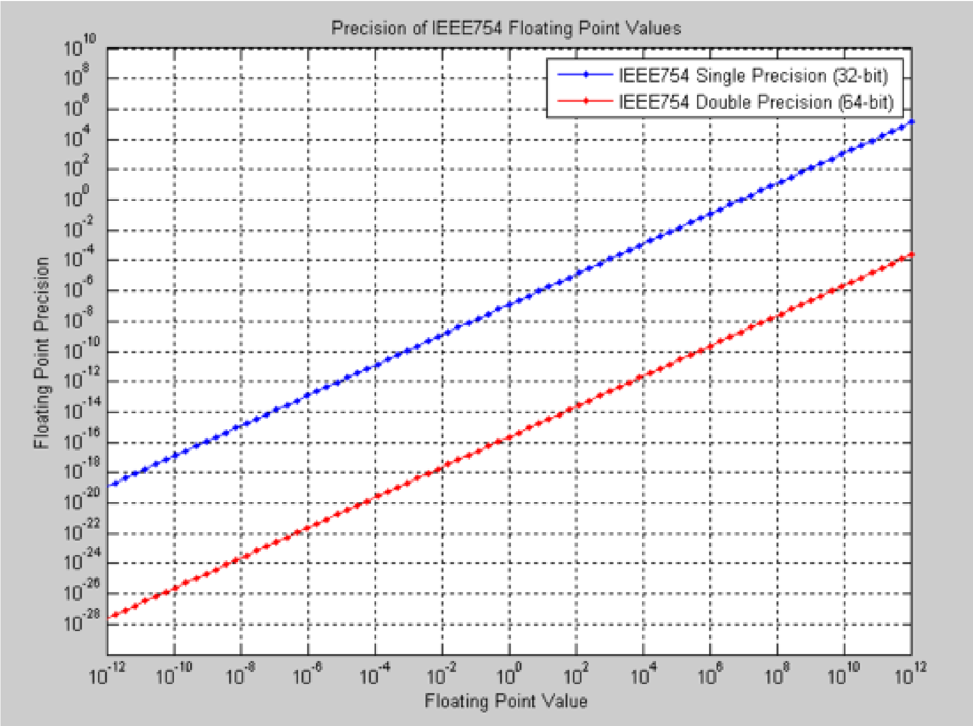
| Name | Common name | Base | Significand bits[b] or digits | Decimal digits | Exponent bits | Decimal E max | Exponent bias[11] | E min | E max | Notes |
|---|---|---|---|---|---|---|---|---|---|---|
| binary16 | Half precision | 2 | 11 | 3.31 | 5 | 4.51 | $2^4-1 = 15$ | −14 | +15 | not basic |
| binary32 | Single precision | 2 | 24 | 7.22 | 8 | 38.23 | $2^7-1 = 127$ | −126 | +127 | |
| binary64 | Double precision | 2 | 53 | 15.95 | 11 | 307.95 | $2^{10}-1 = 1023$ | −1022 | +1023 | |
| binary128 | Quadruple precision | 2 | 113 | 34.02 | 15 | 4931.77 | $2^{14}-1 = 16383$ | −16382 | +16383 | |
| binary256 | Octuple precision | 2 | 237 | 71.34 | 19 | 78913.2 | $2^{18}-1 = 262143$ | −262142 | +262143 | not basic |
| decimal32 | | 10 | 7 | 7 | 7.58 | 96 | 101 | −95 | +96 | not basic |
| decimal64 | | 10 | 16 | 16 | 9.58 | 384 | 398 | −383 | +384 | |
| decimal128 | | 10 | 34 | 34 | 13.58 | 6144 | 6176 | −6143 | +6144 | |

Note that in the table above, the minimum exponents listed are for normal numbers; the special subnormal number representation allows even smaller 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 E min value −1022 and all but one of the 53 significand bits ($2^{-1022 - (53 - 1)} = 2^{-1074}$).

Decimal digits is *digits* × $\log_{10}$ *base*. This gives an approximate precision in number of decimal digits.

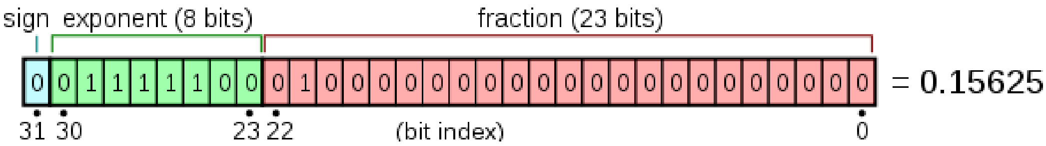
Decimal E max is *Emax* × $\log_{10}$ *base*. This gives an approximate value of the maximum decimal exponent.

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.[12] 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 $b$.[13] 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 $b$.[14] 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 fixed-length bit-string for a given format.

For the exchange of binary floating-point numbers, interchange formats of length 16 bits, 32 bits, 64 bits, and any multiple of 32 bits ≥128 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) 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 (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), and the payload is carried in the remaining bits.

For the exchange of decimal floating-point numbers, interchange formats of any multiple of 32 bits are defined.

The encoding scheme for the decimal interchange formats similarly encodes the sign, exponent, and significand, but two different bit-level representations are defined. Interchange is complicated by the fact that some external indicator of the representation in use is 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, ±infinity, quiet NaNs, and signaling NaNs) have identical binary representations.

# 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; this is the default for binary floating point and the recommended default for decimal.
- **Round to nearest, ties away from zero** – 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); this is intended as an option for decimal floating point.

## 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, for the operations defined by the standard, the rounding must be applied on the exact, i.e. mathematical, value of the operation (the inputs are regarded as exact): the behavior must be as if the result of the operation were computed exactly, then rounded according to the chosen rounding rule. This requirement is called *correct rounding*.

# Required operations

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

- Arithmetic operations (add, subtract, multiply, divide, square root, fused multiply–add, remainder)[15][16]
- Conversions (between formats, to and from strings, *etc.*)[17][18]
- Scaling and (for decimal) quantizing[19][20]
- Copying and manipulating the sign (abs, negate, *etc.*)[21]
- Comparisons and total ordering[22][23]
- Classification and testing for NaNs, *etc.*[24]
- Testing and setting flags[25]
- Miscellaneous operations.

## Total-ordering predicate

The standard provides a predicate *totalOrder* which defines a total ordering for all floating-point data for each format. The predicate agrees with the normal comparison operations when one floating-point number is less than the other one. The normal comparison operations, however, treat NaNs as unordered and compare −0 and +0 as equal. The *totalOrder* predicate orders all floating-point data strictly and totally. When comparing two floating-point numbers, it acts as the ≤ operation, except that $\mathrm{totalOrder}(-0, +0) \wedge \neg\ \mathrm{totalOrder}(+0, -0)$, and different representations of the same floating-point number are ordered by their exponent multiplied by the sign bit. The ordering is then extended to the NaNs by ordering $-\mathrm{qNaN} < -\mathrm{sNaN} < \mathrm{numbers} < +\mathrm{sNaN} < +\mathrm{qNaN}$, with ordering between two NaNs in the same class being based on the integer payload, multiplied by the sign bit, of those data.[22]

# Exception handling

The standard defines five exceptions, each of which returns a default value and has a corresponding status flag that (except in certain cases of underflow) is raised when the exception occurs. No other exception handling is required, but additional non-default alternatives are recommended (see below).

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 result is too large to be represented correctly (*i.e.*, its exponent with an unbounded exponent range would be larger than *emax*). By default, returns ±infinity for the round-to-nearest mode (and

follows the rounding rules for the directed rounding modes).
- Underflow: a result is very small (outside the normal range) and is inexact. By default, returns a subnormal or zero (following the rounding rules).
- 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.

For decimal floating point, there are additional exceptions along with the above:[26][27]

- 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 overflow occurs.
- Rounded: a result's coefficient requires more digits than the destination format provides. The inexact 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.[28]

# 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 which 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 50 operations that language standards should define.[29] These are all optional (not required in order to conform to the standard).

Recommended arithmetic operations, which must round correctly:[30]

- $e^x$, $2^x$, $10^x$
- $e^x - 1$, $2^x - 1$, $10^x - 1$
- $\ln x$, $\log_2 x$, $\log_{10} x$
- $\ln(1+x)$, $\log_2(1+x)$, $\log_{10}(1+x)$
- $\sqrt{x^2+y^2}$
- $\sqrt{x}$
- $(1+x)^n$
- $x^{\frac{1}{n}}$
- $x^n$, $x^y$
- $\sin x$, $\cos x$, $\tan x$
- $\arcsin x$, $\arccos x$, $\arctan x$, $\operatorname{atan2}(y,x)$
- $\operatorname{sinPi} x = \sin \pi x$, $\operatorname{cosPi} x = \cos \pi x$, $\operatorname{tanPi} x = \tan \pi x$ (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 asinPi, acosPi and tanPi functions were not part of the IEEE 754-2008 standard because the feeling was that they were less necessary.[31] The first two were at least mentioned in a paragraph, but this was regarded as an error[32] until they were added in the 2019 revision.

The operations also include setting and accessing dynamic mode rounding direction,[33] and implementation-defined vector reduction operations such as sum, scaled product, and dot product, whose accuracy is unspecified by the standard.[34]

As of 2019, *augmented arithmetic operations* 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.[35][36]

### 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 a single 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 allowed many variations in implementations (such as the encoding of some values and the detection of certain exceptions). IEEE 754-2008 has strengthened up many of these, 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.[37] 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 rounding to even will recover the original number. There is no requirement to preserve the payload of a 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:[38]

- 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

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

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

(Note: as an implementation limit, correct rounding is only guaranteed for the number of decimal digits above 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.)

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,[39] and for testing – by Paxson and Kahan.[40]

## See also

- 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 mantissa 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, a keyword in the Java programming language that restricts arithmetic to IEEE 754 single and double precision to ensure reproducibility across common hardware platforms.
- The table-maker's dilemma for more about the correct rounding of functions.
- Tapered floating point.

## Notes

1. 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 $(-1)^1 \times 12345 \times 10^{-3} = -1 \times 12345 \times 0.001 = -12.345$.
2. including the implicit bit (which always equals 1 for normal numbers, and 0 for subnormal numbers. This implicit bit is not stored in memory), but not the sign bit.

## References


1. "IEEE 754-2019 - IEEE Standard for Floating-Point Arithmetic" (https://standards.ieee.org/content/ieee-standards/en/standard/754-2019.html). *standards.ieee.org*. Retrieved 2019-07-23.
2. "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/IEC/IEEE 60559:2011 - Information technology -- Microprocessor Systems -- Floating-Point arithmetic" (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57469). *www.iso.org*. Retrieved 2018-04-04.

4. "IEEE 754-2008 errata" (https://web.archive.org/web/20191008011711/http://speleotrove.com/misc/IEEE754-errata.html). *speleotrove.com*. Archived from the original (http://speleotrove.com/misc/IEEE754-errata.html) on 2019-10-08. Retrieved 2018-04-04.
5. "Revising ANSI/IEEE Std 754-2008" (http://754r.ucbtest.org/). *ucbtest.org*. Retrieved 2018-04-04.
6. IEEE 754 2008, §2.1.27.
7. "SpiderMonkey Internals" (https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey/Internals). *developer.mozilla.org*. Retrieved 2018-03-11.
8. Klemens, Ben (September 2014). *21st Century C: C Tips from the New School* (https://books.google.fr/books?id=ASuiBAAAQBAJ). O'Reilly Media, Incorporated. p. 160. ISBN 9781491904442. Retrieved 2018-03-11.
9. "zuiderkwast/nanbox: NaN-boxing in C" (https://github.com/zuiderkwast/nanbox). *GitHub*. Retrieved 2018-03-11.
10. IEEE 754 2008, §3.6.
11. Cowlishaw, Mike. "Decimal Arithmetic Encodings" (http://speleotrove.com/decimal/decbits.pdf) (PDF). IBM. Retrieved 2015-08-06.
12. IEEE 754 2008, §3.7.
13. IEEE 754 2008, §3.7 states: "Language standards should define mechanisms supporting extendable precision for each supported radix."
14. 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."
15. IEEE 754 2008, §5.3.1
16. IEEE 754 2008, §5.4.1
17. IEEE 754 2008, §5.4.2
18. IEEE 754 2008, §5.4.3
19. IEEE 754 2008, §5.3.2
20. IEEE 754 2008, §5.3.3
21. IEEE 754 2008, §5.5.1
22. IEEE 754 2008, §5.10
23. IEEE 754 2008, §5.11
24. IEEE 754 2008, §5.7.2
25. IEEE 754 2008, §5.7.4
26. "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.
27. "Decimal Arithmetic - Exceptional conditions" (http://speleotrove.com/decimal/daexcep.html). *speleotrove.com*. Retrieved 2018-04-04.
28. IEEE 754 2008, §7.2(h)
29. IEEE 754 2008, Clause 9.
30. IEEE 754 2008, §9.2.
31. "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.
32. Cowlishaw, Mike, *IEEE 754-2008 errata 2013.12.19* (http://speleotrove.com/misc/IEEE754-errata.html).
33. IEEE 754 2008, §9.3.
34. IEEE 754 2008, §9.4.
35. 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.
36. "754 Revision targeted for 2019" (http://754r.ucbtest.org/background/). *754r.ucbtest.org*. Retrieved 2019-07-23.
37. IEEE 754 2008, §5.12.
38. IEEE 754 2008, §5.12.2.

39. 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, USA: AT&T Laboratories. 90-10.
40. 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).

### Standard

- IEEE Computer Society (2008-08-29). *IEEE Standard for Floating-Point Arithmetic* (https://ieeexplore.ieee.org/document/4610935). *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.
- ISO/IEC/IEEE 60559:2011 (https://www.iso.org/standard/57469.html)

### 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" (https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html). *ACM Computing Surveys*. **23** (1): 5–48. doi:10.1145/103162.103163 (https://doi.org/10.1145%2F103162.103163). Retrieved 2019-03-08.
- 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). Retrieved 2019-03-08.
- Cowlishaw, Mike (June 2003). *Decimal Floating-Point: Algorism for Computers* (http://speleotrove.com/decimal/IEEE-cowlishaw-arith16.pdf) (PDF). *Proceedings 16th IEEE Symposium on Computer Arithmetic*. Los Alamitos, Calif.: IEEE Computer Society. pp. 104–111. ISBN 978-0-7695-1894-7. 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" (http://hal.archives-ouvertes.fr/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).: 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* (http://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, USA. *Numerical Computing with IEEE Floating Point Arithmetic* (1 ed.). Philadelphia, USA: 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, USA: 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).

## External links

- IEEE pages: 754-1985 - IEEE Standard for Binary Floating-Point Arithmetic (https://ieeexplore.ieee.org/document/30711), 754-2008 - IEEE Standard for Floating-Point Arithmetic (https://ieeexplore.ieee.org/document/4610935)
- Online IEEE 754 binary calculators (https://babbage.cs.qc.cuny.edu/IEEE-754/)