IEEE floating point

The **IEEE Standard for Floating-Point Arithmetic** (**IEEE 754**) is a technical standard for floating-point computation established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). Many hardware floating point units use the IEEE 754 standard. The current version, **IEEE 754-2008** published in August 2008, includes nearly all of the original IEEE 754-1985 standard and the IEEE Standard for Radix-Independent Floating-Point Arithmetic (IEEE 854-1987). The international standard **ISO/IEC/IEEE 60559:2011** (with identical content to IEEE 754) has been approved for adoption through JTC1/SC 25 under the ISO/IEEE PSDO Agreement[1] and published.[2]

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 on arithmetic formats
- *exception handling:* indications of exceptional conditions (such as division by zero, overflow, *etc.*)

The standard also includes extensive recommendations for advanced exception handling, additional operations (such as trigonometric functions), expression evaluation, and for achieving reproducible results.

The standard is derived from and replaces IEEE 754-1985, the previous version, following a seven-year revision process, chaired by Dan Zuras and edited by Mike Cowlishaw. The binary formats in the original standard are included in the 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.

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.

A format comprises:

- Finite numbers, which may be either base 2 (binary) or base 10 (decimal). Each finite number is described by three integers: s = a *sign* (zero or one), c = a *significand* (or 'coefficient'), q = an *exponent*. The numerical value of a finite number is
  $(-1)^s \times c \times b^q$
  where b is the base (2 or 10). For example, if the sign is 1 (indicating negative), the significand is 12345, the exponent is −3, and the base is 10, then the value of the number is −12.345.
- Two infinities: +∞ and −∞.
- Two kinds of NaN: a quiet NaN (qNaN) and a signaling NaN (sNaN). A NaN may carry a *payload* that is intended for diagnostic information indicating the source of the NaN. The sign of a NaN has no meaning, but it may be predictable in some circumstances.

The possible finite values that can be represented in a format are determined by the base (b), the number of digits in the significand (precision, p), and the exponent parameter *emax*:

- c must be an integer in the range zero through b^p-1 (*e.g.*, if b=10 and p=7 then c is 0 through 9999999)
- q must be an integer such that $1-emax \le q+p-1 \le emax$ (*e.g.*, if p=7 and *emax*=96 then q is −101 through 90).

Hence (for the example parameters) the smallest non-zero positive number that can be represented is 1×10^{-101} and the largest is 9999999×10^{90} (9.999999×10^{96}), and the full range of numbers is -9.999999×10^{96} through 9.999999×10^{96}. The numbers $-b^{1-emax}$ and b^{1-emax} (here, -1×10^{-95} and 1×10^{-95}) are the smallest (in magnitude) *normal numbers*; non-zero numbers between these smallest numbers are called subnormal numbers.

Zero values are finite values with significand 0. These are signed zeros, the sign bit specifies if a zero is +0 (positive zero) or −0 (negative zero).

Basic 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. A conforming implementation must fully implement at least one of the basic formats.

The typical precision of the basic binary formats is one bit more than the width of its significand. The extra bit of precision comes from an implied (hidden) leading *1* bit. The typical floating point number will be normalized such that the most significant bit will be a one. If the leading bit is known to be one, then it need not be encoded in the interchange format.

Name	Common name	Base	Digits	E min	E max	Notes	Decimal digits	Decimal E max
binary16	Half precision	2	10+1	−14	+15	storage, not basic	3.31	4.51
binary32	Single precision	2	23+1	−126	+127		7.22	38.23
binary64	Double precision	2	52+1	−1022	+1023		15.95	307.95
binary128	Quadruple precision	2	112+1	−16382	+16383		34.02	4931.77
decimal32		10	7	−95	+96	storage, not basic	7	96
decimal64		10	16	−383	+384		16	384
decimal128		10	34	−6143	+6144		34	6144

Decimal digits is $digits \times \log_{10} base$, this gives an approximate precision in decimal.

Decimal E max is $Emax \times \log_{10} base$, this gives the maximum exponent in decimal.

Extended and extendable precision formats

The standard specifies extended and extendable precision formats, which are recommended for allowing a greater precision than that provided by the basic formats.[3] 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 does not require an implementation to support extended or extendable precision formats.

The standard recommends that languages provide a method of specifying *p* and *emax* for each supported base *b*.[4]

The standard recommends that languages and implementations support an extended format which has a greater precision than the largest basic format supported for each radix *b*.[5]

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 $w = \text{floor}(4 \log2(k))-13$. 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, there is some flexibility in the encoding of signaling NaN.

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 the scheme uses a more complex approach to allow the significand to be encoded as a compressed sequence of decimal digits (using densely packed decimal) or as a binary integer. In either case the set of numbers (combinations of sign, significand, and exponent) that may be encoded is identical, and signaling NaNs have a unique encoding (and the same set of possible payloads).

Rounding rules

The standard defines five rounding rules. The first two 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 (zero) least significant bit, which occurs 50% of the time; 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*).

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, *etc.*)
- Conversions (between formats, to and from strings, *etc.*)
- Scaling and (for decimal) quantizing
- Copying and manipulating the sign (abs, negate, *etc.*)
- Comparisons and total ordering
- Classification and testing for NaNs, *etc.*

- Testing and setting flags
- Miscellaneous operations.

Total-ordering predicate

The standard provides a predicate *totalOrder* which defines a total ordering for all floating numbers for each format. The predicate agrees with the normal comparison operations when they say one floating point number is less than another. The normal comparison operations however treat NaNs as unordered and compare −0 and +0 as equal. The totalOrder predicate will order these cases, and it also distinguishes between different representations of NaNs and between the same decimal floating point number encoded in different ways.

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 (*e.g.*, square root of a negative number) (returns qNaN by default).
- Division by zero (an operation on finite operands gives an exact infinite result, *e.g.*, 1/0 or log(0)) (returns ±infinity by default).
- Overflow (a result is too large to be represented correctly) (returns ±infinity by default (for round-to-nearest mode)).
- Underflow (a result is very small (outside the normal range) and is inexact) (returns a denormalized value by default).
- Inexact (returns correctly rounded result by default).

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:[6][7]

- 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 is 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.[8]

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 fifty operations, including log, power, and trigonometric functions, that language standards should define.[9] These are all optional (none are required in order to conform to the standard). The operations include setting and accessing dynamic mode rounding direction,[10] and vector reduction operations such as sum, scaled product, and dot product.[11] Conforming implementations must return correctly rounded results depending on the active rounding mode. The inexact exception need not be set correctly, however the other exceptions must be set as specified.

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 behaviour 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 doubled 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 tightened 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.[12] 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 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:[13]

- 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

$$1 + \text{ceiling}(p \times \log_{10} 2)$$

where p is the number of significant bits in the binary format, e.g. 24 bits for binary32.

(Note: as an implementation limit, correct rounding is only guaranteed for the number of decimal digits above plus 3 for the largest binary format supported. For instance if binary32 is the largest supported binary format supported, 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 in [14] and for testing in.[15]

References

[1] FW: ISO/IEC/IEEE 60559 (IEEE Std 754-2008) (http://grouper.ieee.org/groups/754/email/msg04167.html)

[2] 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)

[3] IEEE 754 2008, §3.7

[4] IEEE 754 2008, §3.7 states, "Language standards should define mechanisms supporting extendable precision for each supported radix."

[5] 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."

[6] 9.4. decimal — Decimal fixed point and floating point arithmetic — Python v2.7.3 documentation (http://docs.python.org/library/decimal.html#signals)

[7] Decimal Arithmetic - Exceptional conditions (http://speleotrove.com/decimal/daexcep.html)

[8] IEEE 2008, §7.2(h)

[9] IEEE 754 2008, Clause 9

[10] IEEE 754 2008, §9.3

[11] IEEE 754 2008, §9.4

[12] IEEE 754 2008, §5.12

[13] IEEE 754 2008, §5.12.2

[14] Gay, David M. (November 30, 1990). *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.

[15] Paxson, Vern; Kahn, William (May 22, 1991). "A Program for Testing IEEE Decimal–Binary Conversion" (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.144.5889). Manuscript. . Retrieved March 28, 2012

Standard

- IEEE Computer Society (August 29, 2008). *IEEE Standard for Floating-Point Arithmetic* (http://ieeexplore.ieee.org/servlet/opac?punumber=4610933). IEEE. doi:10.1109/IEEESTD.2008.4610935. ISBN 978-0-7381-5753-5. IEEE Std 754-2008
- ISO/IEC/IEEE 60559:2011 (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57469)

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 (http://grouper.ieee.org/groups/754/reading.html). Includes historical perspectives.

Further reading

- David Goldberg (March 1991). "What Every Computer Scientist Should Know About Floating-Point Arithmetic" (http://www.validlab.com/goldberg/paper.pdf). *ACM Computing Surveys* **23** (1): 5–48. doi:10.1145/103162.103163. Retrieved 28 April 2008.
- Chris Hecker (February 1996). "Let's Get To The (Floating) Point" (http://chrishecker.com/images/f/fb/Gdmfp.pdf). *Game Developer Magazine*: 19–24. ISSN 1073-922X.
- Charles Severance (March 1998). "IEEE 754: An Interview with William Kahan" (http://www.freecollab.com/dr-chuck/papers/columns/r3114.pdf). *IEEE Computer* **31** (3): 114–115. doi:10.1109/MC.1998.660194. Retrieved 28 April 2008.
- Mike Cowlishaw (June 2003). "Decimal Floating-Point: Algorism for Computers" (http://www.ece.ucdavis.edu/acsel/arithmetic/arith16/papers/ARITH16_Cowlishaw.pdf). *Proceedings 16th IEEE Symposium on Computer Arithmetic* (Los Alamitos, Calif.: IEEE Computer Society): 104–111. ISBN 0-7695-1894-X. Retrieved 31 December 2008.. (Note: *Algorism* is not a misspelling of the title; see also algorism.)
- David Monniaux (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): article #12. doi:10.1145/1353445.1353446. ISSN 0164-0925.: A compendium of non-intuitive behaviours of floating-point on popular architectures, with implications for program verification and testing.
- Michael L. Overton (2001). *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM.

External links

- Online IEEE 754 binary calculators (http://babbage.cs.qc.cuny.edu/IEEE-754/)

Article Sources and Contributors

IEEE floating point *Source*: http://en.wikipedia.org/w/index.php?oldid=538620431 *Contributors*: !Silent, 1exec1, 2001:250:4001:315:E9B7:A13E:B7F7:5A8F, AndrewKepert, Arjun G. Menon, AzraelUK, BBCWatcher, BLueFiSH.as, Batman2000, Betacommand, Brianbjparker, C. A. Russell, CALR, CRGreathouse, Cabyd, Ccvickery, CesarB, Chareverie, Charles Esson, Chris the speller, Copyeditor42, Coredesat, Daavan42, Damicatz, David.Monniaux, DavidWBrooks, Dicklyon, Dictoon, Djcam, Dmcq, EAderhold, ENeville, EdJohnston, Efa, Ekevu, Erud, Everyking, Fox Wilson, Frencheigh, Glrx, Goffrie, Guy Macon, Hairy Dude, HappyInGeneral, Harej, Hpa, JLaTondre, JakeVortex, Japaget, Jason Quinn, Jengelh, Jenks24, Jfgrcar, Jim1138, KelleyCook, Kenb215, Korval, Krich, Kvng, LOL, Lannm, Levin, LilHelpa, Lugia2453, MER-C, MIT Trekkie, Macrakis, MagnusA, MarkSweep, Markerle, Mecanismo, Mfc, Miym, MoraSique, MovGP0, Msnicki, Nasa-verve, Nczempin, NeonMerlin, Nixeagle, Omicronpersei8, Patrick, Quota, Qwerty112233, Rilak, Ruud Koot, Sagaciousuk, Salvidrim, Sam Hocevar, Someone13, Stevenjames53, Suruena, Swat671, Technion, TenOfAllTrades, Thenickdude, Theopolisme, Titodutta, TonyW, Torc2, Tubezone, UU, Urhixidur, Vincent Lefèvre, Wdwd, Wrs1864, Wtshymanski, Xaosflux, Zzyzx11, 168 anonymous edits

License