Double-precision floating-point format

In computing, **double precision** is a computer number format that occupies two adjacent storage locations in computer memory. A **double-precision number**, sometimes simply called a **double**, may be defined to be an integer, fixed point, or floating point (in which case it is often referred to as **FP64**).

Modern computers with 32-bit storage locations use two memory locations to store a 64-bit double-precision number (a single storage location can hold a single-precision number). *Double-precision floating-point* is an IEEE 754 standard for encoding binary or decimal floating-point numbers in 64 bits (8 bytes).

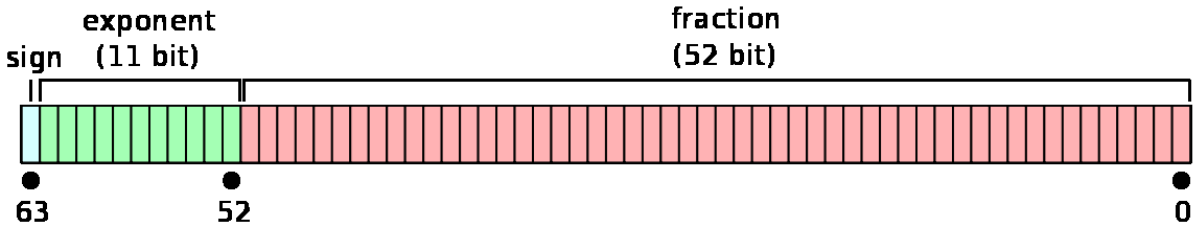
IEEE 754 double-precision binary floating-point format: binary64

Double-precision binary floating-point is a commonly used format on PCs, due to its wider range over single-precision floating point, in spite of its performance and bandwidth cost. As with single-precision floating-point format, it lacks precision on integer numbers when compared with an integer format of the same size. It is commonly known simply as *double*. The IEEE 754 standard specifies a **binary64** as having:

- Sign bit: 1 bit
- Exponent width: 11 bits
- Significand precision: 53 bits (52 explicitly stored)

This gives from 15 - 17 significant decimal digits precision. If a decimal string with at most 15 significant decimal is converted to IEEE 754 double precision and then converted back to the same number of significant decimal, then the final string should match the original; and if an IEEE 754 double precision is converted to a decimal string with at least 17 significant decimal and then converted back to double, then the final number must match the original.[1]

The format is written with the significand having an implicit integer bit of value 1, unless the written exponent is all zeros. With the 52 bits of the fraction significand appearing in the memory format, the total precision is therefore 53 bits (approximately 16 decimal digits, 53 $\log_{10}(2) \approx 15.955$). The bits are laid out as follows:



The real value assumed by a given 64-bit double-precision data with a given biased exponent **e** and a 52-bit fraction is $= (-1)^{\text{sign}}(1.b_{-1}b_{-2}...b_{-52})_2 \times 2^{e-1023}$ or more precisely:

$$\text{value} = (-1)^{\text{sign}}(1 + \sum_{i=1}^{52} b_{-i}2^{-i}) \times 2^{(e-1023)}$$

Between 2^{52}=4,503,599,627,370,496 and 2^{53}=9,007,199,254,740,992 the representable numbers are exactly the integers. For the next range, from 2^{53} to 2^{54}, everything is multiplied by 2, so the representable numbers are the even ones, etc. Conversely, for the previous range from 2^{51} to 2^{52}, the spacing is 0.5, etc.

The spacing as a fraction of the numbers in the range from 2^n to 2^{n+1} is 2^{n-52}. The maximum relative rounding error when rounding a number to the nearest representable one (the machine epsilon) is therefore 2^{-53}.

Exponent encoding

The double-precision binary floating-point exponent is encoded using an offset-binary representation, with the zero offset being 1023; also known as exponent bias in the IEEE 754 standard. Examples of such representations would be:

- E_{min} (1) = −1022
- E (50) = −973
- E_{max} (2046) = 1023

Thus, as defined by the offset-binary representation, in order to get the true exponent the exponent bias of 1023 has to be subtracted from the written exponent.

The exponents 000_{16} and $7ff_{16}$ have a special meaning:

- 000_{16} is used to represent zero (if M=0) and subnormals (if M≠0); and
- $7ff_{16}$ is used to represent ∞ (if M=0) and NaNs (if M≠0),

where M is the fraction mantissa. All bit patterns are valid encoding.

Except for the above exceptions, the entire double-precision number is described by:

$$(-1)^{\text{sign}} \times 2^{\text{exponent}-\text{exponent bias}} \times 1.\text{mantissa}$$

Double-precision examples

$3ff0\ 0000\ 0000\ 0000_{16} = 1$

$3ff0\ 0000\ 0000\ 0001_{16} \approx 1.0000000000000002$, the smallest number > 1

$3ff0\ 0000\ 0000\ 0002_{16} \approx 1.0000000000000004$

$4000\ 0000\ 0000\ 0000_{16} = 2$

$c000\ 0000\ 0000\ 0000_{16} = -2$

$0000\ 0000\ 0000\ 0001_{16} = 2^{-1022-52}$

$\approx 5 \times 10^{-324}$ (Min subnormal positive double, has a precision of only one bit, i.e. $\pm 2^{-1075}$)

$000f\ ffff\ ffff\ ffff_{16} = 2^{-1022} - 2^{-1022-52}$

$\approx 2.2250738585072009 \times 10^{-308}$ (Max subnormal double)

$0010\ 0000\ 0000\ 0000_{16} = 2^{-1022}$

$\approx 2.2250738585072014 \times 10^{-308}$ (Min normal positive double)

$7fef\ ffff\ ffff\ ffff_{16} = (1 + (1 - 2^{-52})) \times 2^{1023}$

$\approx 1.7976931348623157 \times 10^{308}$ (Max Double)

$0000\ 0000\ 0000\ 0000_{16} = 0$

$8000\ 0000\ 0000\ 0000_{16} = -0$

$7ff0\ 0000\ 0000\ 0000_{16} = \infty$

$fff0\ 0000\ 0000\ 0000_{16} = -\infty$

$3fd5\ 5555\ 5555\ 5555_{16} \approx 1/3$

(1/3 rounds down instead of up like single precision, because of the odd number of bits in the significand.)

In more detail:

Given the hexadecimal representation $3FD5\ 5555\ 5555\ 5555_{16}$,

Sign = 0

Exponent = $3FD_{16}$ = 1021

Exponent Bias = 1023 (constant value; see above)

Significand = $5\ 5555\ 5555\ 5555_{16}$

$$
\begin{aligned}
\text{Value} &= 2^{(\text{Exponent} - \text{Exponent Bias})} \times 1.\text{Significand} - \text{Note the Significand must not be converted to decimal here} \\
&= 2^{-2} \times (15\ 5555\ 5555\ 5555_{16} \times 2^{-52}) \\
&= 2^{-54} \times 15\ 5555\ 5555\ 5555_{16} \\
&= 0.333333333333333314829616256247390992939472198486328125 \\
&\approx 1/3
\end{aligned}
$$

Execution speed with double-precision arithmetic

Using floating-point variables and mathematical functions (sin(), cos(), atan2(), log(), exp(), sqrt() are the most popular ones) of double precision as opposed to single precision comes at execution cost: the operations with double precision are usually slower. On average, on a PC of year 2012 build, calculations with double precision are 1.1 - 1.6 times slower than with single precision.

Notes and references

[1] William Kahan (1 October 1987). "Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic" (http://www.cs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF). .

Article Sources and Contributors

Double-precision floating-point format *Source*: http://en.wikipedia.org/w/index.php?oldid=540540557 *Contributors*: 1exec1, 2607:F470:14:2:52E5:49FF:FEE1:966D, A boardley, AXRL, Aklassyguy, Alexey.kudinkin, Alinor, B4hand, Bmsiller, Brianbjparker, Cabyd, Capn ed, Charon, Conversion script, Cutler, Cybercobra, Daniel5Ko, Dicklyon, Dionyziz, Dispenser, DragonLord, Dtgriscom, Emlynoregan, EncMstr, Etoombs, Exir Kamalabadi, Funnyfarmofdoom, Gerbrant, Giftlite, Graham87, Hjherbert, Idunno271828, Illnab1024, Ixfd64, JakeVortex, Jhshukla, Jordi Burguet Castell, Jorge Stolfi, Jshadias, Keka, Kurykh, LOL, Lambyte, Larry_Sanger, LastBall, Lauri.pirttiaho, Lucky17.TW, Mastan20, MattGiuca, Mfc, Michael Angelkovich, MicoFilós, Mild Bill Hiccup, Minesweeper, Ms2ger, Patrick, PierreAbbat, Pmokeefe, Pne, Qwerty112233, R'n'B, Radagast83, Rjstott, RyanEberhart, Scraimer, Sfax.tn, Shadowjams, ShashClp, Smyth, Spitfire ch, Stevenj, Suruena, Theshadow27, TimmmmCam, Tulcod, Vadmium, Yar Kramer, Ylai, Андрей Куликов, 89 anonymous edits

Image Sources, Licenses and Contributors

Image:IEEE 754 Double Floating Point Format.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:IEEE_754_Double_Floating_Point_Format.svg *License*: GNU Free Documentation License *Contributors*: Codekaizen

License