decimal128 floating-point format

From Wikipedia, the free encyclopedia

In computing, **decimal128** is a decimal floating-point computer numbering format that occupies 16 bytes (128 bits) in computer memory. It is intended for applications where it is necessary to emulate decimal rounding exactly, such as financial and tax computations.

Decimal128 supports 34 decimal digits of significand and an exponent range of −6143 to +6144, i.e. ±0.000 000 000 000 000 000 000 000 000 000 000 × 10^{-6143} to ±9.999 999 999 999 999 999 999 999 999 999 999 × 10^{6144}. (Equivalently, ±0 000 000 000 000 000 000 000 000 000 000 000 × 10^{-6176} to ±9 999 999 999 999 999 999 999 999 999 999 999 × 10^{6111}.) Therefore, decimal128 has the greatest range of values compared with other IEEE basic floating point formats. Because the significand is not normalized, most values with less than 34 significant digits have multiple possible representations; $1 \times 10^2 = 0.1 \times 10^3 = 0.01 \times 10^4$, etc. Zero has 12 288 possible representations (24 576 if both signed zeros are included).

Decimal128 floating point is a relatively new decimal floating-point format, formally introduced in the 2008 version[1] of IEEE 754 as well as with ISO/IEC/IEEE 60559:2011.[2]

Floating-point formats

IEEE 754
16-bit: Half (binary16)
32-bit: Single (binary32), decimal32
64-bit: Double (binary64), decimal64
128-bit: Quadruple (binary128), **decimal128**
256-bit: Octuple (binary256)
40-bit or 80-bit: Extended precision

Other
Minifloat
bfloat16
Microsoft Binary Format
IBM floating-point architecture
Posit
G.711 8-bit floats
Arbitrary precision

V • T • E

Contents [hide]

Representation of decimal128 values [edit]

Sign	Combination	Significand continuation
1 bit	17 bits	110 bits
s	mmmmmmmmmmmmmmmmm	cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

IEEE 754 allows two alternative representation methods for decimal128 values. The standard does not specify how to signify which representation is used, for instance in a situation where decimal128 values are communicated between systems.

In one representation method, based on binary integer decimal (BID), the significand is represented as binary coded positive integer.

The other, alternative, representation method is based on densely packed decimal (DPD) for most of the significand (except the most significant digit).

Both alternatives provide exactly the same range of representable numbers: 34 digits of significand and $3 \times 2^{12} = 12\,288$ possible exponent values.

In both cases, the most significant 4 bits of the significand (which actually only have 10 possible values) are combined with the most significant 2 bits of the exponent (3 possible values) to use 30 of the 32 possible values of 5 bits in the combination field. The remaining combinations encode infinities and NaNs.

Combination field	Exponent	Significand Msbits	Other
00mmmmmmmmmmmmmmm	00xxxxxxxxxxxx	0ccc	—
01mmmmmmmmmmmmmmm	01xxxxxxxxxxxx	0ccc	—
10mmmmmmmmmmmmmmm	10xxxxxxxxxxxx	0ccc	—
1100mmmmmmmmmmmmm	00xxxxxxxxxxxx	100c	—
1101mmmmmmmmmmmmm	01xxxxxxxxxxxx	100c	—
1110mmmmmmmmmmmmm	10xxxxxxxxxxxx	100c	—
11110mmmmmmmmmmmm	—	—	±Infinity
11111mmmmmmmmmmmm	—	—	NaN. Sign bit ignored. Sixth bit of the combination field determines if the NaN is signaling.

In the case of Infinity and NaN, all other bits of the encoding are ignored. Thus, it is possible to initialize an array to Infinities or NaNs by filling it with a single byte value.

Binary integer significand field [edit]

This format uses a binary significand from 0 to $10^{34} - 1$ = 9 999 999 999 999 999 999 999 999 999 999 = $1ED09BEAD87C0378D8E63FFFFFFFF_{16}$ = $0111\ 1011010000\ 1001101111\ 1010101101\ 1000011111\ 0000000011\ 0111100011\ 0110001110\ 0110001111\ 1111111111\ 1111111111\ 1111111111_2$. The encoding can represent binary significands up to $10 \times 2^{110} - 1$ = 12 980 742 146 337 069 071 326 240 823 050 239 but values larger than $10^{34} - 1$ are illegal (and the standard requires implementations to treat them as 0, if encountered on input).

As described above, the encoding varies depending on whether the most significant 4 bits of the significand are in the range 0 to 7 (0000_2 to 0111_2), or higher (1000_2 or 1001_2).

If the 2 bits after the sign bit are "00", "01", or "10", then the exponent field consists of the 14 bits following the sign bit, and the significand is the remaining 113 bits, with an implicit leading 0 bit:

```
 s 00eeeeeeeeeeee   (0)ttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt
tttttttttt tttttttttt
 s 01eeeeeeeeeeee   (0)ttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt
tttttttttt tttttttttt
 s 10eeeeeeeeeeee   (0)ttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt
tttttttttt tttttttttt
```

This includes subnormal numbers where the leading significand digit is 0.

If the 2 bits after the sign bit are "11", then the 14-bit exponent field is shifted 2 bits to the right (after both the sign bit and the "11" bits thereafter), and the represented significand is in the remaining 111 bits. In this case there is an implicit (that is, not stored) leading 3-bit sequence "100" in the true significand.

```
 s 1100eeeeeeeeeeee (100)t tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt
tttttttttt tttttttttt
 s 1101eeeeeeeeeeee (100)t tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt
tttttttttt tttttttttt
 s 1110eeeeeeeeeeee (100)t tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt tttttttttt
tttttttttt tttttttttt
```

The "11" 2-bit sequence after the sign bit indicates that there is an *implicit* "100" 3-bit prefix to the significand. Compare having an implicit 1 in the significand of normal values for the binary formats. The "00", "01", or "10" bits are part of the exponent field.

For the decimal128 format, all of these significands are out of the valid range (they begin with $2^{113} > 1.038 \times 10^{34}$), and are thus decoded as zero, but the pattern is same as decimal32 and decimal64.

In the above cases, the value represented is

$$(-1)^{\text{sign}} \times 10^{\text{exponent}-6176} \times \text{significand}$$

If the four bits after the sign bit are "1111" then the value is an infinity or a NaN, as described above:

```
s 11110 xx...x    ±infinity
s 11111 0x...x    a quiet NaN
s 11111 1x...x    a signalling NaN
```

Densely packed decimal significand field [edit]

In this version, the significand is stored as a series of decimal digits. The leading digit is between 0 and 9 (3 or 4 binary bits), and the rest of the significand uses the densely packed decimal (DPD) encoding.

The leading 2 bits of the exponent and the leading digit (3 or 4 bits) of the significand are combined into the five bits that follow the sign bit.

This twelve bits after that are the exponent continuation field, providing the less-significant bits of the exponent.

The last 110 bits are the significand continuation field, consisting of eleven 10-bit *declets*.[3] Each declet encodes three decimal digits[3] using the DPD encoding.

If the first two bits after the sign bit are "00", "01", or "10", then those are the leading bits of the exponent, and the three bits after that are interpreted as the leading decimal digit (0 to 7):

```
   s 00 TTT (00)eeeeeeeeeeee (0TTT)[tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt]
[tttttttttt][tttttttttt][tttttttttt]
   s 01 TTT (01)eeeeeeeeeeee (0TTT)[tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt]
[tttttttttt][tttttttttt][tttttttttt]
   s 10 TTT (10)eeeeeeeeeeee (0TTT)[tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt]
[tttttttttt][tttttttttt][tttttttttt]
```

If the first two bits after the sign bit are "11", then the second two bits are the leading bits of the exponent, and the last bit is prefixed with "100" to form the leading decimal digit (8 or 9):

```
   s 1100 T (00)eeeeeeeeeeee (100T)[tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt]
[tttttttttt][tttttttttt][tttttttttt]
   s 1101 T (01)eeeeeeeeeeee (100T)[tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt]
[tttttttttt][tttttttttt][tttttttttt]
   s 1110 T (10)eeeeeeeeeeee (100T)[tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt][tttttttttt]
[tttttttttt][tttttttttt][tttttttttt]
```

The remaining two combinations (11110 and 11111) of the 5-bit field are used to represent ±infinity and NaNs, respectively.

The DPD/3BCD transcoding for the declets is given by the following table. b9...b0 are the bits of the DPD, and d2...d0 are the three BCD digits.

Densely packed decimal encoding rules[4]

DPD encoded value											Decimal digits					
Code space (1024 states)	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	d2	d1	d0	Values encoded	Description	Occurrences (1000 states)
50.0% (512 states)	a	b	c	d	e	f	0	g	h	i	0abc	0def	0ghi	(0–7) (0–7) (0–7)	Three small digits	51.2% (512 states)
37.5% (384 states)	a	b	c	d	e	f	1	0	0	i	0abc	0def	100i	(0–7) (0–7) (8–9)	Two small digits, one large	38.4% (384 states)
	a	b	c	g	h	f	1	0	1	i	0abc	100f	0ghi	(0–7) (8–9) (0–7)		
	g	h	c	d	e	f	1	1	0	i	100c	0def	0ghi	(8–9) (0–7) (0–7)		
9.375% (96 states)	g	h	c	0	0	f	1	1	1	i	100c	100f	0ghi	(8–9) (8–9) (0–7)	One small digit, two large	9.6% (96 states)
	d	e	c	0	1	f	1	1	1	i	100c	0def	100i	(8–9) (0–7) (8–9)		
	a	b	c	1	0	f	1	1	1	i	0abc	100f	100i	(0–7) (8–9) (8–9)		
3.125% (32 states, 8 used)	x	x	c	1	1	f	1	1	1	i	100c	100f	100i	(8–9) (8–9) (8–9)	Three large digits, bits b9 and b8 are don't care	0.8% (8 states)

The 8 decimal values whose digits are all 8s or 9s have four codings each. The bits marked x in the table above are ignored on input, but will always be 0 in computed results. (The $8 \times 3 = 24$ non-standard encodings fill in the gap between $10^3 = 1000$ and $2^{10} = 1024$.)

In the above cases, with the *true significand* as the sequence of decimal digits decoded, the value represented is

$$(-1)^{\text{signbit}} \times 10^{\text{exponentbits}_2 - 6176_{10}} \times \text{truesignificand}_{10}$$

See also [edit]

- ISO/IEC 10967, Language Independent Arithmetic
- Primitive data type

References [edit]

1. ^ IEEE Computer Society (2008-08-29). *IEEE Standard for Floating-Point Arithmetic*. IEEE. doi:10.1109/IEEESTD.2008.4610935. ISBN 978-0-7381-5753-5. IEEE Std 754-2008.
2. ^ "ISO/IEC/IEEE 60559:2011". 2011. Retrieved 2016-02-08.
3. ^ a b 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* (1 ed.). Birkhäuser. doi:10.1007/978-0-8176-4705-6. ISBN 978-0-8176-4704-9. LCCN 2009939668.
4. ^ Cowlishaw, Michael Frederic (2007-02-13) [2000-10-03]. "A Summary of Densely Packed Decimal encoding". IBM. Archived from the original on 2015-09-24. Retrieved 2016-02-07.

Categories: Computer arithmetic | Data types | Floating point types