Floating Point Numbers

Floats are not Reals

Int's:

eg. 40000 * 40000 --> 1600000000

600000* 600000 --> ?

Floats:

Eg 2 : Is (x + y) + z = x + (y + z)?

eg

(1e20 + -1e20) + 3.14 --> 3.14

1e20 + (-1e20 + 3.14) --> ??

Need to understand details of underlying implementations

Objective

- To understand the fundamentals of floating-point representation
- To know the IEEE-754 Floating Point Standard

Patriot Missile

- Gulf War I
- Failed to intercept incoming Iraqi scud missile (Feb 25, 1991)
- 28 American soldiers killed

GAO Report: GAO/IMTEC-92-26 Patriot Missile Software Problem
http://www.fas.org/spp/starwars/gao/im92026.htm

Patriot Design

- Intended to operate only for a few hours
  - Defend Europe from Soviet aircraft and missile
- Four 24-bit registers (1970s design!)
- Kept time with integer counter: incremented every 1/10 second
- Calculate speed of incoming missile to predict future positions:

  $$\text{velocity} = \text{loc}_1 - \text{loc}_0/(\text{count}_1 - \text{count}_0) * 0.1$$

- But, cannot represent 0.1 exactly!

Floating Imprecision

- 24-bits:

$0.1 = 1/2^4 + 1/2^5 + 1/2^8 + 1/2^9$

$+ 1/2^{12} + 1/2^{13} + 1/2^{16} + 1/2^{17}$

$+ 1/2^{20} + 1/2^{21}$

$= 209715 / 2097152$

Error is 0.2/2097152 = 1/10485760

One hour = 3600 seconds

3600 * 1/10485760 * 10 = 0.0034s

20 hours = 0.0687s

Miss target! (137 meters)

Two weeks before the incident, Army officials received Israeli data indicating some loss in accuracy after the system had been running for 8 consecutive hours. Consequently, Army officials modified the software to improve the system's accuracy. However, the modified software did not reach Dhahran until February 26, 1991--the day after the Scud incident.

GAO Report

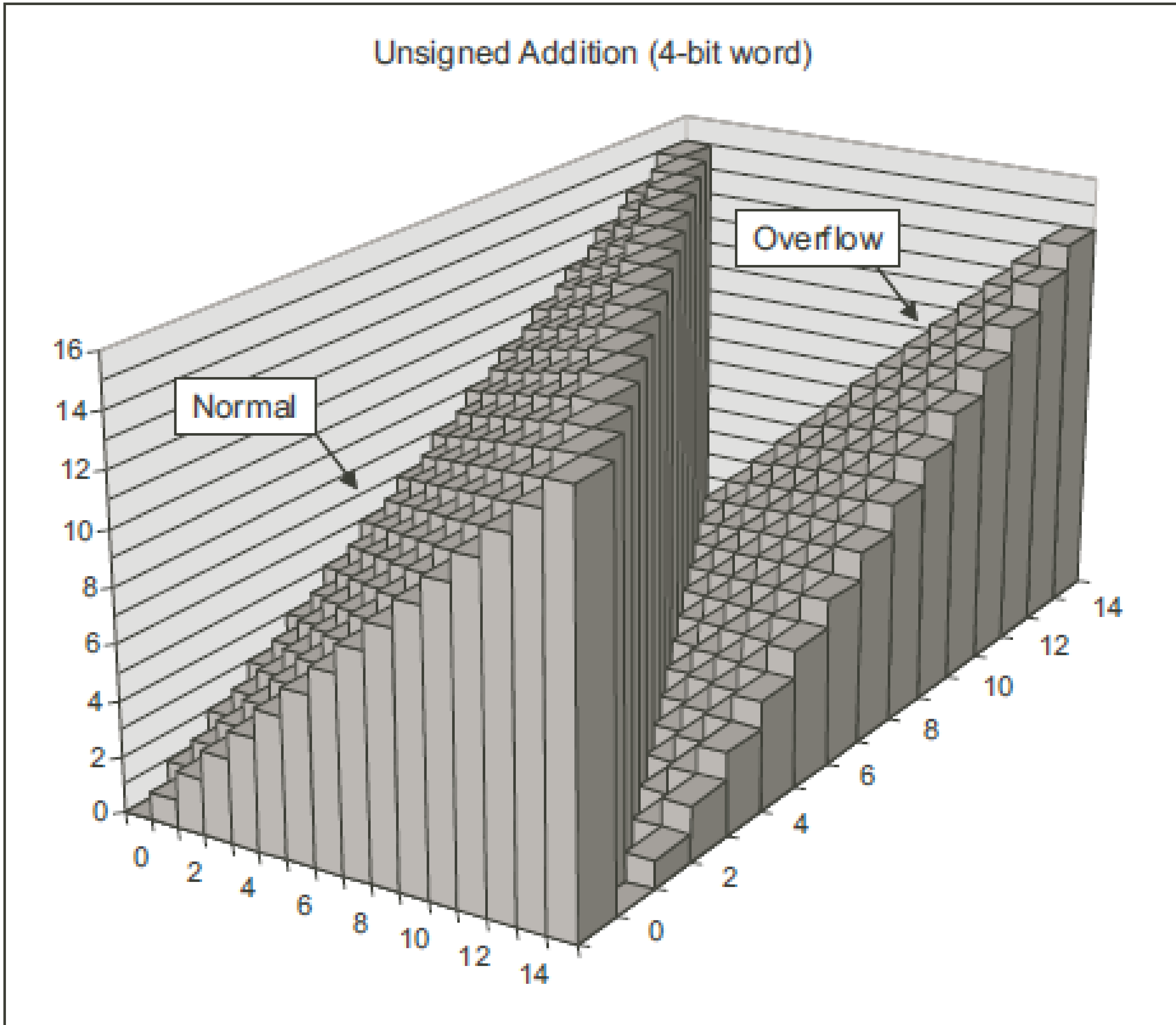
http://fas.org/spp/starwars/gao/im92026.htm

Review of Numbers

- Computers are made to deal with numbers
- What can we represent in N bits?
  - Unsigned integers:

    0 to $2^N - 1$
  - Signed Integers (Two's Complement)

    $-2^{(N-1)}$ to $2^{(N-1)} - 1$


Unsigned Addition (4-bit word)
Normal
Overflow
16
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14

Other Numbers

- What about other numbers?
  - Very large numbers? (seconds/century)

    $3,155,760,000_{10}$ (3.15576_{10} x 10^{9})
  - Very small numbers? (atomic diameter)

    0.00000001_{10} (1.0_{10} x 10^{-8})
  - Rationals (repeating pattern)
  - 2/3 (0.666666666. . .)
  - Irrationals

    $2^{1/2}$ (1.414213562373. . .)
  - Transcendentals
  - e (2.718...), π (3.141...)
- All represented in scientific notation

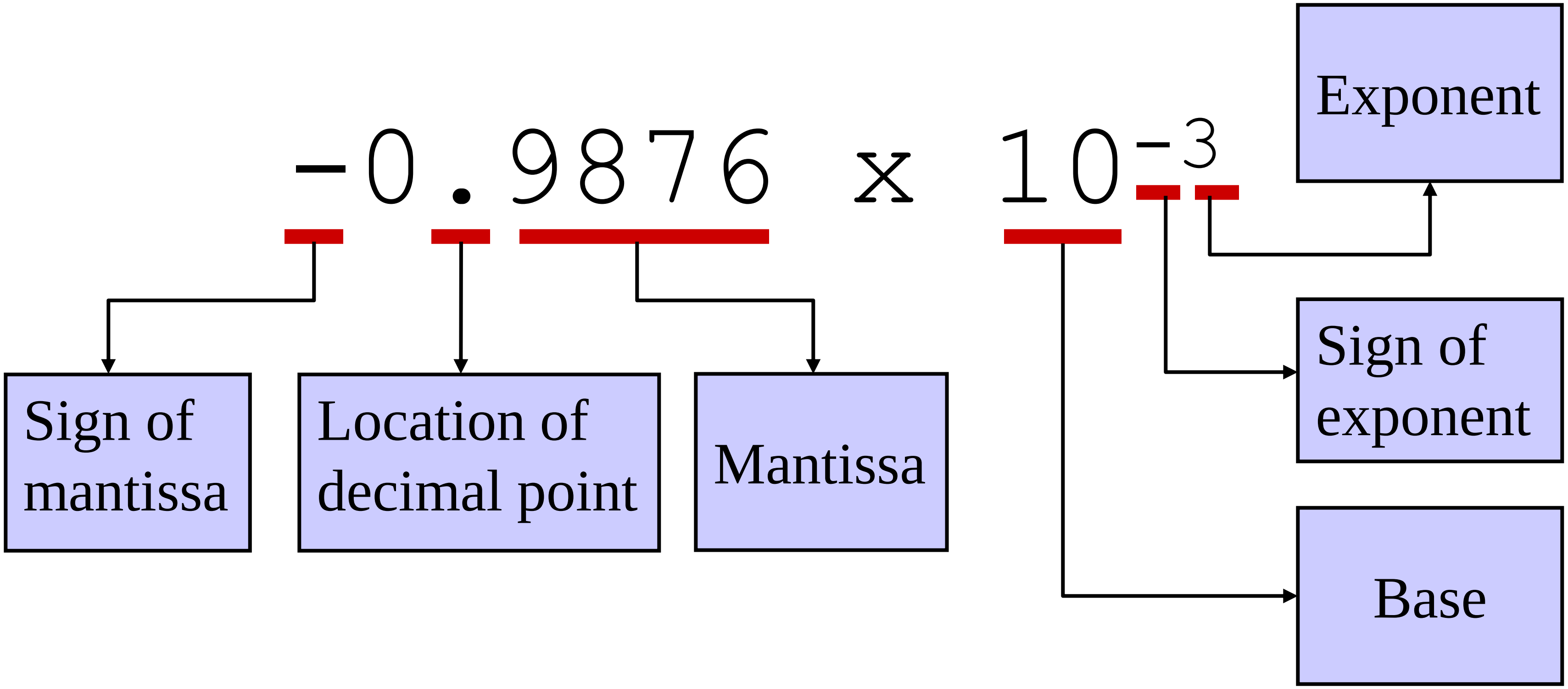
Exponential Notation

- The following are equivalent representations of 1,234

123,400.0	$\times 10^{-2}$
12,340.0	$\times 10^{-1}$
1,234.0	$\times 10^{0}$
123.4	$\times 10^{1}$
12.34	$\times 10^{2}$
1.234	$\times 10^{3}$
0.1234	$\times 10^{4}$

The representations differ in that the decimal place – the “point” -- “floats” to the left or right (with the appropriate adjustment in the exponent).

Parts of a Floating Point Number

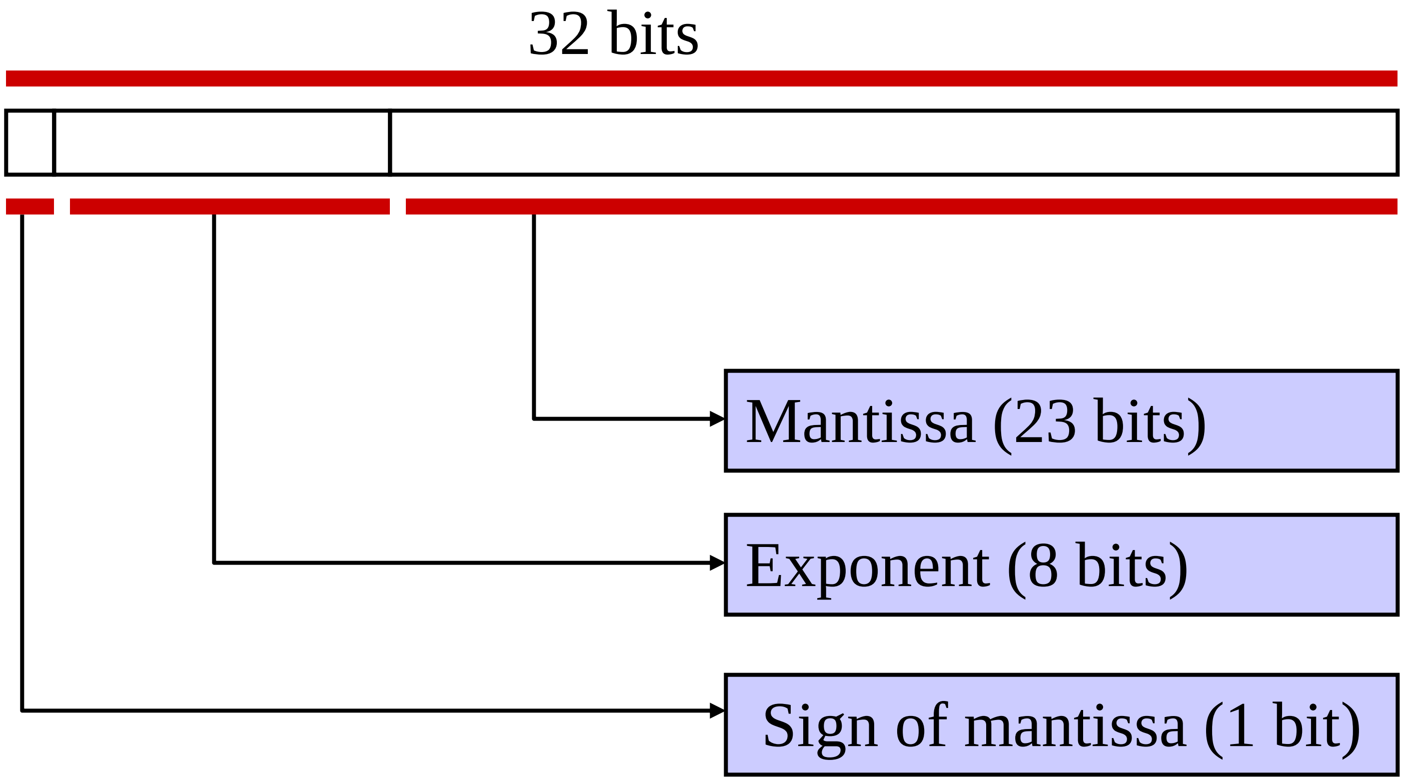


IEEE 754 Standard

- Most common standard for representing floating point numbers
- Single precision: 32 bits, consisting of...
  - Sign bit (1 bit)
  - Exponent (8 bits)
  - Mantissa (23 bits)
- Double precision: 64 bits, consisting of…
  - Sign bit (1 bit)
  - Exponent (11 bits)
  - Mantissa (52 bits)

Prof. Willian Kahan

Single Precision Format



Normalization

- The mantissa is *normalized*
- Has an implied decimal place on left
- Has an implied "1" on left of the decimal place
- E.g.,
  - Mantissa → `10100000000000000000000`
  - Represents... $1.101_2 = 1.625_{10}$
- Normalized form: no leadings 0s
  (exactly one digit to left of decimal point)
  - Normalized: 1.0×10^{-9}
  - Not normalized: $0.1 \times 10^{-8}, 10.0 \times 10^{-10}$

Excess Notation

- To include +ve and –ve exponents, “excess” notation is used
- Single precision: excess 127
- Double precision: excess 1023
- The value of the exponent stored is larger than the actual exponent
- E.g., excess 127,
  - Exponent → `10000111`
  - Represents… `135 - 127 = 8`

Example

- Single precision

```
0 10000010 11000000000000000000000
```

1.11_2

$130 - 127 = 3$

0 = positive mantissa

$+1.11_2 \text{ x } 2^3 = 1110.0_2 = 14.0_{10}$

Hexadecimal

- It is convenient and common to represent the original floating point number in hexadecimal
- The preceding example…

0 100	0001	0 110	0000	0000	0000	0000	0000
4	1	6	0	0	0	0	0

Converting from Floating Point

- E.g., What decimal value is represented by the following 32-bit floating point number?

$\texttt{C17B0000}_{16}$

- Step 1
  - Express in binary and find S, E, and M

$$C17B0000_{16} =$$

$$\underbrace{1}_{S}\ \ \underbrace{10000010}_{E}\ \ \underbrace{11110110000000000000000}_{M}{}_{2}$$

1 = negative
0 = positive

- Step 2
  - Find “real” exponent, n
  - $n = E - 127$

    $= 10000010_2 - 127$

    $= 130 - 127$

    $= 3$

- Step 3
  - Put S, M, and *n* together to form binary result
  - (Don't forget the implied "1." on the left of the mantissa.)

$-1.1111011_2 \text{ x } 2^{n} =$

$-1.1111011_2 \text{ x } 2^{3} =$

-1111.1011_2

- Step 4
  - Express result in decimal

$$\underbrace{-1111}_{-15}.\underbrace{1011}_{}{}_2$$

$2^{-1} = 0.5$

$2^{-3} = 0.125$

$2^{-4} = 0.0625$

0.6875

Answer: -15.6875

Converting from Floating Point

- E.g., What decimal value is represented by the following 32-bit floating point number?

42808000_{16}

Converting to Floating Point

- E.g., Express 36.5625_{10} as a 32-bit floating point number (in hexadecimal)

- Step 1
  - Express original value in binary

$36.5625_{10} =$

100100.1001_2

- Step 2
  - Normalize

$100100.1001_2 =$

$1.001001001_2 \times 2^5$

- Step 3
  - Determine S, E, and M

$$\underbrace{+}_{S}1.\underbrace{001001001}_{M}{}_2 \times 2^{\underbrace{5}_{n}}$$

E $= n + 127$
$= 5 + 127$
$= 132$
$= 10000100_2$

S = 0 (because the value is positive)

- Step 4
  - Put S, E, and M together to form 32-bit binary result

$$\underbrace{\texttt{0}}_{S}\quad \underbrace{\texttt{10000100}}_{E}\quad \underbrace{\texttt{00100100100000000000000}}_{M}{}_{2}$$

$$+1.001001001_2 \times 2^5$$

- Step 5
  - Express in hexadecimal

$$0\ 10000100\ 00100100100000000000000_2 =$$

$$0100\ 0010\ 0001\ 0010\ 0100\ 0000\ 0000\ 0000_2 =$$

$$4\quad 2\quad 1\quad 2\quad 4\quad 0\quad 0\quad 0_{16}$$

Answer: 42124000_{16}

Converting to Floating Point

- E.g., Express 6.5_{10} as a 32-bit floating point number (in hexadecimal)

Converting to Floating Point

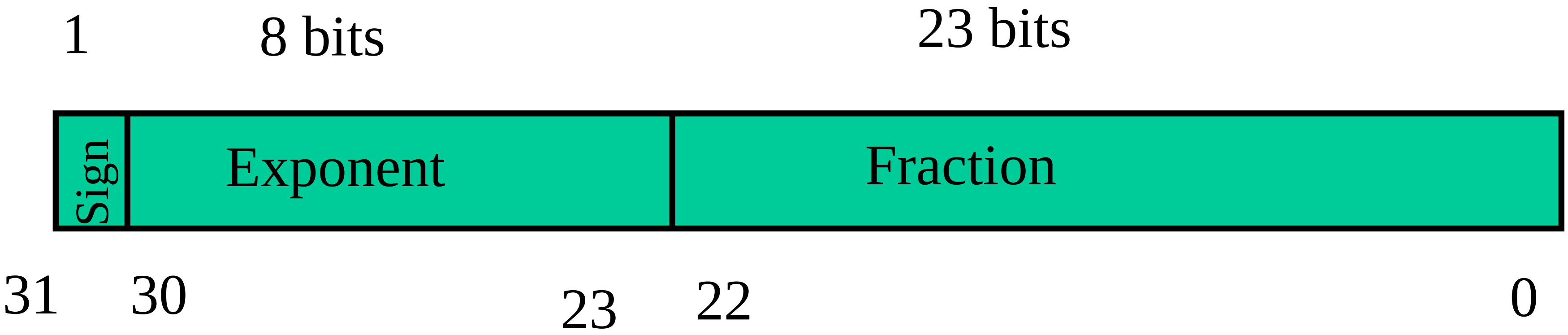
- E.g., Express 0.1 as a 32-bit floating point number (in hexadecimal)

Item	Single precision	Double precision
Bits in sign	1	1
Bits in exponent	8	11
Bits in fraction	23	52
Bits, total	32	64
Exponent system	Excess 127	Excess 1023
Exponent range	−126 to +127	−1022 to +1023
Smallest normalized number	2^{-126}	2^{-1022}
Largest normalized number	approx. 2^{128}	approx. 2^{1024}
Decimal range	approx. 10^{-38} to 10^{38}	approx. 10^{-308} to 10^{308}
Smallest denormalized number	approx. 10^{-45}	approx. 10^{-324}

Figure B-5. Characteristics of IEEE floating-point numbers.

Summary: IEEE Floating Point

Single Precision (32 bits)



Exponent values:

0	zeroes
1-254	exp + 127
255	infinities, NaN

Value = $(1 - 2*\mathit{Sign})\ (1 + \mathit{Fraction})^{\mathit{Exponent} - 127}$

Denormalized Values

- Condition
  - `exp` = `000…0`
- Value
  - Exponent value $E = –Bias + 1$
  - Significand value $M =$ `0.xxx…x`$_2$
    - `xxx…x`: bits of `frac`
- Cases
  - `exp` = `000…0`, `frac` = `000…0`
    - Represents value 0
    - Note that have distinct values +0 and –0
  - `exp` = `000…0`, `frac` ≠ `000…0`
    - Numbers very close to 0.0
    - Lose precision as get smaller

Special Values

- Condition
  - `exp` = `111…1`
- Cases
  - `exp` = `111…1`, `frac` = `000…0`
    - Represents value ∞ (infinity)
    - Operation that overflows
    - Both positive and negative
    - E.g., 1.0/0.0 = –1.0/–0.0 = +∞, 1.0/–0.0 = –∞
  - `exp` = `111…1`, `frac` ≠ `000…0`
    - Not-a-Number (NaN)
    - Represents case when no numeric value can be determined
    - E.g., sqrt(–1), ∞ – ∞

Interesting Numbers

Description	exp	frac	Numeric Value
Zero	00…00	00…00	0.0
Smallest Pos. Denorm.	00…00	00…01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$

- Single ≈ 1.4×10^{-45}
- Double ≈ 4.9×10^{-324}

Description	exp	frac	Numeric Value
Largest Denormalized	00…00	11…11	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$

- Single ≈ 1.18×10^{-38}
- Double ≈ 2.2×10^{-308}

Description	exp	frac	Numeric Value
Smallest Pos. Normalized	00…01	00…00	$1.0 \times 2^{-\{126,1022\}}$

- Just larger than largest denormalized

Description	exp	frac	Numeric Value
One	01…11	00…00	1.0
Largest Normalized	11…10	11…11	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$

- Single ≈ 3.4×10^{38}
- Double ≈ 1.8×10^{308}