Finite representation of real numbers

Floating-point numbers

Dr. Ing. Rodrigo Gonzalez

rodrigo.gonzalez@ingenieria.uncuyo.edu.ar

Control y Sistemas

Ingeniería Mecatrónica,
Facultad de Ingeniería,
Universidad Nacional de Cuyo

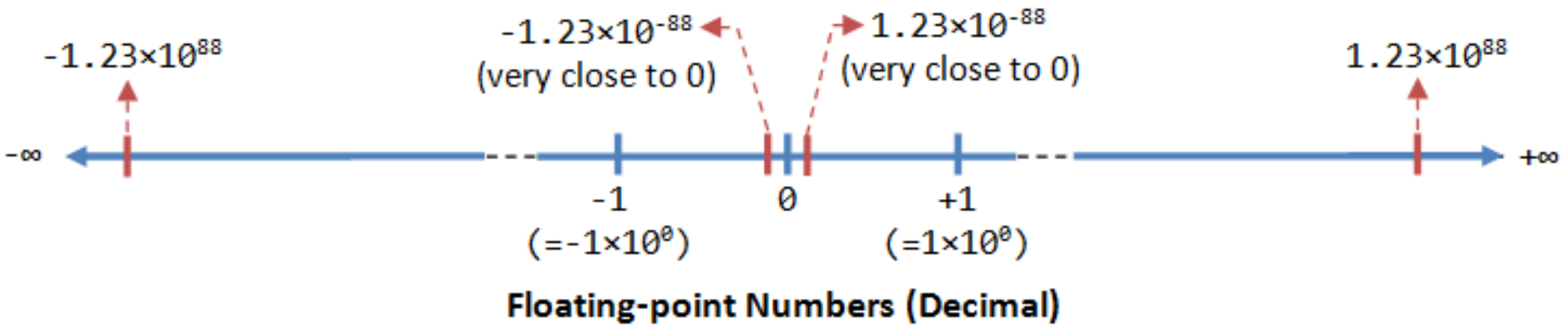
March 2020

UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

Summary

Floating-point Representation

A floating-point number can represent a very large or a very small value, positive and negative.



Floating-point Numbers (Decimal)

A floating-point number is typically expressed in the scientific notation in the form of

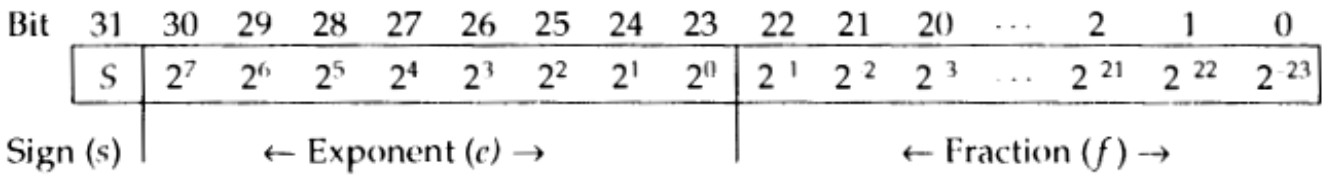
$$(-1)^S \times F \times r^E ,$$

where,

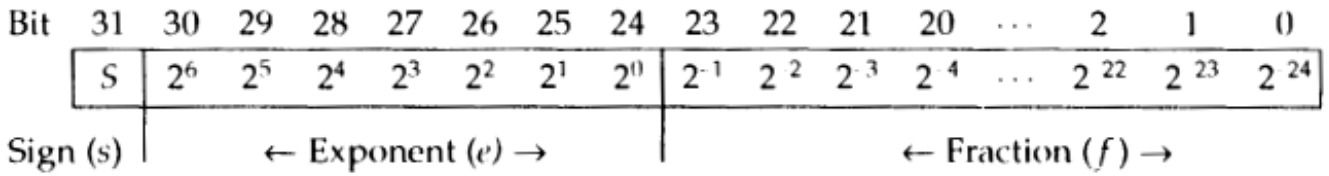
- S, sign bit.
- F, fraction.
- E, **biased** exponent.
- r, certain radix. $r = 2$ for binary; $r = 10$ for decimal.

Old formats

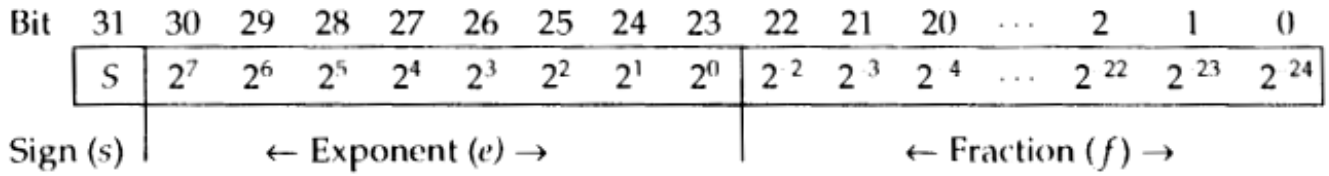
IEEE Standard P754 Format



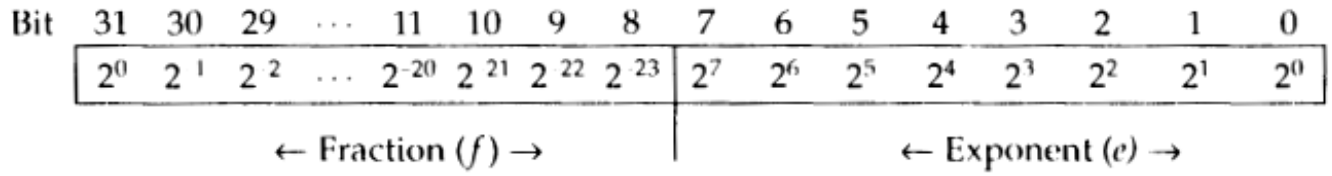
IBM Format



DEC (Digital Equipment Corp.) Format



MIL-STD 1750A Format



IEEE 754 standard

Modern computers adopt the IEEE 754 standard for representing floating-point numbers at the FPU.

First version was published in 1985. Last version in July 2019 (IEEE 754-2019).

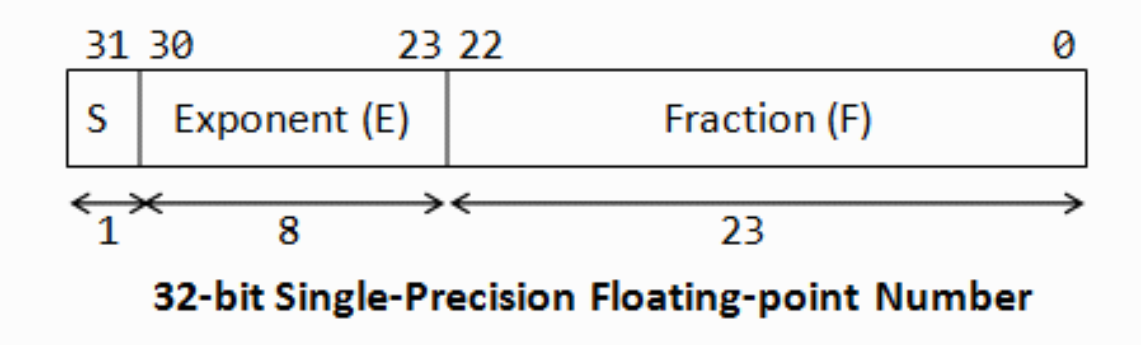
IEEE 754 standard defines several arithmetic formats.

	Binary formats ($B = 2$)				Decimal formats ($B = 10$)		
Parameter	Binary 16	Binary 32	Binary 64	Binary 128	Decimal 132	Decimal 164	Decimal 128
p, digits	$10 + 1$	$23 + 1$	$52 + 1$	$112 + 1$	7	16	34
e_{max}	+15	+127	+1023	+16383	+96	+384	+16,383
e_{min}	−14	−126	−1022	−16382	−95	−383	−16,382
Common name	Half precision	Single precision	Double precision	Quadruple precision			

IEEE 754 standard also defines:

- Rounding rules.
- Arithmetic operations, trigonometric functions.
- Exception handling.

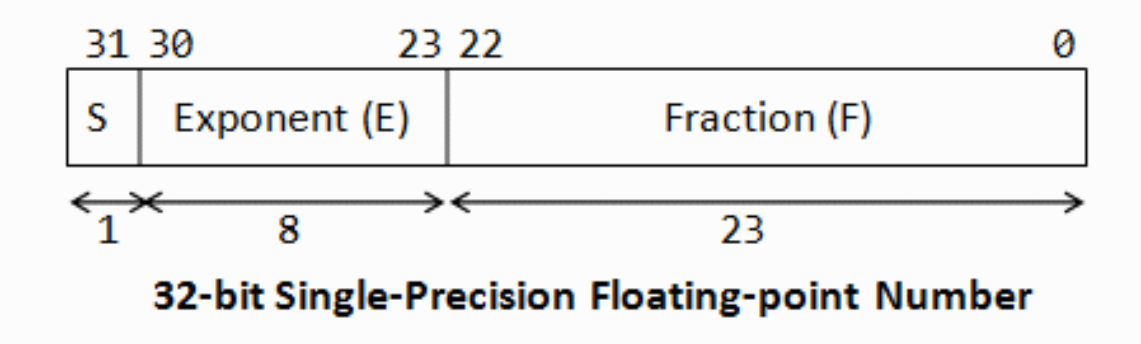
IEEE-754 32-bit Single-Precision



$$(-1)^S \times F \times r^{(E-bias)}$$

- S, sign bit. **0** for positive numbers and **1** for negative numbers.
- F, 23-bits fraction: $\left[2^{-1} \;\; 2^{-2} \cdots 2^{-23}\right]$
- We need to represent both positive and negative exponents.
- E, 8-bits exponent, **no sign bit**.
  - $E = [1, 254]$, $bias = 127$; $\quad -126 \leq E - bias \leq 127$.
  - $E = 0$ and $E = 255$ are reserved.

Normalized Form



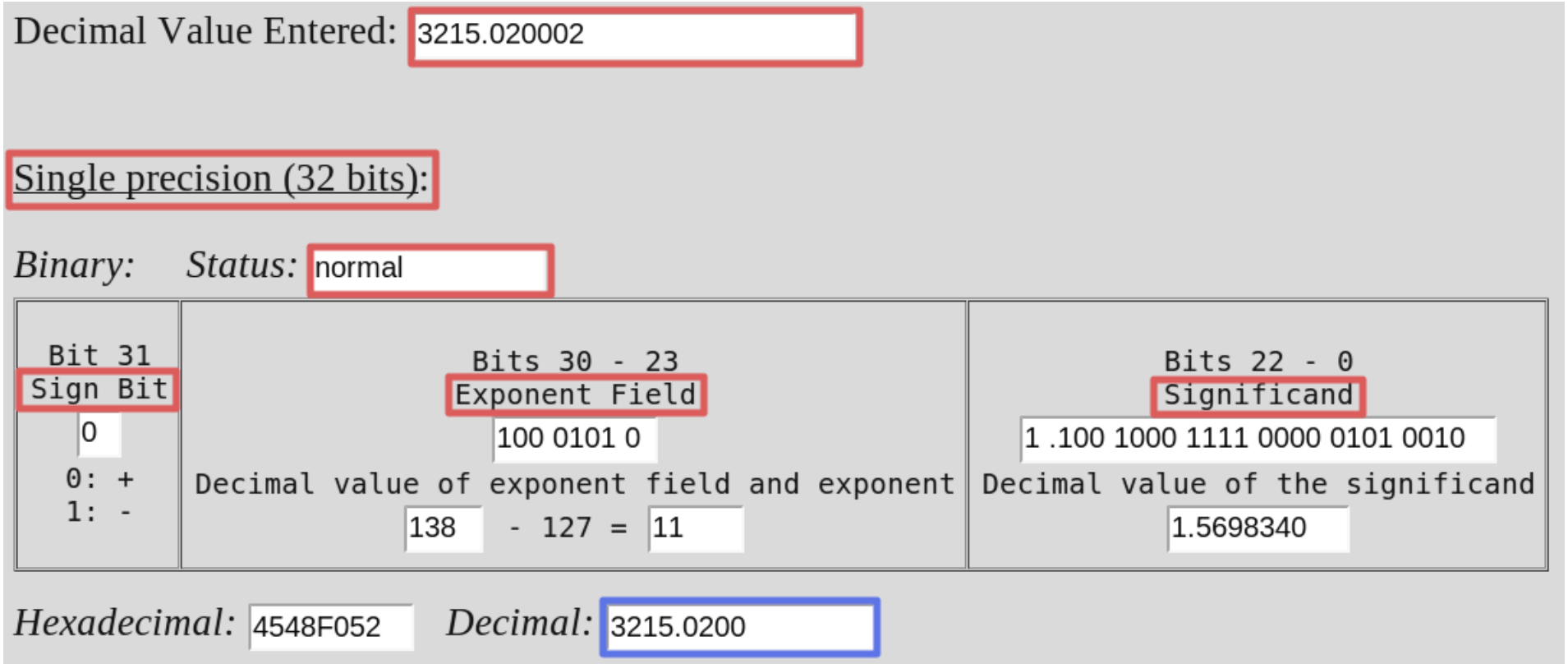
32-bit Single-Precision Floating-point Number

$$(-1)^S \times F \times r^{(E-bias)}$$

- Representation of a floating point number may not be unique:
- For example, the number 13.25 can be represented as
  $1101.01 * (2^0) = 110.101 * (2^1) = 11.0101 * (2^2) = 1.10101 * (2^3)$
- A floating point number is normalized when the integer part of its mantissa is forced to be exactly 1 and its fraction is adjusted accordingly.
- The leading 1 is **implicit**. It is not part of the 32 bits number.
- $1.F = \mathbf{1.}\ [2^{-1}\ \ 2^{-2} \cdots 2^{-23}]$.

Example 1

Represent 3215.020002_{10}



http://babbage.cs.qc.cuny.edu/IEEE-754.old/Decimal.html

Example 2

Represent $3215.020002_{10} \times 2 = 6430.040004_{10}$

Decimal Value Entered: 6430.040004

Single precision (32 bits):

Binary: *Status:* normal

Bit 31 Sign Bit	Bits 30 - 23 Exponent Field	Bits 22 - 0 Significand
0	10001011	1 .10010001111000001010010
0: + 1: -	Decimal value of exponent field and exponent	Decimal value of the significand
	139 - 127 = 12	1.5698340

Hexadecimal: 45C8F052 *Decimal:* 6430.0400

Example 3

Represent $3215.020002_{10}/4 = 803.7550005_{10}$

Decimal Value Entered: 803.7550005

Single precision (32 bits):

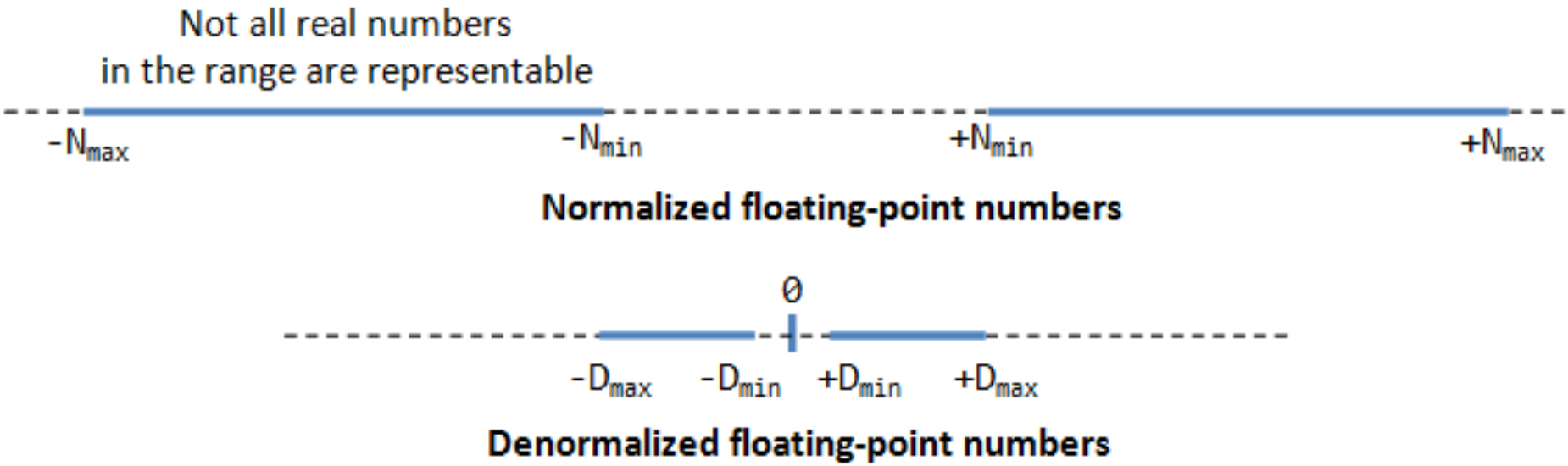
Binary: *Status:* normal

Bit 31 Sign Bit	Bits 30 - 23 Exponent Field	Bits 22 - 0 Significand
0	10001000	1 .10010001111000001010010
0: + 1: -	Decimal value of exponent field and exponent	Decimal value of the significand
	136 - 127 = 9	1.5698340

Hexadecimal: 4448F052 *Decimal:* 803.75500

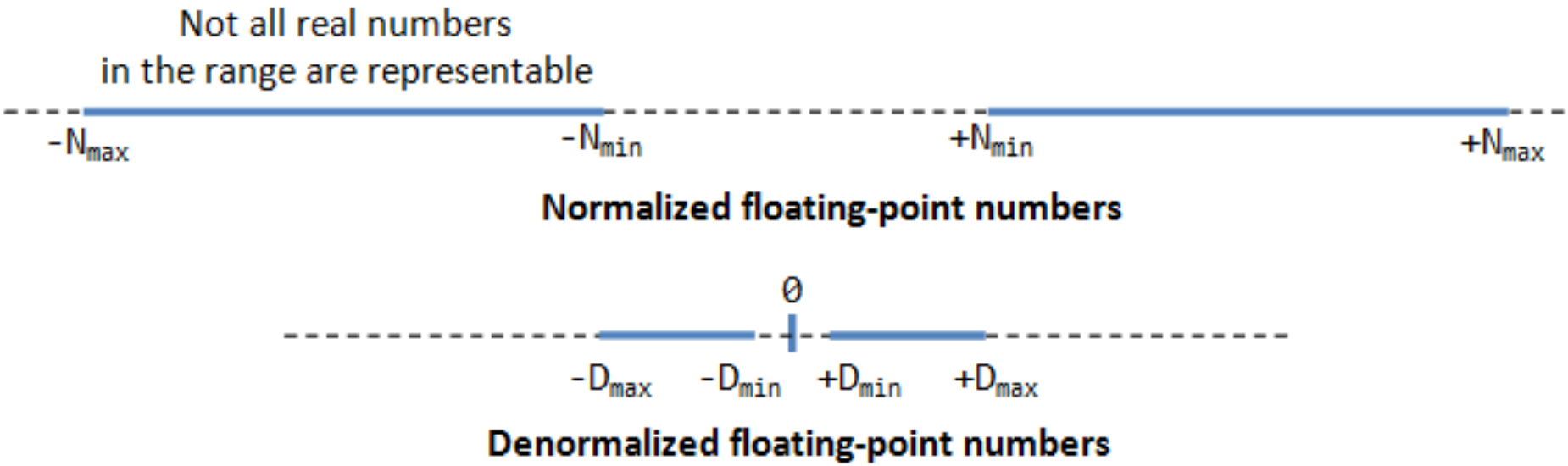
Floating-point numbers are auto-scaled!

De-normalized Form



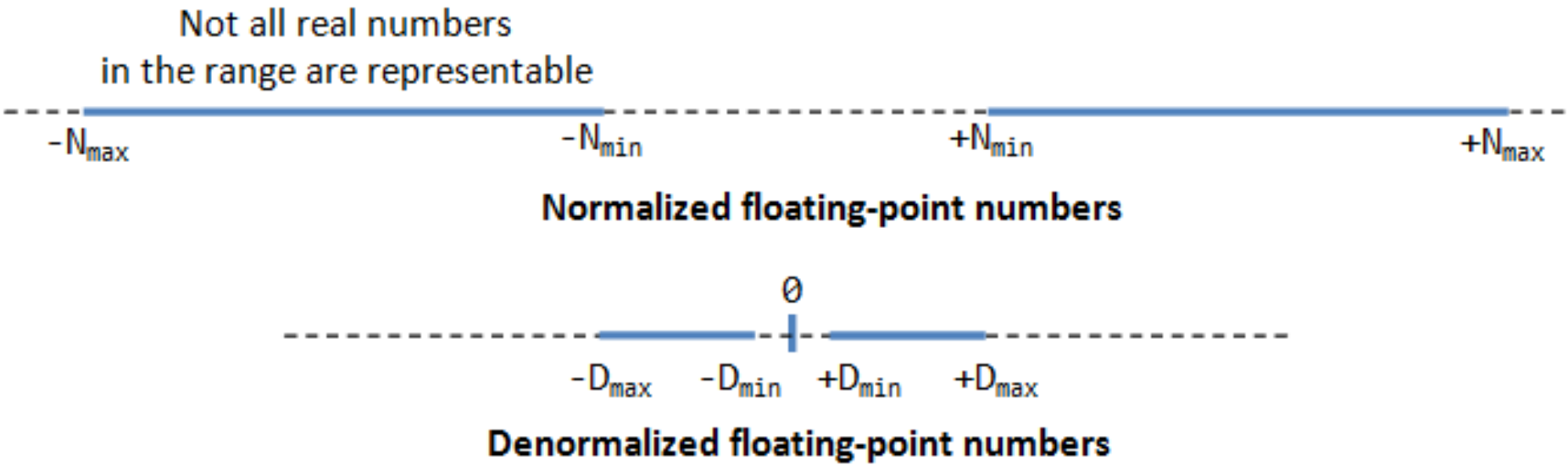
- Normalized form has a serious problem.

De-normalized Form



- Normalized form has a serious problem.
- The number zero cannot be represent with an implicit leading 1!

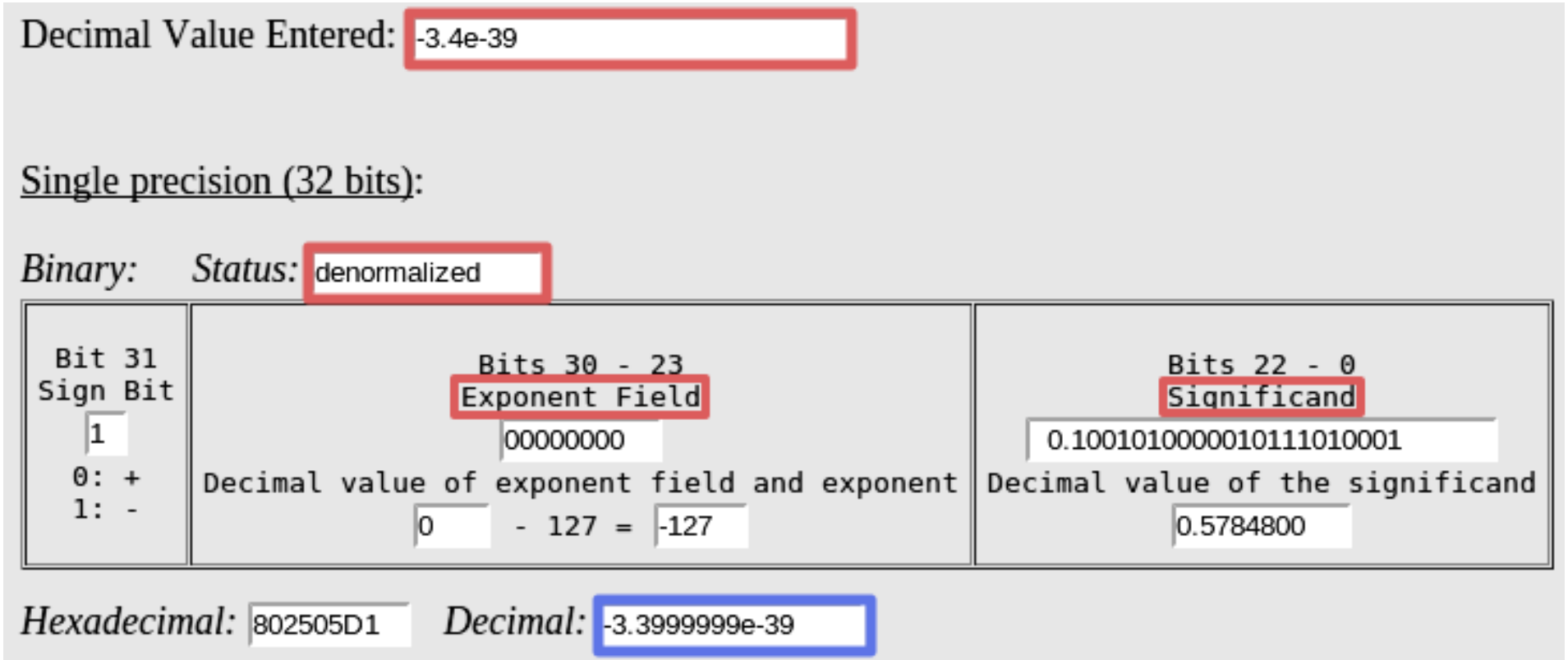
De-normalized Form



- Normalized form has a serious problem.
- The number zero cannot be represent with an implicit leading 1!
- De-normalized form is devised to represent zero and small numbers.
- $E = 0 \Rightarrow 0.F$
- Implicit leading 0: **0.** $\left[2^{-1} \;\; 2^{-2} \cdots 2^{-23}\right]$.

Example 4

Represent $-3.4E\text{-}39_{10}$



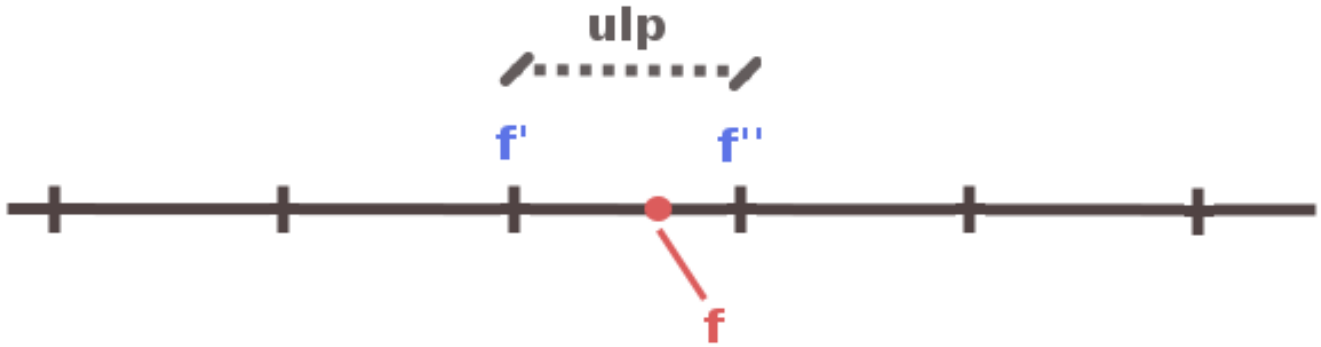
Special values

- **Zero**: $E = 0$, $F = 0$. Two representations: **+0** ($S = 0$) and **-0** ($S = 1$).
- **Inf** (Infinity): $E = 0$xFF, $F = 0$. Two representations: **+Inf** ($S = 0$) and **-Inf** ($S = 1$).
- **NaN** (Not a Number): $E = 0$xFF , $F \neq 0$. A value that cannot be represented as a real number (e.g. 0/0).

MATLAB

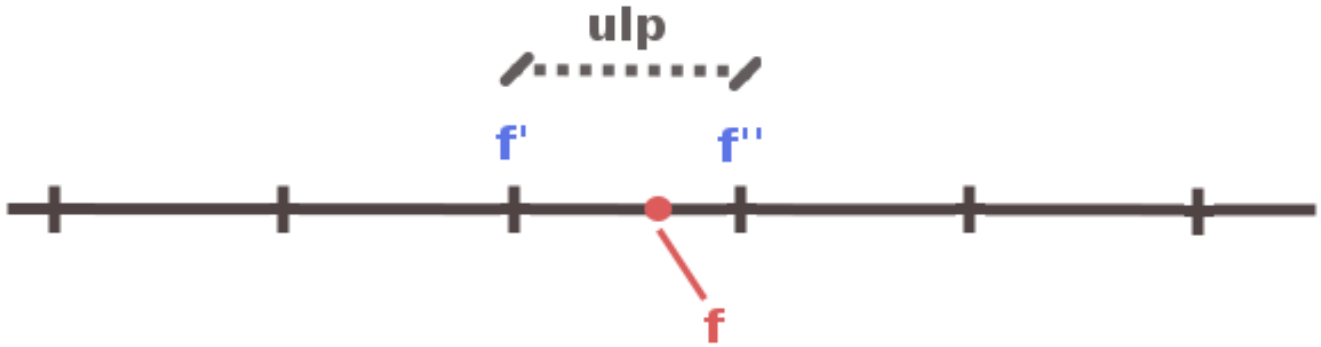
```
» a = 1/0
» ans = Inf
» b = exp(1000)
» ans = Inf
» c = log(0)
» ans = -Inf
» d = -1/0
» ans = -Inf
» e = 0/0
» ans = NaN
» f = Inf/Inf
» ans = NaN
```

Rounding schemes



- *ulp* (unit of least precision). In MATLAB, `eps()`.
- f, significant, $f = 1.F$.
- f' and f'' being two successive multiples of ulp.
- Assume that $f' < f < f''$.
- $f'' = f' + ulp$.
- Then, the rounding function $round(f)$ associates to f either f' or f'', according to some rounding strategy.

Rounding schemes supported by IEEE-754, 2



Rounding schemes are:

1. *Truncation* (also called *round toward 0* or *chopping*):
   - if f is positive, $round(f) = f'$.
   - if f is negative, $round(-f) = f''$.
2. *Round toward plus infinity*: $round(f) = f''$.
3. *Round toward minus infinity*: $round(f) = f'$.
4. *Round to nearest* (default):
   - if $f < f' + ulp/2$, $round(f) = f'$.
   - if $f >= f' + ulp/2$, $round(f) = f''$.

Dynamic range

Dynamic range is defined as,

$$DR_{db} = 20\ log_{10}\left(\frac{\text{largest possible word value}}{\text{smallest possible word value}}\right) \quad [\text{dB}]$$

Dynamic range for floating-point numbers is defined as,

$$DR_{dB} \approx 6.02 \cdot 2^{b_E}$$

where b_E is the number of bits of E.

For single precision (32-bits):

$$DR_{dB} \approx 6.02 \cdot 2^{8} \approx 1541\,\text{dB}$$

For 32-bits fixed point:

$$DR_{dB} \approx 6.02 \cdot 31 \approx 186\,\text{dB}$$

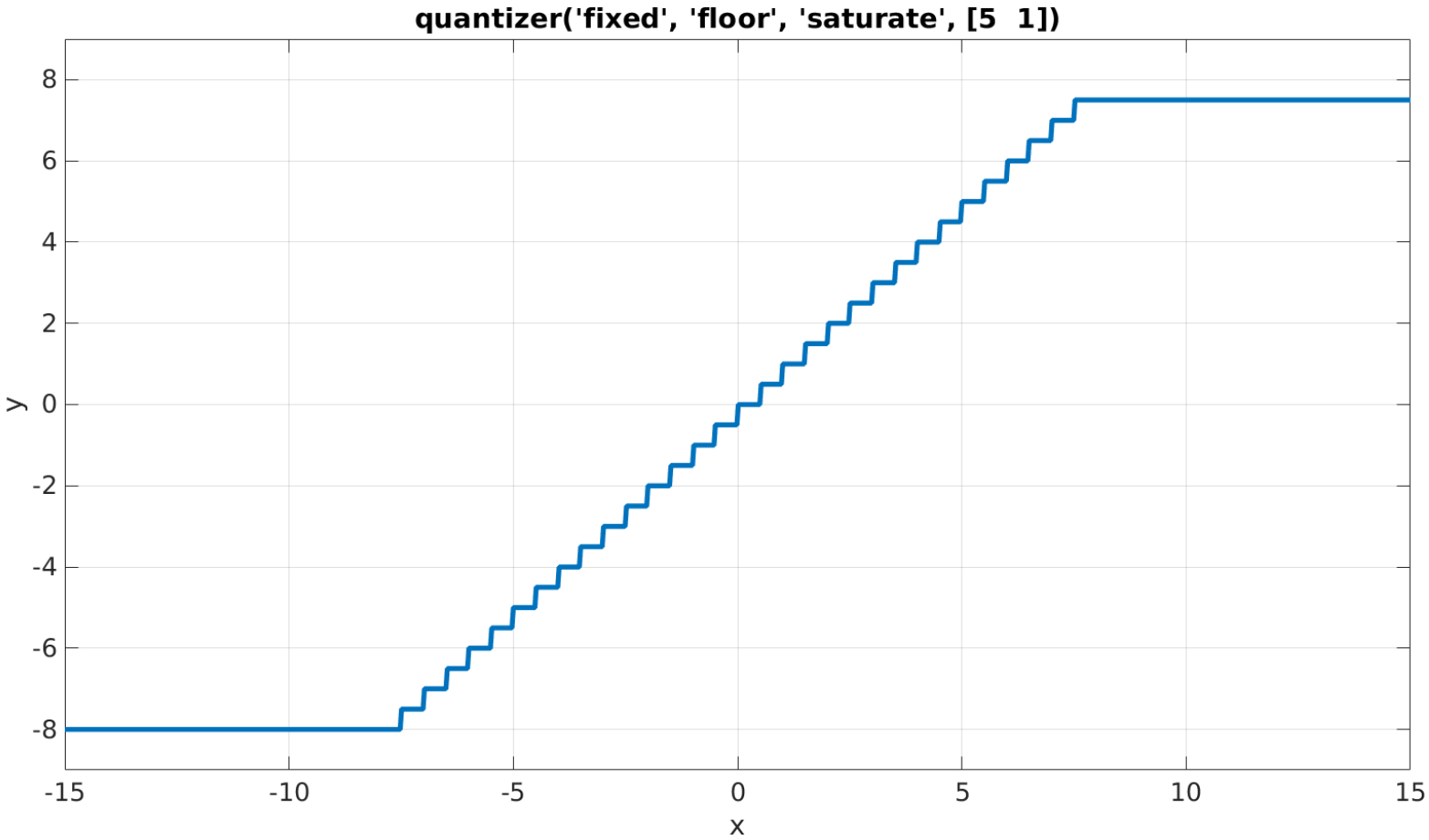
Precision

- Precision is not constant throughout all floating-point numbers' range.
- As the numbers get larger, the precision gets larger as well.

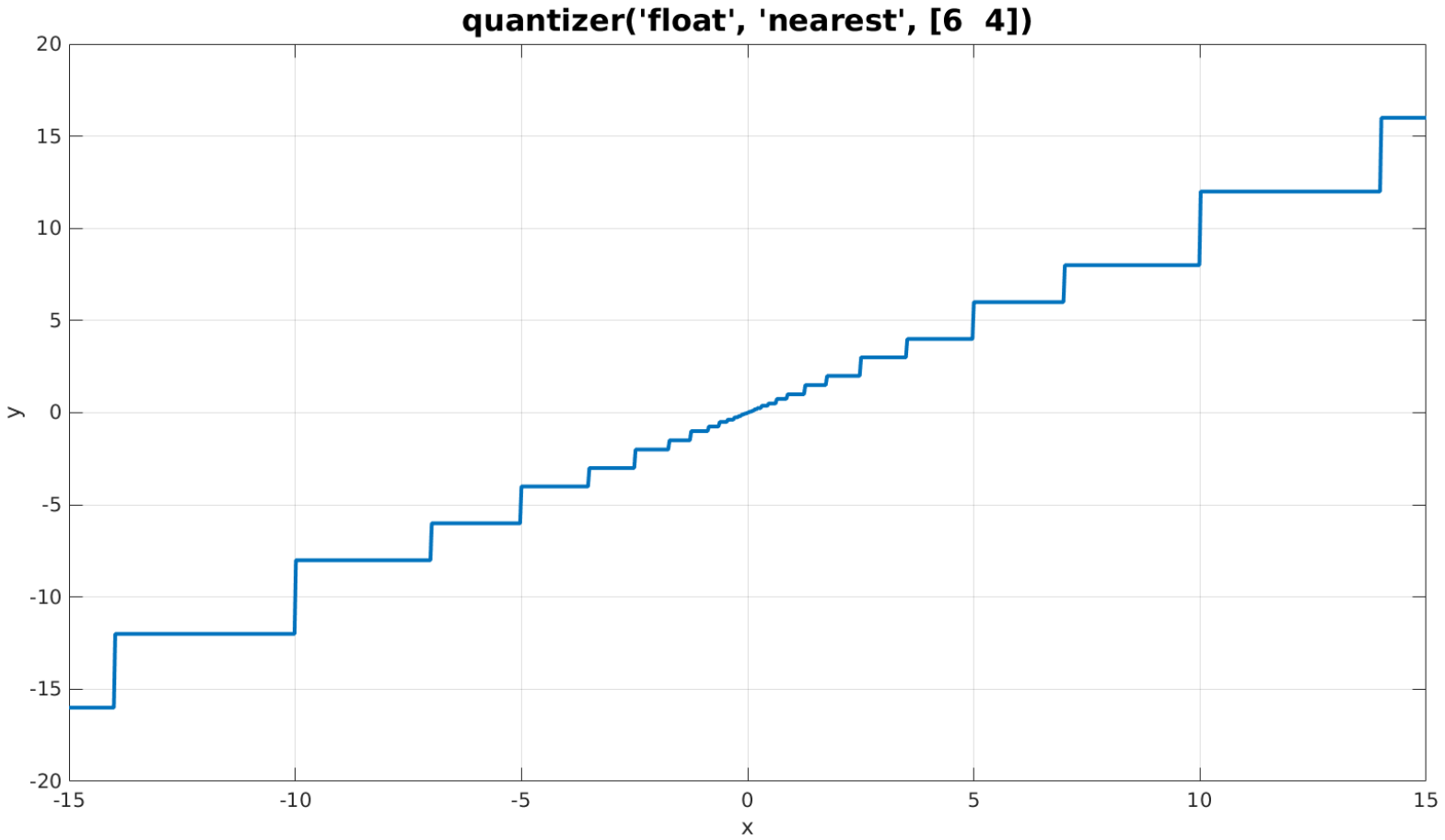
MATLAB

```
» %  Fixed-point quantizer
» q = quantizer('fixed','floor','saturate',[5 1]);
% [wordlength fractionlength]
» u = linspace(-15,15,1000);
» y1 = quantize(q,u);
» plot(u,y1); title(tostring(q))
»
» %  Floating-point quantizer
» q = quantizer([6 4],'float', 'nearest');
% [wordlength exponentlength]
» y2 = quantize(q,u);
» plot(u,y2); title(tostring(q))
```

Precision, 2


quantizer('fixed', 'floor', 'saturate', [5 1])
8
6
4
2
0
-2
-4
-6
-8
y
-15
-10
-5
0
5
10
15
x

Precision, 3


quantizer('float', 'nearest', [6 4])
20
15
10
5
0
-5
-10
-15
-20
y
-15
-10
-5
0
5
10
15
x

Precision, 4

`eps(x)` returns the positive distance from `abs(x)` to the next larger floating point number of the same precision.

MATLAB

```
» e1 = eps(single(1))
» e1 = 1.1920929e-07
```

Precision, 4

`eps(x)` returns the positive distance from `abs(x)` to the next larger floating point number of the same precision.

MATLAB

```
» e1 = eps(single(1))
» e1 = 1.1920929e-07
» e2 = eps(single(1e1))
» e2 = 9.5367432e-07
```

Precision, 4

`eps(x)` returns the positive distance from `abs(x)` to the next larger floating point number of the same precision.

MATLAB

```
» e1 = eps(single(1))
» e1 = 1.1920929e-07
» e2 = eps(single(1e1))
» e2 = 9.5367432e-07
» e3 = eps(single(1e10))
» e3 = 1024
```

Precision, 4

`eps(x)` returns the positive distance from `abs(x)` to the next larger floating point number of the same precision.

MATLAB

```
» e1 = eps(single(1))
» e1 = 1.1920929e-07
» e2 = eps(single(1e1))
» e2 = 9.5367432e-07
» e3 = eps(single(1e10))
» e3 = 1024
» t = single(1e10) + single(1300)
» t = 10000001024.00
```

Sum of two floating-point numbers

Perform $0.5 + (-0.4375)$ using 4 bits for the mantissa.

$$0.5_{10} = 0.1000_2 \times 2^0 = 1.000 \times 2^{-1} \text{ (normalised)}$$
$$-0.4375_{10} = -0.0111_2 \times 2^0 = -1.110 \times 2^{-2} \text{ (normalised)}$$

1. Match exponents to the bigger one.
   Apply n right shifts to -0.4375 where $n = (exponent1 - exponent2) = 1$.
   $-0.4375_{10} = -1.110 \times 2^{-2} = -\mathbf{0.1110} \times \mathbf{2^{-1}}$

2. Add the mantissas.
   $(1.000 - 0.1110) \times 2^{-1} = 0.001 \times 2^{-1}$

3. Normalise the sum, checking for overflow/underflow:
   $0.0625_{10} = 0.001 \times 2^{-1} = 1.000 \times 2^{-4}$
   $-126 <= -4 <= 127$, no overflow or underflow

4. Round the sum.
   The sum fits in 4 bits so rounding is not required

Sum of two floating-point numbers, 2

Perform $1e10 + 1300$ using IEEE-754 single precision.

$$1e10_{10} = (-1)^0 \times 1.00101010000001011111001 \times 2^{33} \text{ (normalised)}$$
$$1300_{10} = (-1)^0 \times 1.01000101000000000000000 \times 2^{10} \text{ (normalised)}$$

1. Match exponents to the bigger one.
   Apply n right shifts to 1300 where $n = (exponent1 - exponent2) = 23$.
   $1300 \approx \mathbf{0.00000000000000000000001} \times \mathbf{2^{33}} = (1.0 \times 2^{-23}) \times 2^{33} = 1.0 \times 2^{10} = 1024$

2. Add the mantissas.
   $(1.00101010000001011111001 + 0.00000000000000000000001) \times 2^{33}$

3. Normalise the sum, checking for overflow/underflow:
   $1.01010100000010111111010 \times 2^{33} = 1.1641533_{10} \times 2^{33}$
   $-126 <= 33 <= 127$, no overflow or underflow

4. Round the sum.
   The sum fits in 23 bits so rounding is not required

More examples

When calculations involve large and small numbers at the same time, the loss of precision affects the small number and the result.

MATLAB

1. `» (2^53 + 1) - 2^53`

More examples

When calculations involve large and small numbers at the same time, the loss of precision affects the small number and the result.

MATLAB

```
» (2^53 + 1) - 2^53
» ans = 0
```

More examples

When calculations involve large and small numbers at the same time, the loss of precision affects the small number and the result.

MATLAB

```
» (2^53 + 1) - 2^53
» ans = 0
» x = 0;
» t = tan(x) - sin(x)/cos(x)
```

More examples

When calculations involve large and small numbers at the same time, the loss of precision affects the small number and the result.

MATLAB

```
» (2^53 + 1) - 2^53
» ans = 0
» x = 0;
» t = tan(x) - sin(x)/cos(x)
» t = 0
```

More examples

When calculations involve large and small numbers at the same time, the loss of precision affects the small number and the result.

MATLAB

```
» (2^53 + 1) - 2^53
» ans = 0
» x = 0;
» t = tan(x) - sin(x)/cos(x)
» t = 0
» x = 1;
» t = tan(x) - sin(x)/cos(x)
```

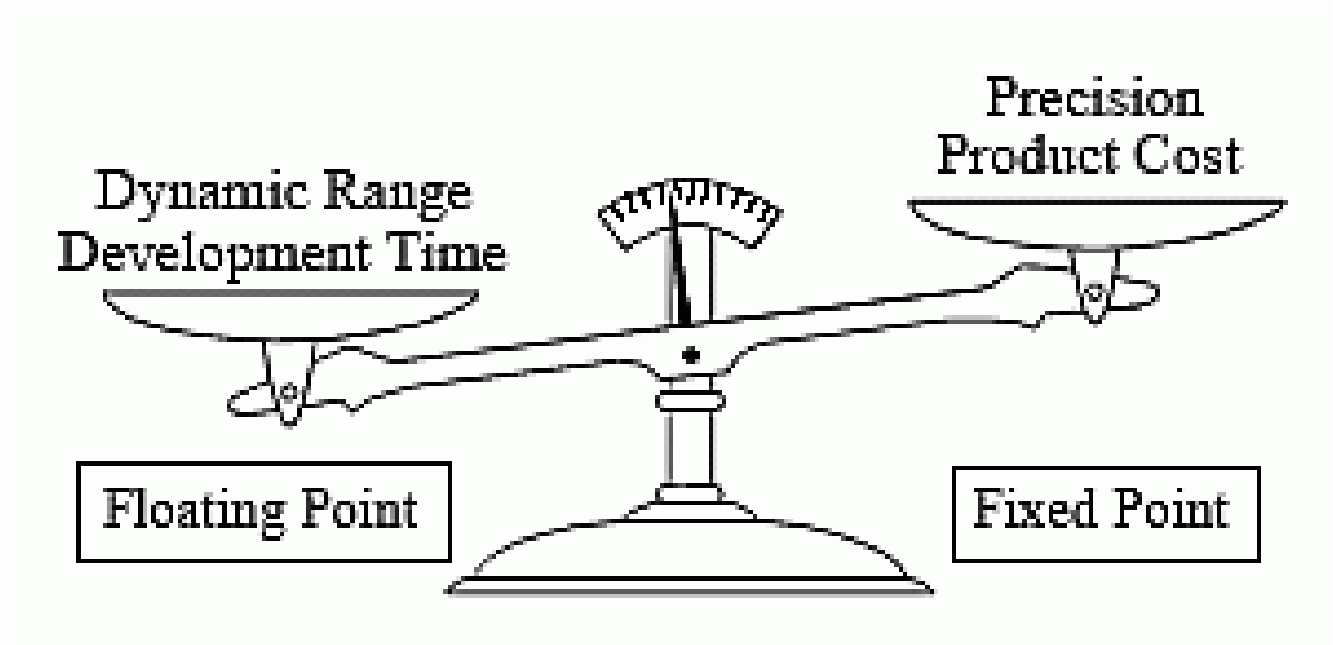
More examples

When calculations involve large and small numbers at the same time, the loss of precision affects the small number and the result.

MATLAB

```
» (2^53 + 1) - 2^53
» ans = 0
» x = 0;
» t = tan(x) - sin(x)/cos(x)
» t = 0
» x = 1;
» t = tan(x) - sin(x)/cos(x)
» t = 2.2204e-16 % eps(1)
```

Fixed-point vs floating-point



Bibliography

1 Jean-Pierre Deschamps, Gustavo D. Sutter, and Enrique Cantó. Guide to FPGA Implementation of Arithmetic Functions, Chapter 12.