Finite representation of real numbers

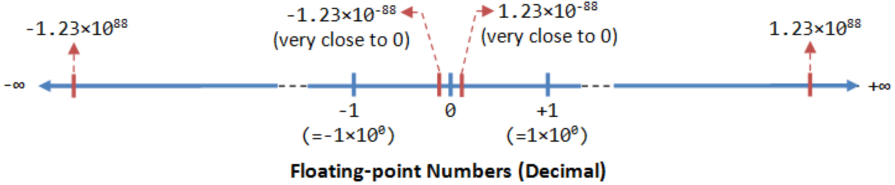
Floating-point numbers

Dr. Ing. Rodrigo Gonzalez

rodralez@frm.utn.edu.ar
rodrigo.gonzalez@ingenieria.uncuyo.edu.ar

Summary

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



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.

In this presentation r=2

IEEE Standard P754 Format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	...	2^{-21}	2^{-22}	2^{-23}
	Sign (s)	← Exponent (e) →								← Fraction (f) →						

IBM Format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	...	2^{-22}	2^{-23}	2^{-24}
	Sign (s)	← Exponent (e) →							← Fraction (f) →							

DEC (Digital Equipment Corp.) Format

Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-2}	2^{-3}	2^{-4}	...	2^{-22}	2^{-23}	2^{-24}
	Sign (s)	← Exponent (e) →								← Fraction (f) →						

MIL-STD 1750A Format

Bit	31	30	29	...	11	10	9	8	7	6	5	4	3	2	1	0
	2^0	2^{-1}	2^{-2}	...	2^{-20}	2^{-21}	2^{-22}	2^{-23}	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	← Fraction (f) →								← Exponent (e) →							

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.

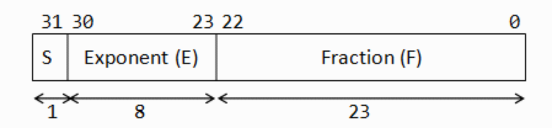
Solo vemos estos estándares base=2

	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			

Nos enfocamos en el de 32 bits que es de sistemas embebidos

IEEE 754 standard also defines:

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



32-bit Single-Precision Floating-point Number

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

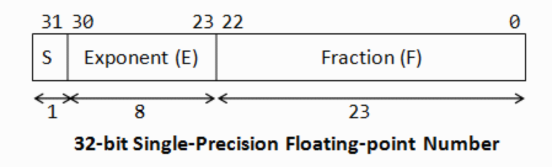
- S, sign bit. **0** for positive numbers and **1** for negative numbers.
- F, 23-bits fraction: $\left[2^{-1} \;\; 2^{-2} \cdots 2^{-23}\right]$
- We need to represent both positive and negative exponents.
- E, 8-bits exponent, **no sign bit**.
  - $E = [1, 254]$, $bias = 127$; $\quad -126 \leq E -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

Reservados para valores filmina 14

Se implementa para evitar bit de signo en el exponente

IEEE 754 standard

Normalized Form



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

Exponente E-bias resulta ser el máximo tal que quede parte entera 1 al pasarlo dividiendo

- Representation of a floating point number may not be unique:
- For example, the number 13.25 can be represented as $1101.01_2 \cdot (2^0) = 110.101_2 \cdot (2^1) = 11.0101_2 \cdot (2^2) = 1.10101_2 \cdot (2^3)$
- 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. Forma normalizada
- The leading 1 is **implicit**. It is not part of the 32 bits number.
- $1.F = \mathbf{1.}\left[2^{-1} \;\; 2^{-2} \cdots 2^{-23}\right]$.

Represent 3215.020002_{10}

Decimal Value Entered: 3215.020002

No tiene representación exacta en 32bits single precision

Single precision (32 bits):

Binary: *Status:* normal

Bit 31 Sign Bit	Bits 30 - 23 Exponent Field	Bits 22 - 0 Significand
0	100 0101 0	1 .100 1000 1111 0000 0101 0010
0: + 1: -	Decimal value of exponent field and exponent 138 - 127 = 11	Decimal value of the significand 1.5698340

Hexadecimal: 4548F052 *Decimal:* 3215.0200

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

Da 3215,020032 pero se pierden últimos bits. Quizás incluso más si pones bien la representación de la parte fraccionaria

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: 139 - 127 = 12	Decimal value of the significand: 1.5698340

Hexadecimal: 45C8F052 *Decimal:* 6430.0400

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 136 - 127 = 9	Decimal value of the significand 1.5698340

Hexadecimal: 4448F052 *Decimal:* 803.75500

- To multiply and divide by 2 is easy using floating-point numbers.
- Not every real number can be represented with floating-point format.
- Floating-point numbers are auto range !

IEEE 754 standard

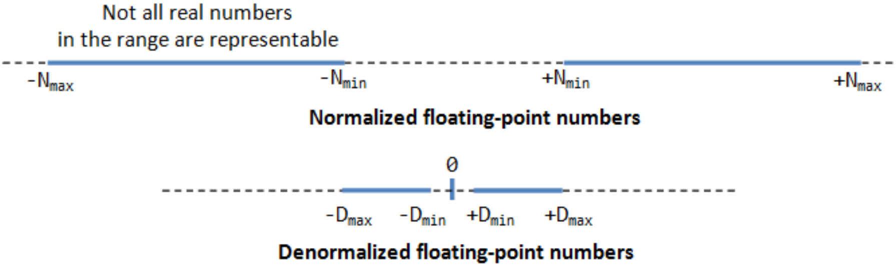
Why auto range?

2^E	**MIN F** 1..00000000000000000000000 (b2) 1 (b10)	**MAX F** 1..11111111111111111111111 (b2) 1.999999881 (b10)
2^-126	1.1755E-38	2.3510E-38
2^-30	9.3132E-10	1.8626E-09
2^-20	9.5367E-07	1.9073E-06
2^-10	9.7656E-04	1.9531E-03
2^-3	0.12500000	0.24999999
2^-2	0.25000000	0.49999997
2^-1	0.50000000	0.99999994
2^0	**1.00000000**	**1.99999988**
2^1	2.00000000	3.99999976
2^2	4.00000000	7.99999952
2^3	8.00000000	15.99999905
2^10	1.0240E+03	2.0480E+03
2^20	1.0486E+06	2.0972E+06
2^30	1.0737E+09	2.1475E+09
2^127	1.7014E+38	3.4028E+38

Aumenta el error absoluto

IEEE 754 standard

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.
- $E = 0 \;\Rightarrow\; E - bias = -127$

  Forma de-normalizada
- Implicit leading $0.F = \mathbf{0.}\left[2^{-1}\;\; 2^{-2} \cdots 2^{-23}\right]$.

Para 2^-127 asume 0 implícito

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

Decimal Value Entered: -3.4e-39

Single precision (32 bits):

Binary: *Status:* denormalized

Bit 31 Sign Bit	Bits 30 - 23 Exponent Field	Bits 22 - 0 Significand
1	00000000	0.1001010000010111010001
0: + 1: -	Decimal value of exponent field and exponent 0 - 127 = -127	Decimal value of the significand 0.5784800

Hexadecimal: 802505D1 *Decimal:* -3.3999999e-39

Special values

E:exponente F:Parte fraccionaria

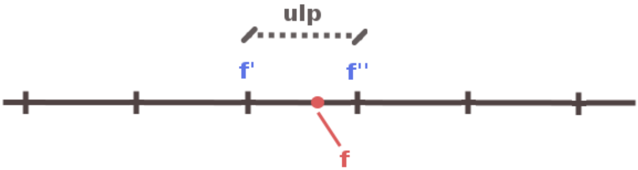
- **Zero**: $E = 0$, $F = 0$. Two representations: **+0** ($S = 0$) and **-0** ($S = 1$).
- **Inf** (Infinity): $E = 0xFF$, $F = 0$. Two representations: **+Inf** ($S = 0$) and **-Inf** ($S = 1$).
- **NaN** (Not a Number): $E = 0xFF$, $F \neq 0$. A value that cannot be represented as a real number (e.g.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

MATLAB

```
1 » a = 1/0
2 » a = Inf
3 » b = exp(1000)
4 » b = Inf
5 » c = log(0)
6 » c = -Inf
```

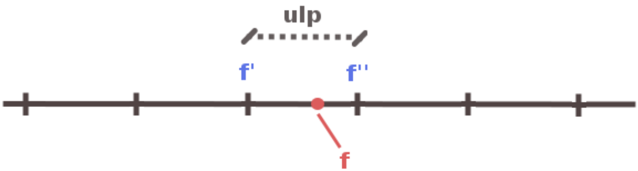
MATLAB

```
1 » d = -1/0
2 » d = -Inf
3 » e = 0/0
4 » d = NaN
5 » f = Inf/Inf
6 » d = NaN
```



- *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



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,

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

I think to be exact it is 6,02*(3+2^bE)

where b_E is the number of bits of E.

For single precision (32-bits):

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

"Con la misma cantidad de bits, podemos representar muchísimos más números"
Creo que quiso decir mayor rango.

For 32-bits fixed point:

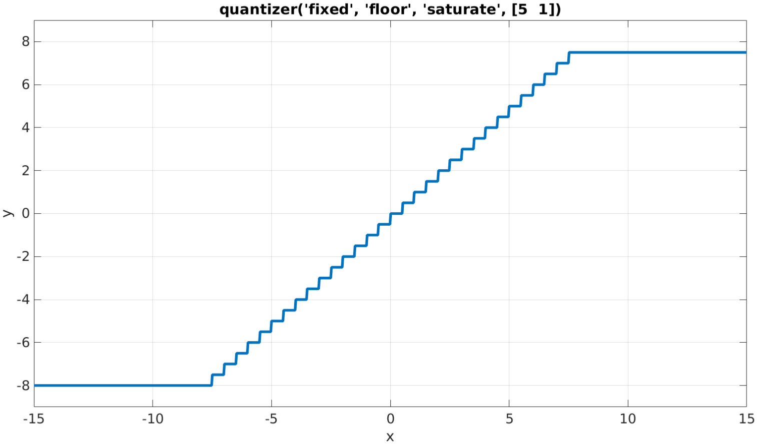
$$DR_{dB} \approx 6.02 \cdot 31 \approx 186\,\text{dB}$$

Me parece que convienen porque por lo general te importa más error relativo chico que absoluto chico.

- Precision is 2^{-n}, where n is the number of bits for the fraction part.
- Precision is constant throughout all fixed-point numbers' range.

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))
```

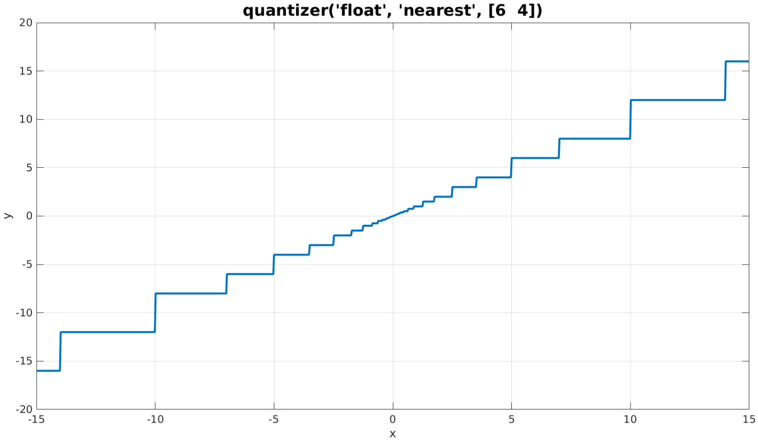

quantizer('fixed', 'floor', 'saturate', [5 1])
8
6
4
2
0
-2
-4
-6
-8
y
-15
-10
-5
0
5
10
15
x

- Precision is $2^E \cdot 2^{-23}$ for single precision.
- Precision is **not** constant throughout all floating-point numbers' range.
- As the numbers get larger, the precision gets larger as well.

MATLAB

```
» % Floating-point quantizer
» q = quantizer([6 4],'float', 'nearest');
% [wordlength exponentlength]
» y2 = quantize(q,u);
» plot(u,y2); title(tostring(q))
```

Quantizer es como un ADC.


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

2^E	MIN F 1..00000000000000000000000 (b2) 1 (b10)	MAX F 1..11111111111111111111111 (b2) 1.999999881 (b10)	PRECISION 2^E * 2^-23
2^-126	1.1755E-38	2.3510E-38	1.4013E-45
2^-30	9.3132E-10	1.8626E-09	1.1102E-16
2^-20	9.5367E-07	1.9073E-06	1.1369E-13
2^-10	9.7656E-04	1.9531E-03	1.1642E-10
2^-3	0.12500000	0.24999999	1.4901E-08
2^-2	0.25000000	0.49999997	
2^-1	0.50000000	0.99999994	5.9605E-08
2^0	**1.00000000**	**1.99999988**	**1.1921E-07**
2^1	2.00000000	3.99999976	2.3842E-07
2^2	4.00000000	7.99999952	4.7684E-07
2^3	8.00000000	15.99999905	9.5367E-07
2^10	1.0240E+03	2.0480E+03	1.2207E-04
2^20	1.0486E+06	2.0972E+06	1.2500E-01
2^30	1.0737E+09	2.1475E+09	1.2800E+02
2^127	1.7014E+38	3.4028E+38	2.0282E+31

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

MATLAB

```
» a = (2^53 + 1) - 2^53
» a = 0
» if (a == 0)
»    disp('Turn off nuclear reactor')
» else
»    disp('Do not turn off nuclear reactor')

» 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)
```

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

$$0.5_{10} = 0.1000_2 \times 2^0 = 1.0000_2 \times 2^{-1} \text{ (normalised)}$$
$$-0.4375_{10} = -0.0111_2 \times 2^0 = -1.1100_2 \times 2^{-2} \text{ (normalised)}$$

1. Match exponents to the bigger one.
   Apply n right shifts to -0.4375 where $n = (exponent1 - exponent2) = (-1 + 2) = 1$.
   $-0.4375 = -1.1100_2 \times 2^{-2} = -\mathbf{0.1110_2} \times \mathbf{2^{-1}}$
2. Add the mantissas.
   $(1.0000_2 - 0.1110_2) \times 2^{-1} = 0.0010_2 \times 2^{-1}$
3. Normalise the sum, checking for overflow/underflow:
   $0.0010_2 \times 2^{-1} = 1.0000_2 \times 2^{-4} = \mathbf{0.0625}$
   $-126 < -4 < 127,$ no overflow or underflow
4. Round the sum.
   The sum fits in 4 bits so rounding is not required

Perform $1e10 + 1500$ using IEEE-754 single precision.

$$10,000,000,000 = 1.00101010000001011111001_2 \times 2^{33} \text{ (normalised)}$$
$$1500 = 1.01110111000000000000000_2 \times 2^{10} \text{ (normalised)}$$

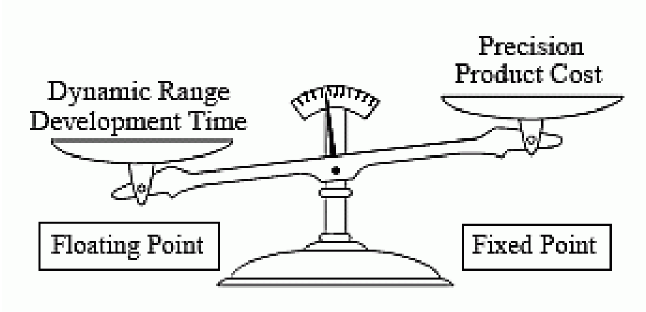
1. Match exponents to the bigger one.
   Apply n right shifts to 1500 where $n = (exponent1 - exponent2) = (33 - 10) = 23$
   $1500 \approx \mathbf{0.00000000000000000000001_2 \times 2^{33}} = (1.0 \times 2^{-23}) \times 2^{33} = 1.0 \times 2^{10} = 1024$

2. Add the mantissas.
   $(1.00101010000001011111001_2 + 0.00000000000000000000001_2) \times 2^{33}$

3. Normalise the sum, checking for overflow/underflow:
   $1.0101010000001011111010_2 \times 2^{33} = 1.16415333710 \times 2^{33} = \mathbf{10,000,001,024}$
   $-126 < 33 < 127, \;\;$ no overflow or underflow

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

Fixed-point vs floating-point



Bibliography

1 IEEE-SA Standards Board. IEEE Standard for Floating-Point Arithmetic. ISBN 978-0-7381-5752-8. Approved 12 June 2008. New York, NY, USA. Link.

2 Jean-Pierre Deschamps, Gustavo D. Sutter, and Enrique Cantó. Floating Point Arithmetic. Guide to FPGA Implementation of Arithmetic Functions, Chapter 12. Springer, 2012.