

Finite representation of real numbers

Floating-point numbers

Dr. Ing. Rodrigo Gonzalez

`rodraz@frm.utn.edu.ar`

Técnicas Digitales III

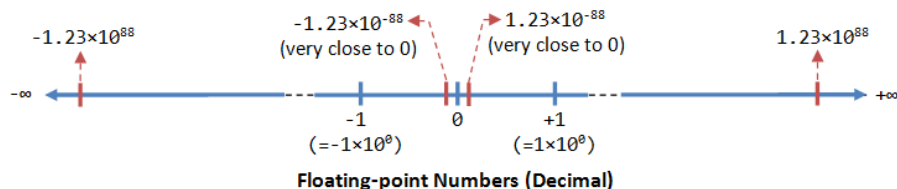
Universidad Tecnológica Nacional,
Facultad Regional Mendoza.



- 1 Floating-point Representation
 - Floating-point Representation
 - Standards
 - Normalized Form
- 2 Floating-point Examples
- 3 De-normalized Form
- 4 Special values
- 5 Rounding schemes
- 6 Dynamic range
- 7 Precision
- 8 Precision problems

Floating-point Representation

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 , exponent.
- r , certain radix. $r = 2$ for binary; $r = 10$ for decimal.

Modern computers adopt IEEE 754-2008 standard for representing floating-point numbers.

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

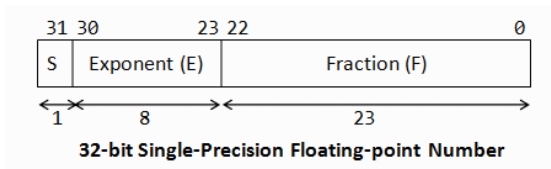
Bit	31	30	29	28	27	26	25	24	23	22	21	20	...	2	1	0
	S	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-2}	2^{-3}	2^{-4}	...	2^{-22}	2^{-23}	2^{-24}
Sign (s)	← Exponent (e) →									← Fraction (f) →						

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) →							

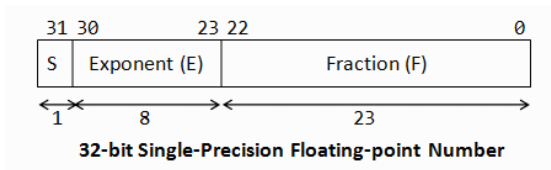
IEEE 754-2008 standard defines several formats.

Parameter	Binary formats ($B = 2$)				Decimal formats ($B = 10$)		
	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			



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

- S, sign bit. 0 for positive numbers and 1 for negative numbers.
- E, 8-bits exponent.
- We need to represent both positive and negative exponents.
- $E = [1, 254]$, $bias = 127$; $-126 \leq E - bias \leq 127$.
- $E = 0$ and $E = 255$ are reserved.
- F, 23-bits fraction.



- Representation of a floating point number may not be unique:
 $11.01_2 = 1.101_2 \times 2^1 = 110.1_2 \times 2^{-1}$.
- Therefore, the fractional part F is normalized.
- $1.F$, implicit leading 1.

Example 1

Represent 3215.020002_{10}

Decimal Value Entered: 3215.020002

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>

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 139 - 127 = 12	Decimal value of the significand 1.5698340

Hexadecimal: 45C8F052 Decimal: 6430.0400

Example 3

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

Decimal Value Entered: 803.7550005

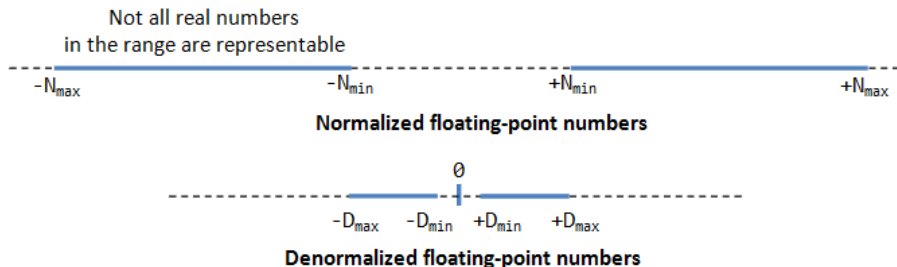
Single precision (32 bits):

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 136 - 127 = 9	Decimal value of the significand 1.5698340

Hexadecimal: 4448F052 Decimal: 803.75500

Floating-point numbers are auto-scaled!



- Normalized form has a serious problem.
- With an implicit leading 1 for the fraction, it cannot represent the number zero!
- De-normalized form is devised to represent zero and small numbers.
- $E = 0 \Rightarrow 0.F$, implicit leading 0.

Example

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

Decimal Value Entered:

Single precision (32 bits):

Binary: Status:

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

Hexadecimal: Decimal:

Special values

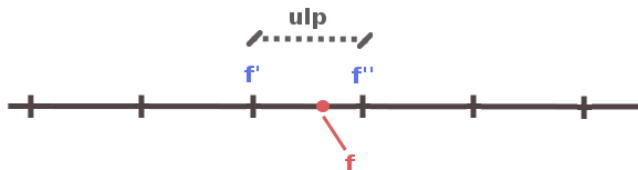
- **Zero:** $E = 0, F = 0$. Two representations: **+0** ($S = 0$) and **-0** ($S = 1$).
- **Inf** (Infinity): $E = 0xFF, F = 0$. Two representations: **+Inf** ($S = 0$) and **-Inf** ($S = 1$).
- **NaN** (Not a Number): $E = 0xFF, F \neq 0$. A value that cannot be represented as a real number (e.g. $0/0$).

MATLAB

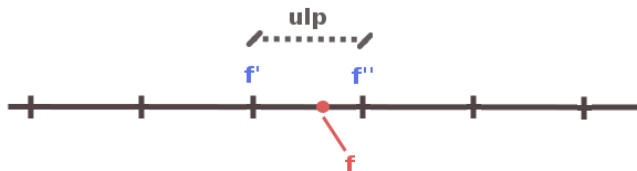
```
❶ » a = 1/0
❷ » ans = Inf
❸ » b = -1/0
❹ » ans = -Inf
❺ » c = 0/0
❻ » ans = NaN
```

C code

```
❶ #include <math.h>
❷ a = NAN;
❸ b = INFINITY;
❹ c = -INFINITY;
```



- ulp (unit of least precision, $eps()$).
- 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 are:

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

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

where b_E is the number of bits of E .

For single precision (32-bits):

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

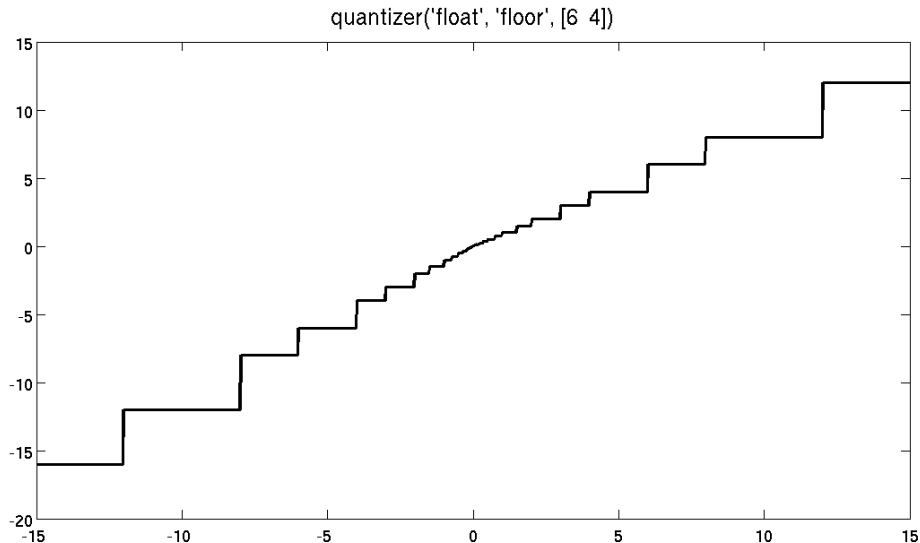
For fixed-point Q31 (32-bits):

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

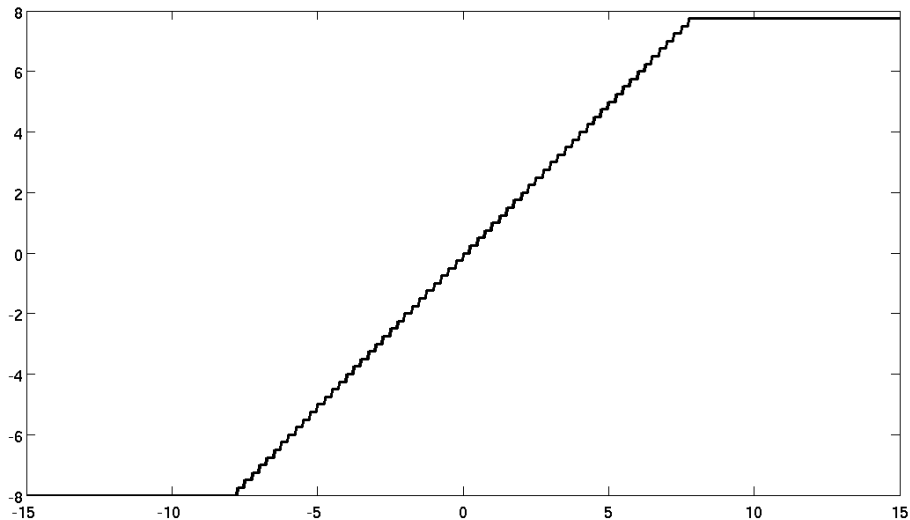
- Precision is not constant throughout floating point numbers' range.
- As the numbers get larger, the precision gets larger.

MATLAB

```
❶ » u = linspace(-15,15,1000);  
❷ » q = quantizer([6 4],'float'); % [wordlength exponentlength]  
❸ » y1 = quantize(q,u);  
❹ » plot(u,y1); title(tostring(q))  
❺ »  
❻ » q = quantizer('fixed',[6 2]); % [wordlength fractionlength]  
❼ » y2 = quantize(q,u);  
❽ » plot(u,y2); title(tostring(q))
```



quantizer('fixed', 'floor', 'saturate', [6 2])



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

MATLAB

```
❶ » e1 = eps(single(1))  
❷ » e1 = 1.1920929e-07  
❸ » e2 = eps(single(1e1))  
❹ » e2 = 9.5367432e-07  
❺ » e3 = eps(single(1e10))  
❻ » e3 = 1024  
❼ » t = single(1e10) + single(1300)  
❽ » t = 10000001024.00
```

Sum of two floating-point positive numbers

Perform $0.5_{10} + (-0.4375_{10})$ using 4 bits for the mantissa.

$$0.5_{10} = 0.1_2 \times 2^0 = 1.000_2 \times 2^{-1} \text{ (normalised)}$$

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

- ❶ Matches with the exponent of the larger number:
Apply n left shifts to -1.110_2 where $n = (\text{exponent1} - \text{exponent2})$.
 $-1.110_2 \times 2^{-2} = -0.1110_2 \times 2^{-1}$
- ❷ Add the mantissas:
 $1.000_2 \times 2^{-1} + -0.1110_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$
- ❸ Normalise the sum, checking for overflow/underflow:
 $0.001_2 \times 2^{-1} = 1.000_2 \times 2^{-4}$
 $-126 \leq -4 \leq 127$ No overflow or underflow
- ❹ Round the sum:
The sum fits in 4 bits so rounding is not required

Sum of two floating-point positive numbers, II

Perform $1e10_{10} + 1300_{10}$ using IEEE-754 single precision.

$$1e10_{10} = (-1)^0 \times 1.00101010000001011111001_2 \times r^{(160-127)} \text{ (normalised)}$$

$$1300_{10} = (-1)^0 \times 1.01000101000000000000000_2 \times r^{(137-127)} \text{ (normalised)}$$

- ❶ Matches with the exponent of the larger number:

Apply n left shifts to $1.01000101000000000000000_2$ where $n = (\text{exponent1} - \text{exponent2})$.

$$1300_{10} = 1.01000101000000000000000_2 \times r^{(137-127)}$$

$$1300_{10} = 0.00000000000000000000000_2 \times r^{(160-127)}$$

- ❷ Add the mantissas:

$$(1.00101010000001011111001_2 + 0.00000000000000000000000_2) \times r^{(160-127)}$$

- ❸ Normalise the sum, checking for overflow/underflow:

$$1.00101010000001011111001_2 \times r^{(160-127)}$$

$-126 \leq (160 - 127) \leq 127$ No overflow or underflow

- ❹ Round the sum:

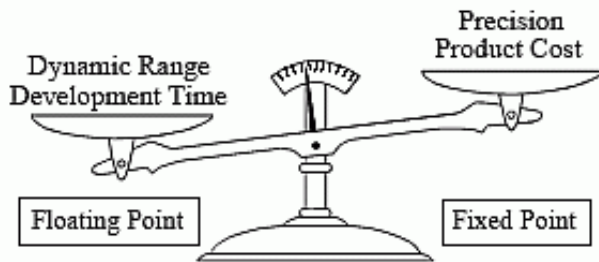
The sum fits in 23 bits so rounding is not required

- 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
2 » ans = 0
3 » x = 1;
4 » t = tan(x) - sin(x)/cos(x)
5 » t = 2.2204e-16 % eps(1)
```

Fixed-point vs floating-point



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