Finite representation of real numbers

Floating-point numbers

Dr. Ing. Rodrigo Gonzalez

rodrigo.gonzalez@ingenieria.uncu.edu.ar

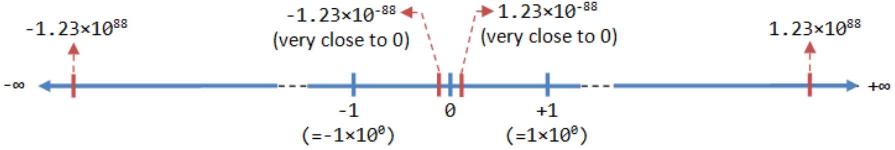
Control y Sistemas

Facultad de Ingeniería,
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.

Standards

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

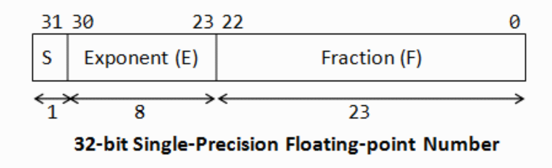
Bit	31	30	29	28	27	26	25	24	23	22	21	20	⋯	2	1	0
	S	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-2}	2^{-3}	2^{-4}	⋯	2^{-22}	2^{-23}	2^{-24}
	Sign (s)	← Exponent (e) →								← Fraction (f) →						

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.

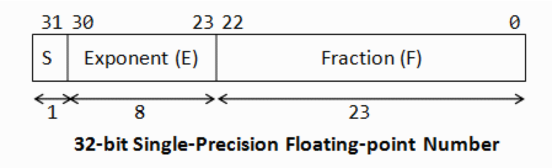
	Binary formats ($B = 2$)				Decimal formats ($B = 10$)		
Parameter	Binary 16	Binary 32	Binary 64	Binary 128	Decimal 132	Decimal 164	Decimal 128
p, digits	$10 + 1$	$23 + 1$	$52 + 1$	$112 + 1$	7	16	34
e_{max}	+15	+127	+1023	+16383	+96	+384	+16,383
e_{min}	−14	−126	−1022	−16382	−95	−383	−16,382
Common name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



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

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

Normalized Form



- 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: $\quad \mathbf{1.}\ \left[2^{-1}\ \ 2^{-2} \cdots 2^{-23}\right]$.

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

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

Decimal Value Entered: 803.7550005

Single precision (32 bits):

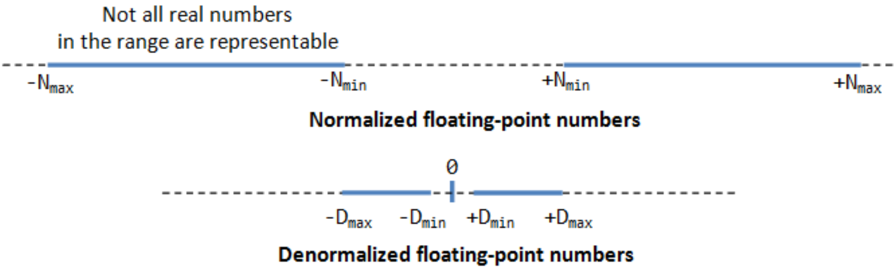
Binary: *Status:* normal

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!

De-normalized Form



- 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: $\quad \mathbf{0.}\left[2^{-1}\;\; 2^{-2}\cdots 2^{-23}\right]$.

Example

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	0.10010100000010111010001
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

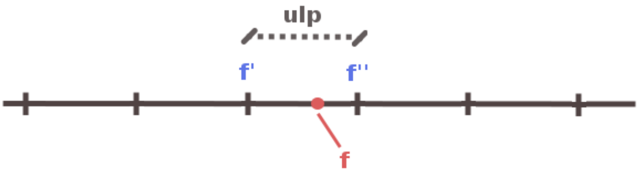
Special values

- **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

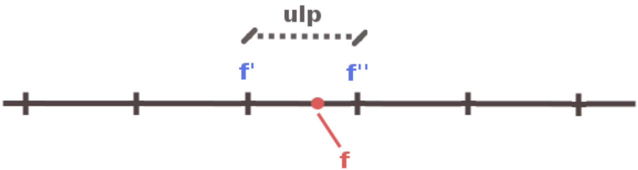
1. » a = 1/0
2. » ans = Inf
3. » b = -1/0
4. » ans = -Inf
5. » c = 0/0
6. » ans = NaN

Rounding schemes



- *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 II



Rounding schemes are:

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

Dynamic range

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

where b_E is the number of bits of E.

For single precision (32-bits):

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

For 32-bits fixed point :

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

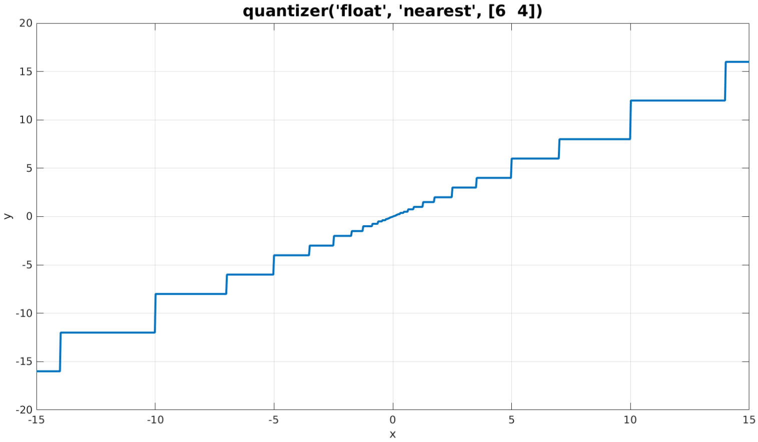
Precision

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

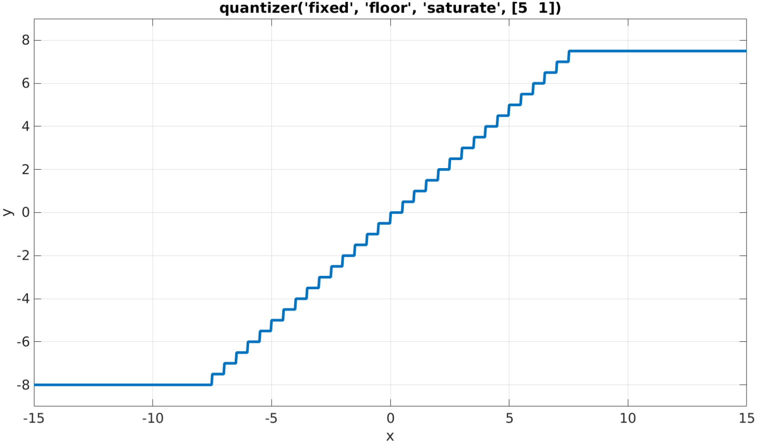
MATLAB

```
1 » u = linspace(-15,15,1000);
2 » q = quantizer([6 4],'float');  % [wordlength exponentlength]
3 » y1 = quantize(q,u);
4 » plot(u,y1); title(tostring(q))
5 »
6 » q = quantizer('fixed',[5 1]); % [wordlength fractionlength]
7 » y2 = quantize(q,u);
8 » plot(u,y2); title(tostring(q))
```

Precision II



Precision III



Precision IV

`eps(x)` returns the positive distance from `abs(x)` to the next larger in magnitude 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 positive numbers

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

$$0.5 = 0.1 \times 2^0 = 1.000 \times 2^{-1} \text{ (normalised)}$$
$$-0.4375 = -0.0111 \times 2^0 = -1.110 \times 2^{-2} \text{ (normalised)}$$

1. Matches with the exponent of the larger number:

   Apply n left shifts to -1.110 where $n = (exponent1 - exponent2)$.

   $-1.110 \times 2^{-2} = -0.1110 \times 2^{-1}$

2. Add the mantissas:

   $1.000 \times 2^{-1} + -0.1110 \times 2^{-1} = 0.001 \times 2^{-1}$

3. Normalise the sum, checking for overflow/underflow:

   $0.001 \times 2^{-1} = 1.000 \times 2^{-4}$

   $-126 <= -4 <= 127$ No overflow or underflow

4. Round the sum:

   The sum fits in 4 bits so rounding is not required

Sum of two floating-point positive numbers, II

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

$$1e10 = (-1)^0 \times 1.00101010000001011111001 \times r^{(160-127)} \text{ (normalised)}$$
$$1300 = (-1)^0 \times 1.01000101000000000000000 \times r^{(137-127)} \text{ (normalised)}$$

1. Matches with the exponent of the larger number:

   Apply n left shifts to 1.01000101000000000000000 where $n = (exponent1 - exponent2)$.

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

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

2. Add the mantissas:

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

3. Normalise the sum, checking for overflow/underflow:

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

   $-126 <= (160 - 127) <= 127$ No overflow or underflow

4. Round the sum:

   The sum fits in 23 bits so rounding is not required

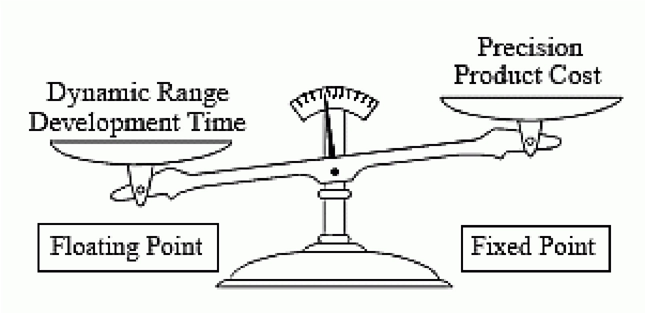
More examples

- 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
2. » ans = 0
3. » x = 0;
4. » t = tan(x) - sin(x)/cos(x)
5. » t = 0
6. » x = 1;
7. » t = tan(x) - sin(x)/cos(x)
8. » 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.