# Finite representation of real numbers

## Fixed-point and floating-point formats

Ing. Rodrigo González
rodralez@frm.utn.edu.ar

Técnicas Digitales III

Universidad Tecnológica Nacional,
Facultad Regional Mendoza.

# Resumen

# Representation

Unsigned integers

- An N-bit binary word can represent a total of $2^N$ separate values.
- Range: 0 to $2^N - 1$
- $n_{10} = 2^{N-1}b_{N-1} + 2^{N-2}b_{N-2} + \cdots + 2^1 b_1 + 2^0 b_0$

2's complement signed integers

- Range: $-2^{N-1}$ to $2^{N-1} - 1$.
- $n_{10} = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$

| Bit Pattern | Unsigned | 2's Complement |
|---|---|---|
| 0000 0000 | 0 | 0 |
| 0000 0001 | 1 | 1 |
| 0000 0010 | 2 | 2 |
| • | • | • |
| • | • | • |
| | | |
| 0111 1110 | 126 | 126 |
| 0111 1111 | 127 | 127 |
| 1000 0000 | 128 | -128 |
| 1000 0001 | 129 | -127 |
| • | • | • |
| • | • | • |
| 1111 1110 | 254 | -2 |
| 1111 1111 | 255 | -1 |

How much bits are needed to represent $-\alpha_{min} \leq \alpha \leq \alpha_{max}$ ?

$$N = \text{floor}(\log_2(\max([\alpha_{min}, \alpha_{max}])) + 2)$$

# Representation, cont'd

$$N = \text{floor}(\log_2(\max([\alpha_{min}, \alpha_{max}])) + 2)$$

## MATLAB

```
» a_m = 15; a_M = 15;
» N = floor (log2 ( max ( [ a_m , a_M] ) ) + 2 );
```

# Representation, cont'd

$$N = \text{floor}(\log_2(\max([\alpha_{min}, \alpha_{max}])) + 2)$$

## MATLAB

```
1 » a_m = 15; a_M = 15;
2 » N = floor (log2 ( max ( [ a_m , a_M] ) ) + 2 );
3 » N = 5.00
```

# "Q" notation

The fractional notation can be applied to the 2's complement notation.

Qm.n

- *m* represents the number of bits to the left of the binary point.
- *n* represents the number of bits to the right of the binary point.
- The weights of bits that are to the right of the binary point are negative powers of 2: $2^{-1} = \frac{1}{2}$, $2^{-2} = \frac{1}{4}$ ... , etc.
- The naming convention does not take the MSB of the number (sign bit) into account. A Qm.n notation therefore uses $m + n + 1$ bits.
- $n_{10} = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^{i-n}$
- Precision: $2^{-n}$.
- Range: $-2^m$ to $2^m - 2^{-n}$.

# "Q" notation, cont'd

For instance:

- Q0.15 (Q15)
  - 16 bits;
  - Range: -1 to 0.99996948;
  - Precision: 1/32768 ($2^{-15}$).
- Q3.12
  - 16 bits;
  - Range: -8 to 7.9998;
  - Precision: 1/4096 ($2^{-12}$).
- Q0.31 (Q31)
  - 32 bits;
  - Range: -1 to 0.999999999534339;
  - Precision: 4.6566129e-10 ($2^{-31}$).

# Scale of representation

- Values represented in Qm.n notation can be seen as an integer simply divided by a power-of-two scale factor, $2^n$.
- The scale factor is in the head of the designer.
- So, the scale factor can be an arbitrary scale that is not a power of two.
- Example: 16-bit 2's complement numbers between 8000H and 7FFFH can represent decimal values between –5 and +5, where the scale factor is 5/32768.

# Dynamic range

Dynamic range is defined as,

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

$$DR_{dB} = 20\ log_{10}\left(\frac{2^b - 1}{1}\right)\ \text{dB}$$

$$DR_{dB} = 20\ \left(log_{10}(2^b - 1) - log_{10}(1)\right)\ \text{dB}$$

$$DR_{dB} = 20\ log_{10}(2) \cdot b\ \text{dB}$$

$$DR_{dB} = 6.02 \cdot b\ \text{dB}$$

# Precision examples

| Format (N.M) | | Largest positive value (0x7FFF) | Least negative value (0x8000) | Precision (0x0001) | | DR(dB) |
|---|---|---|---|---|---|---|
| 1 | 15 | 0,999969482421875 | -1 | 3,05176E-05 | 2^-15 | 90,30873362 |
| 2 | 14 | 1,99993896484375 | -2 | 6,10352E-05 | 2^-14 | 90,30873362 |
| 3 | 13 | 3,9998779296875 | -4 | 0,00012207 | 2^-13 | 90,30873362 |
| 4 | 12 | 7,999755859375 | -8 | 0,000244141 | 2^-12 | 90,30873362 |
| 5 | 11 | 15,99951171875 | -16 | 0,000488281 | 2^-11 | 90,30873362 |
| 6 | 10 | 31,99902344 | -32 | 0,000976563 | 2^-10 | 90,30873362 |
| 7 | 9 | 63,99804688 | -64 | 0,001953125 | 2^-9 | 90,30873362 |
| 8 | 8 | 127,9960938 | -128 | 0,00390625 | 2^-8 | 90,30873362 |
| 9 | 7 | 255,9921875 | -256 | 0,0078125 | 2^-7 | 90,30873362 |
| 10 | 6 | 511,984375 | -512 | 0,015625 | 2^-6 | 90,30873362 |
| 11 | 5 | 1023,96875 | -1024 | 0,03125 | 2^-5 | 90,30873362 |
| 12 | 4 | 2047,9375 | -2048 | 0,0625 | 2^-4 | 90,30873362 |
| 13 | 3 | 4095,875 | -4096 | 0,125 | 2^-3 | 90,30873362 |
| 14 | 2 | 8191,75 | -8192 | 0,25 | 2^-2 | 90,30873362 |
| 15 | 1 | 16383,5 | -16384 | 0,5 | 2^-1 | 90,30873362 |
| 16 | 0 | 32767 | -32768 | 1 | 2^-0 | 90,30873362 |

# Scale factor examples

| Format | Scaling factor ( ) | Range in Hex (fractional value) |
|---|---|---|
| (1.15) | $2^{15} = 32768$ | 0x7FFF (0.99) → 0x8000 (–1) |
| (2.14) | $2^{14} = 16384$ | 0x7FFF (1.99) → 0x8000 (–2) |
| (3.13) | $2^{13} = 8192$ | 0x7FFF (3.99) → 0x8000 (–4) |
| (4.12) | $2^{12} = 4096$ | 0x7FFF (7.99) → 0x8000 (–8) |
| (5.11) | $2^{11} = 2048$ | 0x7FFF (15.99) → 0x8000 (–16) |
| (6.10) | $2^{10} = 1024$ | 0x7FFF (31.99) → 0x8000 (–32) |
| (7.9) | $2^{9} = 512$ | 0x7FFF (63.99) → 0x8000 (–64) |
| (8.8) | $2^{8} = 256$ | 0x7FFF (127.99) → 0x8000 (–128) |
| (9.7) | $2^{7} = 128$ | 0x7FFF (511.99) → 0x8000 (–512) |
| (10.6) | $2^{6} = 64$ | 0x7FFF (1023.99) → 0x8000 (–1024) |
| (11.5) | $2^{5} = 32$ | 0x7FFF (2047.99) → 0x8000 (–2048) |
| (12.4) | $2^{4} = 16$ | 0x7FFF (4095.99) → 0x8000 (–4096) |
| (13.3) | $2^{3} = 8$ | 0x7FFF (4095.99) → 0x8000 (–4096) |
| (14.2) | $2^{2} = 4$ | 0x7FFF (8191.99) → 0x8000 (–8192) |
| (15.1) | $2^{1} = 2$ | 0x7FFF (16383.99) → 0x8000 (–16384) |
| (16.0) | $2^{0} = 1$(Integer) | 0x7FFF (32767) → 0x8000h (–32768) |

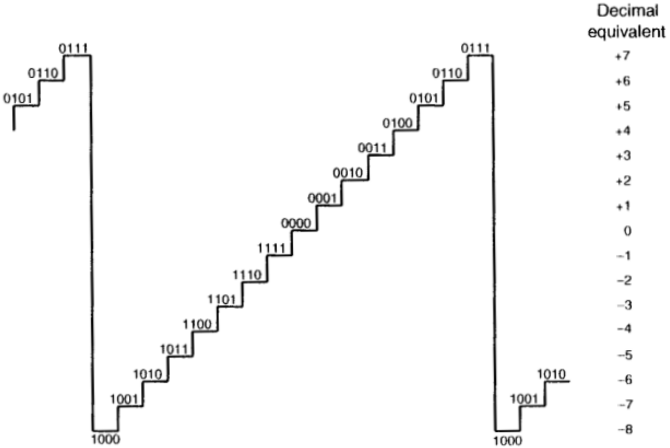
# Addition in 2's complement

- Adding two N-bits numbers can produce a N+1 bits result.
- The last two bits of the carry row show if overflow occurs.
- Saving the result in a N+1 word avoids overflows.
- The general rule is the sum of *m* individual *b*-bit can require as many as $b + log_2(m)$.
  Example: 256 8-bits words requires an accumulator whose word lenght is $8 + log_2(256) = 16$

```
 11111 111   (carry)        0111    (carry)
  0000 1111  (15)            0111   (7)
+ 1111 1011  (-5)          + 0011   (3)
==================         =============
  0000 1010  (10)            1010   (-6)  invalid!
```

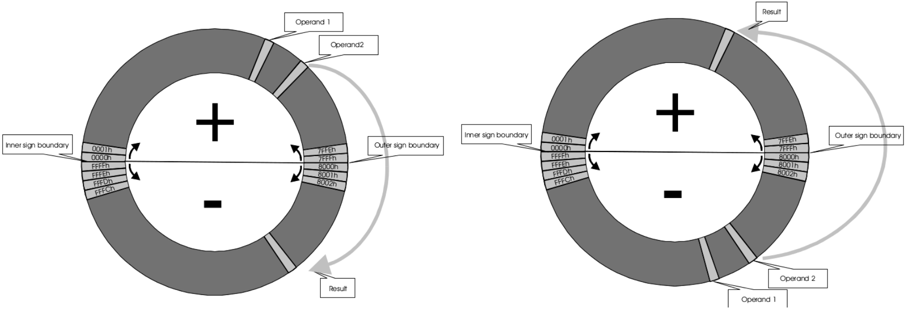
# Overflow

- An **overflow** occurs in an N-bit 2's complement notation when a result is greater than $2^{N-1} - 1$.
- An overflow produces a **roll-over** (wrap).
- An **underflow** occurs if a result is less than $2^{-N}$.



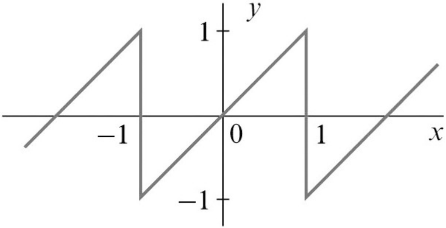
# Overflow, cont'd

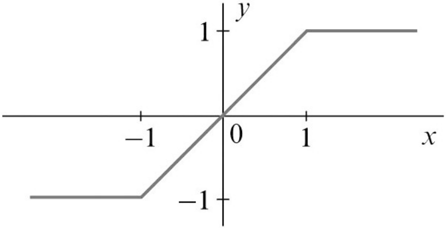
- A roll-over usually has catastrophic consequences on a process.
- Only happen when two very large positive operands, or two very large negative operands are added.
- It can never happen during the addition of a positive operand and a negative operand, whatever their magnitude.



# Saturation

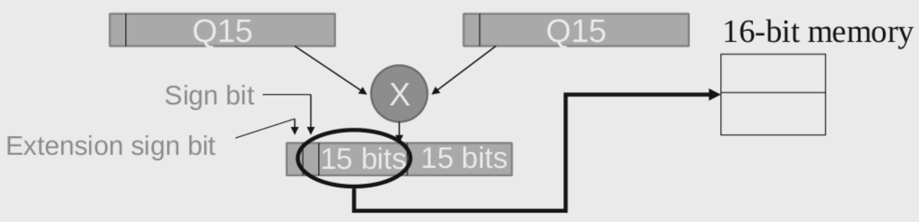
- To avoid a rollover, overflow is detected and the result is saturated to the most positive or most negative value that can be represented.
- This procedure is called **saturation arithmetic**.
- PDSP allows the results to be saturated automatically in hardware.



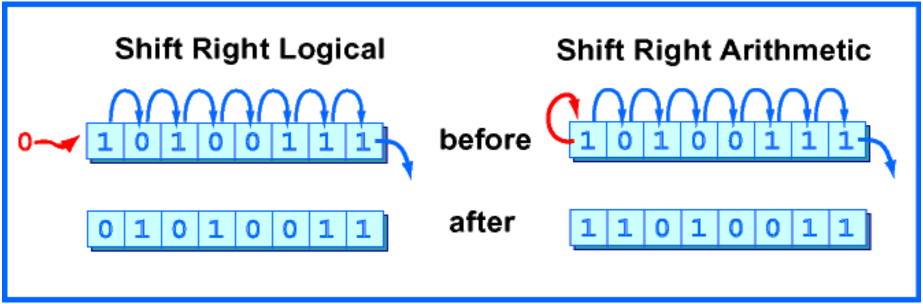
# Multiplication in 2's complement

- The product of two N-bit numbers requires 2N bits to contain all possible values.
- But the two MSB are always equal (sign extension bit).
- Therefore, 2N-1 bits are enough to store the result.
- Q15 will not produce an overflow.



# Multiplication and division by 2 in 2's complement

- Multiplication: all bits are shifted left by one position.
- Division: all bits are shifted right by one position, however the sign bit must be preserved (**arithmetic shift**).
- Arithmetic shift $\neq$ logical shift.



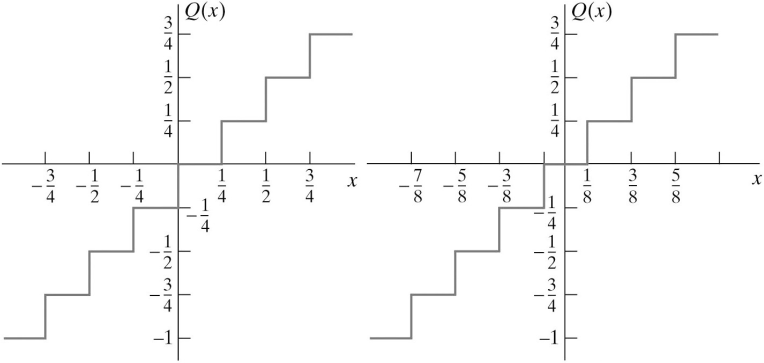
# Accumulator

- PDSP have an accumulator with extra bits to avoid overflow during internal calculations.
- Guard bits: extra bits to avoid addition overflows.
- Only round final results to the final data size and format if possible.

| b39–b32 | b31–b16 | b15–b0 |
|---|---|---|
| G | H | L |
| Guard bits | High-order bits | Low-order bits |

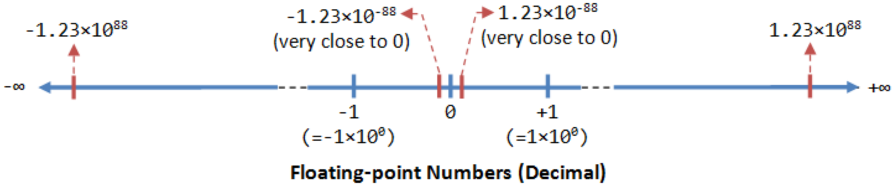
# Truncation and roundoff

- A 2N bits number after multiplication must be stored in memory as a N-bits word.
- Truncation: $e = Q[x] - x, \quad -\Delta \leq e < 0, \quad \mu = -\frac{\Delta}{2}, \quad \sigma^2 = \frac{\Delta}{12}.$
- Roundoff: $e = Q[x + 0.5] - x, \quad -\Delta/2 < e \leq \Delta/2, \quad \underline{\mu = 0}, \quad \sigma^2 = \frac{\Delta}{12}.$



# Number 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

$$F \times r^E,$$

where

- $F$, fraction.
- $E$, exponent.
- $r$, certain radix. Binary, $r = 2$; decimal, $r = 10$.

# Standards

**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 (c) → | | | | | | | | ← 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) → | | | | | | | |

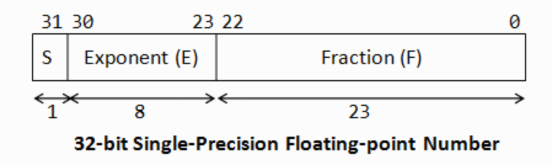
Modern computers adopt IEEE 754 standard for representing floating-point numbers.

# IEEE 754-2008 standard

IEEE 754-2008 standard defines several formats.

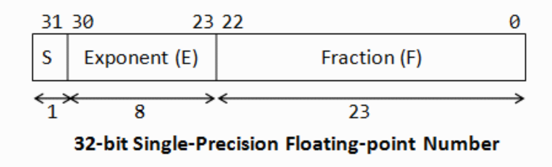
| | Binary formats ($B = 2$) | | | | Decimal formats ($B = 10$) | | |
|---|---|---|---|---|---|---|---|
| Parameter | Binary 16 | Binary 32 | Binary 64 | Binary 128 | Decimal 132 | Decimal 164 | Decimal 128 |
| $p$, digits | $10 + 1$ | $23 + 1$ | $52 + 1$ | $112 + 1$ | 7 | 16 | 34 |
| $e_{max}$ | +15 | +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 | | | |

# IEEE-754 32-bit Single-Precision




- $S$, sign bit. 0 for negative numbers and 1 for positive numbers.
- $E$, 8-bits exponent.
- 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.
- $F$, 23-bits fraction.

# Format




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

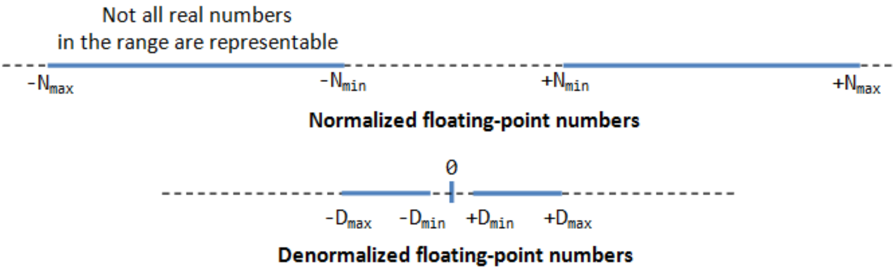
*Binary:* *Status:* normal

| 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!.

# Format




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

# Example

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

Decimal Value Entered: -3.4e-39

Single precision (32 bits):

*Binary:* *Status:* denormalized

| Bit 31<br>Sign Bit | Bits 30 - 23<br>Exponent Field | Bits 22 - 0<br>Significand |
|---|---|---|
| 1 | 00000000 | 0.10010100000010111010001 |
| 0: +<br>1: - | Decimal value of exponent field and exponent<br>0 - 127 = -127 | Decimal value of the significand<br>0.5784800 |

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

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

# Special values

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

## 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()).
- $s$, significant, $s = 1.F$.
- $s'$ and $s''$ being two successive multiples of ulp.
- Assume that $s' < s < s''$, $s'' = s' + ulp$,
- Then, the rounding function $round(s)$ associates to $s$ either $s'$ or $s''$, according to some rounding strategy.

Rounding schemes are:

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

# 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):

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

For fixed-point Q31 (32-bits):

$$DR_{dB} \approx 6.02 \cdot 32 \approx 192\,\text{dB}$$

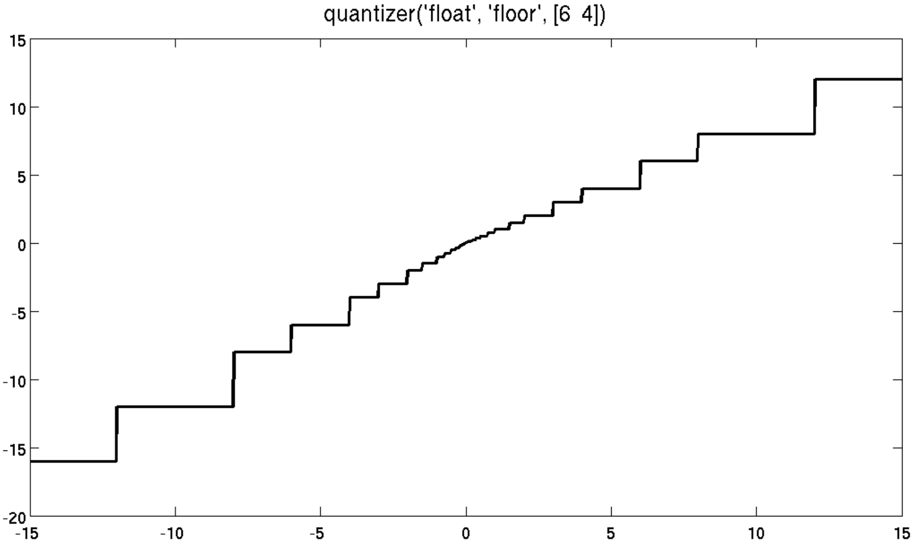
# Precision

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

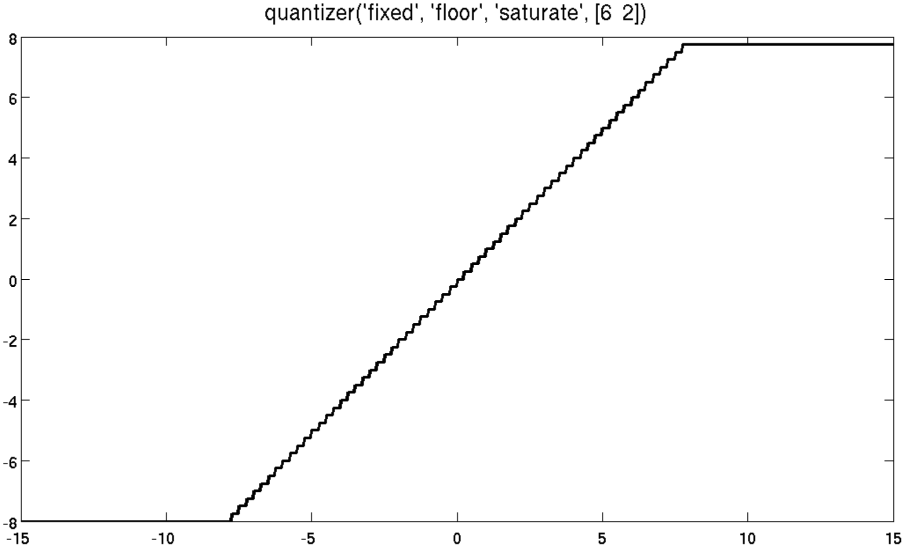
## MATLAB

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

# Precision, cont'd



# Precision, cont'd



# Precision, cont'd

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

## MATLAB

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

# Precision, cont'd

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

## MATLAB

1. » e1 = eps(single(1))
2. » e1 = 1.1920929e-07
3. » e2 = eps(single(1e1))
4. » e2 = 9.5367432e-07

# Precision, cont'd

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

## MATLAB

```
1 » e1 = eps(single(1))
2 » e1 = 1.1920929e-07
3 » e2 = eps(single(1e1))
4 » e2 = 9.5367432e-07
5 » e3 = eps(single(1e10))
6 » e3 = 1024
```

# Precision, cont'd

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 = 1000001024.00
```

# Precision, cont'd

- In floating-point processors scaling the data increases dynamic range, but scaling does not improve precision, and in fact degrades performance.
- 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`

# Precision, cont'd

- In floating-point processors scaling the data increases dynamic range, but scaling does not improve precision, and in fact degrades performance.
- 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`

# Precision, cont'd

- In floating-point processors scaling the data increases dynamic range, but scaling does not improve precision, and in fact degrades performance.
- 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, t = tan(x) - sin(x)/cos(x)

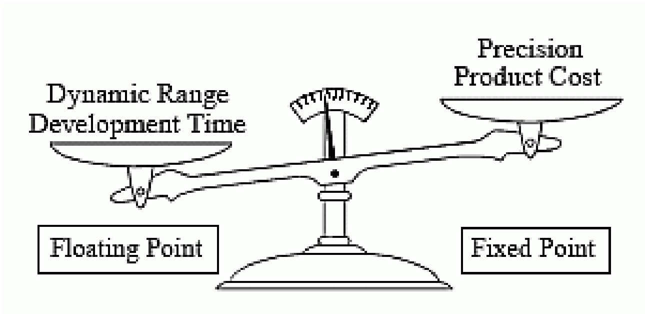
# Precision, cont'd

- In floating-point processors scaling the data increases dynamic range, but scaling does not improve precision, and in fact degrades performance.
- 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, t = tan(x) - sin(x)/cos(x)
4 » t = 2.2204e-16 % eps(1)
```

# Fixed-point vs floating-point



## Bibliography


- Bruno Paillard. An Introduction To Digital Signal Processors, Chapter 5 "Binary representations and fixed-point arithmetic".
- Richard G. Lyons. Understanding Digital Signal, Chapter 12 "Digital data formats and their effects".
- Jean-Pierre Deschamps, Gustavo D. Sutter, and Enrique Cantó. Guide to FPGA Implementation of Arithmetic Functions, Chapter 12 "Floating Point Arithmetic".
- Erick L. Oberstar. Fixed-Point Representation & Fractional Math.
- A Tutorial on Data Representation Integers, Floating-point Numbers, and Characters http://www3.ntu.edu.sg/home/ehchua/programming/java/DataRepresentation.html
- Greg Duckett. Fixed-Point vs. Floating-Point DSP for Superior Audio. http://web.archive.org/web/20060515074349/http://www.rane.com/note153.html