

Finite representation of real numbers

Dr. Ing. Rodrigo Gonzalez

`rodraz@frm.utn.edu.ar`

Técnicas Digitales III

Universidad Tecnológica Nacional,
Facultad Regional Mendoza.

Resumen

1 Coder

2 Integers

3 Fixed-point

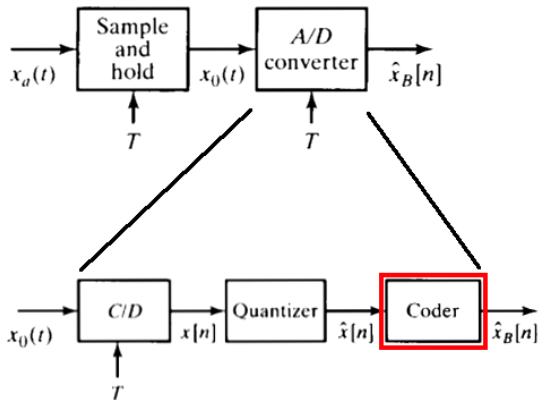
- Fractional point
- Scale factor
- Dynamic range
- Overflow and underflow
- Addition
- How to avoid overflow
- Multiplication
- Accumulator

4 Floating-point

- Number Representation
- Standards
- Normalized Form
- De-normalized Form
- Rounding schemes
- Dynamic range
- Precision
- Sum of two floating-point positive numbers

5 Fixed-point vs floating-point

ADC stages



Representation

Unsigned integers

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

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

Representation (2)

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

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

MATLAB

```
1 » a_m = 15; a_M = 15;
```

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

```
3 » N = 5.00
```

"Q" notation

To represent numbers with fractional parts, one simply has to assume that the [implicit] binary point at various bit positions along the data word allows the scale of representation to be binary point is located somewhere to the left of b_0 . Assume is a key word here.

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.
- This naming convention does not take the MSB of the number (sign bit) into account. A Qm.n notation therefore uses $m + n + 1$ bits.
- Precision: 2^{-n} .
- Range: -2^m to $2^m - 2^{-n}$.

"Q" notation, examples

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}).

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 of representation

- There is no difference at the CPU level between a fractional and an integer representation.
- The difference is based on the concept of *scale*, which is almost completely in the head of the designer.
- Values represented in Qm.n notation can be seen as a signed integer simply divided by 2^n , the inverse of precision.
- In fact, 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 $5/32768$ ($5/2^{15}$).

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)	$2^0 = 1(\text{Integer})$	0x7FFF (32767) → 0x8000h (−32768)

Scale factor, four-bit signed integer multiplication

Four-Bit Integer Multiplication

$$\begin{array}{r}
 0100 \quad 4 \\
 \times 1101 \quad \times -3 \\
 \hline
 0000100 \\
 0000000 \\
 000100 \\
 11100 \\
 \hline
 11110100 \quad -12
 \end{array}$$

Accumulator 11110100

Data Memory ?

Is there another (superior) numbering system? 11-7

Scale factor, Q0.3 multiplication

Four-Bit Multiplication

	0.100	1/2
	x 1.101	x - 3/8
	00000100	
	0000000	
	000100	
	11100	
	11110100	-3/16
Accumulator	1.1110100	
Data Memory	1.110	-1/4

Dynamic range

Dynamic range is defined as,

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

For N-bit signed integers,

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

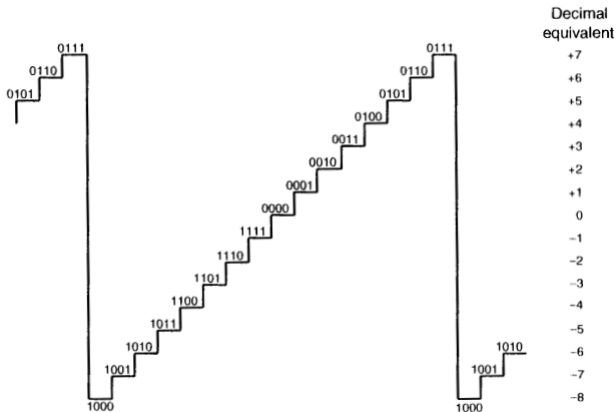
$$DR_{dB} \approx 20 [(N-1) \log_{10}(2)]$$

$$DR_{dB} \approx 20 \log_{10}(2) \cdot (N-1)$$

$$DR_{dB} \approx 6.02 \cdot (N-1) \quad [\text{dB}]$$

Overflow and underflow

- 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} .



Addition in 2's complement

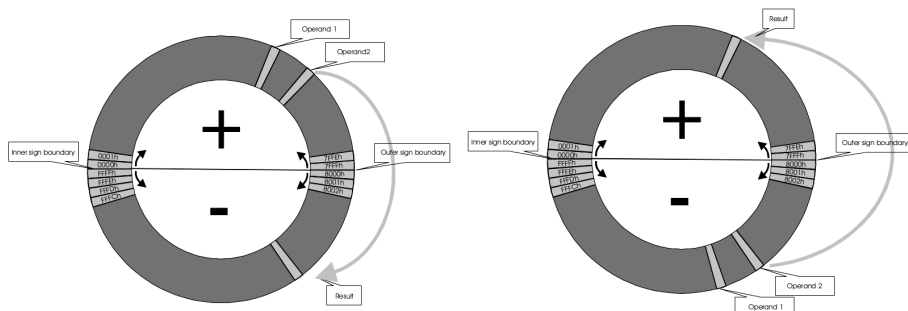
- Adding two N-bits numbers can produce a N+1 bits result.
- The last two bits of the carry row show if overflow occurs.

$$\begin{array}{r}
 \boxed{11}111\ 111\ (\text{carry}) \\
 0000\ 1111\ (15) \\
 +\ 1111\ 1011\ (-5) \\
 \hline
 0000\ 1010\ (10)
 \end{array}$$

$$\begin{array}{r}
 \boxed{01}11\ (\text{carry}) \\
 0111\ (7) \\
 +\ 0011\ (3) \\
 \hline
 1010\ (-6)\ \underline{\text{invalid!}}
 \end{array}$$

Overflow

- 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.

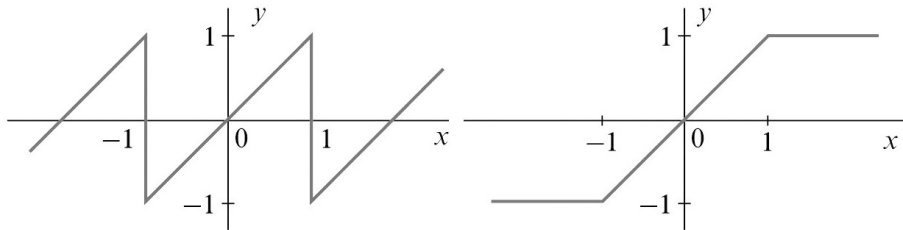


Longer word-length accumulator

- 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 length is $8 + \log_2(256) = 16$.
- DSP processors usually have 40-bit accumulators.
- ¿How many sums are supported by a 40-bits accumulator for 16-bits numbers?

Saturation

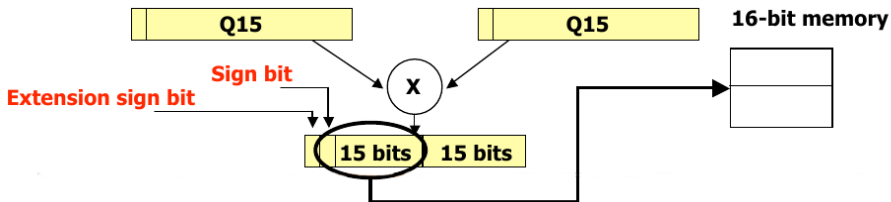
- To avoid a rollover, overflow is detected and the result is saturated to the most positive or most negative value that can be represented.
- This procedure is called **saturation arithmetic**.
- DSP processors allows the results to be saturated automatically in hardware (In TI DSP C5505, SATD Bit of ST1_55 register).



Be aware of non-linearity!

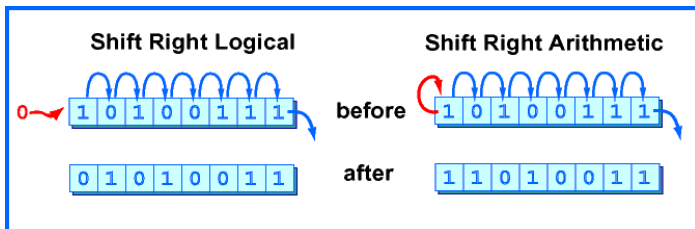
Multiplication in 2's complement

- 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.
- A Q15 multiplication produces Q1.30 result.
- To transform the result into Q31 notation, it must be left-shifted by one bit
- DSP processors have a special mode that allows its ALU to automatically perform the left shift when Q15xQ15.



Logical and Arithmetic shifts

- 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.



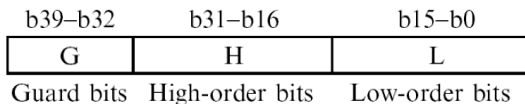
Logical and Arithmetic shifts (2)

In DSP processors:

- ALU can perform logical shifts of 32-bit operands in one cycle, from 16 bits to the right, to 15 bits to the left.
- Sign extension is performed during shifts to the right, if the Sign Extension Mode control bit (in C5505, SXM) is set.
- Result is saturated during shifts to the left if an overflow is detected, and Overflow bit (in C5505, OVM) is set.

Accumulator

- DSP processors have an accumulator with extra bits to avoid overflow during internal calculations (in C5505, 40-bits accumulator).
- Guard bits: extra bits to avoid addition overflows.
- After MAC operations, only final result is adjusted to memory data size.



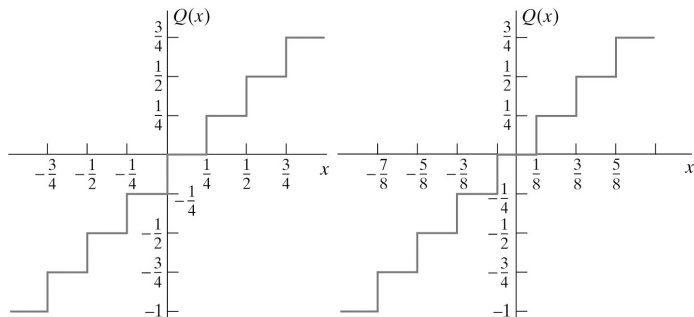
Rounding schemes, truncation and roundoff

- After multiplication, a $2N$ -bits number must be stored in memory of N -bits word.

- Truncation: $e = Q[x] - x$, $-\Delta \leq e < 0$, $\mu = -\frac{\Delta}{2}$, $\sigma^2 = \frac{\Delta}{12}$.

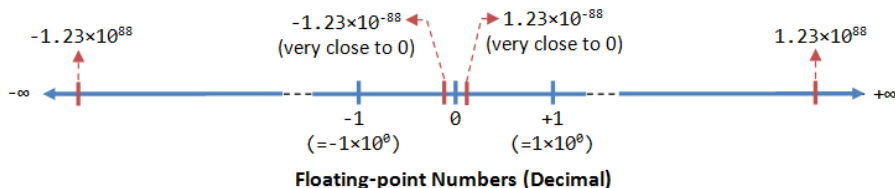
- Roundoff: $e = Q[x + 0.5] - x$, $-\Delta/2 < e \leq \Delta/2$, $\mu = 0$, $\sigma^2 = \frac{\Delta}{12}$.

- DSP processors manage truncation and roundoff automatically.



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

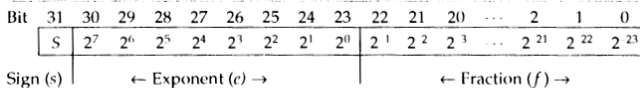
$$(-1)^S \times F \times r^E,$$

where,

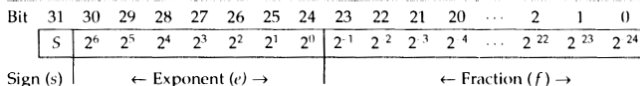
- S , sign bit.
- F , fraction.
- E , exponent.
- r , certain radix. $r = 2$ for binary; $r = 10$ for decimal.

Standards

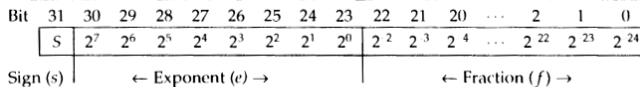
IEEE Standard P754 Format



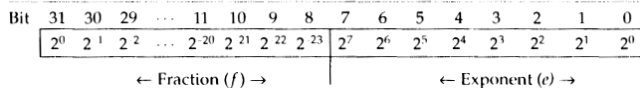
IBM Format



DEC (Digital Equipment Corp.) Format



MIL-STD 1750A Format



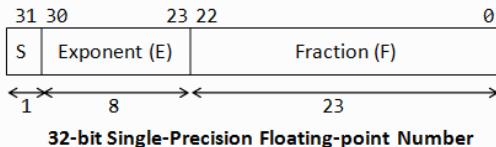
Modern computers adopt IEEE 754-2008 standard for representing floating-point numbers.

IEEE 754-2008 standard

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			

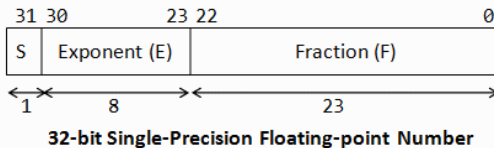
IEEE-754 32-bit Single-Precision



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

- S , sign bit. 0 for positive numbers and 1 for negative numbers.
- 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.

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:

Single precision (32 bits):

Binary: Status:

<p>Bit 31 Sign Bit</p> <p><input type="text" value="0"/></p> <p>0: + 1: -</p>	<p>Bits 30 - 23 Exponent Field</p> <p><input type="text" value="100 0101 0"/></p> <p>Decimal value of exponent field and exponent</p> <p><input type="text" value="138"/> - 127 = <input type="text" value="11"/></p>	<p>Bits 22 - 0 Significand</p> <p><input type="text" value="1.100 1000 1111 0000 0101 0010"/></p> <p>Decimal value of the significand</p> <p><input type="text" value="1.5698340"/></p>
---	---	---

Hexadecimal: Decimal:

<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:

Single precision (32 bits):

Binary: Status:

Bit 31 Sign Bit	Bits 30 - 23 Exponent Field	Bits 22 - 0 Significand
<input type="text" value="0"/>	<input type="text" value="10001011"/>	<input type="text" value="1.10010001111000001010010"/>
0: + 1: -	Decimal value of exponent field and exponent <input type="text" value="139"/> - 127 = <input type="text" value="12"/>	Decimal value of the significand <input type="text" value="1.5698340"/>

Hexadecimal: Decimal:

Example 3

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

Decimal Value Entered:

Single precision (32 bits):

Binary: Status:

Bit 31 Sign Bit	Bits 30 - 23 Exponent Field	Bits 22 - 0 Significand
<input type="text" value="0"/>	<input type="text" value="10001000"/>	<input type="text" value="1.10010001111000001010010"/>
0: + 1: -	Decimal value of exponent field and exponent <input type="text" value="136"/> - 127 = <input type="text" value="9"/>	Decimal value of the significand <input type="text" value="1.5698340"/>

Hexadecimal: Decimal:

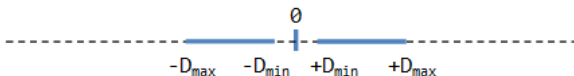
Floating-point numbers are auto-scaled!.

Format

Not all real numbers
in the range are representable



Normalized floating-point numbers



Denormalized floating-point numbers

- 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 small numbers.
- $E = 0 \Rightarrow 0.F$, implicit leading 0.

Example

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

Decimal Value Entered:

Single precision (32 bits):

Binary: Status:

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

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

Rounding schemes are:

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

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):

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

For fixed-point Q31 (32-bits):

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

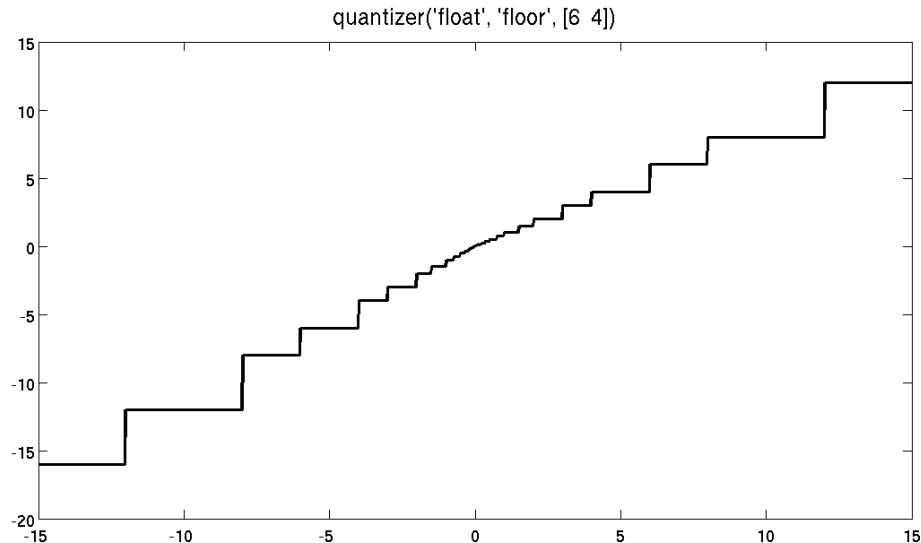
Precision

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

MATLAB

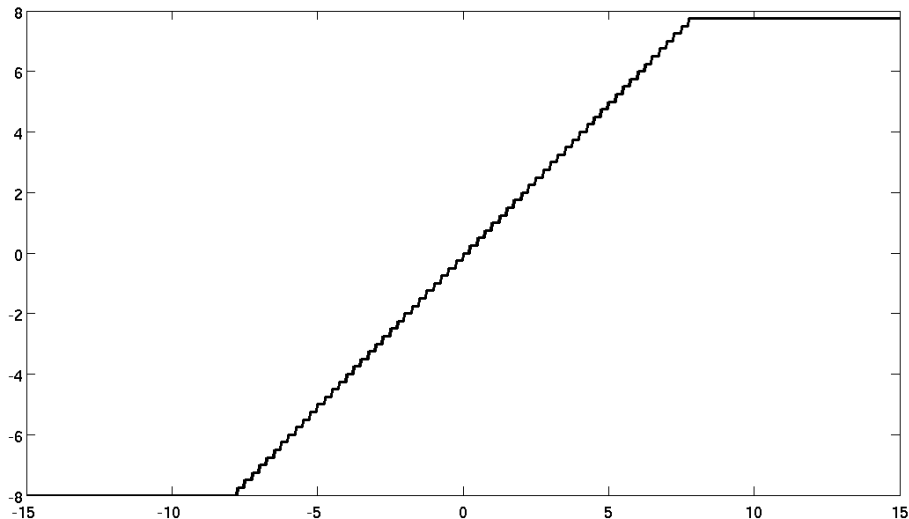
```
❶ » u = linspace(-15,15,1000);  
❷ » q = quantizer([6 4],'float'); % [wordlength exponentlength]  
❸ » y1 = quantize(q,u);  
❹ » plot(u,y1); title(tostring(q))  
❺ »  
❻ » q = quantizer('fixed',[6 2]); % [wordlength fractionlength]  
❼ » y2 = quantize(q,u);  
❽ » plot(u,y2); title(tostring(q))
```

Precision (2)



Precision (3)

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



Precision (4)

`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  
7 » t = single(1e10) + single(1300)  
8 » t = 10000001024.00
```


Sum of two floating-point positive numbers

$$n = n_1 + n_2 = 1.F \times r^{(E-bias)},$$

$$n_1 = 1.F_1 \times r^{(E_1-bias)},$$

$$n_2 = 1.F_2 \times r^{(E_2-bias)}.$$

- if $E_1 \geq E_2$ then,

$$E = E_1, \quad F = F_1 + (F_2 \gg (E_1 - E_2))$$

- else,

$$E = E_2, \quad F = (F_1 \gg (E_2 - E_1)) + F_2$$

- if $F \geq r$ then, (first normalization)

$$E = E + 1, \quad F = F \gg 1$$

- $F = \text{round}(F)$

- if $F \geq r$ then, (second normalization)

$$E = E + 1, \quad F = F \gg 1$$

Example 1

$$n = 1e10 + 1300,$$

$$1e10 = (-1)^0 \times 1.00101010000001011111001 \times r^{(160-127)},$$

$$1300 = (-1)^0 \times 1.111000000000000000000000 \times r^{(131-127)}.$$

- if $160 \geq 131$ then,

$$E = 160,$$

$$F = 1.00101010000001011111001 + (1.111000000000000000000000 \gg 29)$$

$$E = 160, \quad F = 1.00101010000001011111001 + (0.000000000000000000000000)$$

$$E = 160, \quad F = 1.00101010000001011111001$$

$$n = (-1)^0 \times 1.00101010000001011111001 \times r^{(160-127)},$$

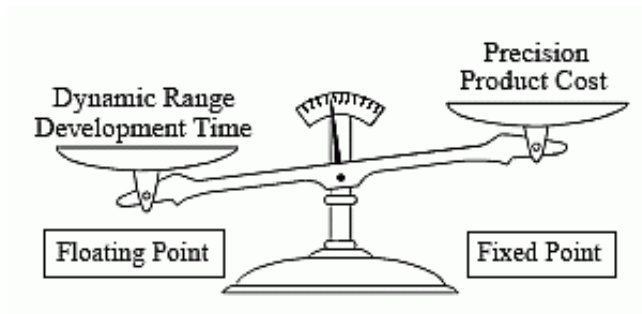
Example 2

- 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

- ❶ Bruno Paillard. An Introduction To Digital Signal Processors, Chapter 5 "Binary representations and fixed-point arithmetic".
- ❷ Richard G. Lyons. Understanding Digital Signal, Chapter 12 "Digital data formats and their effects".
- ❸ Texas Instruments. C28x IQ – Math Library. [Link to document](#).
- ❹ Jean-Pierre Deschamps, Gustavo D. Sutter, and Enrique Cantó. Guide to FPGA Implementation of Arithmetic Functions, Chapter 12 "Floating Point Arithmetic".