Computer Arithmetic

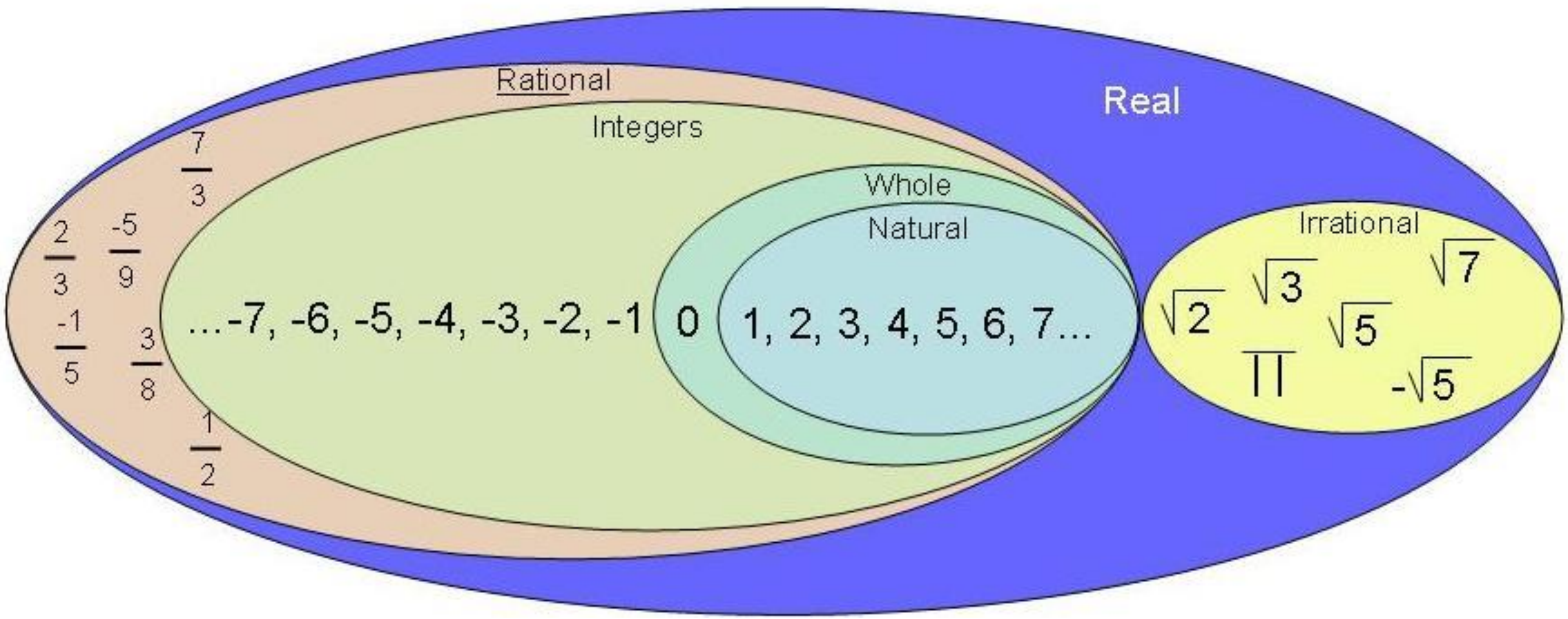
Binary…Integers…Real-Numbers

COMPUTER ARITHMETIC

Binary … Integers … Real - Numbers

- For simplicity we would like to use only binary numbers and …. positive integers.
- But in real life >> Real-Numbers are required.
- Which numbers are “Real-Numbers”?
  - Any type number we can think of is a Real-Number

Binary … Integers … Real - Numbers



Fixed and Floating Point representations

- Because we need to …

  1. Expand the number range
  2. Include smaller numbers than 1
  3. Use Real Numbers (integers, positive and negative numbers that can be written in decimal notation)

- **Fixed-Point** and **Floating-Point** Number representations are used

Fixed-point: binary point fixed

Fixed-Point Numbers

32-bit Fixed-Point Number	
0000 0000 000 0000	0000 0000 000 0000
16-bit Integer	16-bit Fraction

Where are used?

- Fixed-Point arithmetic is used in applications where speed is more important than precision:
  - Digital Signal/Image Processing
  - Control Systems
  - smartDevices
  - Games

Floating-Point and Fixed-Point

- To represent Real Numbers (Numbers with Fractional Part) we can use:
  - Fixed-Point
  - Floating-Point

- Note that, the logic circuits of Fixed-Point hardware are much less complicated than those of Floating-Point hardware

- Fixed-point calculations require less memory and less processor time

Fixed-Point

- Computer: Works with Integers
- Designer (Programmer): Can assign, in the following 8-bit integer, a fraction part.
- Example: $00001000_2 = 8_{10}$
- Insert decimal point: $0000.\overbrace{1000}^{4}{}_2$ =
- Scaled 8 by 2^{-4} = $1/2^4$ = 1/16 = 0.0625
- Therefore: 8 * 0.0625 = 0.5_{10}

Therefore …

- $0000.1000_2 = 0.5_{10}$
- $0.1_2 = 0.5_{10}$

Fixed-Point Numbers

- Fixed-Point Math is integer math that allows fractions
- Fixed-point binary numbers can represent real-numbers (**with fractional or binary point component**)
- They have **fixed width (for example: 8-bit number)**

8-bit number

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

- Divide your range of values to two parts (Integer and fractional)

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}
8	4	2	1	•	1/2	1/4	1/8	1/16

Whole Part

Fractional Part

Fixed-Point

Q<X.Y> format for Fixed-Point numbers

- Q<X.Y>

where,

  - X = whole number + fractional part
  - Y = fractional part

Representation format: Example: Q<8.4>

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}
8	4	2	1	●	1/2	1/4	1/8	1/16

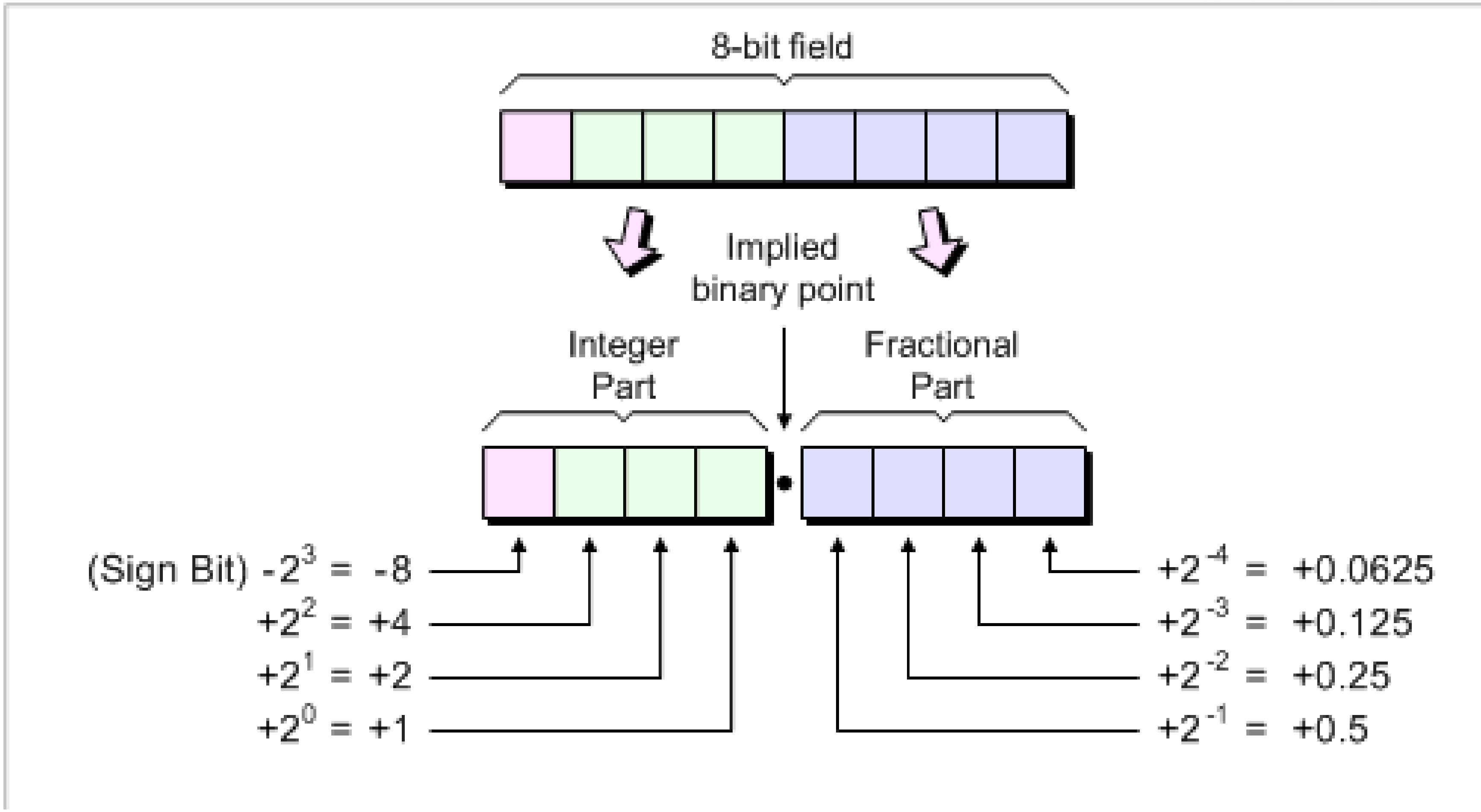
- If X = whole number + fractional part
- If Y= fractional part
  - Q<X.Y> = Q<8.4> fixed-point
  - If Y = 0, then X.Y = X.0, with no fraction part

Precision (Example: Q<8.4>)

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}
8	4	2	1	●	1/2	1/4	1/8	1/16

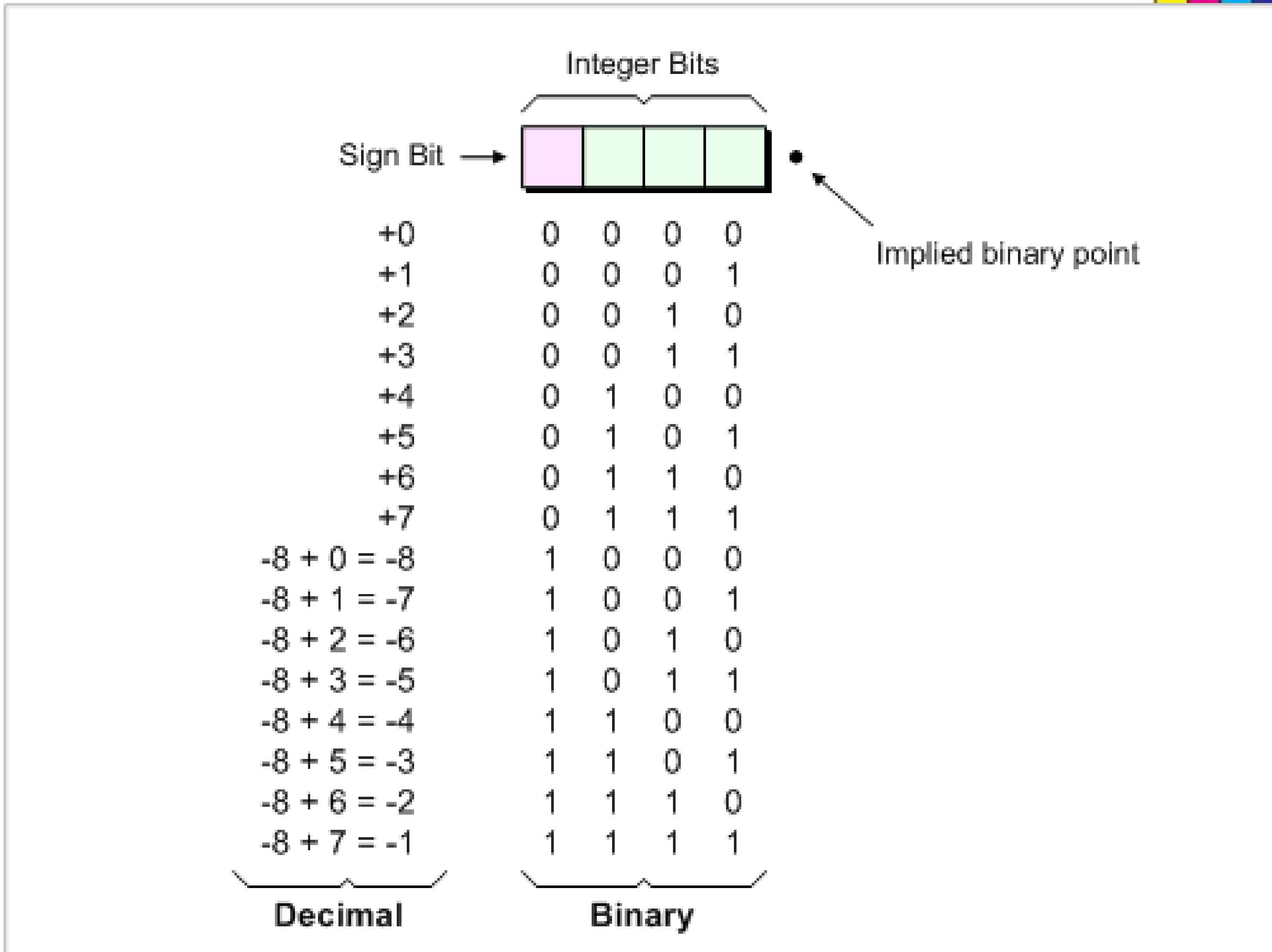
- Precision is the smallest difference between two successive binary numbers, i.e., (1/8-1/16)
- Precision for our Q<8.4> example: (1/8-1/16) = 1/16 = **0.625**

<8.4> Fixed-point (signed) representation



http://www.diycalculator.com/sp-round.shtml

Integer part



Fractional part

Fractional Part

Implied binary point

0.5 0.25 0.125 0.0625

0	0	0	0	=	0.0	+	0.00	+	0.000	+	0.0000	=	+0.0000
0	0	0	1	=	0.0	+	0.00	+	0.000	+	0.0625	=	+0.0625
0	0	1	0	=	0.0	+	0.00	+	0.125	+	0.0000	=	+0.1250
0	0	1	1	=	0.0	+	0.00	+	0.125	+	0.0625	=	+0.1875
0	1	0	0	=	0.0	+	0.25	+	0.000	+	0.0000	=	+0.2500
0	1	0	1	=	0.0	+	0.25	+	0.000	+	0.0625	=	+0.3125
0	1	1	0	=	0.0	+	0.25	+	0.125	+	0.0000	=	+0.3750
0	1	1	1	=	0.0	+	0.25	+	0.125	+	0.0625	=	+0.4375
1	0	0	0	=	0.5	+	0.00	+	0.000	+	0.0000	=	+0.5000
1	0	0	1	=	0.5	+	0.00	+	0.000	+	0.0625	=	+0.5625
1	0	1	0	=	0.5	+	0.00	+	0.125	+	0.0000	=	+0.6250
1	0	1	1	=	0.5	+	0.00	+	0.125	+	0.0625	=	+0.6875
1	1	0	0	=	0.5	+	0.25	+	0.000	+	0.0000	=	+0.7500
1	1	0	1	=	0.5	+	0.25	+	0.000	+	0.0625	=	+0.8125
1	1	1	0	=	0.5	+	0.25	+	0.125	+	0.0000	=	+0.8750
1	1	1	1	=	0.5	+	0.25	+	0.125	+	0.0625	=	+0.9375

Binary **Decimal**

Programmer choice

- It is noted again that the programmer has a clear choice about the range of the numbers involved in the design (application).
- Binary data, for the ALU, are just … a bit pattern
  - $00110111_2 = 55_{10}$
  - $0011011.1_2 = 27.5_{10}$
  - $001101.11_2 = 13.75_{10}$
  - $00110.111_2 = 6.875_{10}$

Example: Fixed-Point Q<X.Y> = Q<8.1>

- Example: Q<X.Y> = Q<8.1>
- { --------.- }
- The decimal number 0.5 can be represented in binary notation with Q<8.1>, as …

2^6	2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}
0	0	0	0	0	0	0	•	1

New Example … 0.75 decimal

- Express decimal 0.75 in a Q<8.1> fixed-point binary format.
- The decimal number 0.75 cannot be represented in a Q<8.1> fixed-point binary format
- Increase the fraction range by one Q<8.2>
- Therefore we decrease the whole number range (**Precision loss**)

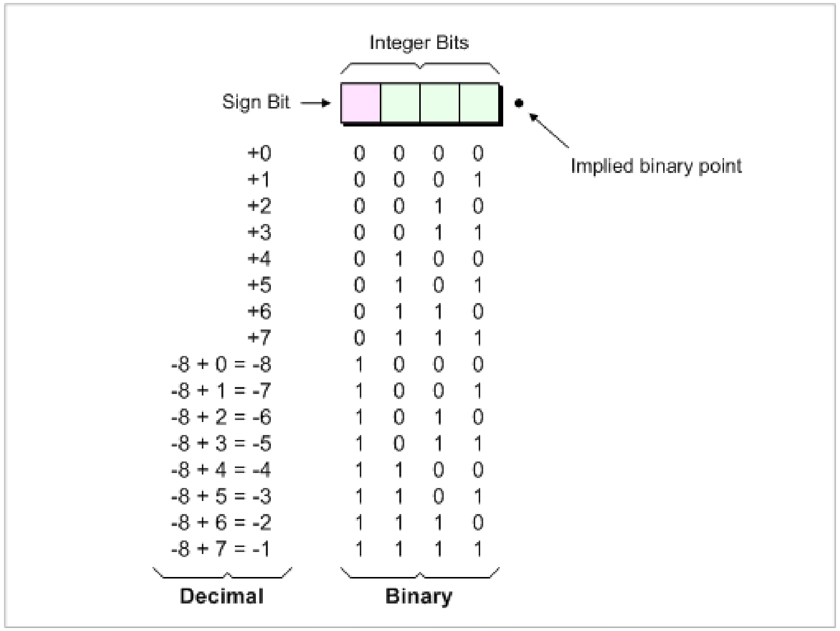
Example: Decimal 0.75 in <8.2> format

2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}
0	0	0	0	0	0	•	1	1

- We lost range on the integer part
- Main disadvantage of Fixed Point numbers

Negative fixed-point numbers

- Use 2’s complement signed-magnitude representation
- Assign one bit of each number for sign. This will decrease the range of numbers by 2
- Why?



Dividing by 2 in binary …

- Dividing a number by 2 is equivalent to shifting the binary number to the right by 1-bit position

Examples:

- 1110_2 (14) → 0111_2 (7)
- 1100_2 (12) → 0110_2 (6)
- 1101_2 (13) → 110.1_2 (6.5)
- 1111_2 (15) → 111.1_2 (7.5)

Why?

- 1101_2 (13) → 110.1_2 (6.5)

Convert decimal (with fraction) to binary:

6.5_{10} in binary?

$6_{10} = 110_2$

0.5_{10} = in binary**?**

With fractional part (0.5)

Multiply by 2 …

$0.5 * 2 = 1.0$

$0 \quad * 2 = 0.0$

Top to bottom

$0.5_{10} = .10_2$

Zeros in the fractions have no real meaning … 2.30000000 = 2.3

Floating-Point Numbers

32-bit Floating-Point Number		
1	0000 0000	0000 0000 0000 0000 0000 000
1-bit	8-bit exponent	23-bit Fraction

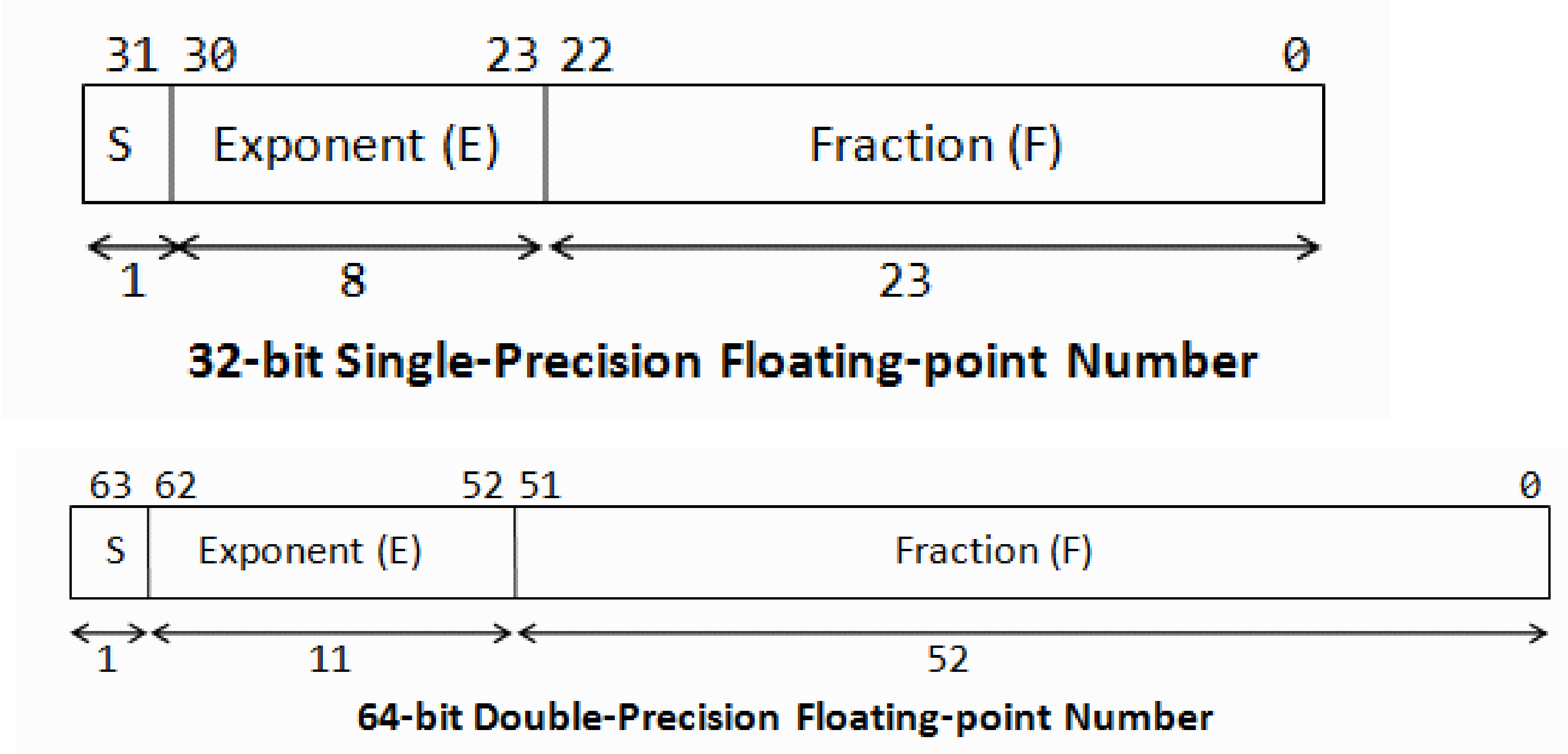
Floating-Point Numbers … range

- Is used to represent numbers:
  - Very large
  - Very small

Decimal Range of the **IEEE 64-bit** Format

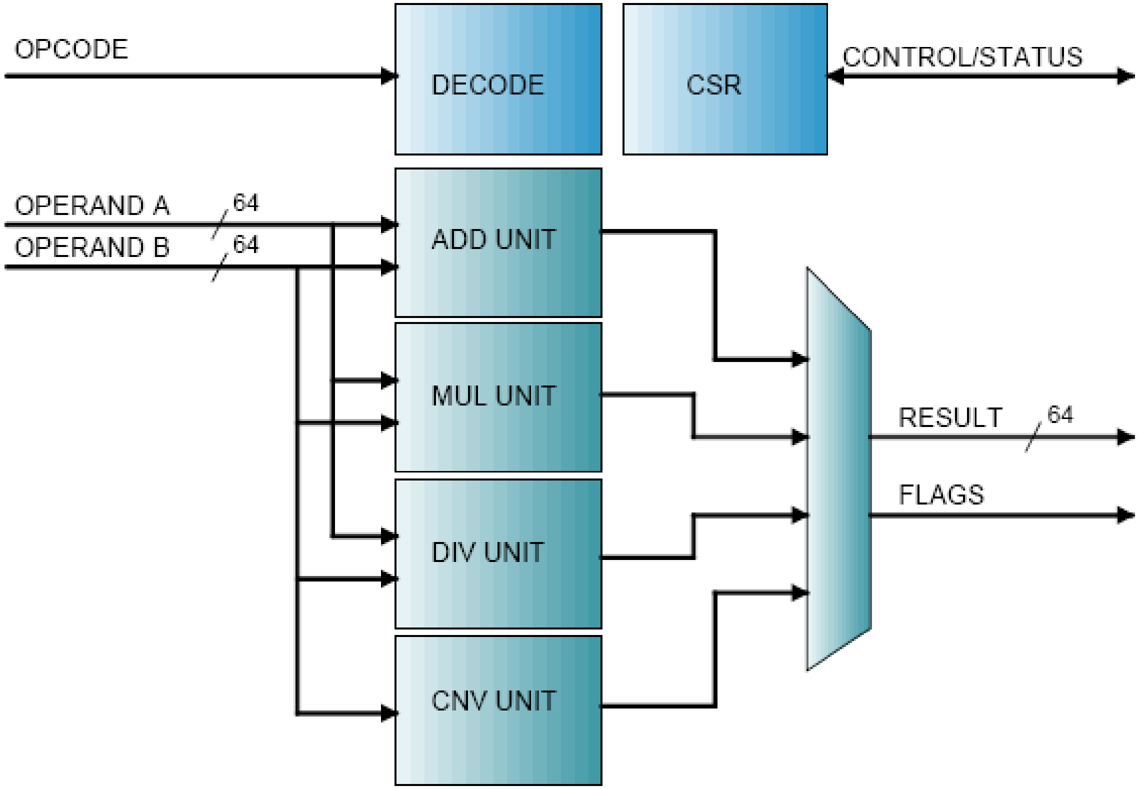
4.9×10^{-307} to $1.8 \times 10^{+308}$

Single/Double-Precision



IEEE-754 FPU

IEEE-754 Floating Point Unit
Single/Double Precision, Full Compliance



Fixed Point and Floating Point Numbers

- A Fixed-Point Number has a fixed number of digits after the binary point
  - Division by zero gives an interrupt or exception
- A Floating-Point Number can have a varying number of digits after the binary point
  - Division by zero gives infinite result

END