

Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C

Chapter 13 Fixed-point Numbers

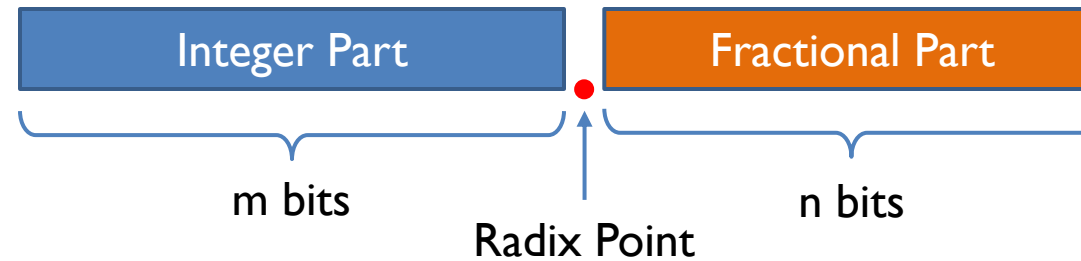
Z. Gu

Fall 2025

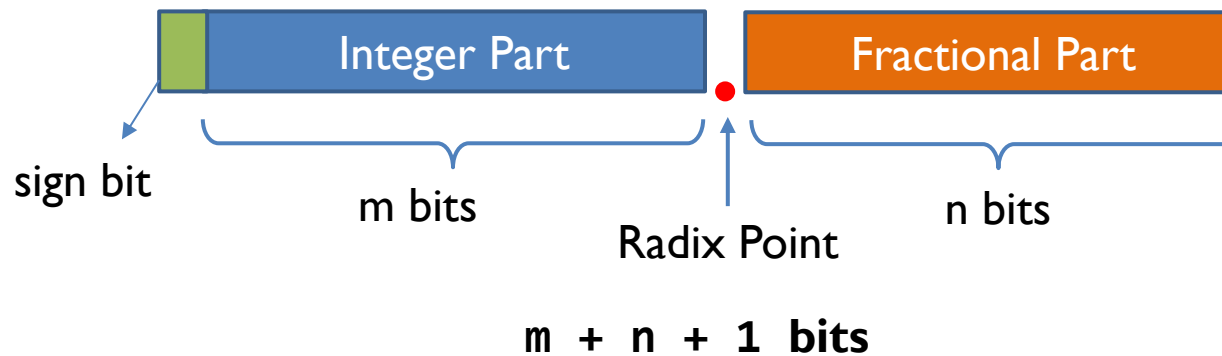
Acknowledgement: Lecture slides based on Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, University of Maine <https://web.eece.maine.edu/~zhu/book/>

Fixed-point Format: Q Notation

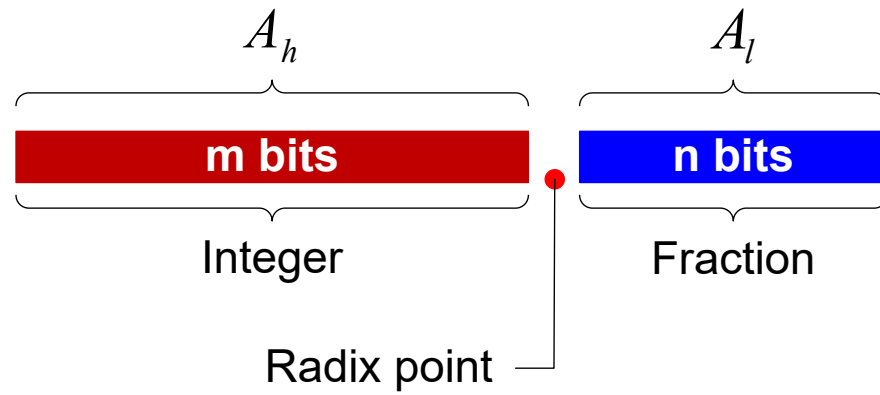
UQm.n for unsigned fixed-point



Qm.n for signed fixed-point

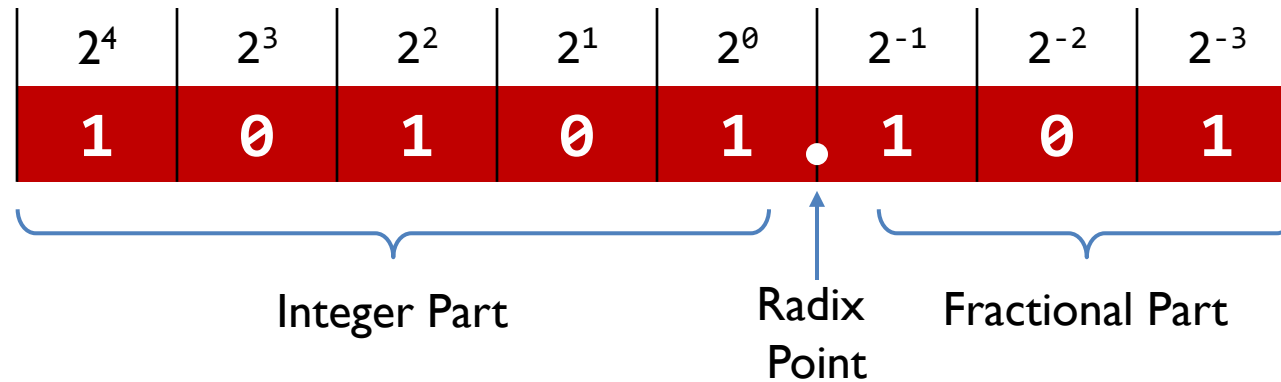


Q Notation: **UQm.n**



$$f = A_h + A_l \times 2^{-n}$$

Converting Fixed-point UQ5.3 to Float



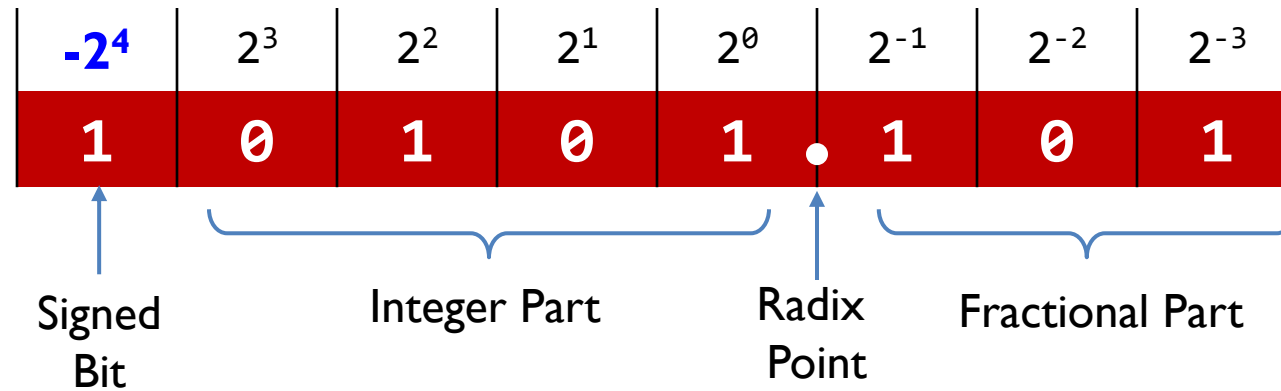
$$10101.101_2 = A_h + A_l \times 2^{-3}$$

$$= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$= 21 + 5 \times 2^{-3} = 21.625$$

$$10101.101_2 = \frac{10101101_2}{2^3} = \frac{173}{8} = 21.625$$

Converting Fixed-point Q4.3 to Float



$$10101.101_2 = A_h + A_l \times 2^{-3}$$

$$= 1 \times (-2^4) + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$

$$= (-16 + 4 + 1) + 0.625 = -10.375$$

$$10101.101_2 = \frac{10101101_2}{2^3} = \frac{-83}{8} = -10.375$$

Another way of computing the int part:
MSB = 1 → number is **negative**

Find magnitude:

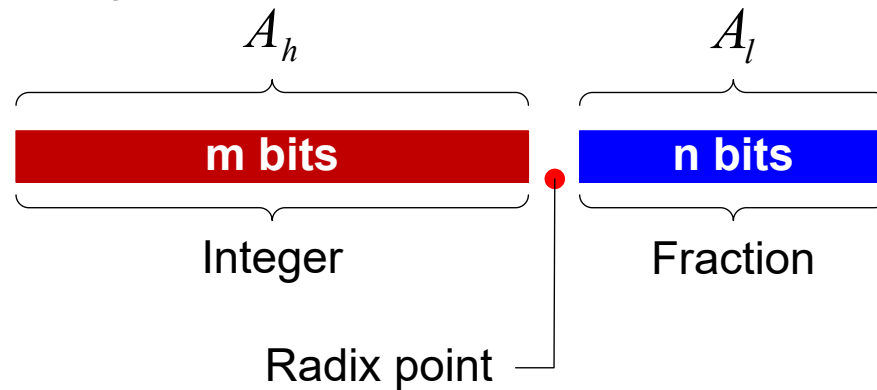
Invert: 10101 → 01010

Add 1: 01010 + 1 = 01011 →
decimal 11

So the value is -11

U5.3 vs. Q4.3

Unsigned Fixed-point Representation **Um.n**



$$f = A_h + A_l \times 2^{-n}$$

$$\begin{aligned} 10101.101_2 &= A_h + A_l \times 2^{-3} \\ &= 21 + 5 \times 2^{-3} \\ &= 21.625 \end{aligned}$$

Signed Fixed-point Representation **Qm.n**



$$f = A_h + A_l \times 2^{-n}$$

$$\begin{aligned} 10101.101_2 &= A_h + A_l \times 2^{-3} \\ &= -11 + 5 \times 2^{-3} \\ &= -10.375 \end{aligned}$$

Two Ways of Calculating Two's Complement (integer)

- ▶ Convert 10101 into decimal with Two's Complement notation
- ▶ Method 1, invert bits and add 1:
 - ▶ $01010 + 1 = 01011 = 11$ in decimal, hence $10101 = -11$ in decimal
- ▶ Method 2, calculate directly:
 - ▶ $1 \times (-2^4) + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = -16 + 4 + 1 = -11$ in decimal
- ▶ The two definitions are mathematically identical (proof omitted)

Convert Float to Fixed-point UQ4.12

UQm.n for unsigned fixed-point

$$\textit{Representation} = \textit{round}(\textit{float} \times 2^n)$$

Example: Convert $f = 3.141593$ to UQ4.12 (4 bits integer, 12 bits fraction)

- ▶ Calculate $f \times 2^{12} = 12867.964928$
- ▶ Round the result to nearest integer, $\textit{round}(12867.964928) = 12868$
- ▶ Convert the integer to binary: $12868 = 0011_0010_0100_0100_2$
- ▶ Organize into UQ4.12: $0011.0010_0100_0100_2$
- ▶ Final result in hex: **0x3244**
- ▶ Approximation Error = reconstructed_value - true_value = $\frac{12868}{2^{12}} - f = 8.5625 \times 10^{-6}$ (due to limited number of fractional bits (12))

Convert Float to Fixed-point Q3.12

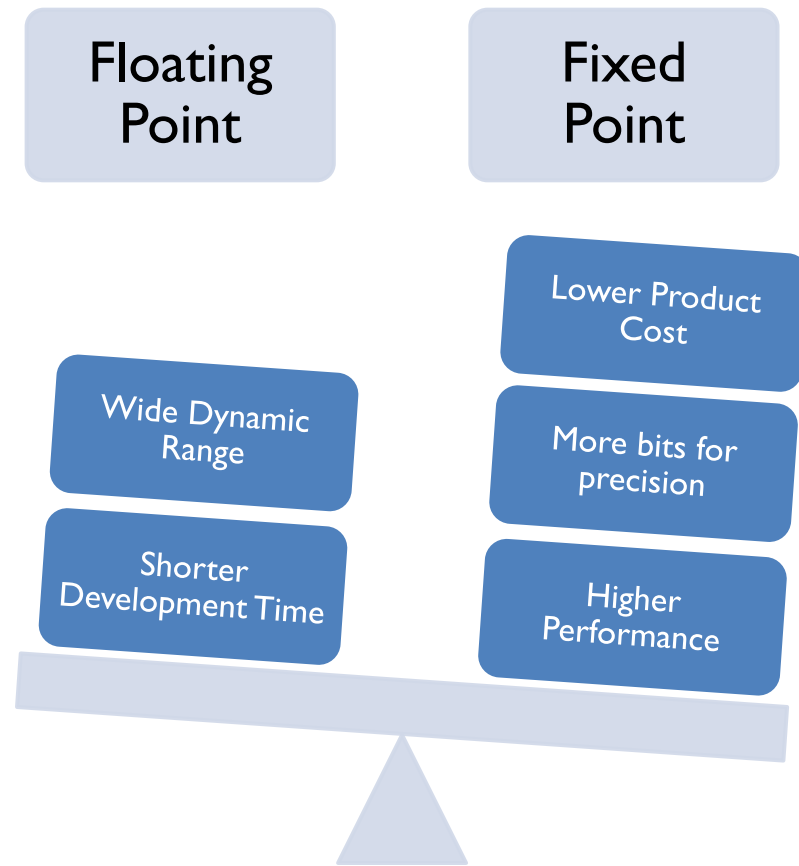
Qm.n for signed fixed-point

$$\text{Representation} = \text{round}(\text{float} \times 2^n)$$

Example: Convert $f = -3.141593$ to Q3.12 (1 sign bit, 3 bits integer, 12 bits fraction)

- ▶ Calculate $f \times 2^{12} = -12867.964928$
- ▶ Round the result to an integer, $\text{round}(-12867.964928) = -12868$
- ▶ Find the 16-bit two's complement: **1100_1101_1011_1100**₂
 - ▶ $12868 = 0x3244 = 0011\ 0010\ 0100\ 0100_2$
 - ▶ Invert bits and add 1 $\rightarrow -12868 = 1100\ 1101\ 1011\ 1100_2$
- ▶ Organize into Q3.12: **1100.1101_1011_1100**₂
- ▶ Final result in hex: **0xCDBC**
- ▶ Approximation Error = reconstructed_value - true_value = $-\frac{12868}{2^{12}} - f = 8.5625 \times 10^{-6}$

Why use fixed-point?



Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C

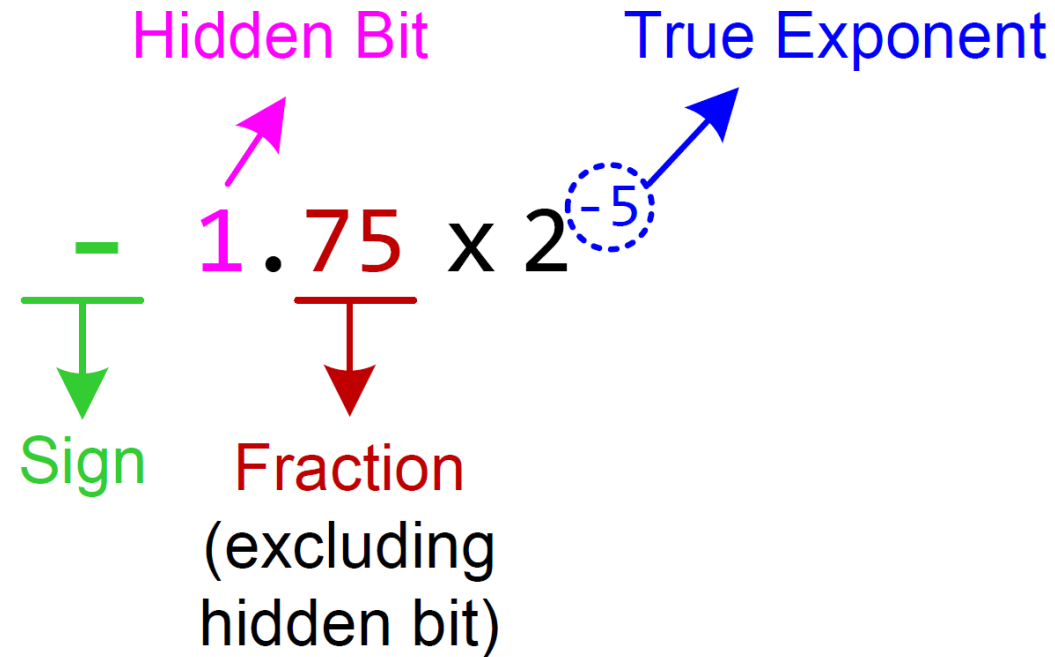
Chapter 13 Floating-point Numbers

Z. Gu

Fall 2025

Acknowledgement: Lecture slides based on Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, University of Maine <https://web.eece.maine.edu/~zhu/book/>

Normalization

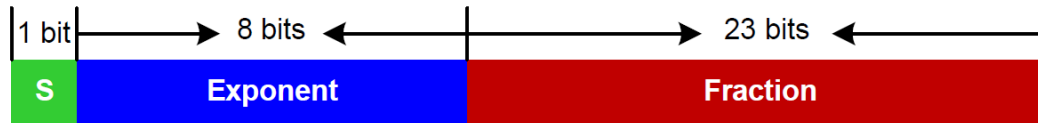


We can convert 10.746×2^6 to normalized format as follows:

$$10.746 \times 2^6 = \frac{10.746}{8} \times 8 \times 2^6 = 1.34325 \times 2^9$$

IEEE Standard 754 (Signed Floating Point)

Single Precision (32 bits)



Double Precision (64 bits)



IEEE 754 value:

$$(-1)^S \times (1 + \textit{Fraction}) \times 2^{\textit{Exponent} - \textit{Bias}}$$

where Bias = $2^7 - 1 = 127$ for single precision FP32

Bias = $2^{10} - 1 = 1023$ for double precision FP64

Hidden bit refer to the implicit leading 1 in (1 + fraction)

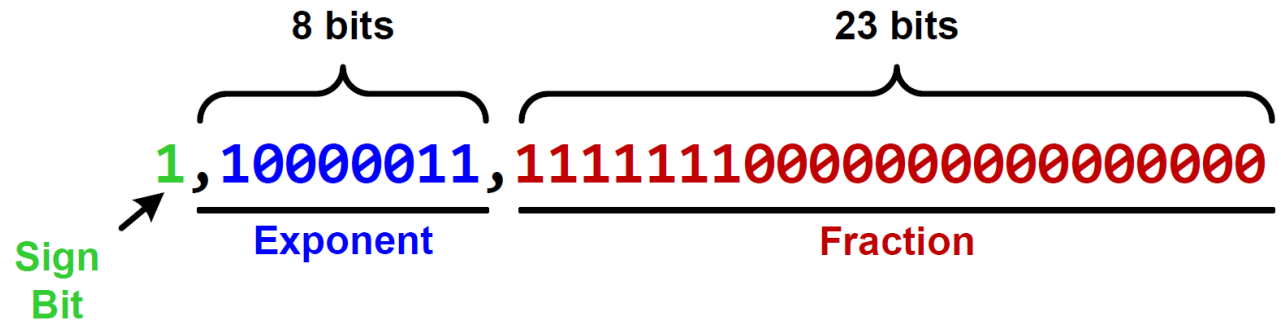
Fraction is also called significand or **mantissa**

Why Bias?

- ▶ FP32's exponent field has 8 bits. So it can represent integers from 0 to 255. But real exponents in normalized floating-point numbers range roughly from -126 to $+127$. To store both negative and positive exponents in that 0–255 range, IEEE-754 adds a bias, shifting all values upward so they become non-negative.
- ▶ For an exponent field of n bits, the bias is $2^{(n-1)} - 1$.
- ▶ For FP32, the bias is chosen as $2^{(8-1)} - 1 = 127$, so half the range is allocated for negative exponents, and half for positive.
 - ▶ Actual Exponent = Stored Exponent - 127
 - ▶ Stored exponent 0 → reserved for zero and subnormal numbers
 - ▶ For subnormal numbers: effective exponent = -126 , and there is no hidden leading 1
 - ▶ Stored exponent 127 → actual exponent = 0
 - ▶ Stored exponent 255 → reserved for special values (like infinity and NaN)

Decoding 0xC1FF0000 into a floating-point number

- ▶ Binary 11000001111111100000000000000000
- ▶ Sign = 1
- ▶ Exponent = $10000011_2 = 131$
- ▶ Fraction = $0.1111111_2 = 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-6} + 1 \times 2^{-7}$
 $= 0.9921875$
- ▶ $f = (-1)^S \times (1 + \text{Fraction}) \times 2^{\text{Exponent} - 127}$
 $= (-1)^1 \times (1 + 0.9921875) \times 2^{131 - 127}$
 $= -1 \times 1.9921875 \times 2^4$
 $= -31.875$
- ▶ If Exponent = $10000101_2 = 133$, then $f = -1 \times 1.9921875 \times 2^6 = -127.5$
 - ▶ Exponent - bias = $133 - 127 = 6$
- ▶ If Exponent = $10000110_2 = 134$, then $f = -1 \times 1.9921875 \times 2^7 = -255$
 - ▶ Exponent - bias = $134 - 127 = 7$



Decoding 0x40920000 into a floating-point number

- ▶ Binary 01000000100100000000000000000000
- ▶ Sign = 0
- ▶ Exponent = $10000001_2 = 129$
- ▶ Fraction = $0.001_2 = 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 0.125$
- ▶ $f = (-1)^S \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127}$
$$= (-1)^0 \times (1 + 0.125) \times 2^{129-127}$$
$$= 1 \times 1.125 \times 2^2$$
$$= 4.5$$
- ▶ How Floating Point Numbers Work (in 7 minutes!)
 - ▶ https://www.youtube.com/watch?v=W_Knvo9NuJY

Decoding 0x3F800000 into a floating-point number

- ▶ Binary 00111111000000000000000000000000
- ▶ Sign = 0
- ▶ Exponent = $01111111_2 = 127$
- ▶ Fraction = 0
- ▶
$$\begin{aligned} f &= (-1)^S \times (1 + \text{Fraction}) \times 2^{\text{Exponent} - 127} \\ &= (-1)^0 \times (1 + 0) \times 2^{127 - 127} \\ &= -1 \times 1 \times 2^0 \\ &= 1.0 \end{aligned}$$
- ▶ How Floating Point Numbers Work (in 7 minutes!)
 - ▶ https://www.youtube.com/watch?v=W_Knvo9NuJY

Decoding 0x41680000 into a floating-point number

- ▶ Binary 01000001011010000000000000000000
- ▶ Sign = 0
- ▶ Exponent = $10000010_2 = 130$
- ▶ Fraction = $0.1101_2 = 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-4} = 0.8125$
- ▶ $f = (-1)^S \times (1 + \text{Fraction}) \times 2^{\text{Exponent} - 127}$
$$\begin{aligned} &= (-1)^0 \times (1 + 0.8125) \times 2^{130 - 127} \\ &= 1 \times 1.8125 \times 2^3 \\ &= 14.5 \end{aligned}$$

Encoding 14.5 into IEEE Std 754 Single-Precision

- ▶ Normalization:
- ▶ $2^3 < 14.5 < 2^4, \frac{14.5}{2^3} = 1.8125$
- ▶ Hence $14.5 = 1.8125 \times 2^3 = (1 + 0.8125) \times 2^3$
- ▶ Conversion:
- ▶ $Sign = 0$
- ▶ $Exponent = 3 + 127 = 130 = 10000110_2$
- ▶ $Fraction = 0.1101_2$ (multiply by 2 repeatedly)
- ▶ Assume $Fraction = b1 \times 2^{-1} + b2 \times 2^{-2} + b3 \times 2^{-3} + b4 \times 2^{-4} + \dots$
 - ▶ $0.8125 \times 2 = 1.625 = 1 + 0.625 \Rightarrow b1 = 1$
 - ▶ $0.625 \times 2 = 1.25 = 1 + 0.25 \Rightarrow b2 = 1$
 - ▶ $0.25 \times 2 = 0.5 = 0 + 0.5 \Rightarrow b3 = 0$
 - ▶ $0.5 \times 2 = 1 \Rightarrow b4 = 1$
- ▶ $14.5 = 01000001011010000000000000000000$ in binary or 0x41680000 in hex

Encoding 1.3 into IEEE Std 754 Single-Precision

- ▶ Normalization:
- ▶ $2^0 < 1.3 < 2^1, \frac{1.3}{2^0} = 1.3$
- ▶ Hence $1.3 = (1 + 0.3) \times 2^0$
- ▶ Conversion:
- ▶ $Sign = 0$
- ▶ $Exponent = 0 + 127 = 127 = 01111111_2$
- ▶ $Fraction = 0.01001100110011 \dots_2$ (multiply by 2 repeatedly)
- ▶ Assume $Fraction = b1 \times 2^{-1} + b2 \times 2^{-2} + b3 \times 2^{-3} + b4 \times 2^{-4} + \dots$
 - ▶ $0.3 \times 2 = 0.6 \Rightarrow b1 = 0$
 - ▶ $0.6 \times 2 = 1.2 \Rightarrow b2 = 1$
 - ▶ $0.2 \times 2 = 0.4 \Rightarrow b3 = 0$
 - ▶ $0.4 \times 2 = 0.8 \Rightarrow b4 = 0$
 - ▶ $0.8 \times 2 = 1.6 \Rightarrow b5 = 1$
 - ▶ $0.6 \times 2 = 1.2 \Rightarrow b6 = 1$

} Repeats infinitely
- ▶ $1.3 = 00111111101001100110011001100110$ in binary or 0x3FA66666 in hex

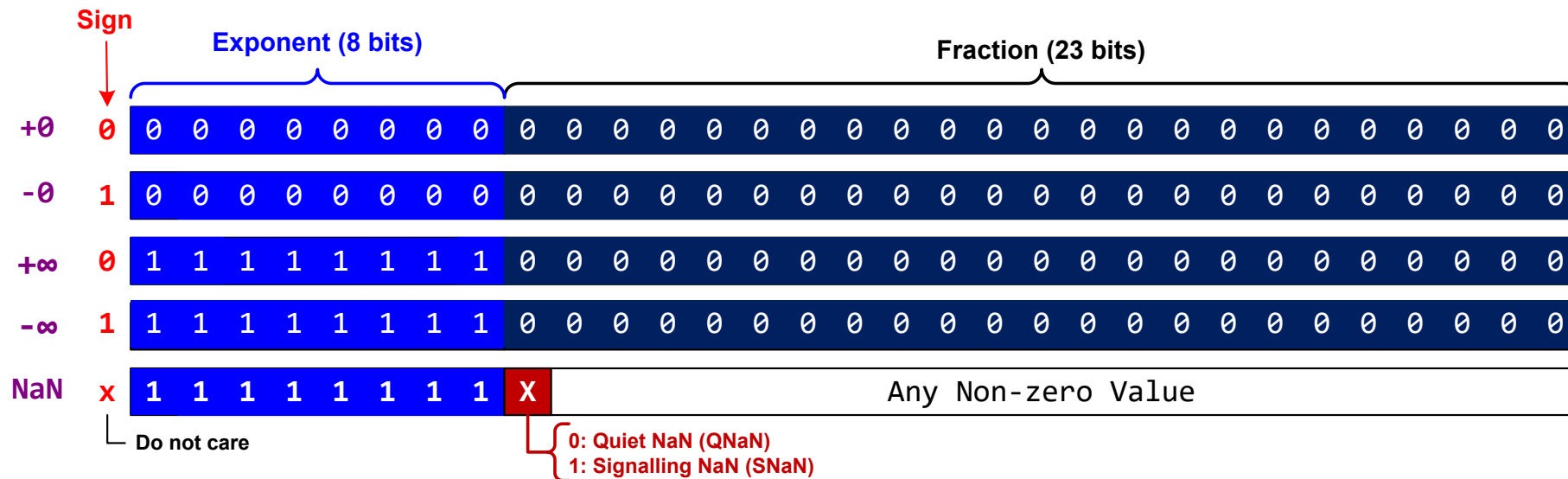
Decoding 0x3FA66666 into a floating-point number

- ▶ Binary 0011111101001100110011001100110
- ▶ Sign = 0
- ▶ Exponent = $0111111_2 = 127$
- ▶ Fraction = 0.29999995 (calculation process skipped)
- ▶ $f = (-1)^S \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127}$
$$= (-1)^0 \times (1 + 0.29999995) \times 2^{127-127}$$
$$= 1.29999995$$
- ▶ Error: $1.3 - 1.29999995 = 5 \times 10^{-8}$

Why Is This Happening?! Floating Point Approximation4
<https://www.youtube.com/watch?v=2glxbTn7GSc>

Special Values

- ▶ Exponents 00000000 and 11111111 are reserved.



- ▶ Example of Not-A-Number (NaN)
 - ▶ $\log(-10.0)$, $\text{sqrt}(-1.0)$, $0.0/0.0$, $-\infty + \infty$,

Subnormalized Float Number

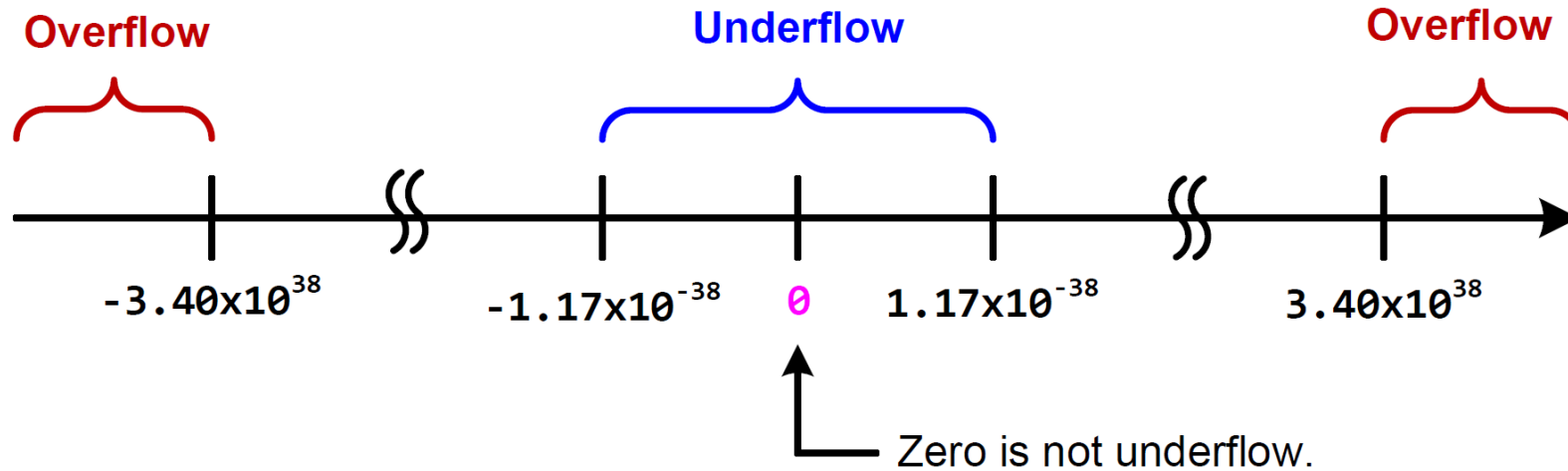
- ▶ To represent numbers between 0 and the minimum positive number that the normalized format can represent.
- ▶ **Normalized Format**

$$(-1)^S \times (\mathbf{1} + Fraction) \times 2^{\mathbf{Exponent}-127}$$

- ▶ **Sub-normalized** Format

$$(-1)^S \times Fraction \times 2^{-126}$$

Overflow and Underflow



Smallest Positive Normal Number:

0 00000001 000000000000000000000000

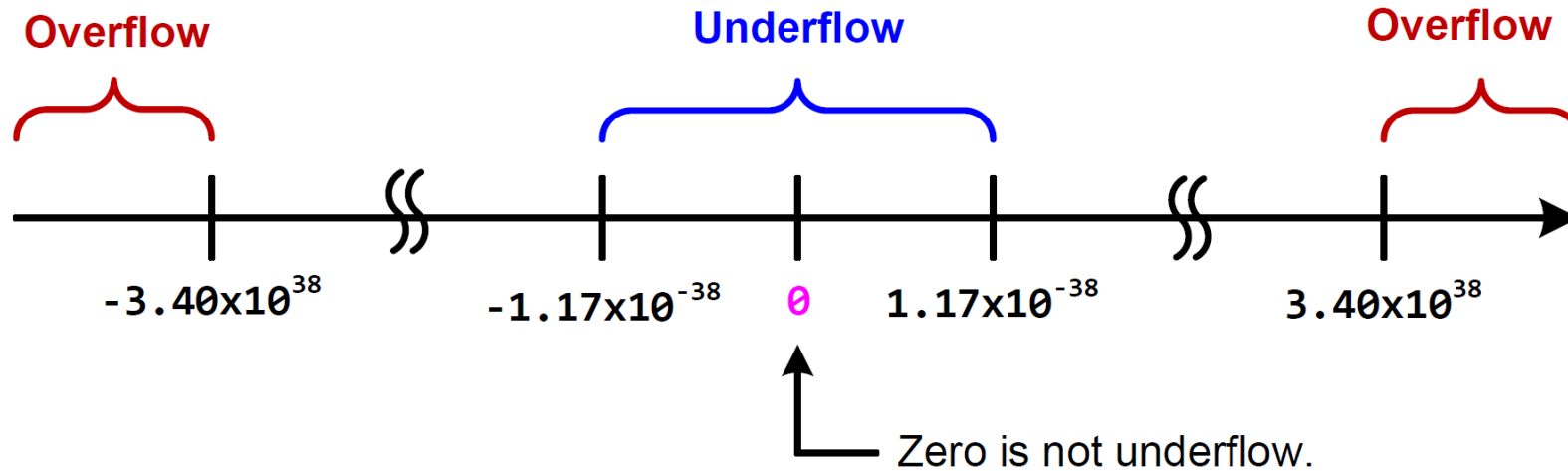
$$(-1)^0 \times (1 + 0) \times 2^{1-127} = 2^{-126} \approx 1.18 \times 10^{-38}$$

Smallest Positive Subnormal Number:

0 00000000 000000000000000000000001

$$(-1)^0 \times (0 + 2^{-23}) \times 2^{1-127} = 2^{-149} \approx 1.40 \times 10^{-45}$$

Overflow and Underflow



To find the largest representable number:

Exponent = largest possible **finite** value = 254 (since 255 is reserved for infinity and NaN)

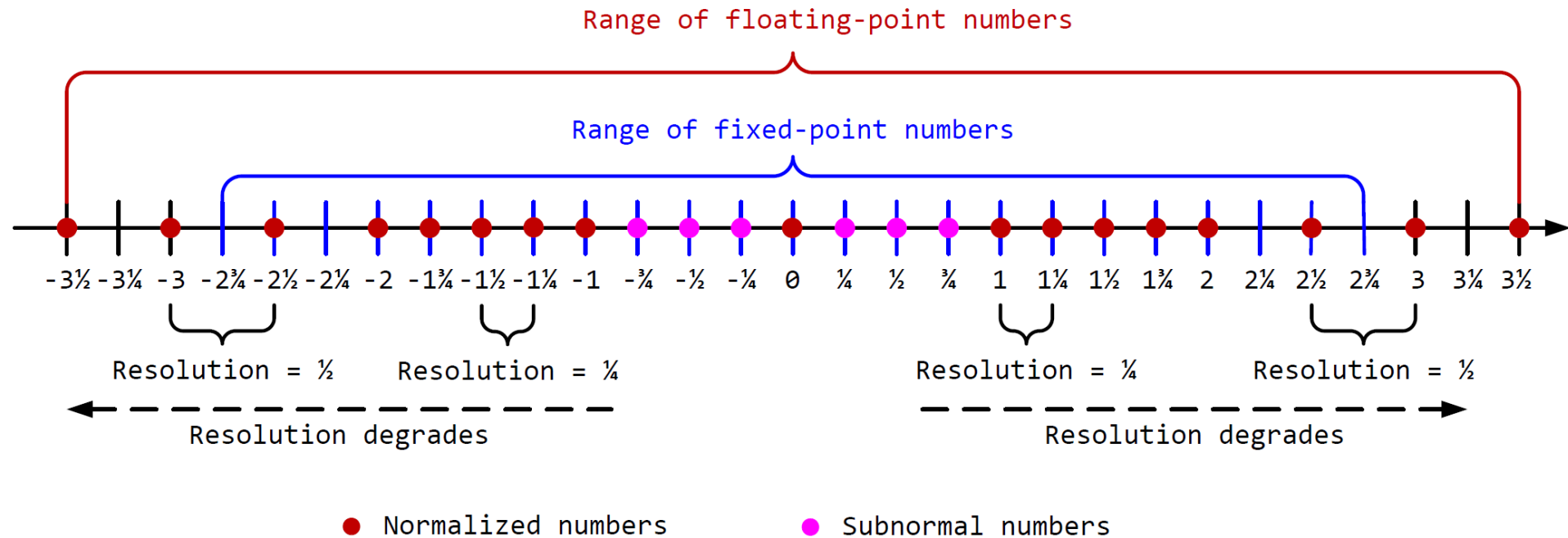
Mantissa = all 1s $\rightarrow 1.11111111111111111111111_2 = 1 + (1 - 2^{-23})$

Numbers farthest from zero:

$$(-1)^S \times (1 + (1 - 2^{-23})) \times 2^{254-127} = \pm(2^{128} - 2^{104}) \approx \pm 3.40 \times 10^{38}$$

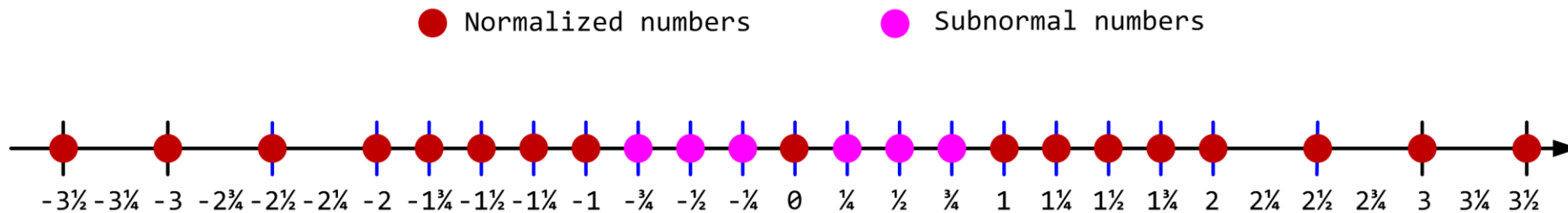
Resolution

- ▶ Given a hypothetical five-bit floating-point system (similar to IEEE 754).
 - ▶ the sign bit, an exponent (2 bits), and a fraction (2 bits)



Tradeoff between Range and Precision

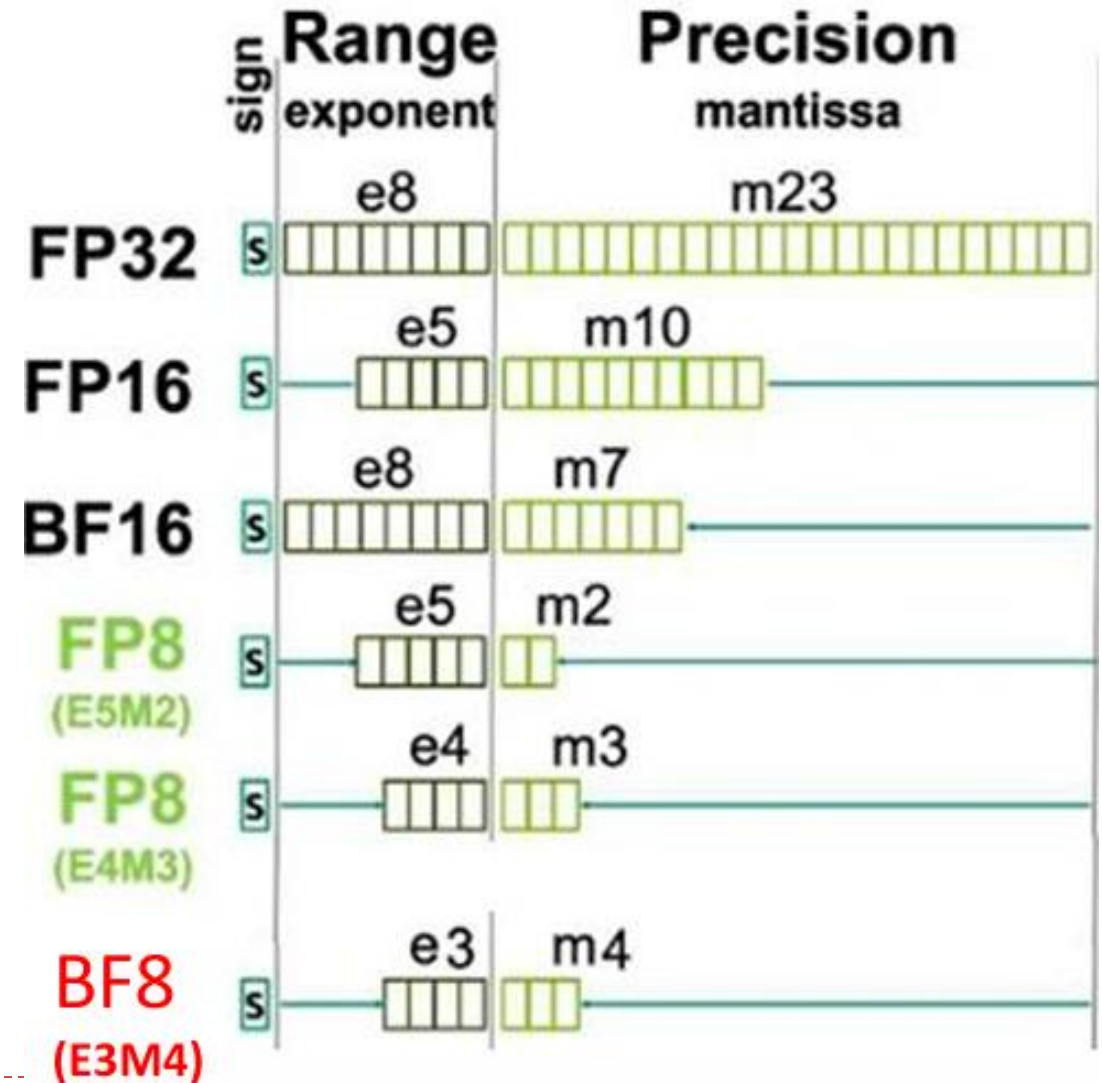
A simplified 5-bit floating-point (IEEE 754 style)



- ▶ Floating-Point
 - ▶ Resolution: difference between two neighbor numbers
 - ▶ Precision decreases as the magnitude increases
- ▶ Fixed-Point
 - ▶ Numbers are evenly distributed among the representable range
 - ▶ Precision is fixed

FP formats used in AI and machine learning

- ▶ bf16 (bfloat16) and bf8 (bfloat8) are floating point formats used in AI and machine learning for efficient computation with lower precision while retaining useful range.
- ▶ bf8 is cutting edge and experimental for very efficient AI deployment where accuracy can be slightly sacrificed for speed and lower memory footprints. bf16 is established as a good practical balance for many ML tasks



References

- ▶ Recommended: How Floating-Point Numbers Are Represented
 - ▶ <https://www.youtube.com/watch?v=bbkcEiUjehk>
- ▶ Recommended: Floating Point Numbers Work (in 7 minutes!)
 - ▶ https://www.youtube.com/watch?v=W_Knvo9NuJY
- ▶ HOW TO: Convert IEEE-754 Single-Precision Binary to Decimal, Steven Petryk
 - ▶ <https://www.youtube.com/watch?v=4DfXdJdaNYs>
- ▶ Why Is This Happening?! Floating Point Approximation
 - ▶ <https://www.youtube.com/watch?v=2glxbTn7GSc>
- ▶ IEEE 754 Binary to Float Conversion
 - ▶ <https://www.youtube.com/watch?v=9k5rdPUzj8&list=PL-ftFcielQtGxBUfzkbz9tWIRZb-rlJ2p&index=2>
- ▶ IEEE 754 Float to Binary Conversion
 - ▶ <https://www.youtube.com/watch?v=9k5rdPUzj8&list=PL-ftFcielQtGxBUfzkbz9tWIRZb-rlJ2p&index=2>