Number representation and floating-point arithmetic

NC 02 • Module 1: Foundations

Central question	How does a computer store and calculate with real numbers?
Lecture focus	Number bases, binary fractions, IEEE 754 and rounding.
Prerequisite	Basic arithmetic and powers of two.

Learning outcomes

By the end of this lecture, students should be able to...

Represent	Convert integers and fractions between decimal and binary.
Decode	Explain normalized binary notation and IEEE 754 fields.
Evaluate	Recognize rounding, overflow, underflow and cancellation.

Positional number systems

A digit’s value depends on its position and the base.

General form

$(d_n \ldots d_0 . d_{-1} \ldots)_\beta = \sum d_i \, b^i$, where $0 \leq d_i < b$.

Common bases

Binary: 2; octal: 8; decimal: 10; hexadecimal: 16.

Example

$(101.01)_2 = 4 + 1 + 1/4 = (5.25)_{10}$.

Decimal integer to binary

Repeated division by 2; read the remainders from bottom to top.

Step 1	13 ÷ 2 = 6 remainder 1; 6 ÷ 2 = 3 remainder 0.
Step 2	3 ÷ 2 = 1 remainder 1; 1 ÷ 2 = 0 remainder 1.
Result	$(13)_{10} = (1101)_2$. Check: 8 + 4 + 1 = 13.

Decimal fraction to binary

Repeated multiplication by 2; read the integer parts in order.

Step 1	$0.625 \times 2 = 1.25 \rightarrow$ digit 1; remainder 0.25.
Step 2	$0.25 \times 2 = 0.5 \rightarrow 0$; $0.5 \times 2 = 1.0 \rightarrow 1$.
Result	$(0.625)_{10} = (0.101)_2 = 1/2 + 1/8$.

Not every fraction terminates

Finite decimal notation does not imply finite binary notation.

Example	$(0.1)_{10} = (0.0001100110011\ldots)_2$; the pattern 0011 repeats.
Reason	A reduced rational fraction terminates in base 2 only if its denominator is a power of 2.
Implication	Most decimal fractions must be rounded when stored in binary floating point.

Fixed-point versus floating-point

Two ways to allocate a finite number of bits.

Fixed point	A fixed scale: integer 1234 with scale 0.01 represents 12.34.
Floating point	A significand multiplied by a base raised to a variable exponent.
Trade-off	Fixed point has constant spacing; floating point supports a wide dynamic range with variable spacing.

Normalized binary notation

For nonzero normal values: $x = (-1)^s \times 1.f \times 2^e$.

Sign	s = 0 for positive; s = 1 for negative.
Significand	1.f contains a leading 1 and fractional binary digits.
Example	$(13.25)_{10} = (1101.01)_2 = 1.10101_2 \times 2^3$.

IEEE 754 binary formats

The sign, biased exponent and fraction occupy separate fields.

Binary32	1 sign bit | 8 exponent bits | 23 fraction bits; bias = 127; precision = 24 bits.
Binary64	1 sign bit | 11 exponent bits | 52 fraction bits; bias = 1023; precision = 53 bits.
Normal value	$x = (-1)^s \times (1 + F/2^p) \times 2^{(E - \text{bias})}$, where p is the fraction field width.

Worked example: encode 13.25

Construct the IEEE 754 binary32 representation.

Normalize	$13.25 = 1.10101_2 \times 2^3$; sign bit = 0.
Exponent	$E = 3 + 127 = 130 = 10000010_2$.
Fraction / result	10101000000000000000000; full word: 0 | 10000010 | 10101000000000000000000 (0x41540000).

Worked example: decode a binary32 word

Given 1 | 10000000 | 01000000000000000000000.

Sign and exponent	$s = 1$; $E = 128$; $e = 128 - 127 = 1$.
Significand	$1.01_2 = 1 + 1/4 = 1.25$.
Result	$x = -1.25 \times 2^1 = -2.5$.

Special values and subnormal numbers

IEEE 754 reserves the extreme exponent fields.

E = 0	Fraction zero: signed zero. Nonzero fraction: subnormal, with leading 0 and exponent 1 − bias.
E = all ones	Fraction zero: ±∞. Nonzero fraction: NaN (not a number).
Normal range	Binary32 positive normal values: about 1.175×10^{-38} to 3.403×10^{38}.

Rounding and representable spacing

Finite precision means that not all real values can be stored exactly.

Default rounding	Round to nearest, ties to even: choose the nearest representable value; at a tie, choose an even last bit.
Spacing near 1	Binary32: $2^{-23} \approx 1.19 \times 10^{-7}$. Binary64: $2^{-52} \approx 2.22 \times 10^{-16}$.
Unit roundoff	Under rounding to nearest: u = 2^{-24} (binary32), 2^{-53} (binary64). Distinct from spacing at 1.

Floating-point arithmetic

Conceptually: compute the exact operation, then round to the format.

Addition	Align exponents, add significands, normalize and round.
Multiplication	Multiply significands, add exponents, normalize and round.
Example	With 3 significant binary bits, $1.00_2 + 0.001_2 = 1.001_2 \rightarrow 1.00_2$ (tie to even).

Numerical hazards

Mathematical identities may behave differently in finite precision.

Overflow / underflow	Overflow: magnitude exceeds range. Underflow: tiny results can become subnormal or round to zero.
Cancellation	Subtracting nearly equal approximate values can magnify relative error.
Non-associativity	In binary64, $(10^{16} - 10^{16}) + 1 = 1$, but $10^{16} + (-10^{16} + 1) = 0$.

Python demonstration

Python float is typically IEEE 754 binary64.

Decimal approximation

```
>>> 0.1 + 0.2
0.30000000000000004
```

Inspect representation

```
>>> (0.1).hex()
'0x1.999999999999ap-4'
```

Compare safely

Use math.isclose(a, b, rel_tol=..., abs_tol=...) with tolerances chosen for the problem.

Practice questions

Attempt each exercise before viewing the solutions.

Q1 Convert $(19.375)_{10}$ into binary and normalized binary notation.

Q2 Encode -6.5 as IEEE 754 binary32: sign, exponent and fraction.

Q3 Why does 0.1 not terminate in binary? What is binary64 spacing immediately above 1?

Solutions and lecture recap

Next lecture: NC 03 — Absolute, Relative and Percentage Errors.

A1

$19.375 = 10011.011_2 = 1.0011011_2 \times 2^4$.

A2

$-6.5 = -1.101_2 \times 2^2$; 1 | 10000001 | 10100000000000000000000.

A3

1/10 has a factor 5 in its reduced denominator. Spacing immediately above 1 is 2^{-52}.