Floating-Point Numbers

Algorithms, Programming, and Data Representation · Module 92313 · Chapter 4

From Fundamentals to AI Apocalypse · personal study guide

Contents

Start from memory 2

How to use this guide 2

How the chapter fits together 3

4.1 The floating-point model 4

4.2 Binary fractions 6

4.3 Floating-point arithmetic and error 8

4.4 IEEE 754 binary formats 10

4.5 Zero, denormalised numbers, infinity, and NaN 14

Formula map 16

Derive it cold 16

Explain it 17

Mixed practice 18

Exam-style problems 19

Slips in the notes 20

Self-check 21

Answers 22

Made by Thanate Amornjiraporn (Focus)

Floating-point formats encode a wide range of magnitudes with a sign, a significand, and an exponent. This guide builds from binary fractions to IEEE single and double precision, then shows why rounding and special values matter.

Start from memory

Try each question before reading.

Mark the answer sure, unsure, or guessed.

1. In a floating-point number, what does the exponent control?
2. Why can the decimal fraction 0.1 have no finite binary expansion?
3. What must happen before two floating-point values can be added?
4. How many stored bits are in IEEE single precision?
5. What is the purpose of the hidden leading bit in a normalised binary value?

Answers: see "Start from memory" under Answers at the end.

How to use this guide

Study one topic, hide it, and reconstruct the rules and examples from memory.

Attempt mixed practice before opening its worked answers.

1. **Day 0:** Try the cold start and study §4.1. Explain range and precision in your own words.
2. **Day 1:** Rebuild the binary fraction place values in §4.2; solve practice 1, 4, 10, 13, 20, 25.
3. **Day 3:** Work through multiplication, addition, rounding, and exponent limits in §4.3; solve practice 6, 7, 11, 15, 19, 23.
4. **About day 7:** Reconstruct the single-precision fields and conversions in §4.4; solve practice 2, 3, 8, 12, 14, 17, 18, 21, 27, 28.
5. **About day 14:** Recall the special-value rules in §4.5; solve practice 5, 9, 16, 22, 24, 26; complete both exam-style problems without notes.
6. **Before the exam:** Tick only skills you can demonstrate cold; revisit the sections named in missed answers.

When stuck: state the base, format, sign, and exponent convention.

Write every shift and rounding step, then check the magnitude of the result.

Made by Thanate Ammonjiraporn (Focus)

How the chapter fits together

Floating-point representation trades a fixed range of equally spaced values for a range whose spacing changes with magnitude.

Binary fractions give the significand its value.

Arithmetic aligns or combines significands and then rounds to the available precision.

Represent · choose a scale and precision

- **Sign, significand, exponent** §4.1

  value = sign × significand × $\text{base}^{\text{exponent}}$

  Rests on: scientific notation separates a number's digits from its scale.

- **Binary fractions** §4.2

  $0.b_1b_2\ldots = \sum_{i=1}^{\infty} b_i 2^{-i}$

  Rests on: each fractional bit has a negative power-of-two place value.

Compute · preserve approximate value

- **Floating-point arithmetic** §4.3

  multiply: multiply significands; add exponents

  Rests on: exponent laws and normalisation after an operation.

- **Rounding and limits** §4.3

  finite precision → rounding; finite range → overflow or underflow

  Rests on: a format can store only finitely many significand and exponent patterns.

Decode · read the stored fields

- **IEEE formats** §4.4

  sign + biased exponent + fraction

  Rests on: a standard layout makes bit patterns portable between systems.

- **Special values** §4.5

  zero · denormalised · infinity · NaN

  Rests on: reserved exponent patterns encode values outside the normalised range.

Question	Useful method
What value does a pattern represent?	Read sign, exponent, fraction, and format rules.
Can a decimal fraction be exact in binary?	Expand it in powers of 2 or test whether the expansion terminates.
How do I add two values?	Restore hidden bits, align exponents, add signed significands, normalise, and round.
Why did a computation change a value?	Check finite precision, exponent range, and special-value rules.

Made by Tiranate Amornjiraporn (Focus)

4.1 The floating-point model

A fixed number of bits cannot represent every real number.

> **In plain words:** a floating-point number is scientific notation in binary: a sign, a short run of significant digits (the **significand**) and an **exponent** that slides the point left or right. A fixed number of bits holds only so many significant digits, so most real numbers are stored as the nearest value the format can hold, and the gaps between neighbouring values grow as the numbers get larger.

Floating point uses a sign, a significand, and an exponent to cover very large and very small magnitudes.

Why not just use more integer bits?

An integer of 30 decimal digits needs about 100 bits in binary, because one decimal digit carries about 3.32 bits ($\log_2 10^{30} \approx 100$). BCD needs $30 \times 4 = 120$ bits, and ASCII needs $30 \times 8 = 240$ bits. Scientific notation avoids this cost by storing digits and scale separately.

Range and precision

For base R (the radix), a floating-point number has the form sign $\times$ significand $\times$ $\mathrm{R}^{\text{exponent}}$.

The exponent controls scale and therefore the approximate range.

The number of significand bits controls precision, or how finely nearby values can be distinguished.

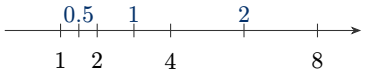
Floating-point values are finite in number, and their spacing changes with magnitude.

Scientific notation model

$$x = \pm M \times R^E$$

Read it as: "x is plus or minus M times R to the E": the significand M carries the digits, and the exponent E says where the point goes.

M is the coefficient (also called the significand or mantissa), R is the base or radix, and E is the exponent. In binary floating point, R=2.



Gaps double with 2 significant bits: the values 1, 1.5, then 2, 3, then 4, 6, 8 lie 0.5, then 1, then 2 apart (the numbers above each zone).

Made by [illegible] Phuriwatpaporn (Focus)

Zones of expressibility and gaps

Take a signed 3-digit coefficient and a signed 2-digit exponent. The positive expressible zone runs from $+.001\times10^{-99}$ to $+.999\times10^{+99}$, and the negative zone mirrors it. Magnitudes above the top lie in the overflow zones. Non-zero magnitudes below the bottom lie in the underflow zones.

The gap between neighbours varies. Next to $.001\times10^{-99}$ the neighbours are 0 and $.002\times10^{-99}$, so the gap is 10^{-102} each way. The same digits at exponent -50 are 10^{-53}, and that value can also be written with a finer exponent. Its neighbours are $.999\times10^{-53}$ (gap 10^{-56}) and $.101\times10^{-52}$ (gap 10^{-55}).

Gap in one binary zone

$$\text{Gap in } [2^e,\ 2^{e+1}) \text{ with } p \text{ significant bits } = 2^{e-p+1}$$

Read it as: "with p significant bits, neighbours in $[2^e, 2^{e+1})$ are 2^{e-p+1} apart": the gap doubles at each power of 2.

(This follows directly from the notes' significand model; the notes only say the gap varies.) The last of the p bits has weight $2^{-(p-1)}$, and the exponent scales it by 2^e.

Normalisation

One value has many forms: $0.023\times10^1=0.230\times10^0=2.3\times10^{-1}$. Hardware needs each number to have one unique form, so the coefficient is normalised to the range [1, R). In binary that range is [1, 2), so the digit before the point is always 1. For example, $1010.11_2\times2^2=1.01011_2\times2^5$: the point moves three places left, so the exponent rises by three.

Worked example · normalise a decimal value

Write 343000×10^{-4} in normalised decimal form with $1\leq M<10$.

1. Move the decimal point in 343000 five places left to make 3.43.
2. Increase the exponent by five to preserve the value: 3.43×10^1.
3. Check: $343000 \times 10^{-4}=34.3$, and $3.43 \times 10^1=34.3$. §4.1

Trap: moving the radix point changes the exponent in the opposite direction if the represented value is to stay the same.

Near miss: 12.5×10^0 and 1.25×10^1 are the same number, but only the second is normalised, with one non-zero digit before the point. A format that allowed both would spend bit patterns on duplicate values.

Recall check · answer without looking

1. Which field chiefly determines range?
2. Which field chiefly determines precision?
3. Why are floating-point values not evenly spaced over the number line?
4. What makes a normalised form useful to hardware?

Answers: see "4.1 The floating-point model" under Answers at the end.

4.2 Binary fractions

Binary fractional place values explain both conversion and approximation.

> **In plain words:** binary fractions work like decimal ones with halves instead of tenths: after the binary point the places are worth 1/2, 1/4, 1/8, …. Some fractions that are short in decimal never end in binary, just as 1/3 never ends in decimal.

The first bit after the binary point has weight 2^{-1}.

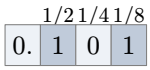
Each following bit has half the previous weight.

Convert binary fractions to decimal

Binary fractional place values

$$0.b_1b_2...b_k = b_1 2^{-1} + b_2 2^{-2} + ... + b_k 2^{-k}$$

Read it as: "$0.b_1b_2…b_k$ is b_1 halves plus b_2 quarters plus …": each bit times its power of one half.
Each b_i is 0 or 1. A leading integer part is evaluated with non-negative powers of 2.



$\mathbf{0.101_2} = 1/2 + 1/8 = 0.625$: each place after the point is worth half the one before.

> **Worked conversion · binary fraction to decimal**
> Convert 0.01101_2 to decimal.
>
> 1. Use the five place values: 1/2, 1/4, 1/8, 1/16, and 1/32.
> 2. Keep the places where the bit is 1: 1/4 + 1/8 + 1/32.
> 3. Add: 0.25+0.125+0.03125=0.40625. §4.2

Made by T

Convert decimal fractions to binary

Repeatedly multiply the fractional part by 2.

At each step, the integer part is the next binary digit, and the new fractional part is carried into the next step.

Worked conversion · terminating binary fraction

Convert 0.6875 to binary.

1. $0.6875 \times 2 = 1.375$, so the first bit is 1; carry 0.375.
2. $0.375 \times 2 = 0.75$, so the next bit is 0; carry 0.75.
3. $0.75 \times 2 = 1.5$, so the next bit is 1; carry 0.5.
4. $0.5 \times 2 = 1.0$, so the next bit is 1; the fraction is now zero.
5. Therefore $0.6875 = 0.1011_2$. Check: $1/2 + 1/8 + 1/16 = 0.6875$. §4.2

Some fractions, such as 0.1 in decimal, repeat in binary. The remainders run 0.1, 0.2, 0.4, 0.8, 0.6, 0.2, and the return to 0.2 means the bits repeat. Cutting off at 0.000110011_2 gives 0.099609375, close to 0.1 but not equal.

A finite significand stores only an approximation of a repeating expansion.

Trap: a decimal fraction that looks short is not necessarily exact in binary.

Near miss: 0.1 has a single digit in decimal, but in binary it is 0.000110011… with the block 0011 repeating forever, so a computer can only store an approximation. $0.5 = 0.1_2$, by contrast, is exact.

Recall check · answer without looking

1. What is the weight of the fourth binary digit after the point?
2. What is 0.101_2 in decimal?
3. Why can a computer store 0.1 only approximately in binary floating point?

Answers: see "4.2 Binary fractions" under Answers at the end.

Made by

4.3 Floating-point arithmetic and error

The exponent and significand must be managed during each operation.

> **In plain words:** to multiply two floating-point numbers, multiply the significands and add the exponents. To add them, first shift the one with the smaller exponent until the exponents match, then add. Either way the result is normalised and rounded, and that rounding is where small errors creep in.

After the exact operation, the result may need normalisation and rounding to fit the available precision.

Multiplication

Multiply normalised values

$$(M_1 \times R^{E_1})(M_2 \times R^{E_2}) = (M_1 M_2) \times R^{E_1+E_2}$$

Multiply the significands and add the exponents. Then normalise and round if needed.

Worked multiplication · normalise the result

Calculate $(2.6\times10^{3})(5.4\times10^{-2})$.

1. Multiply significands: $2.6\times5.4=14.04$.
2. Add exponents: $3+(-2)=1$, giving 14.04×10^{1}.
3. Normalise: $1.404\times10^{2}=140.4$. §4.3

Addition

Before adding, express both numbers with the same exponent.

Shift the significand of the value with the smaller exponent to the right by the exponent difference.

This alignment can discard low-order digits before the addition is even performed.

Align, then add

$$N_1 + N_2 = (M_1 + M_2 \times 10^{E_2-E_1}) \times 10^{E_1}$$

N_2 is the number with the smaller exponent, so E_2-E_1 is negative and M_2 shifts right.

Worked addition · align exponents

Use two significant decimal digits to add 4.5×10^{3} and 6.7×10^{2}.

1. Rewrite the second value at exponent 3: $6.7\times10^{2}=0.67\times10^{3}$.
2. Add significands: $(4.5+0.67)\times10^{3}=5.17\times10^{3}$.
3. Round to two significant digits: 5.2×10^{3}. The exact sum is 5170, while this two-digit result is 5200. §4.3

Rounding, overflow, and comparison

Truncation drops extra digits and tends to bias results in one direction.

Rounding chooses a nearby representable value and usually gives a better approximation.

Worked comparison · truncate or round

With a 2-digit coefficient, store $(2.3\times10^1)(2.3\times10^1)$.

1. Multiply: $2.3\times2.3=5.29$, and $1+1=2$, giving 5.29×10^2, which needs three digits.
2. Truncate: 5.2×10^2. The error is 9, and truncation always errs low for positive values, so the error is biased.
3. Round: 5.3×10^2. The error is only 1, and rounding errs either way, so it is unbiased. §4.3

Exponent overflow occurs when the result is too large for the format.

Exponent underflow occurs when the result is too small for the normalised range.

The slides describe infinity or an exception for overflow, and zero or an exception for underflow in their simplified model.

IEEE formats also provide denormalised numbers near zero.

For approximate comparisons, choose a tolerance ε that fits the scale and precision of the problem.

Absolute tolerance example

$$\text{Treat } a \text{ as close to } b \text{ when } |a-b| < \varepsilon$$
$$\text{For } b = 1 \text{ and } \varepsilon = 0.000005 : 0.999995 < a < 1.000005$$

Read it as: "treat a as close to b when the distance between them is less than ϵ".
A fixed absolute tolerance may be unsuitable when magnitudes vary greatly. The slides also introduce relative closeness.

Sign check: estimate the order of magnitude before accepting a rounded result.

Near miss: 0.1 + 0.2 looks as if it should equal 0.3, but in double precision the computed sum is 0.30000000000000004, so the test 0.1 + 0.2 == 0.3 is false. Compare floating-point results with a tolerance instead.

Recall check · answer without looking

1. What do you do to the exponents when multiplying?
2. Which operand's significand is shifted to align an addition?
3. How do overflow and underflow differ?
4. Why can direct equality comparison be unreliable for computed floating-point values?

Answers: see "4.3 Floating-point arithmetic and error" under Answers at the end.

4.4 IEEE 754 binary formats

A shared format defines how floating-point fields are stored and how arithmetic behaves.

> **In plain words:** IEEE 754 is the shared recipe for storing a floating-point number: one sign bit, an exponent field stored with an offset (the **bias**) so it needs no sign of its own, and the fraction bits after the binary point. A normalised number always starts with 1., so that 1 is not stored at all: it is the **hidden bit**.

The slides introduce IEEE 754 as a widely adopted standard for predictable binary floating-point arithmetic.

Made by Thanate Amornjiraporn (Focus)

Single and double precision

Format	Total	Sign	Exponent	Fraction field	Significand precision	Bias
Single	32 bits	1 bit	8 bits	23 bits	24 bits for normal values	127
Double	64 bits	1 bit	11 bits	52 bits	53 bits for normal values	1023

For a normalised binary value, the leading significand bit is 1 and is omitted from the stored fraction field.

The coefficient is called the significand in the standard. Single precision gives about 7 decimal digits of precision and normalised ranges of roughly $\pm 10^{-38}$ to $\pm 10^{38}$. Double precision gives about 16 digits and roughly $\pm 10^{-308}$ to $\pm 10^{308}$. The notes reserve single precision for when memory is scarce, or for debugging numerical work, because rounding errors show up sooner.

The exponent field stores an excess (biased) value rather than a two's-complement exponent. This lets non-negative floats be compared with simple integer comparisons.

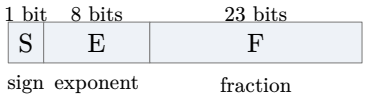
Exponent	8-bit excess-127 pattern
−127	0000 0000
0	0111 1111
1	1000 0000
128	1111 1111

Normalised value rule

$$\text{value} = (-1)^S \times 1.F \times 2^{E-\text{bias}}$$
$$\text{Single bias} = 127$$
$$\text{double bias} = 1023$$

Read it as: "value is −1 to the S, times 1.F, times 2 to the (E minus bias)": the sign, the stored fraction with its hidden 1, and the exponent once the bias is removed.

S is the sign bit, E is the unsigned exponent field, and F is the stored fraction interpreted after the binary point. This rule is for normalised values.



IEEE 754 single precision: 1 sign bit, 8 exponent bits stored with bias 127, and 23 fraction bits after the hidden 1, 32 bits in all.

Convert decimal to single precision

Worked conversion · 42.6875

1. Convert to binary: $42.6875 = 101010.1011_2$.
2. Normalise: 1.010101011×2^5, so the sign bit is 0.
3. Store the fraction after the hidden leading 1: 01010101100000000000000.
4. Bias the exponent: $5+127 = 132 = 10000100_2$.
5. The fields are 0 | 10000100 | 01010101100000000000000, or hexadecimal 422A C000. §4.4

Decode single precision

Worked decoding · BEC0 0000

1. The sign bit is 1, so the value is negative.
2. The exponent field is $01111101_2 = 125$; subtract the bias: $125 - 127 = -2$.
3. Restore the hidden bit: $1.1000\ldots_2 = 1.5$.
4. Scale: $1.5 \times 2^{-2} = 0.375$. Apply the sign: -0.375. §4.4

Made by Thanate Amornj (Focus)

Add in single precision

Worked addition · 42.6875 + 0.375

1. Read the fields: 42.6875 has exponent field 132 and 0.375 has 125. Restore the hidden bits: 1.010101011_2 and 1.1_2.
2. The exponents differ by 132−125=7, so shift the smaller significand 7 places right: 1.1_2 becomes 0.00000011_2.
3. Add: $1.010101011+0.00000011=1.010110001_2$, at exponent 5.
4. Check: $1.010110001_2\times2^5=101011.0001_2=43.0625$, which is 42.6875+0.375.
5. The fields are 0 | 10000100 | 01011000100000000000000, or hexadecimal 422C 4000. §4.4

Trap: restore the hidden leading 1 when decoding a normalised value, but do not apply that rule to a denormalised value.

Near miss: the single-precision exponent field 10000000 looks like a large exponent, but it means 128 − 127 = 1. Subtract the bias before reading the exponent.

Recall check · answer without looking

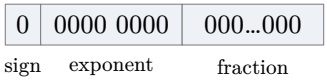
1. How many bits are stored in the single-precision fraction field?
2. What is the single-precision exponent bias?
3. Why can a normalised value have 24 bits of significand precision while storing only 23 fraction bits?

Answers: see "4.4 IEEE 754 binary formats" under Answers at the end.

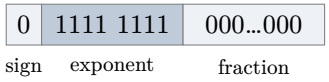
Made by Thana

4.5 Zero, denormalised numbers, infinity, and NaN

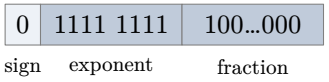
Reserved exponent patterns extend useful behaviour beyond normalised values.



+0: every bit is 0.



$+\infty$: exponent all 1s, fraction 0.



NaN: exponent all 1s, fraction not 0.

> **In plain words:** some exponent patterns are kept for special cases. An exponent of all zeros means zero or a tiny **subnormal** number that fills the gap just above zero; an exponent of all ones means infinity when the fraction is zero, and NaN, "not a number", for results such as 0/0.

IEEE formats encode five kinds of value: zero, normalised numbers, denormalised numbers, infinity and NaN. In single precision, exponent field zero and exponent field all ones have special interpretations, so normalised exponents run from −126 to 127 rather than −127 to 128.

Exponent field	Fraction field	Meaning	True exponent
All zeroes	All zeroes	Positive or negative zero	—
All zeroes	Non-zero	Denormalised (subnormal) number	−126
Neither all zeroes nor all ones	Any pattern	Normalised number	−126 to 127
All ones	All zeroes	Positive or negative infinity	—
All ones	Non-zero	NaN, or not-a-number	—

For a denormalised single-precision number, the leading bit is 0 and the effective exponent is −126.

Denormalised values provide a gradual step from the smallest normalised values toward zero.

With the default round-to-nearest mode, overflow produces a signed infinity; other rounding modes can produce the largest finite value. Division of a finite non-zero value by zero also produces a signed infinity. You can do basic arithmetic with infinity, such as $\infty+5=\infty$.

NaN represents an operation without a real-number result, such as 0/0, infinity−infinity, $0\times\infty$, or the square root of a negative number.

Operations on infinity and NaN follow format rules, not ordinary real-number algebra in every case.

Made by Innate Amorimborn (Inc.

Special arithmetic examples

Operation	Result
finite x ÷ ±∞	signed 0
finite non-zero ÷ ±0	signed ∞
+∞ + +∞	+∞
±∞ × ±∞	±∞
±0 ÷ ±0	NaN
+∞ − +∞	NaN
±∞ ÷ ±∞	NaN
±∞ × 0	NaN

The signs of zero and infinity in these results depend on the operand signs. The notes say an invalid operation yields either a NaN result (quiet NaN) or an exception (signalling NaN); this is simplified, as the slips list explains.

Worked classification · single precision fields

Classify sign 0, exponent 11111111, fraction 000…000.

1. The exponent field is all ones.
2. The fraction field is all zeroes.
3. This encodes positive infinity. §4.5

Trap: an all-ones exponent does not always mean infinity; inspect the fraction field to distinguish infinity from NaN.

Near miss: +0 and −0 have different bit patterns and compare as equal, yet they are not interchangeable: 1/(+0) gives +∞ while 1/(−0) gives −∞. The sign bit still matters at zero.

Recall check · answer without looking

1. How do you distinguish infinity from NaN?
2. What is the main purpose of denormalised numbers?
3. What does zero exponent and zero fraction encode?

Answers: see "4.5 Zero, denormalised numbers, infinity, and NaN" under Answers at the end.

Formula map

Idea	Relation	Use
Floating-point model	$x = \pm M \times R^E$	Separate sign, significand, and scale.
Binary fraction	$0.b_1b_2\ldots = \sum_{i=1}^{\infty} b_i 2^{-i}$	Convert binary fractions to decimal.
Multiplication	$(M_1R^{E_1})(M_2R^{E_2}) = M_1M_2R^{E_1+E_2}$	Multiply significands and add exponents.
Addition	$N_1+N_2 = (M_1+M_2\times10^{E_2-E_1})\times10^{E_1}$	Align to the larger exponent, then add coefficients.
Gap in one zone	2^{e-p+1}	Spacing in $[2^e, 2^{e+1})$ with p significant bits.
Single normalised value	$(-1)^S \times 1.F \times 2^{E-127}$	Decode a normalised single-precision pattern.
Double normalised value	$(-1)^S \times 1.F \times 2^{E-1023}$	Decode a normalised double-precision pattern.
Denormalised single value	$(-1)^S \times 0.F \times 2^{-126}$	Decode a denormalised single-precision pattern.
Absolute closeness	$\lvert a-b\rvert<\varepsilon$	Compare computed values with a chosen tolerance.

Check every answer

State the format and whether the value is normalised

Track the exponent when shifting the radix point

Check sign, scale, range, and final rounding

These are representation calculations, so verify the bit fields and magnitude rather than physical units.

Derive it cold

Cover the explanations and rebuild each result on blank paper.

1. **Binary fractions:** derive each bit's place value as a negative power of 2. §4.2
2. **Decimal fraction conversion:** use repeated multiplication by 2 and track the fractional remainder. §4.2
3. **Multiplication:** use exponent laws, then normalise. §4.3
4. **Addition:** align exponents, combine significands, and round. §4.3
5. **Single-precision decoding:** recover the sign, unbiased exponent, and hidden bit. §4.4
6. **Single-precision addition:** restore hidden bits, shift by the exponent difference, add, and re-encode. §4.4
7. **Special values:** derive the classification from exponent and fraction fields. §4.5

Made by Thanate

Explain it

Answer each why or when question aloud before opening its model response.

1. Why does increasing the exponent change range more than precision?
2. Why can two nearby real values have different spacing when stored at different magnitudes?
3. Why does 0.1 decimal not have a finite binary fraction?
4. Why must exponents be aligned before adding two floating-point values?
5. Why can rounding during addition lose a small addend?
6. Why is the leading 1 omitted for normalised binary significands?
7. When would a denormalised number be used?
8. Why should an approximate comparison use a tolerance chosen for the problem?

Answers: see "Explain it" under Answers at the end.

Made by Thanate Amornjiraporn (Focus)

Mixed practice

Try all 28 problems before opening the worked answers.

The topics are interleaved so you must choose the method each time.

1. Convert 0.1101_2 to decimal.
2. How many bits of significand precision does normalised IEEE single precision provide?
3. Normalise 0.00111_2 as $M \times 2^E$ with $1 \le M < 2$.
4. Convert 0.375 decimal to binary.
5. In a floating-point format, which field chiefly determines range?
6. Multiply $(3 \times 10^2)(5 \times 10^{-4})$ and give a normalised result.
7. Align 7.2×10^5 to exponent 7.
8. How many stored fraction bits does double precision have?
9. Classify single-precision fields S=1, E=all ones, F=0.
10. Convert 0.01011_2 to decimal.
11. Use two significant decimal digits to round $8.46 \times 10^3 + 2.1 \times 10^2$.
12. What is the single-precision exponent bias?
13. Convert decimal 0.4375 to binary.
14. For normalised single precision with exponent field E=130, what is the unbiased exponent?
15. State one reason truncation can be less desirable than rounding.
16. Classify single-precision fields E=0 and F non-zero.
17. For 1.25×2^3, give the decimal value.
18. How many total bits are used in double precision?
19. Give an absolute-tolerance test for a value a to be close to b using ε.
20. Convert 1.101_2 to decimal.
21. Why is 1.25×2^3 an exact binary value?
22. What do $\pm$non-zero $\div 0$ and $\pm 0 \div \pm 0$ produce?
23. What happens to the exponent when multiplying two floating-point values?
24. When the exponent field is all ones and the fraction is zero, what is the value when S=0?
25. Is 0.1 finite in binary? Give the first 8 bits after the point.
26. With fixed p, what happens to the gap from [8, 16) to [16, 32)?
27. Encode 0.375 in single precision, in hexadecimal.
28. Decode the single-precision word C154 0000.

Answers: see "Mixed practice" under Answers at the end.

Made by Thanate Arornjitaporn (Focus)

Exam-style problems

Allow about 20 minutes per problem.

Show the format, fields, and each conversion step.

1 · Encode and decode single precision

Encode 13.25 as IEEE single precision, then decode the resulting fields to verify the value.

1. Convert 13.25 to binary.
2. Normalise it and identify sign, exponent, and stored fraction.
3. Give the exponent field and verify by decoding.

Answers: see "Exam-style problems" under Answers at the end.

2 · Align, add, and assess precision

Using two significant decimal digits, calculate $3.4\times10^{4} + 5.6\times10^{2}$.

1. Align both values to exponent 4.
2. Add the significands and round to two significant digits.
3. Compare the rounded result with the exact decimal sum and explain the source of the difference.
4. Carry the same alignment into single precision: add 13.25 and 0.375, and give the fields of the sum.

Answers: see "Exam-style problems" under Answers at the end.

Made by Thanate Angrnjiraporn (Focus)

Slips in the notes

Each item was checked on the rendered slide. The guide uses the corrected form; numbers are PDF slide numbers.

- **Slide 3 · light year:** printed as 9,130,000,000,000 km. The accepted value is about 9.46×10^{12} km.
- **Slide 3 · solar mass:** printed as 2 followed by nine groups of 000, which is 2×10^{27} kg. The accepted value is about 2×10^{30} kg.
- **Slide 3 · electron diameter:** printed as 10^{-20} m. This is probably only illustrative, because an electron has no established diameter. The guide does not use it.
- **Slide 40 · single-precision range endpoints are reversed:** it prints 10^{38} to 10^{-38} for the positive range. In increasing order the range is approximately 10^{-38} to 10^{38}, and the negative range is -10^{38} to -10^{-38} as printed.
- **Slide 42 · double-precision range endpoints are reversed:** it prints 10^{308} to 10^{-308} for the positive range. In increasing order the range is approximately 10^{-308} to 10^{308}.
- **Slide 49 · signalling NaN (simplified):** it presents a signalling NaN as the outcome of an operation that raises an exception. In IEEE 754 a signalling NaN is an encoding that raises the exception when it is used. Learn the slide's wording, but know that it is a simplification.
- **Slides 41 and 47–48 · notation clash, not an error:** slide 41 lists −127 and 128 as excess values, but those two patterns are reserved. Slide 48 reconciles this: the normalised range is −126 to 127.
- **Slides 40 and 42 · notation clash, not an error:** F is labelled "Significand", but it holds only the stored bits after the hidden 1. The guide calls F the fraction field.

The decimal ranges are rough magnitudes, not exact endpoint values.

Made by Thanate Angkifcharoen (Focus)

Self-check

Tick a skill only in a later session, at least one day after studying, when you can demonstrate it without notes.

- ☐ Explain how significand and exponent affect precision and range. (5, 26)
- ☐ Normalise a floating-point value and preserve its magnitude. (3)
- ☐ Convert binary fractions to decimal. (1, 10, 20)
- ☐ Convert terminating decimal fractions to binary. (4, 13)
- ☐ Explain why some decimal fractions repeat in binary. (25)
- ☐ Multiply floating-point numbers and normalise. (6)
- ☐ Align exponents before adding. (7, 11)
- ☐ Round results to a stated precision. (11, 15)
- ☐ State an approximate comparison using tolerance. (19)
- ☐ State the single-precision field sizes and bias. (2, 12)
- ☐ Encode a decimal value in single precision. (27, exam 1)
- ☐ Decode a normalised single-precision value. (14, 28, exam 1)
- ☐ Classify a denormalised pattern and explain its role. (16)
- ☐ Classify special-value fields and predict basic infinity/NaN arithmetic. (9, 22, 24)
- ☐ Rebuild the derivations from the list.
- ☐ Complete both exam-style problems with checks.

Made by Thanate Angkinjaporn (Focus)

Answers

Start from memory

1. The exponent controls scale and therefore the range of magnitudes. §4.1
2. Its binary expansion repeats, so a finite significand must approximate it. §4.2
3. Align the exponents by shifting the significand of the value with the smaller exponent. §4.3
4. Thirty-two bits: one sign, eight exponent, and twenty-three stored fraction bits. §4.4
5. For normalised binary numbers the leading significand bit is always 1, so it can be omitted from storage and treated as implicit. §4.4

4.1 The floating-point model

1. The exponent field. §4.1
2. The significand field. §4.1
3. Each exponent scales the significand by a different power of the base, so the gap between representable values grows with magnitude. §4.1
4. Each number gets one unique representation. §4.1

4.2 Binary fractions

1. $2^{-4}=1/16$. §4.2
2. $1/2+1/8=0.625$. §4.2
3. The binary expansion repeats, but the significand has a finite number of bits and must be rounded or truncated. §4.2

4.3 Floating-point arithmetic and error

1. Add them. §4.3
2. Shift the significand belonging to the smaller exponent to the right. §4.3
3. Overflow exceeds the maximum magnitude; underflow falls below the smallest normalised magnitude. §4.3
4. Finite precision can make a computed result differ slightly from its exact mathematical value, so a suitable tolerance may be needed. §4.3

4.4 IEEE 754 binary formats

1. Twenty-three. §4.4
2. 127. §4.4
3. The leading binary 1 is implicit, giving 24 significant bits including the 23 stored fraction bits. §4.4

Made by Thanate Amornjiraporn (Focus)

4.5 Zero, denormalised numbers, infinity, and NaN

1. Both use an all-ones exponent. A zero fraction means infinity; a non-zero fraction means NaN. §4.5
2. They represent values between zero and the smallest normalised magnitude, making the transition toward zero gradual. §4.5
3. Positive or negative zero, according to the sign bit. §4.5

Explain it

1. The exponent scales a significand by powers of the base, expanding which magnitudes can be represented. Precision depends chiefly on how many significand bits distinguish values at a given scale.
2. At a larger exponent, the same significand step is multiplied by a larger power of 2. Therefore adjacent representable values are farther apart.
3. A finite binary fraction has a denominator that is a power of 2 after reduction. One tenth has a factor of 5 in its denominator, so its binary expansion repeats.
4. Only values with the same exponent have significands describing the same scale. Alignment expresses both at a shared scale before combining them.
5. Alignment shifts the smaller significand to the right. If the exponent difference is large, its significant bits may fall beyond the available precision.
6. Every normalised binary number begins with 1 before the point, so the format can infer that bit and use its storage for another fraction bit.
7. A denormalised represents a non-zero value smaller than the minimum normalised magnitude, close to zero.
8. Absolute or relative error that is acceptable depends on the scale and purpose of the calculation. One fixed tolerance may be too strict for large values or too loose for small ones.

Made by Thanate Anonnijaporn (Focus)

Mixed practice

1. $1/2+1/4+1/16=0.8125$. §4.2
2. 24 bits: the 23 stored fraction bits plus the implicit leading 1. §4.4
3. $1.11_2\times2^{-3}$. The point moves three places right to make the significand 1.11, so the exponent is -3. §4.1
4. $0.375=3/8=0.011_2$. §4.2
5. The exponent field. §4.1
6. Multiply significands: 15. Add exponents: -2. Result 15×10^{-2}, normalised as 1.5×10^{-1}. §4.3
7. $7.2\times10^5=0.072\times10^7$; shift the significand two places right as the exponent rises by two. §4.3
8. 52 stored fraction bits. §4.4
9. Negative infinity: all-ones exponent and zero fraction mean infinity; sign bit 1 makes it negative. §4.5
10. $1/4+1/16+1/32=0.34375$. §4.2
11. Align $2.1\times10^2=0.21\times10^3$. Sum 8.67×10^3; to two significant digits this is 8.7×10^3. §4.3
12. 127. §4.4
13. $0.4375=1/4+1/8+1/16=0.0111_2$. §4.2
14. $130-127=3$. §4.4
15. Truncation always discards in the same direction for positive values and can introduce a biased error. §4.3
16. A denormalised number. The exponent is zero and the fraction is non-zero. §4.5
17. $1.25\times8=10$. §4.1
18. 64 bits: one sign, eleven exponent, and 52 fraction bits. §4.4
19. For example, treat a as close to b when $|a-b|<\varepsilon$, with ε chosen for the expected scale and precision. §4.3
20. $1+1/2+1/8=1.625$. §4.2
21. $1.25=1.01_2$ is a finite binary fraction, and multiplying by 2^3 only shifts the binary point. §4.2
22. $\pm\infty$ and NaN, respectively. The first is division of a non-zero value by zero; zero divided by zero has no real result. §4.5
23. Add the exponents and multiply the significands, then normalise if needed. §4.3
24. Positive infinity. §4.5
25. No. Doubling the remainders gives $0.00011001\ldots_2$, and the remainder 0.2 recurs, so the bits repeat without end. §4.2
26. It doubles: e rises from 3 to 4 while p stays fixed, and the gap is 2^{e-p+1}. §4.1
27. $0.375=1.1_2\times2^{-2}$. E=125=01111101_2, F=1000…, so the word is 3EC0 0000. §4.4
28. S=1, E=10000010_2=130, so the exponent is 3. F=10101000…, so the value is $-1.10101_2\times2^3=-13.25$. §4.4

Made by Thanatee Amornjiraporn (Focus)

Exam-style problems

1. $13.25{=}1101.01_2$, since $13{=}1101_2$ and $0.25{=}0.01_2$. §4.2
2. Normalise: $1.10101{\times}2^3$. The sign is 0. The stored fraction is 10101000000000000000000 after padding to 23 bits. §4.4
3. Biased exponent is $3{+}127{=}130{=}10000010_2$. The pattern is 0 | 10000010 | 10101000000000000000000. Decoding restores $1.10101_2{\times}2^3{=}13.25$. §4.4

Exam-style problems

1. $5.6{\times}10^2{=}0.056{\times}10^4$. §4.3
2. $(3.4{+}0.056){\times}10^4{=}3.456{\times}10^4$, which rounds to $3.5{\times}10^4$. §4.3
3. The exact sum is 34560 and the rounded result is 35000. The smaller addend contributes digits beyond the two-digit precision, so rounding changes the stored result. §4.3
4. $13.25{=}1.10101_2{\times}2^3$ and $0.375{=}1.1_2{\times}2^{-2}$. The exponents differ by 5, so 1.1_2 becomes 0.000011_2. The sum is $1.101101_2{\times}2^3{=}13.625$. The fields are 0 | 10000010 | 10110100000000000000000. §4.4

Made by Thanate Amornjiraporn (Focus)

Personal study guide based on the supplied “04_floating_point_numbers.pdf” (50 slides; slide 2 is dated 22 April 2024). Written in my own words; no lecture pages or figures are reproduced. Not an official FAU publication; check course details against the current notes. Made by **Thanate Amornjiraporn (Focus)**.