

13.3 Floating-point numbers, representation and manipulation

Name: _____ Class: _____ Date: _____

Total: 21 marks

KEY WORDS exponent 指数 • mantissa 尾数 • floating-point 浮点 • two's complement 补码 • floating-point number 浮点数

Objective

Build the skills to answer exam questions on **floating-point numbers** 浮点数 — the format, conversions, normalisation and rounding error.

You must be able to:

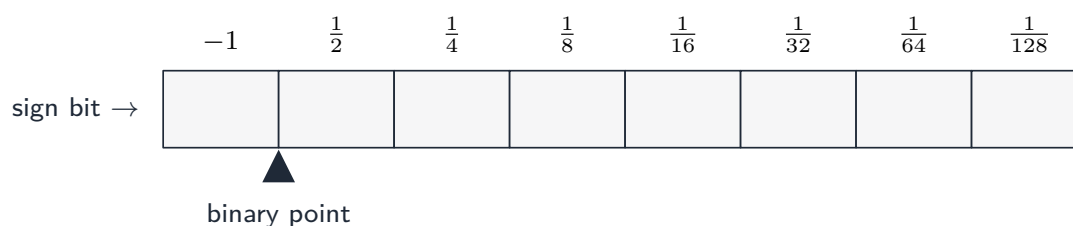
- describe the **mantissa** + **exponent** format
- convert **binary** ↔ **denary** floating-point
- **normalise** a number and explain **rounding errors**

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ The format

A floating-point number is $\text{mantissa} \times 2^{\text{exponent}}$. Both are stored in **two's complement**. The mantissa is a fraction with the binary point just after the **sign bit**:



■ Binary → denary

Mantissa 01010000, exponent 00000010:

- sign bit 0 → positive; fraction $.1010000 = \frac{1}{2} + \frac{1}{8} = 0.625$
- exponent $00000010 = +2$
- value $= 0.625 \times 2^2 = 0.625 \times 4 = \mathbf{2.5}$

■ Denary $\rightarrow$ binary (normalised)

Convert 6.0 (8-bit mantissa, 8-bit exponent):

- 6 in binary = 110.0
- move the point left until it is just after the first 1: 0.110×2^3
- mantissa = 01100000, exponent = 3 = 00000011

■ Normalising

A positive mantissa is **normalised** when it begins 0.1... (no wasted leading zeros — maximum precision). Shift left and decrease the exponent:

- 00011000 exp 00000101 = 0.0011×2^5
- shift mantissa left 2 $\rightarrow$ 01100000, exponent $5 - 2 = 3 = 00000011$ (same value).

■ Negative numbers

A **negative** mantissa is in **two's complement** and is normalised when it begins 1.0... (the first two bits differ). To represent -0.75 : take $+0.75 = 0.1100000$, then two's-complement it (invert, add 1) $\rightarrow 1.0100000$. The sign/integer bit has place value -1 , so $1.0100000 = -1 + \frac{1}{4} = -0.75$.

■ Rounding error

Some denary reals (like 0.1) are **repeating** binary fractions, so they are stored only **approximately**. Errors build up, so test $\text{ABS}(x - 0.3) < 1\text{e-}9$, not $x = 0.3$.

2 Practice

Now apply the methods above.

2.1 State the **two** parts of a floating-point number and what each represents. [2]

2.2 A mantissa 01000000 has exponent 00000011. State its **denary** value. [2]

2.3 State what it means for a number to be **normalised**. [1]

2.4 State **one** reason some denary real numbers cannot be stored exactly. [1]

3 Exam-style questions

A system uses an **8-bit mantissa** and an **8-bit exponent**, both in two's complement.

3.1 Convert mantissa 01010000, exponent 00000010 to **denary**. Show your working. [3]

3.2 Convert 6.0 to its **normalised** floating-point representation. Show your working. [4]

3.3 Normalise the mantissa 00011000 with exponent 00000101. Give the new mantissa and exponent. [3]

3.4 Explain why $0.1 + 0.2$ might **not** equal exactly 0.3 when stored in floating-point. [2]

3.5 Convert the floating-point number with mantissa 10110000 and exponent 00000010 to **denary**. Show your working. [3]

Solutions

2.1 The **mantissa** holds the **significant digits**; the **exponent** is the **power of 2** to multiply by.

2.2 $0.1000000 = 0.5$; exponent = 3; $0.5 \times 2^3 = 4$.

2.3 The first significant bit is **immediately after the binary point** (0.1... for a positive number) — no wasted leading zeros, so precision is maximised.

2.4 Some denary fractions (e.g. 0.1) are **repeating/infinite** in binary, so they must be **truncated** to fit the mantissa.

3.1 Sign bit 0 $\rightarrow$ positive; $.1010000 = \frac{1}{2} + \frac{1}{8} = 0.625$, exponent = 2; $0.625 \times 2^2 = \mathbf{2.5}$.

3.2 $6 = 110.0_2$; normalise to 0.110×2^3 ; mantissa 01100000; exponent 00000011.

3.3 $00011000 = 0.0011 \times 2^5$; shift mantissa **left 2**, exponent $5 - 2 = 3$; new mantissa 01100000; new exponent 00000011.

3.4 0.1 and 0.2 cannot be stored **exactly** in binary (they are repeating fractions); the small stored errors **add up**, so the result is very close to but **not exactly** 0.3.

3.5 Sign bit 1 $\rightarrow$ negative; $1.0110000 = -1 + \frac{1}{4} + \frac{1}{8} = -0.625$, exponent = 2; $-0.625 \times 2^2 = \mathbf{-2.5}$.