

Exercise walk-through

13.3

Floating-point numbers, representation and manipulation

A-Level Computer Science

Mantissa $\times$ two-to-the-exponent

A **floating-point** number is binary scientific notation:

$$\text{value} = \text{mantissa} \times 2^{\text{exponent}}$$

Both fields are **two's complement** integers.

- more mantissa bits $\Rightarrow$ more **precision**
- more exponent bits $\Rightarrow$ more **range**

The mantissa is a binary *fraction*: the first bit after the point is $\frac{1}{2}$, then $\frac{1}{4}$, $\frac{1}{8}$, ...

mantissa

0.1010000

exponent

00000010

$$0.625 \times 2^2 = 2.5$$

Worked example – binary to denary

Mantissa 10110000, exponent 00000011

Exponent 00000011 = +3.

Mantissa starts with 1, so it is **negative**. Sign bit = -1, fraction $\frac{1}{4} + \frac{1}{8} = 0.375$:

$$m = -1 + 0.375 = -0.625$$

$$\text{value} = -0.625 \times 2^3 = -5.0$$

A negative mantissa follows two's-complement rules. Convert the exponent first – it is the easy part.

Worked example – store +2.5

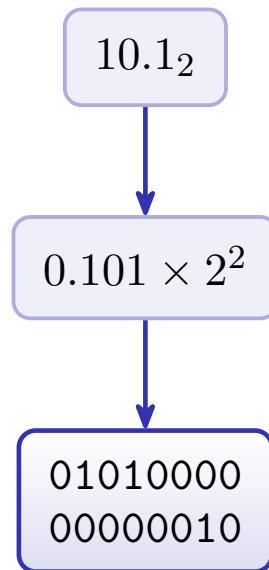
Write +2.5 in this 8+8-bit format

In binary $2.5 = 10.1$.

As a normalised fraction: $2.5 = 0.101 \times 2^2$.

mantissa 01010000 (sign 0, then .101)

exponent 00000010 ($= 2$)



Normalisation maximises precision

A number is **normalised** when the first significant bit sits **immediately after the binary point** – no wasted leading zeros.

- positive: starts 0.1 ...
- negative (two's complement): starts 1.0 ...

Shift the mantissa left and decrease the exponent (or right and increase it) until that pattern holds. The value is unchanged.

Normalise 0.0101

Shift left until it reads 0.1 ...:

$$0.0101 \rightarrow 0.101 \times 2^{-1}$$

mantissa 0.101, exponent -1

Each **left** shift subtracts one from the exponent.

Approximation and rounding errors

Many denary reals **cannot be stored exactly** in binary – e.g.

- **rounding errors** build up over many operations ($0.1 + 0.2$ is not exactly 0.3).
- **comparisons fail** – test $\text{ABS}(x - 0.3) < 1\text{e-}9$ instead of $x = 0.3$.
- subtracting two nearly-equal values loses precision.
- **overflow** (a result too large for the exponent's range) and **underflow** (a result too small, rounding to zero) occur when the exponent runs out of range.

Exercises 2.1-2.3

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]

Exercises 2.1-2.3

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

2.1

- the **mantissa** holds the **significant digits**
- the **exponent** is the **power of 2** to multiply by

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

2.2

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

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

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

Exercises 2.4·3.1

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

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

Exercises 2.4·3.1

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

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

3.1

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

Exercises 3.2-3.3

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]

Exercises 3.2-3.3

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

3.2

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

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

3.3

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

Exercises 3.4-3.5

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]

Exercises 3.4-3.5

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

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

3.5

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