ANSWERS

A7.3 Fixed point and floating point

Data representation · A level · OCR H446 1.4.1, AQA 7517 4.5.4.4, Eduqas A500QS 2.3 · about 35 min

What this lesson is about

Binary fractions, mantissa and exponent, normalisation, and adding floating point numbers.

- Binary fractions
- Fixed point
- Floating point: mantissa and exponent
- Normalisation
- Adding and subtracting floating point
- The robot's view

Questions

6 marks in all

1. An unsigned fixed point byte has 4 bits before the binary point and 4 after. What is the value of 10110110? [1 mark]

> **Answer:** 11.375. 1011.0110 is 8 + 2 + 1 + 0.25 + 0.125 = 11.375.

2. A floating point number has an 8-bit two's complement mantissa 01010000 and a 4-bit two's complement exponent 0011. What is its denary value? [1 mark]

> **Answer:** 5. The mantissa is 0.1010000 = 0.625 and the exponent is 3, so the value is 0.625 x 8 = 5.

3. A floating point number has an 8-bit two's complement mantissa 10100000 and a 4-bit two's complement exponent 0001. What is its denary value? [1 mark]

> **Answer:** -1.5. The mantissa is -1 + 0.25 = -0.75 and the exponent is 1, so the value is -0.75 x 2 = -1.5.

4. Which of these 8-bit two's complement mantissas are normalised? [1 mark]

Tick every answer that is true.

- ☐ **A** 01101000
- ☐ **B** 00110000
- ☐ **C** 10110000
- ☐ **D** 11010000

> **Answer:** A, C. A normalised mantissa's first two bits differ: 01 for a positive number, 10 for a negative one.

5. Normalise the floating point number with 8-bit mantissa 00011000 and 4-bit exponent 0011. Give the mantissa, a space, then the exponent. [1 mark]

> **Answer:** 01100000 0001. Shift the mantissa left 2 places so its first two bits differ, and subtract 2 from the exponent. Both forms are 1.5.

6. Why are floating point numbers normalised? [1 mark]

Tick every answer that is true.

- ☐ **A** It gives the greatest precision for the number of mantissa bits
- ☐ **B** Each value has only one representation
- ☐ **C** It increases the range of the exponent
- ☐ **D** It stops rounding errors completely

> **Answer:** A, B. Removing redundant leading bits uses every mantissa bit for precision and gives one form per value. It does not change the exponent's range or remove rounding.

The task: decode a float

Write two functions. - `float_value(mantissa, exponent)` : `mantissa` is a string of 8 characters `0` or `1`, a two's complement fixed point number with the binary point after the first bit (place values -1, ½, ¼ down to 1/128). `exponent` is a string of 4 characters `0` or `1`, a two's complement integer from -8 to 7. It returns the value, mantissa × 2 to the power of exponent, as a `float`. - `is_normalised(mantissa)` : returns `True` when the first two bits of the mantissa are different, otherwise `False`. The loop prints each line as `01101000 0011 = 6.5 normalised` or `00011000 0101 = 6.0 not normalised`, with the value exactly as Python prints the float.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def float_value(mantissa, exponent):
    return 0.0

def is_normalised(mantissa):
    return False

for mantissa, exponent in [("01101000", "0011"), ("10110000", "0010"), ("00011000",
"0101"), ("11100000", "1111")]:
    word = "normalised" if is_normalised(mantissa) else "not normalised"
    print(mantissa, exponent, "=", float_value(mantissa, exponent), word)
```

The hint students can ask for: The mantissa's place values start at -1 and then halve: 1/2, 1/4, 1/8 and so on. The exponent is an ordinary two's complement integer with -8 as its first place value. The value is the mantissa multiplied by 2 to the power of the exponent. A normalised mantissa's first two bits are different.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def float_value(mantissa, exponent):
    m = -1.0 if mantissa[0] == "1" else 0.0
    place = 0.5
    for bit in mantissa[1:]:
        if bit == "1":
            m = m + place
        place = place / 2
    e = -8 if exponent[0] == "1" else 0
    e = e + int(exponent[1:], 2)
    return m * 2 ** e

def is_normalised(mantissa):
    return mantissa[0] != mantissa[1]

for mantissa, exponent in [("01101000", "0011"), ("10110000", "0010"), ("00011000",
"0101"), ("11100000", "1111")]:
    word = "normalised" if is_normalised(mantissa) else "not normalised"
    print(mantissa, exponent, "=", float_value(mantissa, exponent), word)
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.