ANSWERS

A15.7 Calculation and theory questions

Exam preparation · A level · OCR H446 1.4.1, AQA 7517 4.4.1.2, Eduqas A500QS 1.8 · about 40 min

What this lesson is about

Number representation, floating point, Boolean logic, state machines and complexity answered quickly and checked.

- Always show your working
- Two's complement
- Floating point
- Boolean logic
- State machines
- Complexity

Questions

6 marks in all

1. What is the denary value of the 8-bit two's complement number 11101100? [1 mark]

> **Answer:** -20. -128 + 64 + 32 + 8 + 4 = -20. Check: flip and add 1 gives 00010100, which is 20.

2. A normalised floating point number has mantissa 01101000 (point after the first bit) and exponent 0011. What is its denary value? [1 mark]

> **Answer:** 6.5 (accept within 0.001). The mantissa is 0.8125 and the exponent is 3, so 0.8125 x 8 = 6.5.

3. Which mantissa is normalised? [1 mark]

- ☐ **A** 10100000
- ☐ **B** 11010000
- ☐ **C** 00110000
- ☐ **D** 11100000

> **Answer:** A. A normalised mantissa starts 01 if positive or 10 if negative.

4. In how many rows of its truth table is NOT (A AND B) OR C true? [1 mark]

> **Answer:** 7. It is false only when A = 1, B = 1 and C = 0.

5. An algorithm has a loop over n items inside another loop over n items. What is its time complexity? [1 mark]

- ☐ **A** $O(n^2)$
- ☐ **B** $O(n)$
- ☐ **C** $O(\log n)$
- ☐ **D** $O(2^n)$

> **Answer:** A. n passes of an inner loop of n steps gives n x n steps.

6. What does this finite state machine print? [1 mark]

```
t = {("even", "0"): "even", ("even", "1"): "odd", ("odd", "0"): "odd", ("odd", "1"): "even"}
for text in ["110", "111", ""]:
    state = "even"
    for symbol in text:
        state = t[(state, symbol)]
    print(repr(text), state)
```

Answer:

```
'110' even
'111' odd
'' even
```

The machine tracks whether the number of 1s so far is even; the empty string stays in the start state.

The task: theory at speed

Write each method as a function, then print the answers. - `twos(bits)`: `bits` is a string of `0`s and `1`s of any length. Return its value as a two's complement integer: the leftmost bit is worth minus its normal place value. - `float_value(mantissa, exponent)`: both are strings of bits. The mantissa is two's complement with the binary point just after the leftmost bit; the exponent is a two's complement integer. Return the number's value as a `float`. - `run_fsm(transitions, state, accepting, text)`: `transitions` is a dictionary from `(state, symbol)` tuples to the next state, `state` is the start state, `accepting` is a set of accepting states, and `text` is a string of symbols. Return `True` if the machine ends in an accepting state. Then print exactly these six lines, using the functions for the values: 1. `11101100 = <value>` using `twos`; 2. `01101000 x 2^0011 = <value>` and 3. `10100000 x 2^1111 = <value>` using `float_value`; 4. `1101: <result>` and 5. `1001: <result>`, where the result is `accepted` or `rejected`, using `run_fsm` with the even-number-of-1s machine above (states `"even"` and `"odd"`, start `"even"`, accepting `{"even"}`); 6. `true rows: <n>`, counting the rows where `NOT (A AND B) OR C` is true with three nested loops over A, B and C. The robot stays still.

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

def twos(bits):
    pass
```

The hint students can ask for: For two's complement, the leftmost bit is worth minus the place value it would normally have. The mantissa has its binary point just after the sign bit, so its value is the two's complement integer divided by the right power of two. For the FSM, look up each (state, symbol) pair in turn and check where you end.

A solution

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

def twos(bits):
    value = int(bits, 2)
    if bits[0] == "1":
        value -= 2 ** len(bits)
    return value

def mantissa_value(bits):
    return twos(bits) / 2 ** (len(bits) - 1)

def float_value(mantissa, exponent):
    return mantissa_value(mantissa) * 2 ** twos(exponent)

def run_fsm(transitions, state, accepting, text):
    for symbol in text:
        state = transitions[(state, symbol)]
    return state in accepting

print("11101100 =", twos("11101100"))
for m, e in [("01101000", "0011"), ("10100000", "1111")]:
    print(f"{m} x 2^{e} =", float_value(m, e))

even_ones = {("even", "0"): "even", ("even", "1"): "odd", ("odd", "0"): "odd", ("odd",
```

```
"1"): "even"}
for text in ["1101", "1001"]:
    print(f"{text}:", "accepted" if run_fsm(even_ones, "even", {"even"}, text) else
"rejected")

count = 0
for a in (0, 1):
    for b in (0, 1):
        for c in (0, 1):
            if (not (a and b)) or c:
                count += 1
print("true rows:", count)
```

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