ANSWERS

A7.5 Bitwise operations and characters

Data representation · A level · OCR H446 1.4.1, AQA 7517 4.5.5.1, Eduqas A500QS 2.3 · about 30 min

What this lesson is about

Masks with AND, OR and XOR, logical, arithmetic and circular shifts, and characters as codes in ASCII and Unicode.

- AND, OR and XOR on whole bytes
- Shifts
- Characters
- A digit as a character and as a number

Questions

6 marks in all

1. What is 10110110 AND 00001111? [1 mark]

> **Answer:** 00000110. AND keeps a bit only where both bytes have a 1, so the mask keeps the low four bits, 0110, and clears the rest.

2. Which operation and mask toggles (flips) the lowest four bits of a byte and leaves the rest unchanged? [1 mark]

- ☐ **A** XOR with 00001111
- ☐ **B** AND with 00001111
- ☐ **C** OR with 00001111
- ☐ **D** AND with 11110000

> **Answer:** A. XOR with 1 flips a bit and XOR with 0 leaves it alone. AND with 0 clears and OR with 1 sets.

3. Give the result of an arithmetic shift right by one place of the 8-bit two's complement number 11110000. [1 mark]

> **Answer:** 11111000. An arithmetic shift copies the sign bit in on the left, so -16 becomes -8.

4. What does this program print? [1 mark]

```
print(0b10110110 >> 2, 0b00010110 << 1)
```

> **Answer:**
>
> ```
> 45 44
> ```
>
> 10110110 (182) shifted right 2 is 101101 (45). 00010110 (22) shifted left 1 is 00101100 (44).

5. The ASCII code for the character 0 is 48. What is the denary ASCII code for the character 7? [1 mark]

> **Answer:** 55. The digit characters are in order, so 7 is 48 + 7 = 55. That differs from the pure binary value of 7, which is 00000111.

6. Why was Unicode introduced? [1 mark]

- ☐ **A** ASCII could not represent the characters of most of the world's languages
- ☐ **B** ASCII codes were too long
- ☐ **C** Unicode uses fewer bits for every character
- ☐ **D** ASCII could not store digits

> **Answer:** A. 7-bit ASCII has only 128 codes, enough for English. Unicode gives codes to characters from all writing systems, plus symbols such as emoji.

The task: unpack a colour

`colour` holds `0xFD20`, a 16-bit RGB565 colour: bits 15 to 11 are red, bits 10 to 5 green, bits 4 to 0 blue. Using only shifts (`>>`, `<<`) and masks (`&`, `|`): 1. Extract the fields into `r5` (0 to 31), `g6` (0 to 63) and `b5` (0 to 31) and print `r5=<r5> g6=<g6> b5=<b5>`. 2. Widen each to 8 bits (0 to 255): `r8` is `r5` shifted left 3, ORed with `r5` shifted right 2; `g8` is `g6` shifted left 2, ORed with `g6` shifted right 4; `b8` is made like `r8`. Print `rgb=<r8>,<g8>,<b8>` with no spaces. 3. Light the LED with `led(r8, g8, b8)`. Do not use `//`, `%`, `bin` or `format`, and do not type the answers.

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

colour = 0xFD20

r5 = 0
g6 = 0
b5 = 0
print("r5=" + str(r5), "g6=" + str(g6), "b5=" + str(b5))
```

The hint students can ask for: Shift the 16-bit value right until the field you want sits at the bottom, then AND it with a mask of that field's width. To widen a field, shift it left to fill the new width and OR in its own top bits to fill the gap at the bottom.

A solution

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

colour = 0xFD20

r5 = (colour >> 11) & 0x1F
g6 = (colour >> 5) & 0x3F
b5 = colour & 0x1F
print("r5=" + str(r5), "g6=" + str(g6), "b5=" + str(b5))

r8 = (r5 << 3) | (r5 >> 2)
g8 = (g6 << 2) | (g6 >> 4)
b8 = (b5 << 3) | (b5 >> 2)
print("rgb=" + str(r8) + "," + str(g8) + "," + str(b8))
led(r8, g8, b8)
```

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