ANSWERS

A9.6 AQA assembly language and bitwise operations

Computer architecture · A level · OCR H446 1.2.4, AQA 7517 4.7.3.5 · about 25 min

What this lesson is about

AQA's instruction set, compare and branch, masks and shifts, and the bit fields that drive BugBot's motors.

- The instruction set
- Selection
- Iteration
- Running AQA assembly in Python
- What the bitwise operations are for
- Bits that drive motors

Questions

6 marks in all

1. What is the difference between MOV R1, #50 and LDR R1, 50? [1 mark]

- ☐ **A** MOV puts the number 50 in R1; LDR loads the value stored at address 50
- ☐ **B** They do the same thing
- ☐ **C** MOV loads from address 50; LDR puts the number 50 in R1
- ☐ **D** LDR moves R1 to address 50

> **Answer:** A. # means immediate; a plain number in LDR is a direct memory address.

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

```
r0 = 0b11001010
print(format(r0 & 0b00001111, '08b'))
print(format(r0 | 0b00000001, '08b'))
print(format(r0 >> 3, '08b'))
```

> **Answer:**
>
> ```
> 00001010
> 11001011
> 00011001
> ```
>
> AND keeps the bottom four bits, OR sets bit 0, and a right shift by 3 moves every bit three places down.

3. R0 holds 13. What does R1 hold after LSL R1, R0, #2? [1 mark]

> **Answer:** 52. Shifting left by 2 places multiplies by 4.

4. Which instruction toggles (flips) the lowest four bits of R2, leaving the others alone? [1 mark]

- ☐ **A** EOR R2, R2, #15
- ☐ **B** AND R2, R2, #15
- ☐ **C** ORR R2, R2, #15
- ☐ **D** MVN R2, R2

> **Answer:** A. XOR with 1 flips a bit; XOR with 0 leaves it unchanged.

5. A program does CMP R0, #10 then BLT small. When does it branch? [1 mark]

- ☐ **A** When the value in R0 is less than 10
- ☐ **B** When 10 is less than the value in R0
- ☐ **C** When R0 holds 10
- ☐ **D** Always

> **Answer:** A. CMP compares the register with operand2; BLT branches if the register was less.

6. Why does an if-else in AQA assembly need an unconditional B after the 'if' part? [1 mark]

- ☐ **A** Otherwise the processor falls through into the else part as well
- ☐ **B** B stops the program
- ☐ **C** Branch instructions must come in pairs
- ☐ **D** It resets the flags

> **Answer:** A. Instructions run in order unless the PC is changed, so the else code must be skipped explicitly.

The task: decode the control bytes

`commands` holds four motor control bytes, each a whole number from 0 to 255 laid out as in the table above. For each byte, use bitwise operators (`>>` and `&`) to split it into its speed (0 to 63) and its mode, and: - print a line in the form `161: speed 40 forward`: the byte in denary, the speed, then the mode as `coast`, `forward`, `reverse` or `brake`; - then act on it: `forward` drives forward 10 cm at that speed, `reverse` drives backward 10 cm at that speed, `brake` calls `stop()`, and `coast` does nothing. Do not convert the byte to a string of bits.

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

commands = [161, 98, 3, 0]
modes = ["coast", "forward", "reverse", "brake"]
```

The hint students can ask for: Draw the byte as eight boxes. Work out which way and how far to shift so only the speed bits are left, and which mask keeps only the bottom two bits. The mode number can then choose a word from the list and decide what the robot does.

A solution

```
from bugbot import *
connect()
commands = [161, 98, 3, 0]
modes = ["coast", "forward", "reverse", "brake"]
for byte in commands:
    speed = byte >> 2
    mode = modes[byte & 0b11]
    print(f"{byte}: speed {speed} {mode}")
    if mode == "forward":
        forward(speed, distance=10)
    elif mode == "reverse":
        backward(speed, distance=10)
    elif mode == "brake":
        stop()
```

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