ANSWERS

A9.2 The processor and its registers

Computer architecture · A level · OCR H446 1.1.1, AQA 7517 4.7.3.1, Eduqas A500QS 2.1 · about 20 min

What this lesson is about

The ALU, control unit and clock, general-purpose and dedicated registers, and the status flags an 8-bit ALU sets.

- The components of the processor
- General-purpose and dedicated registers
- The status register's flags
- How flags make decisions
- Registers, cache and memory

Questions

5 marks in all

1. Which register holds the instruction currently being decoded and executed? [1 mark]

- ☐ **A** Current instruction register
- ☐ **B** Program counter
- ☐ **C** Memory address register
- ☐ **D** Status register

> **Answer:** A. The CIR keeps the instruction while the MDR is reused for data.

2. What does the control unit do? [1 mark]

- ☐ **A** Decodes instructions and sends control signals to coordinate the other components
- ☐ **B** Performs arithmetic and logic operations
- ☐ **C** Stores the result of calculations
- ☐ **D** Generates the clock pulses

> **Answer:** A. The ALU calculates; the control unit decodes and directs.

3. An 8-bit ALU adds 200 and 100. Which flag is set? [1 mark]

- ☐ **A** Carry
- ☐ **B** Zero
- ☐ **C** Negative
- ☐ **D** Overflow

> **Answer:** A. 300 does not fit in 8 unsigned bits, so C = 1 and the stored result is 44, which is positive and not zero.

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

```
def flags(a, b):
    r = (a + b) & 255
    return r, r >> 7, int(r == 0)

print(flags(100, 50))
print(flags(128, 128))
```

Answer:

```
(150, 1, 0)
(0, 0, 1)
```

150 has its top bit set, so N is 1; 128 + 128 wraps round to 0, so Z is 1.

5. What is the difference between a general-purpose register and a dedicated register? [1 mark]

☐ **A** A general-purpose register holds whatever the program chooses; a dedicated register has one fixed job, such as the PC

☐ **B** General-purpose registers are in main memory; dedicated ones are in the processor

☐ **C** Dedicated registers can only hold addresses

☐ **D** There is no difference

Answer: A. R0 to R12 in AQA's assembly are general-purpose; the PC, CIR, MAR, MBR and SR are dedicated.

The task: the flags

Write `alu_add(a, b)` yourself for an 8-bit ALU. Its inputs `a` and `b` are whole numbers from 0 to 255 (bytes). It works out the 8-bit result (0 to 255, anything that does not fit is lost) and the four flags N, Z, C and V, each 0 or 1, exactly as the table above defines them. For each pair in `pairs`, print one line in the form `200 + 100 = 44 N=0 Z=0 C=1 V=0`: the two inputs, the 8-bit result, then the flags in the order N, Z, C, V. The robot does not move.

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

pairs = [(20, 30), (100, 50), (200, 100), (128, 128), (255, 1)]

def alu_add(a, b):
    return a + b
```

The hint students can ask for: Work out the full sum first: carry is whether it went past what 8 bits can hold, and the stored result is only the part that fits. For overflow, read both inputs and the result as two's complement and ask whether two numbers of the same sign gave a result of the other sign.

A solution

```
from bugbot import *
connect()
pairs = [(20, 30), (100, 50), (200, 100), (128, 128), (255, 1)]

def signed(byte):
    return byte - 256 if byte >= 128 else byte

def alu_add(a, b):
    total = a + b
    result = total & 255
    n = result >> 7
    z = 1 if result == 0 else 0
    c = 1 if total > 255 else 0
    v = 1 if (signed(a) < 0) == (signed(b) < 0) and (signed(result) < 0) != (signed(a) < 0)
else 0
    return result, n, z, c, v

for a, b in pairs:
    result, n, z, c, v = alu_add(a, b)
    print(f"{a} + {b} = {result} N={n} Z={z} C={c} V={v}")
```

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