Exercise walk-through

Bit manipulation ▪ 4.3

eXercise

You must be able to

- perform **logical binary shifts** (left and right) and state their effect on the value
- use a **mask** 掩码 to **set**, **clear**, **toggle** and **test** a single bit
- write the **assembly instructions** that do this: `AND`, `OR`, `XOR`, `LSL`, `LSR`
- read the three operand forms: `#` denary, `B` binary, `&` hexadecimal

Method — Binary shifts

A **logical shift** moves every bit left or right, filling the new positions with 0.

- **left shift** by $n \rightarrow \times\ 2^n$ (for an unsigned number).
- **right shift** by $n \rightarrow$ **integer** $\div\ 2^n$.
- An **arithmetic** right shift keeps the **sign bit**, so a negative signed number stays negative.
- A **cyclic shift** 循环移位 (rotate) feeds the bit that falls off one end back in at the other end, so no bits are lost.

Method — Binary shifts

$= 11$

LSL #1: each bit moves left, a 0 enters on the right

$= 22 \quad (11 \times 2)$

Method — Binary shifts

set bit 2

```
  01001000
OR 00000100
------------
=  01001100
```

clear bit 6

```
  01001000
AND 10111111
------------
=  00001000
```

toggle bit 3

```
  01001000
XOR 00001000
------------
=  01000000
```

Bit masking with AND/OR isolates or sets chosen bits

Method — Bit masks (monitor and control)

A device often uses **one bit per signal** (bit n = LED n). Combine the register with a **mask** using a logic operation:

Goal	**Operation**	**Mask (for bit n)**
set bit n to 1	`R OR mask`	bit $n = 1$, rest 0
clear bit n to 0	`R AND mask`	bit $n = 0$, rest 1
toggle bit n	`R XOR mask`	bit $n = 1$, rest 0
test bit n	`R AND mask`, then check $\neq 0$	bit $n = 1$, rest 0

Example — turn on bit 0 of `00000100` without changing the rest: `00000100 OR 00000001 = 00000101`.

Method — Writing it as an assembly instruction

The exam asks for the **instruction**, not just the mask. Every one of these works on the **accumulator**, ACC, and there is only one general-purpose register:

Instruction	**What it does to ACC**
`AND #n` / `AND Bn` / `AND &n`	bitwise AND of ACC with the operand
`AND <address>`	bitwise AND of ACC with the contents of that address
`OR` and `XOR`	the same two forms, with OR and XOR
`LSL #n`	shift ACC **left** *n* places, zeros in on the right
`LSR #n`	shift ACC **right** *n* places, zeros in on the left

Method — Writing it as an assembly instruction

The operand prefix says which base the number is written in:

- `#123` — a **denary** number;
- `B01001010` — a **binary** number;
- `&4A` — a **hexadecimal** number.

So "set the least significant bit of `ACC` to 1" is `OR B00000001` (or `OR #1`, or `OR &01` — all the same instruction with the operand written three ways). "Clear bit 3" is `AND B11110111`. "Test bit 3" is `AND B00001000`, then check whether `ACC` is zero.

A `<label>:` in front of an instruction names it so a jump can reach it, and a `<label>: <data>` line gives a symbolic address to a memory location holding that data.

Practice 2.1 ▪ 2 marks

Perform a **logical shift left by 1** on `00000101`, and give the denary value before and after.

Solution 2.1

Method: Binary shifts

Worked steps

00000101 (5) $\rightarrow$ 00001010 M1; denary before **5**, after **10** A1.

Practice 2.2 ▪ 1 mark

State the arithmetic effect of a **logical shift right by 1** on an unsigned number.

Solution 2.2

Method: Binary shifts

Worked steps

It **halves** the number (integer $\div$ 2) B1.

Practice 2.3 ▪ 2 marks

Give the mask and the operation needed to **set bit 2** of a register to 1, leaving the other bits unchanged.

Solution 2.3

Method: Bit masks (monitor and control)

Worked steps

Mask `00000100`, operation **OR**: `R OR 00000100` M1 A1 *mask, OR.*

Practice 2.4 ▪ 1 mark

A logical shift **left by 3** multiplies an unsigned number by what?

Solution 2.4

Method: Binary shifts

Worked steps

By **8** (that is 2^3) B1.

Exam-style 3.1 ▪ 2 marks

(a) Perform a logical shift **left by 2** on `00000110`, and give the denary result.
(b) State the effect this shift has on the value.

Solution 3.1

Method: Binary shifts

Worked steps

(a) 00000110 $\rightarrow$ 00011000 M1; denary **24** A1.

(b) It **multiplies the value by 4** B1.

Exam-style 3.2 ▪ 2 marks

An 8-bit register controls 8 LEDs (bit n = LED n).
(a) Give the mask and operation to turn **on only LED 0**, leaving the others unchanged.
(b) Give the mask and operation to **test whether LED 3 is on**.

Solution 3.2

Worked steps

(a) Mask `00000001`, operation **OR** (`R OR 00000001`) B1 B1.
(b) Mask `00001000`, operation **AND** (`R AND 00001000`); if the result is non-zero, LED 3 is on B1 B1.

Exam-style 3.3 ▪ 2 marks

Explain why an **arithmetic** right shift, not a logical one, is used when halving a **negative** signed number.

Solution 3.3

Method: Binary shifts

Worked steps

A logical shift puts a **0** into the top bit, which would make a negative number look positive B1; an arithmetic shift **copies the sign bit**, so the number stays negative B1.

Exam-style 3.4 ▪ 2 marks

State the difference between a **logical** shift and a **cyclic** shift.

Solution 3.4

Method: Binary shifts

Worked steps

A logical shift fills the vacated end with **0** and the bit shifted off the other end is **lost**; a cyclic shift feeds that bit **back in** at the other end, so no bits are lost B1 B1.

Exam-style 3.5 ▪ 2 marks

A microcontroller uses an 8-bit accumulator `ACC` to control a set of sensors and indicator lamps. Bit 0 is the least significant bit.

(a) `ACC` currently contains `01101100`. **Write** the single bit-manipulation instruction that **sets the least significant bit to 1** without changing any other bit, and **give** the resulting contents of `ACC`.

(b) **Write** the single instruction that **clears bit 6 to 0**, leaving the other bits unchanged, using a **binary** operand.

(c) **Write** the instruction that would **test whether bit 2 is set**, and **describe** how the program then decides the answer.

(d) **Complete** the table by giving the contents of `ACC` after each instruction. Each row starts from the value shown in the row above.

instruction	**contents of `ACC`**
starting value	`10011110`
`LSR #3`	_____

Exam-style 3.5 ▪ 2 marks

(e) The value 10011110 is now treated as a **two's complement** integer. **Show** the result of an **arithmetic** right shift of 3 places on it, and **explain** why the result differs from your answer in (d).
(f) **Complete** the binary addition below, showing your working and any **overflow** bit. **State** whether the answer is valid if the values are unsigned 8-bit integers.

$$10110101 + 01011010$$

Solution 3.5

Method: Writing it as an assembly instruction

Worked steps

(a) `OR B00000001` (or `OR #1`, or `OR &01`) B1; ACC becomes `01101101` B1.
(b) `AND B10111111` B1 — the mask is 1 everywhere **except** bit 6, so every other bit survives the AND unchanged B1.
(c) `AND B00000100` B1. The AND leaves every other bit zero, so ACC is **non-zero** only if bit 2 was 1 B1; the program then branches on whether ACC is zero B1.
(d) `LSR #3` on `10011110` gives `00010011` B1; `XOR B00001111` flips the low four bits, giving `00011100` B1; `LSL #2` gives `01110000` B1.
(e) An arithmetic shift copies the **sign bit** into the vacated places, so `10011110`

Solution 3.5

becomes 11110011 M1 A1. The logical shift in (d) put zeros in, turning a negative number positive; the arithmetic shift keeps it negative, which is what halving a negative number must do B1.
(f) $10110101 + 01011010 = 100001111$ — nine bits M1. The 8-bit answer is 00001111 with an **overflow** bit of 1 A1. As unsigned 8-bit integers the result is **not valid**: $181 + 90 = 271$, which will not fit in 8 bits B1.