

4.3 Bit manipulation

Name: _____ Class: _____ Date: _____

Total: **33** marks

KEY WORDS bit manipulation 位操作 • logical shift 逻辑移位 • mask 掩码
• cyclic shift 循环移位 • bit 位

Objective

Build the skills to answer exam questions on **bit manipulation** 位操作 — binary shifts and using bit masks to monitor and control a device.

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

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ Binary shifts

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

0	0	0	0	1	0	1	1
---	---	---	---	---	---	---	---

 = 11

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

0	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---

 = 22 (11 × 2)

set bit 2	clear bit 6	toggle bit 3
01001000	01001000	01001000
OR 00000100	AND 10111111	XOR 00001000
= 01001100	= 00001000	= 01000000

Bit masking with AND/OR isolates or sets chosen bits

- **left shift** by $n \rightarrow \times 2^n$ (for an unsigned number).

- **right shift** by $n \rightarrow \text{integer} \div 2^n$.
- 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.

■ Bit masks (monitor and control)

A device often uses **one bit per signal** (bit $n = \text{LED } n$). 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.

■ 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 B n / 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

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.

2 Practice

Now apply the methods above.

2.1 Perform a **logical shift left by 1** on 00000101, and give the denary value before and after. [2]

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

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

2.4 A logical shift **left by 3** multiplies an unsigned number by what? [1]

3 Exam-style questions

3.1 (a) Perform a logical shift **left by 2** on 00000110, and give the denary result. [2]

(b) State the effect this shift has on the value. [1]

3.2 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. [2]

(b) Give the mask and operation to **test whether LED 3 is on**. [2]

3.3 Explain why an **arithmetic** right shift, not a logical one, is used when halving a **negative** signed number. [2]

3.4 State the difference between a **logical** shift and a **cyclic** shift. [2]

3.5 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**. [2]

(b) **Write** the single instruction that **clears bit 6 to 0**, leaving the other bits unchanged, using a **binary** operand. [2]

(c) **Write** the instruction that would **test whether bit 2 is set**, and **describe** how the program then decides the answer. [3]

(d) **Complete** the table by giving the contents of **ACC** after each instruction. Each row starts from the value shown in the row above. [3]

instruction	contents of ACC
starting value	10011110
LSR #3	_____
XOR B00001111	_____
LSL #2	_____

(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). [3]

(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. [3]

$$10110101 + 01011010$$