

4.3.1 Arithmetic and cyclic shifts, and when a shift stops being a multiply or a divide

Name: _____ Class: _____ Date: _____ **Total: 29 marks**

KEY WORDS logical shift 逻辑移位 • arithmetic 算术移位 • binary shift 二进制移位
• cyclic 循环移位 • mask 掩码

Objective

Build the skills to answer the **binary shift** 二进制移位 questions that sheet 4.3 does not reach. That sheet performs **logical** shifts and uses a **mask** 掩码 to set, clear, toggle and test a bit; this one is about the other two shifts the syllabus names and that sheet only mentions – the **arithmetic** 算术移位 and the **cyclic** 循环移位 shift – what each one does to the **value**, and when a shift stops being a multiplication or a division.

You must be able to:

- perform a **logical**, an **arithmetic** and a **cyclic** shift, left and right, by any number of places
- state the effect of each shift on the value, and say which reading of the bits it is correct for
- say when a shift is no longer a multiplication or a division, and why
- extract a field from a register by combining a shift with a mask

1 Worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

■ The three shifts, on one byte

A shift question gives you a bit pattern and a number of places. What differs between the three kinds is only **what enters the vacated end**.

Shift	What enters	What leaves
logical (LSL, LSR)	a 0 , at either end	falls off and is lost
arithmetic right (ASR)	a copy of the sign bit	falls off and is lost
cyclic (rotate)	the bit that just left the other end	comes back round – nothing is lost

Worked example. *Shift 10110100 by two places.*

	Result	Unsigned	Signed
LSL #2	11010000	208	–48
LSR #2	00101101	45	45
ASR #2	11101101	237	–19

- Only the **arithmetic** right shift keeps the number negative. 10110100 is –76 signed, and $-76 \div 4 = -19$, which is what **ASR #2** gives.

■ Why there are two right shifts

start

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

 = 240 unsigned, –16 signed

LSR #1

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

 = 120 halves the **unsigned** value

ASR #1

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

 = –8 halves the **signed** value

The two results differ in **one bit** — the one that entered on the left. **LSR** always brings in a 0; **ASR** copies the sign bit, so a negative number stays negative and $-16 \div 2$ comes out right.

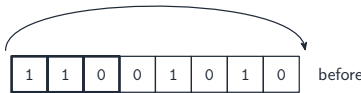
The same byte reads as two different numbers, and each right shift halves one of them.

11110000 is **240** read as unsigned and **–16** read as two's complement.

- LSR #1** brings in a 0: $01111000 = 120$, which is half of 240.
- ASR #1** copies the sign bit: $11111000 = -8$, which is half of –16.
- Neither is wrong.** The question tells you which reading applies, and that chooses the shift. Signed data takes the arithmetic shift; unsigned data takes the logical one.

■ The cyclic shift loses nothing

out of the left-hand end, back in at the right-hand end



after a cyclic left shift of 3

A cyclic left shift of 3 on 11001010 takes the leading 110 off the left and puts it back on the right, giving 01010110.

- Because nothing is lost, a cyclic shift is **reversible**: a cyclic shift of 8 places on an 8-bit register returns the original pattern exactly.
- That is what it is used for: rotate a register one place at a time, testing the same single bit each time, and you can inspect all 8 bits without ever destroying the data.
- A cyclic shift is **not** a multiplication or a division. 11001010 is 202, and the rotated 01010110 is 86.

■ **A shift is only a multiply or a divide while nothing important falls off**

- LSL #n multiplies an unsigned value by 2^n – but only while the bits leaving the left are **zeros**. 00001101 is 13, and LSL #3 gives 01101000 = 104 = 13×8 , correct because only 0s left.
- The same shift on 10110100 (180) should give $180 \times 4 = 720$, which needs ten bits. LSL #2 gives 11010000 = 208: the two 1s that fell off the left are gone, the sign bit has changed, and the answer is simply wrong. **Overflow**.
- ASR #n divides a signed value by 2^n , but it rounds **down**, towards the more negative number, not towards zero. -9 is 11110111; ASR #1 gives 1111011 = -5, not -4.

■ **Shift and mask together**

To pull one **field** out of a register, shift it down to the bottom and mask off what is left.

With ACC holding 10110100, where the top four bits are a sensor number and the bottom four a reading:

- the **sensor number**: LSR #4 gives 00001011 = 11. No mask is needed, because a logical shift brings 0s in on the left.
- the **reading**: AND B00001111 gives 00000100 = 4. No shift is needed, because the field is already at the bottom.

2 Practice

Now use the methods above.

2.1 A register holds 10110100. Give the contents after each of these shifts. [3]

- LSL #2
- LSR #2
- ASR #2

2.2 Perform a cyclic left shift of 3 places on 11001010. [2]

2.3 A register holds 11110000.

Give the result of LSR #1 and of ASR #1, and state the denary value each one is the correct half of. [3]

2.4 State the contents of an 8-bit register after a cyclic shift of 8 places, and explain your answer. [1]

2.5 A register holds 00001101. Perform LSL #3 and state whether the result is still eight times the original. [2]

3 Exam-style questions

3.1 A register holds the 8-bit two's complement integer 10110100.

- (a) State its denary value. [1]

- (b) Give the contents of the register after **ASR #2**, and show that the result is the correct value. [2]

- (c) State why **LSR #2** would **not** give the correct answer here. [1]

3.2 A programmer halves a signed integer by performing an arithmetic right shift of one place.

The register holds 11110111.

- (a) State the denary value before and after the shift. [2]

- (b) The programmer expected -4 . Explain why the answer is different. [1]

3.3 A microcontroller must test each of the 8 bits of a register in turn, and must still hold the original value when it has finished.

- (a) Name the type of shift that makes this possible, and explain why the other two cannot be used. [3]

- (b) State how many shifts return the register to its starting contents. [1]

3.4 For each task, name the shift to use and give one reason. [3]

- (a) multiply an unsigned counter by 4
(b) halve a signed temperature reading that may be negative
(c) move every bit one place left without losing the bit at the far end

3.5 An 8-bit accumulator holds 10110100. The top four bits are a sensor number and the bottom four bits are that sensor's reading.

Write the instruction, or instructions, that leave **only** the sensor number in the accumulator, and state the denary value that results. Then do the same for the reading. [4]

Solutions

Each answer shows the steps, then the **result**.

2.1

- (a) LSL #2 = 11010000 – 0s enter on the right, the top two bits are lost
- (b) LSR #2 = 00101101 – 0s enter on the left
- (c) ASR #2 = 11101101 – the sign bit 1 is copied into both vacated places

2.2

- the leading 110 leaves the left and re-enters on the right
- 01010110

2.3

- LSR #1 = 01111000 = 120
- ASR #1 = 11111000 = -8
- 11110000 is 240 unsigned and -16 signed: the logical shift halves **240** to 120, the arithmetic shift halves **-16** to -8

2.4

- it is **unchanged** – every bit has been round the whole register once and is back where it started

2.5

- LSL #3 = 01101000 = 104
- yes: the original is 13 and $13 \times 8 = 104$; only 0s fell off the left, so nothing was lost

3.1

(a) the sign bit is 1: invert 10110100 to 01001011, add 1 to get 01001100 = 76, so the value is **-76**

(b) ASR #2 = 11101101

- read it back: invert to 00010010, add 1 to get 00010011 = 19, so the result is -19, and $-76 \div 4 = -19$

(c) LSR #2 brings in **0s**, so the sign bit becomes 0 and the number becomes positive (00101101 = 45); the value is no longer negative, so it is not a quarter of -76

3.2

(a) before: the sign bit is 1, so invert 11110111 to 00001000, add 1 to get 00001001 = 9, giving **-9**

- after ASR #1: 11111011, which reads back as **-5**

(b) an arithmetic right shift rounds **down**, towards the more negative number, not towards zero: $-9 \div 2 = -4.5$, and rounding down gives **-5**

3.3

(a) a **cyclic** shift (a rotate)

- a cyclic shift loses no bits: each one that leaves an end comes back at the other end, so after the test the original value can be restored
- a logical shift fills the vacated end with 0s and an arithmetic shift with copies of the sign bit, and in both the bits that fall off the end are lost, so the original value is destroyed

(b) **8** shifts

3.4

(a) a **logical left shift of 2** – shifting left n places multiplies an unsigned value by 2^n , and $2^2 = 4$

(b) an **arithmetic right shift** – it copies the sign bit into the vacated place, so a negative reading stays negative; a logical right shift would bring in a 0 and make it positive

(c) a **cyclic left shift** – the bit leaving the left re-enters on the right instead of being lost

3.5

- the sensor number: **LSR #4**
- this gives 00001011 = **11**; no mask is needed because a logical shift brings 0s in on the left
- the reading: **AND B00001111** (also accepted: **AND &0F**, or **AND #15**)
- this gives 00000100 = **4**; no shift is needed because that field is already at the bottom of the register