Exercise walk-through

4.3.1

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

A-Level Computer Science

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 The method

Method — 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**

Method — The three shifts, on one byte

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

Method — The three shifts, on one byte

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

Method — Why there are two right shifts

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.

Method — Why there are two right shifts

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

$= 240$ unsigned, -16 signed

$= 120$ halves the **unsigned** value

$= -8$ 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.

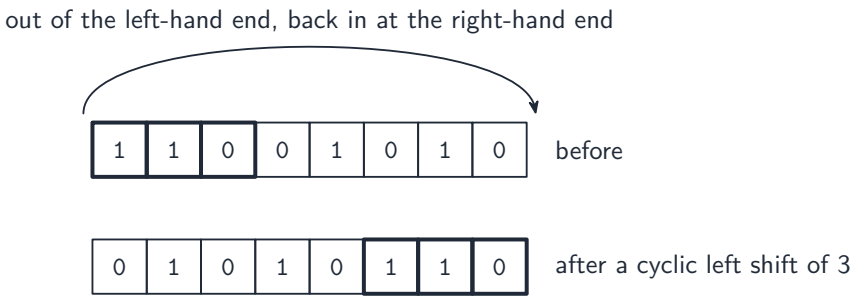
Method — The cyclic shift loses nothing

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.

Method — The cyclic shift loses nothing

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



Method — 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` $= 104 = 13 \times 8$, 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 `11111011` $= -5$, not -4.

Method — 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

Practice 2.1 ▪ 3 marks

A register holds `10110100`. Give the contents after each of these shifts.

Practice 2.1 (a)

```
LSL #2
```

Solution 2.1 (a)

Method: The cyclic shift loses nothing — p.9

Worked steps

`LSL #2` = `11010000` – 0s enter on the right, the top two bits are lost B1

Practice 2.1 (b)

```
LSR #2
```

Solution 2.1 (b)

Worked steps

`LSR #2` = `00101101` – 0s enter on the left B1

Practice 2.1 (c)

```
ASR #2
```

Solution 2.1 (c)

Worked steps

`ASR #2` = `11101101` – the sign bit 1 is copied into both vacated places B1

Practice 2.2 ▪ 2 marks

Perform a **cyclic left shift of 3 places** on 11001010.

Solution 2.2

Worked steps

- the leading 110 leaves the left and re-enters on the right M1
- 01010110 A1

Practice 2.3 ▪ 3 marks

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.

Solution 2.3

Worked steps

- `LSR #1` $= \texttt{01111000} = 120$ **B1**
- `ASR #1` $= \texttt{11111000} = -8$ **B1**
- 11110000 is 240 unsigned and -16 signed: the logical shift halves **240** to 120, the arithmetic shift halves $-\mathbf{16}$ to -8 **B1**

Practice 2.4 ▪ 1 mark

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

Solution 2.4

Method: The cyclic shift loses nothing — p.9

Worked steps

- it is **unchanged** – every bit has been round the whole register once and is back where it started B1

Practice 2.5 ▪ 2 marks

A register holds `00001101`. Perform `LSL #3` and state whether the result is still eight times the original.

Solution 2.5

Method: The cyclic shift loses nothing — p.9

Worked steps

- `LSL #3` $= \texttt{01101000} = 104$ B1
- yes: the original is 13 and $13 \times 8 = 104$; only 0s fell off the left, so nothing was lost B1

3 Exam-style

Exam-style 3.1 ▪ 4 marks

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

Exam-style 3.1 (a) ▪ 1 mark

State its denary value.

Solution 3.1 (a)

Method: The cyclic shift loses nothing — p.9

Worked steps

the sign bit is 1: invert 10110100 to 01001011, add 1 to get $01001100 = 76$, so the value is $-\mathbf{76}$ B1

Exam-style 3.1 (b) ▪ 2 marks

Give the contents of the register after `ASR #2`, and show that the result is the correct value.

Solution 3.1 (b)

Worked steps

`ASR #2 = 11101101` B1

- read it back: invert to 00010010, add 1 to get $00010011 = 19$, so the result is -19, and $-76 \div 4 = -19$ B1

Exam-style 3.1 (c) ▪ 1 mark

State why `LSR #2` would **not** give the correct answer here.

Solution 3.1 (c)

Worked steps

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 B1

Exam-style 3.2 ▪ 3 marks

A programmer halves a signed integer by performing an arithmetic right shift of one place.
The register holds 11110111.

Exam-style 3.2 (a) ▪ 2 marks

State the denary value before and after the shift.

Solution 3.2 (a)

Method: Why there are two right shifts — p.7

Worked steps

before: the sign bit is 1, so invert 11110111 to 00001000, add 1 to get 00001001 $= 9$, giving $-\mathbf{9}$ B1

- after `ASR #1`: 11111011, which reads back as $-\mathbf{5}$ B1

Exam-style 3.2 (b) ▪ 1 mark

The programmer expected -4. Explain why the answer is different.

Solution 3.2 (b)

Worked steps

an arithmetic right shift rounds **down**, towards the more negative number, not towards zero: $-9 \div 2 = -4.5$, and rounding down gives -5 B1

Exam-style 3.3 ▪ 4 marks

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.

Exam-style 3.3 (a) ▪ 3 marks

Name the type of shift that makes this possible, and explain why the other two cannot be used.

Solution 3.3 (a)

Method: The cyclic shift loses nothing — p.9

Worked steps

a **cyclic** shift (a rotate) B1

- 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 B1
- 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 B1

Exam-style 3.3 (b) ▪ 1 mark

State how many shifts return the register to its starting contents.

Solution 3.3 (b)

Worked steps

8 shifts B1

Exam-style 3.4 ▪ 3 marks

For each task, name the shift to use and give one reason.

Exam-style 3.4 (a)

multiply an unsigned counter by 4

Solution 3.4 (a)

Worked steps

a **logical left shift of 2** – shifting left n places multiplies an unsigned value by 2^n, and $2^2 = 4$ B1

Exam-style 3.4 (b)

halve a signed temperature reading that may be negative

Solution 3.4 (b)

Worked steps

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 B1

Exam-style 3.4 (c)

move every bit one place left without losing the bit at the far end

Solution 3.4 (c)

Worked steps

a **cyclic left shift** – the bit leaving the left re-enters on the right instead of being lost B1

Exam-style 3.5 ▪ 4 marks

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.

Solution 3.5

Method: Shift and mask together — p.12

Worked steps

- the sensor number: `LSR #4` M1
- this gives 00001011 = **11**; no mask is needed because a logical shift brings 0s in on the left A1
- the reading: `AND B00001111` (also accepted: `AND &0F`, or `AND #15`) M1
- this gives 00000100 = **4**; no shift is needed because that field is already at the bottom of the register A1