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