Chapter 5 · Section 5.2 — Exercises (Mazidi)

Chapter 5 · Section 5.2 — Exercises (Mazidi)

> Problems are paraphrased to respect copyright. Byte arithmetic means **8-bit two's-complement** (range −128…+127).

---

3) Find the overflow flag (V) for each; do byte-sized calculations

Rule of thumb (ADD): **same sign in, different sign out ⇒ V=1**; otherwise **V=0**.

item	operation (8-bit)	numeric sum	8-bit result	V
(a)	(+15) + (−12)	+3	0x03	**0**
(b)	(−123) + (−127)	−250	0x06 (wrap)	**1**
(c)	(+0x25) + (+34)	+71	0x47	**0**
(d)	(−127) + (+127)	0	0x00	**0**
(e)	(+100) + (−100)	0	0x00	**0**

Notes: In (b) both operands are **negative** yet the 8-bit result has a **positive sign bit (0)** → overflow.

---

4) Sign-extend the following to 32 bits and show a tiny program to verify

Assumptions on source widths: decimal values within |N|≤128 are treated as **8-bit**; `0x999` is treated as **12-bit**; `-129` is shown for **16-bit** (also give optional 9-bit note).

item	source width	original value	32-bit sign-extended
(a) −122	8-bit	0x86	**0xFFFFFF86**
(b) −0x999	12-bit	0x999	**0xFFFFF999**
(c) +0x17	8-bit	0x17	**0x00000017**
(d) +127	8-bit	0x7F	**0x0000007F**
(e) −129	16-bit	0xFF7F	**0xFFFFFF7F**

*If you instead assume a 9-bit source for (e), `0x017F` → `0xFFFFFE7F`, still representing **−129**.*

Verification snippets (Thumb):

- 8-bit to 32-bit (use `SXTB`), example for **−122**:

```
THUMB
MOVS    r0,#0x86        ; 8-bit pattern for -122
SXTB    r1,r0           ; r1 = 0xFFFFFF86
```

- 12-bit to 32-bit (generic `LSL`/`ASR`), example for **0x999**:

```
LDR     r0,=0x00000999  ; treat as 12-bit signed
LSL     r0,r0,#20       ; move sign bit to bit31
ASR     r0,r0,#20       ; arithmetic right shift back ⇒ 0xFFFFF999
```

- 16-bit to 32-bit (use `SXTH`), example for **−129**:

```
LDR     r0,=0xFF7F
SXTH    r1,r0           ; r1 = 0xFFFFFF7F (-129)
```

---

5) Modify Program 5-2 to find the highest temperature (signed bytes)

Assume an array of **N signed bytes** at `TEMPS` (e.g., −40…+125°C). We scan with signed loads and keep the **maximum**.

```
        AREA    |.text|, CODE, READONLY
        EXPORT  find_max_temp
        THUMB
TEMPS   EQU     0x20000000      ; array base
N       EQU     64              ; number of samples

find_max_temp:
        LDR     r0, =TEMPS
        LDR     r1, =N
        LDRSB   r2, [r0], #1    ; r2 = current max (first element), sign-extended
        SUBS    r1, r1, #1

.loop:
        CBZ     r1, .done
        LDRSB   r3, [r0], #1    ; signed load
        CMP     r3, r2          ; signed compare (works because both are 32-bit signed)
        BLE     .skip           ; if r3 <= r2 keep old max
        MOV     r2, r3          ; else update max
.skip:  SUBS    r1, r1, #1
        BNE     .loop
.done:  BX      lr              ; max in r2
        END
```

Why this works: `LDRSB` performs **sign extension** from byte to 32-bit; `CMP` and the conditional `BLE` use signed interpretation when comparing general registers, so we correctly track the **highest** signed temperature.

Notes for learners

- On ARM, `V` reflects **signed overflow**, while `C` reflects **unsigned carry/no-borrow**.
- Sign-extend with: `SXTB` (8→32), `SXTH` (16→32), or the **`LSL`+`ASR` trick** for arbitrary widths.