Section 2.4 — ARM CPSR (Current Program Status Register) (Mazidi)

Chapter 2 · Section 2.4 — Exercises (Mazidi)

> Problems are paraphrased to respect copyright. For theory and examples, see **Mazidi, Ch. 2 §2.4**.

---

30) The status register is a(n) ______-bit register.

Answer: 32-bit.
Why: The **CPSR** (Current Program Status Register) is 32 bits; the top four bits hold **N Z C V**.

---

31) Which bits of the status register are used for the C and Z flag bits, respectively?

Answer: C = bit 29, **Z = bit 30**.
Why: The high nibble of CPSR is `N(31) Z(30) C(29) V(28)`.

---

32) Which bits of the status register are used for the V and N flag bits, respectively?

Answer: V = bit 28, **N = bit 31**.

---

33) In the `ADD` instruction, when is C raised?

Answer: When there is an **unsigned carry out of bit 31** (i.e., the 32-bit addition exceeds `0xFFFFFFFF`).

---

34) In the `ADD` instruction, when is Z raised?

Answer: When the **result is zero** (all 32 bits are 0).

---

35) What is the status of the C and Z flags after the following code?

```
LDR   R0, =0xFFFFFFFF
LDR   R1, =0xFFFFFFF1
ADDS  R1, R0, R1
```

Answer: C = 1, **Z = 0**.
Why: `0xFFFFFFFF + 0xFFFFFFF1 = 0x1FFFFFFF0` → result (low 32 bits) `0xFFFFFFF0` (**non-zero**) with a **carry out** → `C=1`, `Z=0`.

---

36) Find the C flag value after each of the following codes.

(a)

```
LDR   R0, =0xFFFFFF54
LDR   R5, =0xFFFFFFC4
ADDS  R2, R5, R0
```

Answer: C = 1.
Why: Sum `0xFFFFFF54 + 0xFFFFFFC4 = 0x1FFFFFF18` → carry out.

(b)

```
MOVS  R3, #0
LDR   R6, =0xFFFFFFFF
ADDS  R3, R3, R6
```

Answer: C = 0.
Why: Sum `0 + 0xFFFFFFFF = 0xFFFFFFFF` → **no carry**.

(c)

```
LDR   R3, =0xFFFFFF??    ; (value shown near 0xFFFFFFxx in the problem)
LDR   R8, =0xFFFFFF05
ADDS  R2, R3, R8
```

Answer: C = 1 (for the given near-`0xFFFFFFxx` values).
Why: Adding two values both close to `0xFFFFFFFF` **exceeds 32 bits**, producing a carry out (**unsigned overflow**).

---

37) Write a simple program in which the value `0x55` is added 5 times.

Approach: Accumulate in a loop using `ADDS` so flags update; keep result in `R0`.

```
        AREA    |.text|, CODE, READONLY
        EXPORT  _start
        THUMB
_start:
        MOVS    r0, #0x00         ; accumulator
        MOVS    r1, #0x55         ; value to add
        MOVS    r2, #5            ; loop count

add_loop:
        ADDS    r0, r0, r1        ; r0 += 0x55
        SUBS    r2, r2, #1
        BNE     add_loop

        ; Result: r0 = 5 * 0x55 = 0x1A9 (low 32 bits), C set if a carry occurred
        B       .
        END
```

Explanation: The loop adds `0x55` five times (`0x1A9` total). Using `ADDS` updates flags each step; the final flags depend on intermediate carries/zero results (none here).

Notes for learners

- Remember the CPSR high bits order: **N(31) Z(30) C(29) V(28)**.
- Use the **S suffix** (e.g., `ADDS`, `SUBS`) to **update flags**.
- **Unsigned vs. signed: C** indicates **unsigned carry/borrow**; **V** indicates **signed overflow**.