Section 2.3 — Load and Store Instructions in ARM (Mazidi)

Chapter 2 · Section 2.3 — Exercises (Mazidi)

> Problems are paraphrased to respect copyright. For theory and examples, see **Mazidi, Ch. 2 §2.3**.

---

25) Show a simple code to store values 0x30 and 0x97 into locations 0x20000015 and 0x20000016, respectively.

Approach: Use **byte stores** (`STRB`) because the addresses are **unaligned**. Load the base address with `LDR r0,=imm`.

```
        AREA    |.text|, CODE, READONLY
        EXPORT  _start
        THUMB
_start:
        LDR     r0, =0x20000015   ; first byte address
        MOVS    r1, #0x30
        STRB    r1, [r0]          ; [0x20000015] = 0x30

        ADDS    r0, r0, #1        ; next byte address 0x20000016
        MOVS    r1, #0x97
        STRB    r1, [r0]          ; [0x20000016] = 0x97

        B       .
        END
```

Explanation: `STRB` writes a single byte; no alignment issues.

---

26) Show a simple code to load the value 0x55 into locations 0x20000030–0x20000038.

Approach: Fill a **range of 9 bytes** starting at 0x20000030 with `0x55` via a loop.

```
        AREA    |.text|, CODE, READONLY
        EXPORT  _start
        THUMB
_start:
        LDR     r0, =0x20000030   ; start address
        MOVS    r1, #0x55         ; byte to store
        MOVS    r2, #9            ; count: 0x30..0x38 inclusive

fill55:
        STRB    r1, [r0]          ; *r0 = 0x55
        ADDS    r0, r0, #1        ; next byte
        SUBS    r2, r2, #1
        BNE     fill55

        B       .
        END
```

Explanation: Because the range is byte-wise and inclusive, `count = last - first + 1 = 0x38 - 0x30 + 1 = 9`.

---

27) True or False. We cannot load immediate values into the data SRAM directly.

Answer: True (in the strict instruction sense).
Why: There is **no store-immediate** form. You first load the immediate into a **register** (e.g., `MOVS r1,#imm`) and then **store** it with `STR/STRB` to SRAM.

---

28) Show a simple code to load the value 0x11 into locations 0x20000010–0x20000015.

Approach: Fill a **range of 6 bytes** with `0x11` using `STRB` and a counter.

```
        AREA    |.text|, CODE, READONLY
        EXPORT  _start
        THUMB
_start:
        LDR     r0, =0x20000010   ; start address
        MOVS    r1, #0x11         ; byte to store
        MOVS    r2, #6            ; 0x10..0x15 inclusive

fill11:
        STRB    r1, [r0]
        ADDS    r0, r0, #1
        SUBS    r2, r2, #1
        BNE     fill11

        B       .
        END
```

Explanation: `count = 0x15 - 0x10 + 1 = 6`.

---

29) Repeat Problem 28, except load the value into locations 0x20000034–0x2000003C.

Approach: Same loop; **9 bytes** from 0x34 to 0x3C inclusive.

```
        AREA    |.text|, CODE, READONLY
        EXPORT  _start
        THUMB
_start:
        LDR     r0, =0x20000034
        MOVS    r1, #0x11         ; requested value
        MOVS    r2, #9            ; 0x34..0x3C inclusive

fill_more:
        STRB    r1, [r0]
        ADDS    r0, r0, #1
        SUBS    r2, r2, #1
        BNE     fill_more

        B       .
        END
```

Explanation: `count = 0x3C - 0x34 + 1 = 9`.

Notes for learners

- Use **`STRB`** for byte writes; `STR` for word (aligned) writes.
- The pseudo-instruction **`LDR Rd,=imm`** loads 32-bit addresses/constants via a literal pool—handy for SRAM addresses like **0x2000_XXXX**.
- For inclusive ranges: **count = last − first + 1**.