Chapter 2 · Section 2.9 — Exercises (Mazidi)

Chapter 2 · Section 2.9 — Exercises (Mazidi)

> Problems are paraphrased to respect copyright. For background on addressing forms, see **Mazidi, Ch. 2 §2.9**.

---

61) Give the addressing mode for each

(a) `MOV R5,R3` → **Register (register-to-register move)** — Operand is a **register**.
(b) `MOV R0,#56` → **Immediate** — Operand is an **immediate literal**.
(c) `LDR R5,[R3]` → **Register indirect** (single data transfer, **[Rn]** = base with **offset 0**, pre-indexed, no write-back).
(d) `ADD R9,R1,R2` → **Register** (three-register data processing; `Operand2` is a **register**).
(e) `LDR R7,[R2]` → **Register indirect** (as in **(c)**).
(f) `LDRB R1,[R4]` → **Register indirect (byte)** — loads a **byte** from address in **R4**.

---

62) Show the contents of the memory locations after execution (assume little-endian, the common MCU default).

(a)

```
LDR R2, =0x129F
LDR R1, =0x1450
LDR R2, [R1]
```

- `0x1450 = 0x9F`
- `0x1451 = 0x12`

Why: The constant `0x129F` is laid out in memory as bytes **9F 12 00 00** (little-endian). Loading from `[R1]` reads the word; the memory bytes remain as shown.

(b)

```
LDR  R4, =0x8C63
LDR  R1, =0x2400
LDRH R4, [R1]
```

- `0x2400 = 0x63`
- `0x2401 = 0x8C`

Why: Halfword `0x8C63` is stored as bytes **63 8C** in little-endian order.

Notes for learners

- **Register indirect** means the **effective address** comes from a register (e.g., `[R3]`). Adding an offset uses forms like `[R3,#imm]` or `[R3,Rm{,shift}]`.
- **Immediate form** uses a literal (`#imm`) as the operand; for memory, there is **no "store-immediate"**—load the immediate into a register, then store.
- Endianness controls **byte order** in memory; ARM MCUs are typically **little-endian**.