Section 2.8 — The Program Counter and Program ROM Space in ARM (Mazidi)

2025-09-01

Chapter 2 · Section 2.8 — Exercises (Mazidi)

> Problems are paraphrased to respect copyright. For background, see **Mazidi, Ch. 2 §2.8**.

---

52) Every ARM family member wakes up at address ______ when it is powered up.

Answer: `0x00000000` (reset vector base).
Why: On reset, execution starts from the **vector table at address 0x00000000** (PC is loaded from that table). Some MCUs can remap, but the architectural default is **0x00000000**.

---

53) A programmer puts the first opcode at address `0x100`. What happens when the microcontroller is powered up?

Answer: It **does not start at `0x100`**. The CPU **fetches from the reset vector at `0x00000000`**. To run code at `0x100`, the reset vector (or early code) must **branch/jump** there; otherwise the CPU executes whatever is at/pointed to by address 0x0.

---

54) ARM instructions are ______ bytes.

Answer: 4 bytes (32-bit) in ARM state.
Note: Thumb instructions are **16-bit** (some 32-bit encodings), but this section refers to **ARM** instructions.

---

55) Program: add each digit of your 5-digit ID and store the sum at `0x4000100`

Approach: Define the digits with `EQU` (replace `D1..D5` with your own digits), sum them, and store the result as a **word** in SRAM.

```
        AREA    |.text|, CODE, READONLY
        EXPORT  _start
        THUMB

; === Replace these with your actual ID digits (0-9) ===
D1      EQU     1
D2      EQU     2
D3      EQU     3
D4      EQU     4
D5      EQU     5

DST     EQU     0x40000100

_start:
        MOVS    r0, #0          ; accumulator
        MOVS    r1, #D1
        ADDS    r0, r0, r1
        MOVS    r1, #D2
        ADDS    r0, r0, r1
        MOVS    r1, #D3
        ADDS    r0, r0, r1
        MOVS    r1, #D4
        ADDS    r0, r0, r1
        MOVS    r1, #D5
        ADDS    r0, r0, r1

        LDR     r2, =DST
        STR     r0, [r2]        ; store sum as 32-bit word

        B       .
        END
```

Explanation: Five immediate adds accumulate the digit sum; `STR` writes the 32-bit result to `0x40000100`.

---

56) Show the placement of data for:

```
LDR R1, =0x22334455
LDR R2, =0x20000000
STR R1, [R2]
```

- **Little-endian (ARM MCUs default):**
  `0x20000000: 55`, `0x20000001: 44`, `0x20000002: 33`, `0x20000003: 22`
- **Big-endian:**
  `0x20000000: 22`, `0x20000001: 33`, `0x20000002: 44`, `0x20000003: 55`

Why: Little-endian stores the **least-significant byte** at the **lowest address**.

---

57) Show the placement of data for:

```
LDR R1, =0xFFEEDDCC
LDR R2, =0x2000002C
STR R1, [R2]
```

- **Little-endian:**
  `0x2000002C: CC`, `0x2000002D: DD`, `0x2000002E: EE`, `0x2000002F: FF`
- **Big-endian:**
  `0x2000002C: FF`, `0x2000002D: EE`, `0x2000002E: DD`, `0x2000002F: CC`

---

58) How wide is the memory in the ARM chip?

Answer: 8 bits (byte-addressable).
Why: Memory is organized in **bytes**; loads/stores can access **byte/halfword/word**, but the fundamental addressable

unit is **8-bit**.

---

59) How wide is the data bus between the CPU and the program memory in the ARM7 chip?

Answer: 32 bits.
Why: ARM7 implements a **32-bit data path** for instruction and data accesses (device-specific memories may vary, but the core bus is 32-bit).

---

60) In `ADD Rd,Rn,operand2`, how many bits are allocated for Rd and how does that cover all GPRs?

Answer: 4 bits for **Rd** → **16 possible values** → **R0–R15**.
Why: A 4-bit field in the encoding (bits **[15:12]** in ARM state) selects any of the **16 architected registers**.

Notes for learners

- **Reset/PC:** On Cortex-M, the **initial SP** is at `0x00000000` and the **reset handler address** is at `0x00000004` inside the vector table; execution still **originates from the table at 0x0**.
- **Endianness:** Most microcontrollers ship **little-endian**; big-endian layouts simply reverse byte order at consecutive addresses.
- **Instruction sizes:** ARM (A32) = 32-bit instructions; Thumb (T32) = mostly 16-bit with some 32-bit encodings.