8.1 Microcontroller architecture

Name: _____ Class: _____ Date: _____ **Total: 22 marks**

KEY WORDS microcontroller • memory-mapped register

Objective

Size memory maps from address widths, trace the fetch–decode–execute cycle, and read a register-level datasheet description – every answer checked against bit widths and address bounds.

1 Key knowledge and worked examples

■ **The memory map is arithmetic**

N address lines address 2^N locations. A region spans from base to base + size − 1; regions cannot overlap, and the top address bits often select the region.

■ **Worked example: placing two regions**

Flash 128 KB at `0x0000_0000`; SRAM 32 KB at `0x2000_0000`. Flash spans `0x0000_0000`–`0x0001_FFFF` (2^{17} = 131 072 bytes); SRAM spans `0x2000_0000`–`0x2000_7FFF` (2^{15}). Check: 128 KB needs 17 address lines ✓ (widths match the sizes); the regions' top bits differ (0x000… vs 0x200…) so they cannot overlap ✓; and the decoder can simply compare address bits [31:29] – the map was chosen to make the decoding cheap.

■ **The cycle and the register file**

Fetch (PC → instruction), decode (control unit), execute (datapath), PC advance or jump. The **register file** holds the working set: 16 registers of 32 bits in a classic embedded core – the fastest storage the programmer names directly.

2 Practice

2.1 How many address lines for 2 KB? For 1 MB? [2]

2.2 A peripheral's 32-bit register has bits [7:4] as a field. What range of values fits, and what mask extracts it? [3]

2.3 A 4 GB address space places peripherals in the top 16 MB. Give the base address and the number of bits that must be checked to recognise it. [3]

3 Exam-style questions

3.1 A microcontroller has 512 KB flash at `0x0000_0000` and 96 KB SRAM at `0x2000_0000`.

Give each region's end address (exact hex). Then a third region: peripherals mapped in a 64 KB window at `0x4000_0000` – give its range and state, for each of the three regions, how many address lines the region itself uses and which single bit pattern in the top 4 bits [31:28] identifies it. Verify no two regions overlap. [7]

3.2 A 16-bit timer/counter's register map: CR1 (control), SR (status), CNT (count), ARR (auto-reload) – each 16 bits at consecutive word addresses from 0x4000_1000.

Write the C declarations mapping these registers (volatile pointers). Then: the ARR is loaded with 4999 and the counter starts at 0 – state the number of counts per full period and why the period register is often "N−1". Finally, compute the timer's output frequency if the counter's input clock is 8 MHz and the prescaler register (also 16-bit) is set to 7 (the divider is prescaler + 1). [7]

Solutions

2.1

- 2 KB = 2^{11} → 11 lines.
- 1 MB = 2^{20} → 20 lines.

2.2

- Bits [7:4] hold 4 bits → values 0–15.
- Extract mask: 0xF0 (then shift right by 4).

2.3

- Base: 0xFF00_0000 (top 16 MB = 0x0100_0000 bytes).
- The top 8 bits [31:24] = 0xFF identify it (8 bits for 16 MB).

3.1

- Flash 512 KB = 0x80000 bytes: `0x0000_0000`–`0x0007_FFFF`.
- SRAM 96 KB = 0x18000: `0x2000_0000`–`0x2001_7FFF`.
- Peripherals 64 KB = 0x10000: `0x4000_0000`–`0x4000_FFFF`.
- Lines used: flash 19 (2^{19} = 512 K), SRAM 17 (96 K < 128 K = 2^{17}), peripherals 16.
- Top 4 bits [31:28]: flash/SRAM base 0x0 vs 0x2 vs 0x4 – each region's top nibble is distinct (0x0, 0x2, 0x4) so no overlap is possible ✓.

3.2

```
volatile uint16_t *const CR1  = (uint16_t *)0x40001000;
volatile uint16_t *const SR   = (uint16_t *)0x40001004;
volatile uint16_t *const CNT  = (uint16_t *)0x40001008;
volatile uint16_t *const ARR  = (uint16_t *)0x4000100C;
```

- ARR = 4999 → counts 0, 1, …, 4999 then wraps: **5000 counts** per period.
- "N−1" because the count INCLUDES zero: to get N ticks, load N−1 (an ARR of 0 would give a 1-count period).
- Divider = prescaler + 1 = 8 → counter clock `8 MHz/8 = 1 MHz`; period = 5000 counts → output `1 MHz/5000 = 200 Hz`.