Section 4.3 — ARM Time Delay and Instruction Pipeline (Mazidi)

Chapter 4 · Section 4.3 — Exercises (Mazidi)

> Problems are paraphrased to respect copyright. We show the cycle math and the final wall-clock delay using **F = 1/T**.

Timing assumptions used

- One **machine cycle** = one CPU clock period `T = 1/F`.
- Instruction costs used in this section:
  `NOP = 1`, `SUBS (reg,#imm) = 1`, `LDR (literal) = 3`, `BNE taken = 3`, `BNE not-taken = 1`.
- For a nested delay with outer count **M**, inner count **N**, and **k** NOPs in the inner body, the total cycles are:
  TotalCycles = M × (k+4) × N + 5M − 1
  (derivation: inner loop costs = (k+4)N − 2; per outer iteration add LDR=3 and the outer SUBS/BNE; include first MOV once).

---

22) Oscillator frequency if the machine cycle = 1.25 ns

Answer: 800 MHz (i.e., **800MHz**).

---

23) Machine cycle if `F = 200 MHz`

Answer: 5.00 ns (i.e., **5ns**).

---

24) Machine cycle if `F = 100 MHz`

Answer: 10.00 ns (i.e., **10ns**).

---

25) Machine cycle if `F = 160 MHz`

Answer: 6.25 ns (i.e., **6.25ns**).

---

26) Delay of the subroutine (`M=200`, `N=4,000,000,000`, inner has `k=1` NOP) at 80MHz

- **Total cycles:** 4,000,000,000,999
- **Delay:** 50000.000012488 s ≈ 13 h 53 min 20.000012 s

---

27) Delay of the subroutine (`M=100`, `N=50,000,000`, inner has `k=2` NOPs) at 50MHz

- **Total cycles:** 30,000,000,499
- **Delay:** 600.000009980 s ≈ 10 min 0.000010 s

---

28) Delay of the subroutine (`M=200`, `N=20,000,000`, inner has `k=3` NOPs) at 40MHz

- **Total cycles:** 28,000,000,999
- **Delay:** 700.000024975 s ≈ 11 min 40.000025 s

---

29) Delay of the subroutine (`M=500`, `N=20,000`, inner has `k=3` NOPs) at 100MHz

- **Total cycles:** 70,002,499
- **Delay:** 0.700024990 s ≈ 0.700025 s

---

30) *"ARM chip does not have the NOP instruction"* — what is used instead?

Answer: Assemblers accept the mnemonic **`NOP`**, which they assemble to a no-effect data-processing instruction such as **`MOV r0,r0`** (on classic ARM/ARM7). Newer architectures add a real NOP encoding, but on older parts it's this **`MOV`** pseudo-op.

Cross-checks

- If your core/flash adds wait states, timing will be **longer** than the idealized values above.
- If your toolchain lists different cycle counts (e.g., `LDR` latency), redo the math with those numbers; the structure stays the same.