Embedded Systems Assignment 2: Comprehensive Lab Report

Simulator Used: EDSIM51DI (8051 Simulator)

1. Lab 4: Architecture and Memory Layout

1.1 Introduction & Objectives

This laboratory exercise explores the fundamental architecture and memory organization of the 8051 microcontroller. The 8051 implements a Harvard architecture, which physically separates code (program) memory from data memory, contrasting with the unified memory space of Von Neumann architecture.

Objectives:

- Compare Von Neumann and Harvard architectures and understand their fundamental differences.
- Explore the 8051 memory map, identifying code memory, data memory, and Special Function Registers (SFRs).
- Demonstrate code versus data access using appropriate assembly instructions.
- Analyze the impact of interrupt service routines (ISRs) on stack usage.

1.2 Architecture Comparison

Feature	**Harvard Architecture (8051)**	**Von Neumann Architecture**
Memory Org.	Separate code and data memory	Unified memory for code and data
Buses	Separate buses for instructions and data	Single shared bus for instructions and data
Performance	Higher; allows simultaneous access	Lower; suffers from bus bottleneck
Security	Better; code is protected from data writes	Lower; accidental code modification possible

1.3 Memory Map and Access

- **Code Memory (0000H-FFFFH):** Stores program instructions and constant data; accessed via the MOVC instruction
- **Data Memory:** Consists of internal RAM (00H-7FH) and optional external RAM; accessed via MOV or MOVX
- **Stack:** Resides in internal RAM and grows upward toward higher addresses. The default Stack Pointer (SP) value is 07H.

---

2. Lab 5: Timers, Interrupts, and Real-Time Behavior

2.1 Introduction & Objectives

This lab compares two methods for handling I/O operations: Polling and Interrupt-Driven I/O. Choice of method significantly impacts CPU efficiency and system responsiveness.

Objectives:

- Compare the efficiency of Polling versus Interrupt-driven execution.
- Configure Special Function Registers (SFRs) for interrupt control (IE, TCON, TMOD).
- Demonstrate real-time background tasking while a peripheral handles timing.

2.2 Polling vs. Interrupt-Driven Comparison

- **Polling (polling.asm):** The CPU is trapped in a "busy-waiting" loop, continuously monitoring the TF0 flag. This wastes CPU cycles and does not scale well.
- **Interrupt-Driven (interrupt-driven.asm):** Hardware signals the CPU only when service is required, allowing the CPU to perform other tasks in the main loop.

2.3 Timer 0 Calculation (12 MHz Crystal)

To generate a 50ms delay for an LED toggle:

1. **Machine Cycle:** 12 MHz / 12 = 1 MHz (1µs per tick).

2. **Required Ticks:** 50,000µs / 1µs = 50,000 ticks.
3. **Initial Value:** 65,536 - 50,000 = 15,536.
4. **Hexadecimal Conversion:** 15,536 = 3CB0H.
    - TH0 = 3CH
    - TL0 = B0H

---

3. Lab 6: Data Acquisition and Logging

3.1 System Overview

A temperature data acquisition system was implemented using Python to simulate a sensor with threshold-based control and CSV data export.

3.2 Data Integrity Implementation

- **Timestamping:** ISO 8601 format with millisecond precision to provide an audit trail.
- **Sequential Numbering:** Each sample is indexed to detect missing data and ensure completeness.
- **Data Validation:** Range checks ($-50^{\circ}C$ to $150^{\circ}C$) and NaN detection ensure 100% data quality.
- **Checksums:** Generated using hash(temperature + timestamp) to detect corruption or modification.

4. Reflection on Data Integrity and Reproducibility

4.1 Reproducibility Analysis

Reproducibility tests showed that statistical properties and system behavior are reproducible across multiple runs. However, exact temperature values are not reproducible due to simulated random sensor noise. This provides a realistic representation of scientific simulation.

4.2 Data Integrity Summary

- **Stack Integrity:** In Lab 4 and 5, the Stack Pointer was initialized to safe locations (60H) to avoid overwriting register banks.

- **Hardware Automation:** The 8051's automatic PC saving on the stack during interrupts ensures execution context is preserved without manual intervention.
- **Validation:** Lab 6 demonstrated that metadata preservation (session start, sampling rate, total alerts) is essential for complete session reconstruction.

---

5. Conclusion

The laboratory series successfully demonstrated the architectural advantages of the 8051's Harvard design and the efficiency of interrupt-driven systems over polling. Understanding these concepts is critical for writing efficient code, debugging memory issues, and preventing stack overflows in real-time embedded applications.

Deliverables

1. Source Code: polling.asm, interrupt-driven.asm, temperature_daq.py
2. Screenshots: Memory maps, Timer waveforms, and DAQ graphs
3. Data Logs: 3 CSV files (run1, run2, run3)
4. Flowcharts: VI diagrams for the DAQ system