Capstone Assignment (Final Project)

	School of Engineering & Technology	
	Department: Computer Science	**Session: 2025-2026**
	Programme: BCA	**Semester: V**
	Course Code: ENCA302	**Number of students:**
	Course Name: Introduction to Computer Organization & Architecture	**Faculty: Dr. Kishore Ayyala**

Title: Designing a Real-Time Embedded Computer System for a Smart Traffic Controller
Weightage: 5% of Course Grade
Total Marks: 10
Deadline: [Insert Date]
Duration: 2.5 to 3 Weeks
Submission Mode: GitHub Repository

Learning Objectives / CO Mapping

- **CO1**: Instruction set-level analysis and design decisions using RISC-V and x86.
- **CO2**: Functional unit design and architectural abstraction of a real-time processor.
- **CO3**: Performance evaluation using pipeline efficiency, cache behavior, and memory performance metrics.
- **CO4**: Pipelining and hazard mitigation for time-bound execution in embedded systems.
- **CO5**: Data representation, memory access techniques, and I/O configuration for real-time interfacing.

Problem Context

You are part of a team designing a Smart Traffic Signal Controller (STSC) for deployment in a smart city grid. The controller must:

- Read sensor input (vehicle count, pedestrian buttons, emergency override)
- Execute logic to control signal timing and mode
- Optimize performance with limited hardware resources

- Log traffic data using efficient I/O and memory mechanisms

The embedded controller must be implemented and simulated in a modular, verifiable manner using RISC-V architecture concepts, pipelining, and cache-aware execution.

Problem Statement

Design and simulate a full compute subsystem for a Smart Traffic Signal Controller using a pipelined RISC-V processor. Your system must:

1. Handle real-time sensor inputs (using simulated traces)
2. Execute traffic management algorithms on pipelined instruction execution
3. Use efficient memory and I/O strategies (DMA, interrupts, caches)
4. Evaluate performance using metrics like CPI, AMAT, and stall cycles
5. Be modular and visualized using simulation tools

Project Breakdown and Deliverables

Phase 1: Architectural Design & ISA Simulation (Week 1)

- Design a minimal RISC-V instruction set to implement traffic logic:
    - Read sensors
    - Branch based on traffic density
    - Schedule light sequences (Red → Yellow → Green)
    - Log traffic flow in memory
- Deliverables:
    - isa_spec.pdf: Custom RISC-V ISA subset used
    - traffic_logic.asm: Traffic algorithm in assembly
    - instruction_trace.txt: Simulated input/output instructions trace

Phase 2: Processor Pipeline Design (Week 1–2)

- Design a 5-stage pipelined datapath using Logisim Evolution or Verilog:
    - IF, ID, EX, MEM, WB stages

    - Handle hazards (data and control) using forwarding, stalls, branch prediction
- Deliverables:
    - /pipeline/: Circuit or Verilog source files
    - pipeline_layout.png: Annotated diagram of datapath
    - hazard_handling_report.pdf: Summary of data/control hazard mitigation

Phase 3: Memory Hierarchy Simulation & Analysis (Week 2)

- Implement a cache simulator in Python:
    - L1 and L2 caches
    - LRU replacement
    - Write-back vs write-through policies
- Run traces of your traffic logic and report:
    - Hit/miss rate
    - AMAT
    - Cache vs memory latency
- Deliverables:
    - /sim/: Python simulator files
    - config.json: Cache config parameters
    - cache_trace_analysis.pdf: Plots and analysis

Phase 4: I/O Strategy and System Integration (Week 2–3)

- Simulate two I/O methods:
    - Programmed I/O for sensor input
    - DMA for logging traffic snapshots
- Compare cycle counts and efficiency
- Deliverables:
    - io_strategy_comparison.pdf: Charts and write-up

- integration_summary.md: How CPU, cache, memory, and I/O integrate

Final Report & Demo Submission

- Performance metrics:
  - CPI (Ideal vs Real)
  - Number of stalls/flushes
  - Cache effectiveness
  - I/O overhead (Programmed I/O vs DMA)
- Deliverables:
  - final_report.pdf: System design, analysis, and performance summary
  - demo_walkthrough.mp4: 3–5 minute narrated screen recording of simulation and outputs

Submission Instructions (LMS Upload)

- All deliverables must be **uploaded to the LMS** under **"Assignment 1"** folder.
- File Naming Convention:
  - Report: YourName_AbstractionReport.pdf
  - Code/Spreadsheet: YourName_PerformanceCode.py or .xlsx
- Late submissions will incur penalties unless approved with valid justification.
- Ensure all documents are **final, complete, and virus-free** before upload.
- Only one submission per student/team is allowed.

Evaluation Rubric (10 Marks Total)

Criteria	Weight
ISA & Assembly Code (functionality and clarity)	2 marks
Pipeline Design (correctness and hazard handling)	2 marks
Cache Simulator and Analysis	2 marks
I/O Strategy Implementation and Evaluation	2 marks
Final Report (clarity, insights, metrics)	1 mark
GitHub Repository (organization, code quality)	1 mark

Marking Workflow

1. **Automated pass/fail gate** (GitHub Actions script run by TA).
2. **Rubric sheet** completed independently by two graders; discrepancies > 0.5 → third reviewer.
3. **Feedback** pushed as GitHub review within 7 days; includes inline comments + rubric snapshot.
4. **Regrade window** 3 days after feedback; students submit a PR with rebuttal.

Resources

Core Textbook

- **Computer Organization and Design: The Hardware/Software Interface**
  David A. Patterson, John L. Hennessy, 5th Edition, Elsevier
  Chapters:
  - Chapter 1–2: ISA, RISC-V
  - Chapter 4–5: Pipelining and Hazard Handling
  - Chapter 6: Memory Hierarchy and Caching
  - Chapter 7: I/O Strategies

Supplemental Textbooks

- **Computer Architecture: A Quantitative Approach**
  Hennessy & Patterson (6th Edition) – In-depth on performance metrics and cache design
- **Digital Design and Computer Architecture** by Harris & Harris
  Excellent for beginners using Logisim and RTL concepts

Simulators and Tools

Tool	Purpose	Link
Logisim Evolution	CPU datapath and pipelining design	https://github.com/logisim-evolution/logisim-evolution
Ripes Simulator	RISC-V instruction simulation and pipeline visualization	https://github.com/mortbopet/Ripes
Venus	Online RISC-V IDE and simulator	https://venus.cs61c.org
Cache Simulator Reference	Understand cache design strategies in Python	https://github.com/geohot/cache-sim

🔧 Python Libraries (for cache sim and plotting)

- `numpy` – For array operations and trace processing
- `pandas` – For data analysis and configuration files
- `matplotlib` / `seaborn` – For graphing cache metrics
- `json` – For reading `config.json` files

Public GitHub Repositories (Reference)

1. **RISC-V Pipeline in Verilog (MIT 6.004)**
   https://github.com/MassachusettsInstituteOfTechnology/6.004-riscv
2. **Simple Cache Simulators (Python)**
   - https://github.com/dguo/cache-simulator
   - https://github.com/aldur/CacheSim
3. **Smart Traffic Controller Algorithm (Logic Reference)**
   - https://github.com/arthurpaulino/smart-traffic-light-controller

Online Course Lectures

- **MIT 6.004 Computation Structures**
  https://ocw.mit.edu/courses/6-004-computation-structures-spring-2017
  - Week 4–7: Pipelining
  - Week 8–9: Memory systems
  - Week 10: I/O Devices and Control
- **NPTEL: Computer Organization**
  Prof. Smruti R. Sarangi (IIT Delhi)
  https://nptel.ac.in/courses/106102062

Research Papers (Optional Advanced Reading)

- “Evaluating Instruction-Level Parallelism in Embedded Systems” – ACM
- “Memory Hierarchy Performance in Real-Time Systems” – IEEE Transactions on Computers
- “DMA vs Interrupts for I/O in Embedded Devices” – Journal of Embedded Computing Systems