CMPE-220 Sec 01
CPU Design Report

SUBMITTED BY TEAM 09

Bhavika Sodagum	(017506567)
Om Bharatbhai Kapadiya	(018282355)
Shreya Pudukkottai	(017416724)
Parth Gala	(018198661)

Date : Nov 29, 2025

GitHub Repository

Project Repository

The complete source code, documentation, and demo video for this project are available on GitHub:

Repository URL:

https://github.com/shreyapbk0622/CMPE220_Project

Repository Contents

The repository includes all project components:

- Source Code: CPU emulator (cpu.c), Assembler (assembler.c), Demo programs
- Documentation: ISA specification, Memory map, CPU schematic, Build instructions
- Assembly Programs: Timer, Hello World, and Fibonacci implementations
- Report: This project report and comprehensive status documentation

How to Download, Compile, and Run

Prerequisites

Before building the project, ensure you have:

- GCC Compiler with C11 support or later
- Git for cloning the repository
- Terminal/Command Prompt access

Installing Prerequisites

On macOS:

```
xcode-select --install
```

On Linux (Ubuntu/Debian):

```
sudo apt-get update
sudo apt-get install build-essential git
```

On Windows: - Install MinGW-w64 from https://mingw-w64.org - Add MinGW bin folder to system PATH

Download Instructions

Step 1: Clone the Repository

Open a terminal and run:

```
git clone https://github.com/shreyapbk0622/CMPE220_Project.git
cd CMPE220_Project
git checkout complete-cpu-implementation
```

Step 2: Verify Files

Confirm all required files are present:

```
ls -la
```

You should see: - cpu.c - Main CPU emulator - assembler.c - Assembler implementation - run_hello.c - Hello World demo - run_fibonacci.c - Fibonacci demo - *.asm - Assembly source files - *.md - Documentation files

Compilation Instructions

Compile the CPU Emulator

```
gcc -std=c11 cpu.c -o cpu
```

Compile the Assembler

```
gcc -std=c11 assembler.c -o assembler
```

Compile the Hello World Demo

```
gcc -std=c11 run_hello.c -o hello
```

Compile the Fibonacci Demo

```
gcc -std=c11 run_fibonacci.c -o fibonacci
```

Compile All Programs (One Command)

On Linux/macOS:

```
gcc -std=c11 cpu.c -o cpu && \
gcc -std=c11 assembler.c -o assembler && \
gcc -std=c11 run_hello.c -o hello && \
gcc -std=c11 run_fibonacci.c -o fibonacci
```

On Windows (MinGW):

```
gcc -std=c11 cpu.c -o cpu.exe
gcc -std=c11 assembler.c -o assembler.exe
gcc -std=c11 run_hello.c -o hello.exe
gcc -std=c11 run_fibonacci.c -o fibonacci.exe
```

Running the Programs

Run the Main CPU Emulator

This demonstrates all CPU opcodes with a comprehensive test program:

```
./cpu
```

Expected Output: - Step-by-step instruction execution - Register values after each instruction - Final memory dump - CPU flags (ZR, NG, OV, CY)

Run the Hello World Program

This demonstrates memory-mapped I/O:

./hello

Expected Output:

```
=== HELLO, WORLD! Program ===
Output: HELLO, WORLD!
=== Program Complete ===
```

Run the Fibonacci Sequence Program

This shows detailed Fetch-Decode-Execute-Store cycles:

./fibonacci

Expected Output: - Detailed cycle-by-cycle execution trace - FETCH: Instruction pointer and instruction register values - DECODE: Opcode and operand extraction - EXECUTE: ALU computation details - STORE: Register state updates - Final Fibonacci numbers in registers

Use the Assembler

To assemble an assembly program:

./assembler timer.asm timer_output.h

This generates: - timer_output.h - C header file with machine code - timer_output.bin - Binary machine code file

Viewing Documentation

The project includes comprehensive documentation:

- ISA.md - Instruction Set Architecture specification
- MEMORY_MAP.md - Memory organization and I/O mapping
- CPU_SCHEMATIC.md - CPU architecture diagram
- README.md - Quick start guide
- INSTRUCTIONS.md - Detailed build and test instructions
- PROJECT_STATUS.md - Complete project status report

View any documentation file:

```
cat ISA.md
# or open in a text editor
```

Troubleshooting

Common Issues

Issue: "gcc: command not found"

Solution: Install GCC compiler (see Prerequisites section)

Issue: "Permission denied" when running programs

Solution: Make files executable:

```
chmod +x cpu assembler hello fibonacci
```

Issue: Compilation warnings or errors

Solution: Ensure you're using C11 standard:

```
gcc -std=c11 -Wall -Wextra cpu.c -o cpu
```

Issue: Programs don't produce expected output

Solution: Check that you're in the correct directory and all source files are present.

Testing the Installation

Run this quick test to verify everything works:

```
# Test 1: CPU emulator
./cpu | grep "HALTED"

# Test 2: Hello World
./hello | grep "HELLO"

# Test 3: Assembler
./assembler timer.asm timer_test.h && ls timer_test.h
```

 # Test 4: Fibonacci
./fibonacci | head -20

If all tests pass, the installation is successful.

Team Member Contributions

Project Overview

This project was a collaborative effort to design and implement a complete 16-bit software CPU in C. The team worked together to create a fully functional emulator, assembler, and demonstration programs.

Individual Contributions

Shreya Pudukkottai:

Major Responsibilities:

- CPU Architecture Design - Designed the 16-bit CPU architecture including register organization (8 general-purpose registers, stack pointer, instruction pointer)
- ALU Implementation - Implemented the Arithmetic Logic Unit with all control flags (zx, nx, zy, ny, f, no) and status flags (ZR, NG, OV, CY). Developed the n-bit adder using gate-level logic
- Instruction Set Design - Designed the 15-instruction ISA with 4-bit opcode, 3-bit register fields, and 6-bit immediate values. Defined instruction encoding format
- Control Unit Logic - Implemented the fetch-decode-execute cycle, instruction decoding, and program counter management in the control unit

Specific Code Contributions:

- Implemented struct ALU and alu_compute() function (~150 lines)
- Created struct CPU with all component structures
- Developed fetch_decode_execute() control logic (~100 lines)
- Implemented arithmetic operations (ADD, SUB, MUL, DIV)
- Wrote ALU test cases in main CPU test program

Documentation:

- Authored CPU_SCHEMATIC.md with complete architecture diagram
- Created detailed ALU operation explanations in ISA.md
- Documented flag semantics and instruction encoding

Time Commitment: Approximately 25-30 hours

Om Kapadiya

Major Responsibilities:

- Memory System Design - Designed the 400-word memory architecture with distinct regions for I/O, code/data, and stack
- Memory-Mapped I/O - Implemented the memory-mapped I/O system with character output port at address 0x20. Created memory_write() and memory_read() functions
- Assembler Development - Designed and implemented the complete two-pass assembler with label resolution, symbol table, and code generation
- Load/Store Instructions - Added LOAD and STORE instructions to enable memory access beyond stack operations

Specific Code Contributions:

- Implemented complete assembler.c (~400 lines) including:
  - Two-pass assembly algorithm
  - Label symbol table and resolution
  - Opcode parser and encoder
  - Output generation (C header and binary formats)
- Created memory-mapped I/O functions (~30 lines)
- Implemented LOAD/STORE instruction handlers
- Developed load_program() and dump_memory() functions

Documentation:

- Authored MEMORY_MAP.md with complete memory layout
- Created assembler usage examples
- Documented memory-mapped I/O specifications
- Wrote INSTRUCTIONS.md build guide

Time Commitment: Approximately 25-30 hours

Parth Gala

Major Responsibilities:

- Example Programs - Designed and implemented all three required assembly programs (Timer, Hello World, Fibonacci)
- Demo Programs - Created standalone C programs (run_hello.c, run_fibonacci.c) demonstrating the CPU with detailed execution traces
- ISA Documentation - Authored comprehensive ISA specification including all instruction details, addressing modes, and examples
- Testing and Integration - Performed system integration testing, verified all components work together, and created test cases

Specific Code Contributions:

- Implemented run_hello.c (~200 lines) with memory-mapped I/O demonstration
- Implemented run_fibonacci.c (~300 lines) with detailed cycle tracing showing FETCH-DECODE-EXECUTE-STORE phases
- Created timer.asm with extensive comments explaining CPU cycles
- Created hello.asm demonstrating character output via STORE
- Created fibonacci.asm with algorithm documentation and trace examples

- Implemented register dump and CPU state display functions

Documentation:

- Authored ISA.md with complete instruction set specification
- Created detailed instruction encoding examples
- Documented all addressing modes with examples
- Wrote comprehensive README.md with quick start guide
- Created PROJECT_STATUS.md tracking all requirements
- Documented Fibonacci algorithm and execution trace in comments

Testing:

- Verified compilation on multiple platforms (macOS, Linux, Windows)
- Created test procedures and verification steps
- Tested assembler with all example programs
- Validated output correctness for all demo programs

Time Commitment: Approximately 25-30 hours

Bhavika Sodugum

- Architecture Decisions - All major design decisions were made collectively through team meetings
- Code Reviews - Each team member reviewed others' code before integration
- Integration Testing - All members participated in testing the complete system
- Documentation Review - All documentation was peer-reviewed for accuracy

- Demo Video - recorded the video with input from all team members

Tools and Technologies Used

- Programming Language: C (C11 standard)
- Version Control: Git and GitHub
- Development Environment: VSCode, Vim, terminal editors
- Compiler: GCC
- Documentation: Markdown
- Video Recording: [QuickTime/OBS/Screen Recorders

Conclusion

Learning Outcomes

Through this project, the team gained hands-on experience with:

- Computer architecture design principles
- Instruction set architecture development
- Fetch-decode-execute cycle implementation
- ALU design and arithmetic operations
- Memory organization and management
- Assembler design and implementation
- Low-level programming and bit manipulation
- System integration and testing
- Technical documentation and presentation

Acknowledgments

We would like to thank:

- Ishie Eswar for guidance and project requirements
- Course TAs for technical support and feedback
- Open Source Community for GCC compiler and development tools

References

- Computer Organization and Design (Patterson & Hennessy)
- Nand2Tetris Course Materials
- GCC Documentation
- C11 Standard Specification