CPU Design Project: Software CPU

CMPE 220 – System Software

Team Members

Abdul Muqtadir Mohammed

Akash Kishorbhai Devani

Faisal Barkatali Budhwani

Venkata Sai Anjana Karthikeya Nimmala Sri Naga

GitHub Repository

The full source code for the Tiny16 software CPU project is available at:

Repository URL:

https://github.com/F1804/software$_{c}pu_{d}esign$

To download the project, you may either:

- Visit the GitHub link above and download the ZIP file, or
- Clone the repository using Git:

  `git clone https://github.com/F1804/software`$_{c}pu_{d}esign.git$

  `cd software_cpu_design`

How to Download, Compile, and Run the Program

All instructions below are guaranteed to work on macOS or Linux with a C++17-compatible compiler.

1. Navigate to the Project Directory

Open a terminal and run:

```
cd  /desktop/software_cpu_design
```

2. Compile the Tiny16 CPU

The main source file downloaded from GitHub is `tiny16.cpp`. Compile it using:

```
g++ -std=c++17 -O2 -o tiny16 tiny16.cpp
```

This creates the executable:

```
tiny16
```

3. Run the Example Programs

Hello World Example

```
./tiny16 run examples/hello.asm
```

Expected output: `Hello, World!`

Timer Example

```
./tiny16 run examples/timer.asm
```

Expected output: `STimer`

Fibonacci Example

Assemble the Fibonacci program:

```
./tiny16 asm examples/fib.asm -o fib.bin
```

Then emulate it:

```
./tiny16 emu fib.bin --base 0x0000 --pc 0x0100 --dump 0x0100 0x0140
```

This prints the first ten 16-bit Fibonacci numbers stored in memory.

Team Member Contributions

Abdul Muqtadir Mohammed

- Implemented the memory subsystem including RAM, MMIO regions, UART output handling, and timer interrupt logic.
- Developed and tested all example assembly programs (`hello.asm`, `timer.asm`, `fib.asm`) on the Tiny16 emulator.
- Verified the full end-to-end workflow (assembling, running, emulating, memory dumps) and prepared the run instructions and documentation.

Akash Kishorbhai Devani

- Implemented the two-pass assembler including parsing, instruction encoding, fixup processing, and symbol table logic.
- Added support for directives such as `.org`, `.word`, and `.stringz`.
- Designed the virtual file mapping system and assisted in debugging the assembler.

Faisal Barkatali Budhwani

- Implemented the CPU execution pipeline including fetch–decode–execute behavior and program counter updates.
- Developed ALU operations (ADD, SUB, AND, OR, XOR, NOT, SHL, SHR) and implemented accurate flag semantics (Z, N, C, V).
- Implemented branching logic, call/return instructions, and stack operations to support structured program behavior.

Venkata Sai Anjana Karthikeya Nimmala Sri Naga

- Organized the overall project file structure and integrated the example programs for consistent testing and grading clarity.
- Helped test CPU execution correctness including instruction behavior, timer logic, and memory interactions.
- Improved readability by polishing documentation, adding comments, and verifying program organization for final submission quality.