SimpleCPU Project Report

Team Members:

Jaya Manikanta Rohit Dosapati

Kishan Kumar Reddy Konreddy

Rishita Konda

Venkata Sai Sri Harsha Kata

GitHub Repository

Repository Link:

https://github.com/D-J-M-Rohit/SimpleCPU

How to Download, Compile, and Run the Program

1. Clone the repository:

```
git clone https://github.com/D-J-M-Rohit/SimpleCPU.git

cd SimpleCPU
```

2. Build the project using Make:

```
make all
```

3. Run the factorial program:

```
./simple-cpu run build/factorial.bin
```

Expected Output:

```
3! = 6
```

Team Member Contributions

Jaya Manikanta Rohit Dosapati:

Jaya Manikanta Rohit Dosapati designed and implemented the full factorial assembly program in `programs/factorial.asm`. He planned how to express factorial computation using SimpleCPU instructions, chose register roles, and wrote the loop that multiplies the running product by each decreasing integer until it reaches 1. He carefully used register A as the countdown value and register B as the running product, wiring these into the multiply and compare instructions used inside the loop.

He also implemented the digit extraction and printing logic. Jaya wrote the code that repeatedly divides the final factorial result by 10 to obtain each digit, converts the remainder into an ASCII character, pushes digits onto the stack in reverse order, and then pops and prints them so that the number appears in the correct left-to-right order. He handled formatting for the "n! = " prefix and newline, ensured registers were preserved across sub-steps, and added comments so the flow of the assembly program is easy to follow.

Kishan Kumar Reddy Konreddy:

Kishan Kumar Reddy Konreddy implemented and refined key CPU instructions in `src/cpu.c` that the factorial program relies on. He built the MUL instruction handler that multiplies two 16-bit registers, stores the result, and updates the CPU flags correctly. This instruction is central to factorial because it is used in every loop iteration as the product grows.

He also implemented and tested the DIV instruction, which the digit printing logic uses to compute

quotient and remainder when splitting the result into decimal digits. Kishan verified that quotient and remainder behavior matched expectations and protected against divide-by-zero cases. In addition, he worked on DEC and CMPI handling, so the CPU can decrement registers reliably and compare against immediate values, which the loop uses to detect when the countdown reaches 0 or 1.

Kishan used the trace and debug modes to step through execution of the factorial program, inspected register contents and flags across each instruction, and fixed bugs related to incorrect arithmetic or flag propagation to ensure stable and predictable program behavior.

Rishita Konda:

Rishita Konda developed the command-line interface and main driver in `src/main.c` that allow users to assemble and run the factorial program easily. She wrote the main function that parses command-line arguments such as `assemble`, `run`, `debug`, and `asm-run`, and mapped each command to the correct helper function.

Rishita implemented the `cmd_assemble` function that calls the assembler on a source file like `programs/factorial.asm` and writes the resulting binary to the `build/` directory. She also wrote the `cmd_run` function that loads a binary into memory starting at address 0x0100, initializes the CPU, and then invokes the emulator to execute until HALT.

She implemented the `cmd_debug` mode to print register state around program execution and the `cmd_asm_run` path for quick iteration that assembles and runs in one step. Rishita tested these commands with multiple sample programs, refined error messages, and made the workflow for

running the factorial program clear and user friendly.

Venkata Sai Sri Harsha Kata:

Venkata Sai Sri Harsha Kata designed the CPU structure and coordinated overall integration across modules using `src/cpu.h` and related files. He defined the CPU register set (A, B, C, D, SP, PC), the FLAGS register bits, the memory array, and opcode values for all instructions. These definitions ensure that the assembler and the CPU emulator agree on how instructions and addressing modes are represented.

Harsha also implemented core helper functions such as register accessors, memory read/write operations, and stack push/pop behavior, which the factorial program indirectly uses through CALL, RET, and stack-based digit printing. He contributed to defining the memory map, including the program region starting at 0x0100 and the stack region growing downward from high memory.

In addition to code, Harsha worked on documentation that explains the memory layout, instruction set, and execution flow so that the whole team had a shared understanding of how the factorial program moves from assembly source to binary execution on SimpleCPU. He participated in end-to-end testing, verifying that the assembled factorial program runs correctly under the emulator and prints the expected result.