Assembly Language Interpreter in C++

Team Members

Pradyumn Kejriwal
Group-coordinator

Anmol Goklani
Member

Ayushka Garg
Member

Project Overview (Original Requirements)

This project involved developing an interpreter in C to execute a basic assembly language for a machine using a single register, the accumulator. The interpreter reads and processes assembly instructions for arithmetic (ADD, SUB, MUL, DIV), logical (AND, OR, NOT), and memory operations (LD, ST). Each command modifies the accumulator or specified memory locations to perform computations.

Project Expansions

To extend functionality, we introduced new instructions for branching, input/output operations, stack operations, comparison (CMP), bit-shifting, etc. These enhancements required adding additional registers, allowing the interpreter to support more complex control flow, data manipulation, and input/output handling. This expansion transformed the interpreter from a basic single-register machine to a more versatile assembly language interpreter capable of handling a wider range of operations and use cases. Our realtime visualisation also aids with debugging.

Project Workflow

- **Input Handling**: Program starts by reading input from a file containing assembly instructions.
- **Instruction Parsing**: Each line of the file is parsed into an Instruction object based on its opcode and operands. We create a vector of these Instruction objects.
- **Execution Flow**: We initialize a main execution loop where instructions are processed step-by-step.

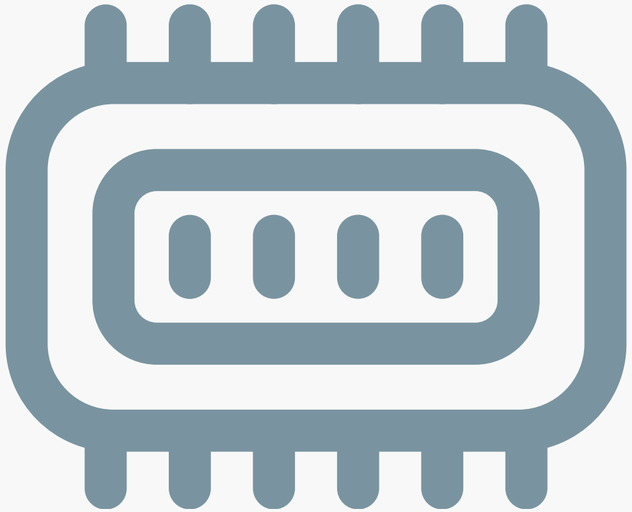
Memory Implementation

Map-Based Memory Model

Memory is represented as a map structure, allowing flexible storage and quick access to memory locations by address.

Register Integration

Additional registers support data movement and processing between registers and memory, enabling complex operations.



Efficient Data Management

The memory model allows efficient storage and retrieval, essential for handling diverse instructions and data manipulations.

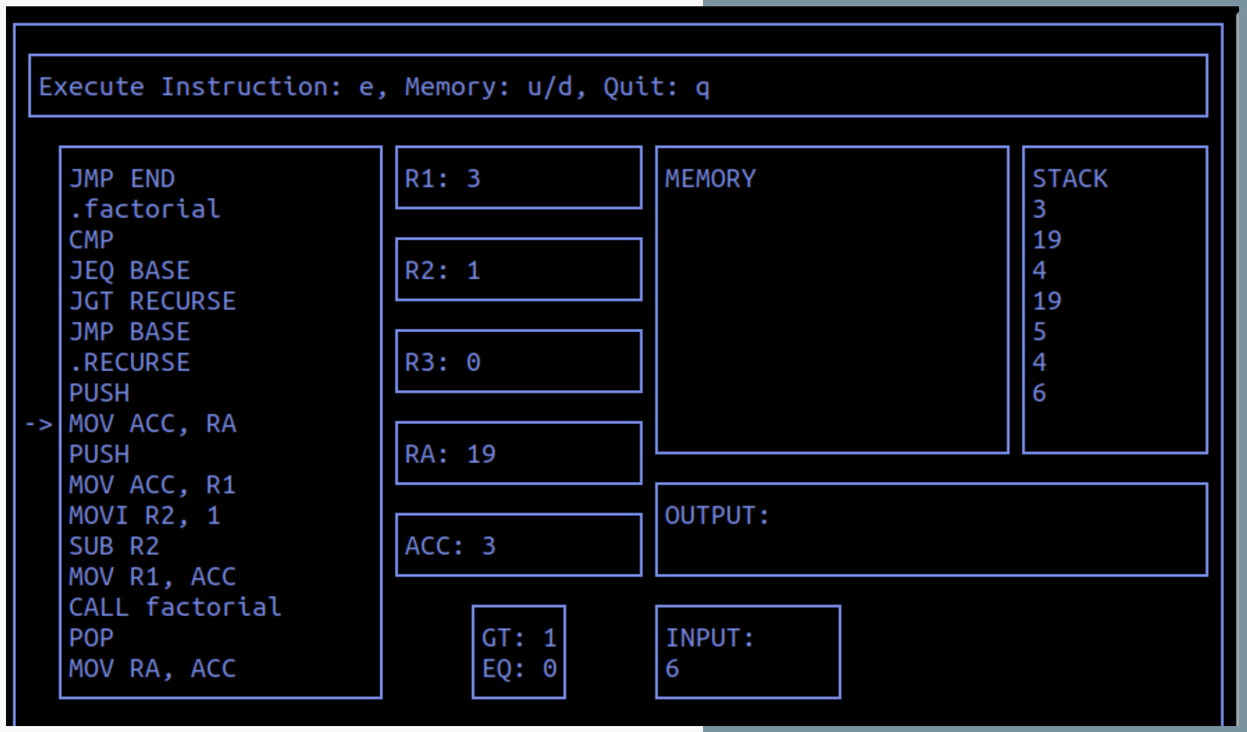
Real Time Visualisation

Step-by-Step Execution

ASCII art displays live updates of memory, stack, registers, input, output and the program counter during execution.

Enhanced Debugging

The visual format provides a clear, real-time view of program state, aiding debugging and understanding of instruction flow.



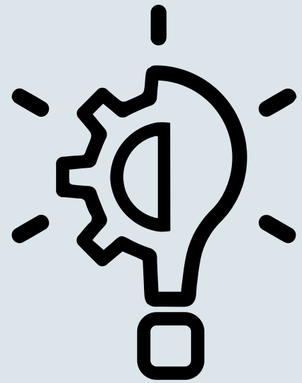
Key Features and Enhancements

Modularity and Extensibility

Expanded instruction set, additional registers, and flexible memory mapping for complex programs.

Interactivity

Users can interact with the interpreter through input instructions and observe execution step-by-step.



Educational Focus

Visualized, interactive execution improves understanding of assembly language and machine-level operations.

Conclusion

Evolution from a basic single-register interpreter to a versatile assembly interpreter.
Enhanced instruction set, real-time visualization, and memory management make this tool both practical and educational.

Thank you