Lab 4 Report

Design and Implementation of a Stack-Based Virtual Machine

Astitwa Saxena
2025MCS3005

Abhishek Gupta
2025MCS2963

1 Introduction

This report describes the design and implementation of a stack-based virtual machine (VM) along with its assembler. The VM executes a custom instruction set and supports arithmetic computation, control flow, and function calls. The system was validated using multiple assembly programs and benchmarked for performance.

2 Architecture of the Virtual Machine

The virtual machine follows a stack-based architecture.

2.1 Program Counter

The Program Counter (PC) stores the address of the next instruction to be executed. It is incremented sequentially and updated explicitly during control flow instructions such as jumps and calls.

2.2 Stack

The stack is used for:

- Arithmetic operations
- Passing arguments to functions
- Storing return addresses and local variables

All ALU operations pop operands from the stack and push results back.

2.3 Memory Model

The VM uses a linear memory array to store instructions and data. Instructions are fetched using the PC, while data memory is accessed via explicit load and store instructions.

3 Instruction Dispatch Strategy

Instruction dispatch is implemented using a switch-based execution loop. Each opcode is decoded and executed sequentially. This design prioritizes simplicity and clarity over aggressive optimization. The dispatch loop continues until a HALT instruction is encountered.

4 Call Frames and Return Mechanism

Function calls are implemented using explicit CALL and RET instructions.

4.1 Call Frames

A call frame consists of:

- Return address
- Function arguments
- Local variables

The return address is pushed onto the stack before transferring control to the function.

4.2 Return Mechanism

The RET instruction restores the return address from the stack and updates the PC accordingly. The function result is left on top of the stack for the caller.

5 Benchmarking and Results

The VM was benchmarked using iterative execution of assembly programs.

5.1 Benchmark Screenshots

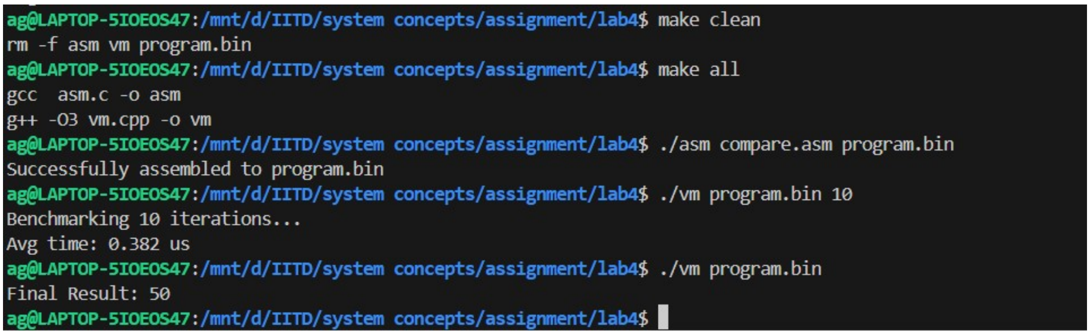

Figure 1: Benchmark execution and average timing

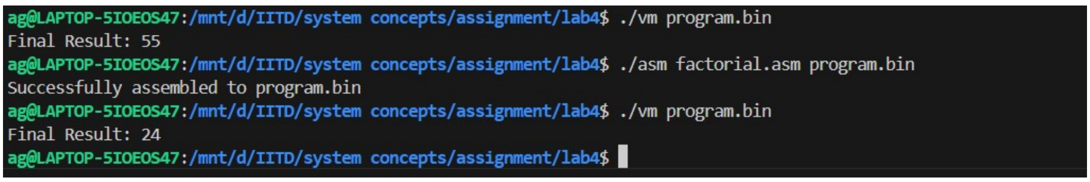

Figure 2: Program execution result after benchmarking

6 Limitations and Possible Enhancements

6.1 Limitations

- Switch-based dispatch limits performance scalability

- No support for dynamic memory allocation
- Limited debugging and error reporting
- No register-based optimization

6.2 Possible Enhancements

- Implement threaded or JIT-based dispatch
- Add heap memory support
- Improve assembler diagnostics
- Introduce registers for hybrid execution
- Add instruction-level profiling

7 Conclusion

The implemented VM demonstrates a clean and functional execution model suitable for educational purposes. Despite its simplicity, it supports non-trivial programs and provides a strong foundation for future extensions.