BENCHMARK REPORT

Bytecode Virtual Machine (BVM)

Assignment: Lab 4
Date: 07-01-2026

Team: Abhinav Gupta (2025MCS2089)
Pratik Chaudhari (2025MCS2096)

Overview

This report documents the performance benchmarking of the Bytecode Virtual Machine (BVM). The objective is to verify the efficiency of the VM implementation and analyze the relationship between program size, instruction count, and execution time across various algorithmic complexities.

Methodology

A custom benchmarking tool (**src/benchmark_main.cpp**) was developed to automate the testing process. The methodology ensures high-resolution timing for the lightweight VM operations:

1. **Assembly:** Each test case (.asm) is assembled into memory at runtime to determine the **program size.**
2. **Execution Loop:** The VM executes the bytecode. To capture microsecond-level precision for fast operations, each test is run **100,000 times**.
3. **Metric Collection:**

- **Instructions Executed:** The VM tracks the total number of fetch cycles performed during a single run.
- **Time:** The total duration is measured using std::chrono::high_resolution_clock and averaged over the iterations to determine **Time per Run (μs)**.

Benchmark Results

The following data was collected on the host machine (MacBook Air/Standard Linux Environment).

Test Case	Program Size (Bytes)	Instructions Executed	Time (µs)	Description
Arithmetic	24	8	0.08312	Simple stack arithmetic (Add, Mul, Div)
Circle Area	14	6	0.05888	Formula calculation using DUP and MUL
Loop Sum	79	135	1.02542	Loop control flow overhead and accumulation
Factorial (Func)	73	63	0.5227	Recursive function calls with base cases
Fibonacci	94	120	0.92555	Iterative calculation using heavy memory swapping
Nested Calls	26	10	0.08925	Multiple function calls testing return stack depth
Memory Ops	64	16	0.15832	Frequent LOAD/STORE global memory access
Conditional	52	11	0.09306	Branching logic using CMP and JZ to find max value
Stack Ops	32	12	0.11554	Stack manipulation testing DUP, POP, and arithmetic
Complex Calc	82	86	0.68477	Mixed workload: loops, calls, and arithmetic

(Note: Instruction counts refer to the number of opcodes executed in the final successful logic path.)

Performance Analysis

Execution Efficiency

The VM demonstrates high efficiency, with basic arithmetic programs (Arithmetic, Circle Area) executing in less than **1 microsecond**. This confirms the low overhead of the switch-based dispatch mechanism and direct pointer arithmetic used for operand fetching.

Instruction Density

There is a clear correlation between algorithmic complexity and execution time.

- **Linear Programs:** Tests like "Arithmetic" have an instruction count roughly proportional to their byte size.

- **Branching Programs:** "Loop Sum" and "Fibonacci" show higher execution times relative to their size. This is due to the JMP and CMP instructions causing the VM to execute the same block of code repeatedly.

Memory vs. Stack Overhead

The "Fibonacci" test relies heavily on LOAD and STORE to swap variables in memory slots 0, 1, and 2. The performance remains stable (~0.9 µs), indicating that the simulated memory access (array indexing) is nearly as fast as stack operations, validating the efficiency of the memory model design.

Conclusion

The BVM performs deterministically and efficiently across all tested scenarios. The metrics validate that the VM correctly handles control flow, recursion, and memory operations without significant performance bottlenecks. The instruction execution rate is consistent, making the VM suitable for the intended educational and simulation purposes.