Stack-Based Bytecode Virtual Machine

Gaurav Jain (2025MCS2121)
Aman Singh (2025MCS2122)

January 8, 2026

1 Architecture of the VM

The Virtual Machine operates on a stack-based architecture, utilizing a split-stack design to separate data manipulation from control flow. The internal state is managed through the `VM` structure defined in `vm.c`.

1.1 Memory Model

The VM's memory is segmented into four distinct regions:

- **Operand Stack:** A fixed-size array (`int stack[STACK_SIZE]`) used for all arithmetic and logical operations. Instructions such as `ADD` or `SUB` pop operands from this stack and push the result back.
- **Global Memory:** A direct-mapped storage area (`int memory[MEMORY_SIZE]`) that acts as the VM's Random Access Memory (RAM). It allows for data persistence across different stack operations via `LOAD` and `STORE` instructions.
- **Return Stack:** A dedicated stack (`int return_stack[RETURN_STACK_SIZE]`) used exclusively for managing function calls. This separation prevents stack corruption where data might be mistaken for return addresses.
- **Code Segment:** A byte array (`unsigned char *code`) storing the raw bytecode instructions.

1.2 Registers

The VM uses three primary architectural registers to track execution state:

- **PC (Program Counter):** A pointer (`unsigned char *pc`) that tracks the next instruction byte to be executed.
- **SP (Stack Pointer):** An integer index (`int sp`) pointing to the next free slot on the operand stack.
- **RSP (Return Stack Pointer):** An integer index (`int rsp`) pointing to the next free slot on the return stack.

2 Instruction Dispatch Strategy

The VM employs a standard **Fetch-Decode-Execute** cycle, implemented within the `run` function.

2.1 The Dispatch Loop

The execution is driven by a `while` loop that continues as long as the `running` flag is set and the `pc` is within valid bounds of the code segment.

```c
while (vm->running && (vm->pc - vm->code) < vm->code_size)
{
    unsigned char opcode = *vm->pc;
    vm->pc++; // Fetch and Advance

    switch (opcode) // Decode and Dispatch
    {
        case OP_PUSH: // Execute
            // ... implementation ...
            break;
        // ... other opcodes
    }
}
```

Listing 1: Dispatch Loop Structure

2.2 Operand Handling

The dispatch strategy varies based on instruction type:

- **Zero-Operand Instructions (e.g., ADD, POP):** Directly manipulate the stack.
- **Immediate-Operand Instructions (e.g., PUSH, JMP):** The `read_bytes` function is called to read a 4-byte integer directly from the instruction stream immediately following the opcode. The `pc` is automatically advanced by 4 bytes during this process.

3 Call Frames and Return Mechanism

To support subroutines, the VM implements a hardware-like call stack mechanism using the `return_stack`.

3.1 Function Calls (`OP_CALL`)

When the `CALL` instruction is executed; The target address is read from the bytecode. The current execution offset (return address) is calculated: `current_offset = pc - code_base`. This offset is pushed onto the `return_stack`. The `pc` is updated to the target address, effectively transferring control to the function.

3.2 Function Returns (`OP_RET`)

When the `RET` instruction is executed; The VM checks for return stack underflow. The return address offset is popped from the `return_stack`. The `pc` is restored to `code_base + offset`, resuming execution immediately after the original `CALL` instruction.

4 Instruction Set Architecture (ISA)

The Virtual Machine executes a defined set of instructions. The opcodes are defined in `src/isa.h` and are categorized by functionality.

Hex	**Mnemonic**	**Category**	**Description**
		Data Movement	
0x01	`PUSH`	Stack	Push a 32-bit integer onto the stack.
0x02	`POP`	Stack	Remove the top element of the stack.
0x03	`DUP`	Stack	Duplicate the top element of the stack.
0xFF	`HALT`	Control	Terminate VM execution.
		Arithmetic & Logic	
0x10	`ADD`	Math	Pop a, b; Push $a + b$.
0x11	`SUB`	Math	Pop a, b; Push $a - b$.
0x12	`MUL`	Math	Pop a, b; Push $a \times b$.
0x13	`DIV`	Math	Pop a, b; Push a/b.
0x14	`CMP`	Logic	Pop a, b; Push 1 if $a < b$, else 0.
		Control Flow	
0x20	`JMP`	Branch	Unconditional jump to address.
0x21	`JZ`	Branch	Jump to address if top of stack is 0.
0x22	`JNZ`	Branch	Jump to address if top of stack is NOT 0.
		Memory & Functions	
0x30	`STORE`	Memory	Store top of stack into global memory at index.
0x31	`LOAD`	Memory	Load value from global memory index to stack.
0x40	`CALL`	Function	Push return address and jump to target.
0x41	`RET`	Function	Pop return address and jump back.

Table 1: Complete Instruction Set Definitions

5 Limitations and Enhancements

5.1 Current Limitations

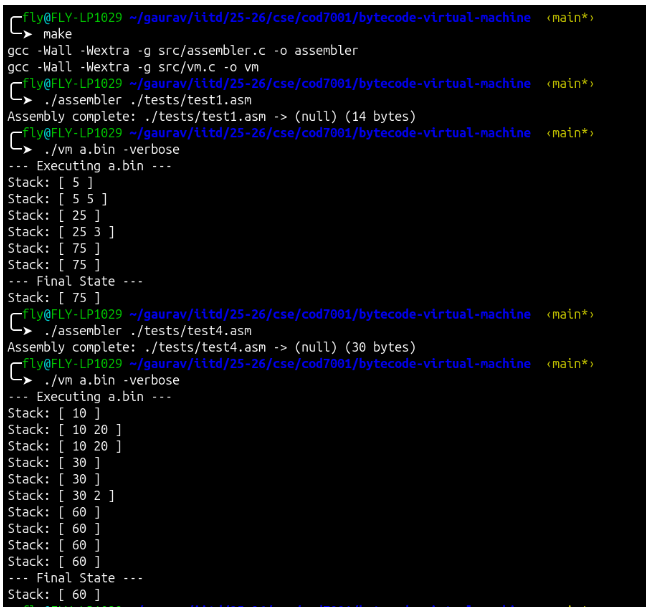
- **Static Memory Allocation:** The stack sizes (256 integers) and memory size (1024 integers) are hardcoded definitions. Recursion depth and dataset size are strictly limited by compile-time constants.
- **Single Data Type:** The VM only supports 32-bit signed integers. There is no support for floating-point arithmetic, characters, or complex data structures.
- **No Standard I/O:** The VM lacks instructions for input or output (e.g., `PRINT`, `READ`). Output is currently limited to a debug print of the stack trace.

5.2 Possible Enhancements

- **Dynamic Resizing:** Implementing `realloc` logic for the stacks and memory would allow the VM to handle larger programs dynamically.
- **Type System:** Introducing a tagged union structure for stack values would enable support for floats and strings.
- **Native Interface (FFI):** Adding a `SYSCALL` opcode could bridge the VM with the host OS, enabling file I/O and console interaction.
- **String Pool:** Implementing a separate string table in the bytecode to support text processing operations.

Appendix

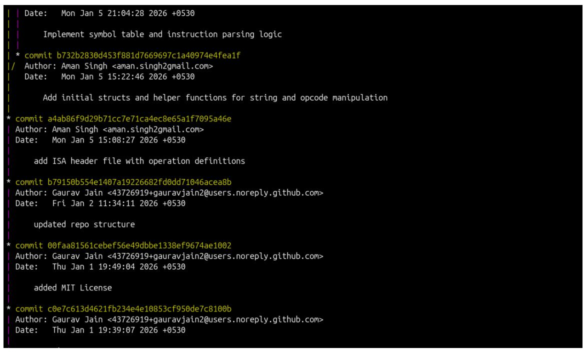
A1. Demo Screenshot



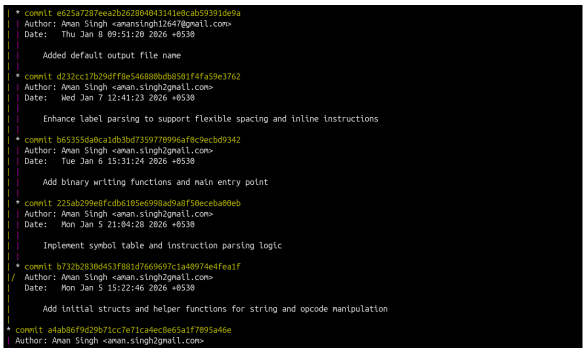
```
┌fly@FLY-LP1029 ~/gaurav/iitd/25-26/cse/cod7001/bytecode-virtual-machine  ‹main*›
└➤ make
gcc -Wall -Wextra -g src/assembler.c -o assembler
gcc -Wall -Wextra -g src/vm.c -o vm
┌fly@FLY-LP1029 ~/gaurav/iitd/25-26/cse/cod7001/bytecode-virtual-machine  ‹main*›
└➤ ./assembler ./tests/test1.asm
Assembly complete: ./tests/test1.asm -> (null) (14 bytes)
┌fly@FLY-LP1029 ~/gaurav/iitd/25-26/cse/cod7001/bytecode-virtual-machine  ‹main*›
└➤ ./vm a.bin -verbose
--- Executing a.bin ---
Stack: [ 5 ]
Stack: [ 5 5 ]
Stack: [ 25 ]
Stack: [ 25 3 ]
Stack: [ 75 ]
Stack: [ 75 ]
--- Final State ---
Stack: [ 75 ]
┌fly@FLY-LP1029 ~/gaurav/iitd/25-26/cse/cod7001/bytecode-virtual-machine  ‹main*›
└➤ ./assembler ./tests/test4.asm
Assembly complete: ./tests/test4.asm -> (null) (30 bytes)
┌fly@FLY-LP1029 ~/gaurav/iitd/25-26/cse/cod7001/bytecode-virtual-machine  ‹main*›
└➤ ./vm a.bin -verbose
--- Executing a.bin ---
Stack: [ 10 ]
Stack: [ 10 20 ]
Stack: [ 10 20 ]
Stack: [ 30 ]
Stack: [ 30 ]
Stack: [ 30 2 ]
Stack: [ 60 ]
Stack: [ 60 ]
Stack: [ 60 ]
Stack: [ 60 ]
--- Final State ---
Stack: [ 60 ]
```

Figure 1: Execution flow

A2. Git Commit History

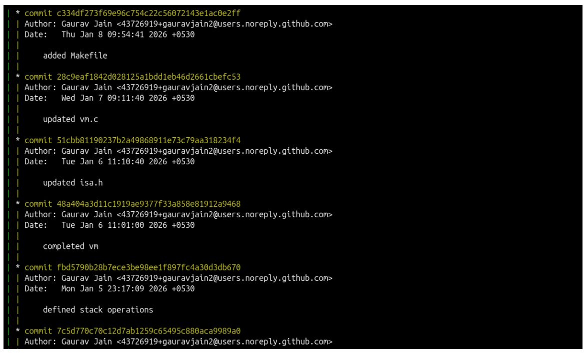


```
| | Date:   Mon Jan 5 21:04:28 2026 +0530
| |
| |     Implement symbol table and instruction parsing logic
| |
| * commit b732b2830d453f881d7669697c1a40974e4fea1f
|/  Author: Aman Singh <aman.singh2gmail.com>
|   Date:   Mon Jan 5 15:22:46 2026 +0530
|
|       Add initial structs and helper functions for string and opcode manipulation
|
* commit a4ab86f9d29b71cc7e71ca4ec8e65a1f7095a46e
| Author: Aman Singh <aman.singh2gmail.com>
| Date:   Mon Jan 5 15:08:27 2026 +0530
|
|     add ISA header file with operation definitions
|
* commit b79150b554e1407a19226682fd0dd71046acea8b
| Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| Date:   Fri Jan 2 11:34:11 2026 +0530
|
|     updated repo structure
|
* commit 00faa81561cebef56e49dbbe1338ef9674ae1002
| Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| Date:   Thu Jan 1 19:49:04 2026 +0530
|
|     added MIT License
|
* commit c0e7c613d4621fb234e4e10853cf950de7c8100b
| Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| Date:   Thu Jan 1 19:39:07 2026 +0530
|
```

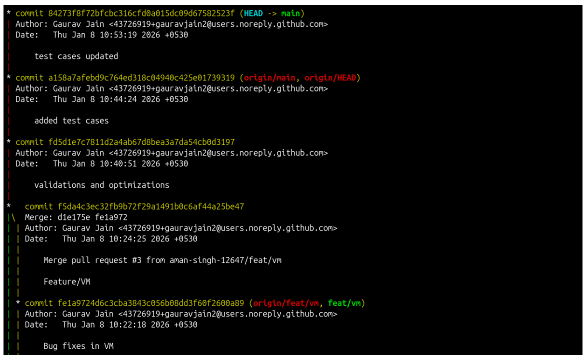


```
| * commit e625a7287eea2b262804043141e0cab59391de9a
| | Author: Aman Singh <amansingh12647@gmail.com>
| | Date:   Thu Jan 8 09:51:20 2026 +0530
| |
| |     Added default output file name
| |
| * commit d232cc17b29dff8e546880bdb8501f4fa59e3762
| | Author: Aman Singh <aman.singh2gmail.com>
| | Date:   Wed Jan 7 12:41:23 2026 +0530
| |
| |     Enhance label parsing to support flexible spacing and inline instructions
| |
| * commit b65355da0ca1db3bd7359770996af0c9ecbd9342
| | Author: Aman Singh <aman.singh2gmail.com>
| | Date:   Tue Jan 6 15:31:24 2026 +0530
| |
| |     Add binary writing functions and main entry point
| |
| * commit 225ab299e8fcdb6105e6998ad9a8f50eceba00eb
| | Author: Aman Singh <aman.singh2gmail.com>
| | Date:   Mon Jan 5 21:04:28 2026 +0530
| |
| |     Implement symbol table and instruction parsing logic
| |
| * commit b732b2830d453f881d7669697c1a40974e4fea1f
|/  Author: Aman Singh <aman.singh2gmail.com>
|   Date:   Mon Jan 5 15:22:46 2026 +0530
|
|       Add initial structs and helper functions for string and opcode manipulation
|
* commit a4ab86f9d29b71cc7e71ca4ec8e65a1f7095a46e
| Author: Aman Singh <aman.singh2gmail.com>
```

Figure 2: Git Commit History



```
| * commit c334df273f69e96c754c22c56072143e1ac0e2ff
| | Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| | Date:   Thu Jan 8 09:54:41 2026 +0530
| |
| |     added Makefile
| |
| * commit 28c9eaf1842d028125a1bdd1eb46d2661cbefc53
| | Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| | Date:   Wed Jan 7 09:11:40 2026 +0530
| |
| |     updated vm.c
| |
| * commit 51cbb81190237b2a49868911e73c79aa318234f4
| | Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| | Date:   Tue Jan 6 11:10:40 2026 +0530
| |
| |     updated isa.h
| |
| * commit 48a404a3d11c1919ae9377f33a858e81912a9468
| | Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| | Date:   Tue Jan 6 11:01:00 2026 +0530
| |
| |     completed vm
| |
| * commit fbd5790b28b7ece3be98ee1f897fc4a30d3db670
| | Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| | Date:   Mon Jan 5 23:17:09 2026 +0530
| |
| |     defined stack operations
| |
| * commit 7c5d770c70c12d7ab1259c65495c880aca9989a0
| | Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
```



```
* commit 84273f8f72bfcbc316cfd0a015dc09d67582523f (HEAD -> main)
| Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| Date:   Thu Jan 8 10:53:19 2026 +0530
|
|     test cases updated
|
* commit a158a7afebd9c764ed318c04940c425e01739319 (origin/main, origin/HEAD)
| Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| Date:   Thu Jan 8 10:44:24 2026 +0530
|
|     added test cases
|
* commit fd5d1e7c7811d2a4ab67d8bea3a7da54cb0d3197
| Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| Date:   Thu Jan 8 10:40:51 2026 +0530
|
|     validations and optimizations
|
*   commit f5da4c3ec32fb9b72f29a1491b0c6af44a25be47
|\  Merge: d1e175e fe1a972
| | Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| | Date:   Thu Jan 8 10:24:25 2026 +0530
| |
| |     Merge pull request #3 from aman-singh-12647/feat/vm
| |
| |     Feature/VM
| |
| * commit fe1a9724d6c3cba3843c056b08dd3f60f2600a89 (origin/feat/vm, feat/vm)
| | Author: Gaurav Jain <43726919+gauravjain2@users.noreply.github.com>
| | Date:   Thu Jan 8 10:22:18 2026 +0530
| |
| |     Bug fixes in VM
```

Figure 3: Git Commit History