1 Lab 4: Bytecode Virtual Machine

Week 5 Marks: 15

1.1 Problem Statement

Design and implement a stack-based bytecode virtual machine capable of executing a defined instruction set with support for arithmetic, control flow, function calls, and memory operations.

1.2 Functional Requirements

1. Instruction set definition
2. Stack-based execution
3. Call frames and returns
4. Bytecode loader
5. Assembler or bytecode generator

1.3 Non-Functional Requirements

Deterministic execution, no memory leaks.

1.4 Optional Enhancements

JIT compilation, standard library.

1.5 Deliverables

VM implementation, assembler, benchmarks, report.

1.6 Instruction Set Definitions

This is a representative Instruction Set which you can use as a template — you free to **add** more instructions if you do feel the need.

1.6.1 Data Movement and Stack Management

These instructions facilitate the movement of literal values onto the stack and manage stack depth.

Mnemonic	**Opcode**	**Description**	**Stack Effect**
PUSH *val*	0x01	Push 32-bit integer *val* onto stack.	$[\,] \to [val]$
POP	0x02	Remove the top element.	$[val] \to [\,]$
DUP	0x03	Duplicate the top element.	$[a] \to [a, a]$
HALT	0xFF	Terminate VM execution.	N/A

1.6.2 Arithmetic and Logical Operations

Arithmetic instructions pop the required operands from the stack and push the result.

Mnemonic	**Opcode**	**Description**	**Stack Effect**
ADD	0x10	Pop b, pop a, push $a + b$.	$[a, b] \to [a + b]$
SUB	0x11	Pop b, pop a, push $a - b$.	$[a, b] \to [a - b]$
MUL	0x12	Pop b, pop a, push $a \times b$.	$[a, b] \to [a \times b]$
DIV	0x13	Pop b, pop a, push a/b.	$[a, b] \to [a/b]$
CMP	0x14	Push 1 if $a < b$, else push 0.	$[a, b] \to [0/1]$

1.6.3 Control Flow

Control flow instructions modify the PC to enable branching and looping.

Mnemonic	Opcode	Description
JMP *addr*	0x20	Unconditional jump to address.
JZ *addr*	0x21	Jump to *addr* if top of stack is 0.
JNZ *addr*	0x22	Jump to *addr* if top of stack is NOT 0.

1.6.4 Memory and Function Calls

These instructions manage global data storage and subroutine execution.

Mnemonic	Opcode	Description	Stack Effect
STORE *idx*	0x30	Store top of stack in Memory[*idx*].	$[val] \to [\,]$
LOAD *idx*	0x31	Push value from Memory[*idx*] to stack.	$[\,] \to [val]$
CALL *addr*	0x40	Push PC+1 to return stack and jump.	N/A
RET	0x41	Pop return stack into PC.	N/A

1.6.5 Programming Example

The following bytecode calculates the area of a circle $A = \pi r^2$ where $r = 5$ and $\pi \approx 3$ (integer math).

Listing 1: Bytecode Example

```
PUSH 5      ; Push radius
DUP         ; Duplicate for squaring
MUL         ; Square radius (25)
PUSH 3      ; Push pi constant
MUL         ; Multiply (75)
HALT        ; Result is 75 on top of stack
```