Integrated Programming Assignment

End-to-End Execution, Debugging, and Memory Management Framework

Overview

In this assignment, you will design and implement a **single, integrated system** that unifies the concepts developed across **Labs 1–5**. The goal is not to implement standalone components, but to build a coherent execution framework in which *every lab component is essential to the functioning of the system.* **The key requirement is that you reuse most if not all of the code that you have developed in Labs 1 to 5.**

Your system will manage the complete lifecycle of a user program: submission, execution, debugging, monitoring, and memory management.

Important: Submissions that merely combine lab features without enforcing interdependence will not receive full credit.

Learning Objectives

By completing this assignment, you will:

- Understand how shells, parsers, execution engines, debuggers, and memory managers interact in real systems.
- Design clean interfaces between system components.
- Build a unified abstraction for program execution across different execution models.
- Reason about program behavior, memory usage, and debugging across layers.

System Concept

A **program** is treated as a first-class entity whose lifecycle is managed by the system.

```
Program → Parse & Instrument → Execute → Debug & Trace → Memory Accounting
```

Each lab contributes *one indispensable stage* in this pipeline.

Component Responsibilities

Lab 1: Shell and Process Control

The shell is the **entry point and program manager**.

It must:

- Accept program submissions.
- Assign a unique program ID (PID).
- Track program state (submitted, running, paused, terminated).
- Dispatch programs to the appropriate execution engine.

Required commands:

```
submit <program>
run <pid>
debug <pid>
kill <pid>
memstat <pid>
```

Without this component, programs cannot be managed or controlled.

Lab 2: Parsing, Instrumentation, and Debug Metadata

Parsing is not limited to syntax checking.
Your parser must:

- Construct an AST.
- Emit debug metadata (line numbers, variable bindings).
- Annotate allocation sites and control-flow points.

This metadata is consumed by the debugger and execution engine.
Without this component, execution is opaque and cannot be debugged meaningfully.

Lab 3: Intermediate Representation

All programs must be lowered to a **common intermediate representation (IR)**.
The IR must encode:

- Instruction semantics.
- Source-level mappings.
- Memory effects.

This IR acts as the unifying abstraction between native execution and virtual execution.
Without this component, execution models cannot be unified.

Lab 4: Virtual Machine and Execution Control

The virtual machine provides a **controlled execution environment**.
It must support:

- Instruction-level stepping.
- Breakpoints.
- Pausing and resuming execution.

All debugger actions operate through this execution layer.
Without this component, debugging commands have no effect.

Lab 5: Memory Management and Garbage Collection

Memory management is system-wide.
Your system must:

- Track all dynamic allocations.
- Maintain a root set from execution state.
- Reclaim unreachable objects.
- Report memory usage per program.

Required commands:

```
memstat <pid>
gc <pid>
leaks <pid>
```

Without this component, memory behavior cannot be analyzed or controlled.

Integration Requirement

Your design must ensure that:

- Removing any one lab component breaks the system.
- Data flows across components through well-defined interfaces.
- Debugging and memory analysis depend on parsing and execution metadata.

Standalone implementations of individual labs will not be sufficient.

Example Workflow

```
myshell> submit example.lang
PID = 4

myshell> run 4
Output: 42

myshell> debug 4
(debug) break 12
(debug) step
(debug) memstat
Heap objects: 37
(debug) continue
```

Each command exercises multiple subsystems simultaneously.

Submission Instructions

Submit a single archive containing all deliverables. Clearly indicate how to build and run your system.

Additional Submission Instructions will be updated.