1-week VM exercise

Goal

Design and implement a register-based VM with a tiered set of features, and develop a compiler that targets RISC-V architecture. The task is structured into core requirements, a control flow extension, and bonus challenges.

Spec:

1. Core Requirements

The core requirements focus on essential arithmetic operations and register-based design, building on low-level VM design principles. The inclusion of a minimal set of registers and basic arithmetic operations (ADD, SUB, MUL) is focused on implementing foundational VM functionality.

- Design a register-based VM with the following basic arithmetic instructions:
  a) ADD: Addition
  b) SUB: Subtraction
  c) MUL: Multiplication
- Implement a minimal set of registers (e.g., 4-8 general-purpose registers)
- Develop a basic compiler that translates these VM instructions to RISC-V assembly
- Create a simple assembler for your VM instruction set
- Write a small program demonstrating these operations and show its compiled RISC-V output

2. Control Flow Extension

To allow for slightly more sophisticated program logic the next step is to implement control flow instructions (JMP, JZ/JNZ).

- Add control flow instructions to your VM:
  a) JMP: Unconditional jump
  b) JZ/JNZ: Conditional jumps (Jump if Zero/Jump if Not Zero)

- Extend your compiler to handle these new instructions
- Implement a stack for function calls
- Write an example program that demonstrates loops and function calls
- Show the compiled RISC-V output for this extended program

3. Testing/Integration/Validation

- Provide instructions for running your compiled programs on a RISC-V emulator of your choice
- Demonstrate successful execution of your programs on the emulator
- Develop test cases for each tier of your implementation
- Create a simple test runner that compiles test programs and verifies their output on the RISC-V emulator

Bonus Challenge (Transactions/Cryptography)

- Add instructions for basic transaction or cryptographic operations:
  a) HASH: Compute a simple hash
  b) VERIFY: Verify a basic signature
- Extend your compiler to handle these new instructions
- Write a simple program that simulates a basic blockchain transaction with signature verification
- Demonstrate the compiled RISC-V output for this advanced program

Deliverables

1. Specification of the VM instruction set (for all implemented tiers)
2. Compiler and assembler source code
3. Example programs for each tier (simple arithmetic, control flow, and if implemented, transactions)
4. Compiled RISC-V assembly output for each example program
5. Test suite covering implemented features

6. A simple summary summarizing the implementation, challenges faced, and any design decisions