Project Name: Build a Virtual CPU Emulator

Week 07: Advanced Features

Objective:

To enhance the functionality and performance of the virtual CPU emulator by integrating advanced features such as branching and control flow instructions, subroutines and interrupt handling, and a simple pipeline mechanism to mimic modern CPU behavior.

---

1. Implement Branching and Control Flow Instructions

Branching and control flow instructions enable the CPU to execute non-linear instruction sequences based on conditions or explicit jumps.

1.1 **Unconditional Branching**:

- Implement instructions like `JUMP` to move the program counter (PC) to a specific memory address.

1.2 **Conditional Branching**:

- Create instructions like `BEQ` (branch if equal), `BNE` (branch if not equal), `BLT` (branch if less than), and `BGT` (branch if greater than).
- Integrate comparison operations to set condition flags for branching.

1.3 **Program Counter Management**:

- Update the program counter logic to support jumps and branching while ensuring seamless instruction execution.

---

2. Add Support for Subroutines and Interrupts

Subroutines and interrupts improve modularity and responsiveness in the virtual CPU.

2.1 **Subroutines**:

- Implement `CALL` instruction to push the return address onto a stack before jumping to a subroutine.
- Implement `RET` instruction to pop the return address from the stack and resume execution.

2.2 **Interrupt Handling**:

- Define an interrupt vector table with predefined memory addresses for interrupt routines.
- Create an `ISR` (Interrupt Service Routine) mechanism to save the CPU state, handle the interrupt, and restore the state before resuming normal execution.
- Enable prioritization and masking of interrupts to prevent conflicts.

---

3. Integrate a Simple Pipeline Mechanism

A pipeline mechanism divides the instruction execution cycle into stages, improving throughput by overlapping operations.

3.1 **Pipeline Stages**:

- Divide execution into Fetch, Decode, Execute, and Write-back stages.
- Design mechanisms to ensure smooth data flow between stages.

3.2 **Hazard Detection**:

- Implement basic hazard detection to identify and resolve data, structural, or control hazards.
- Add stalling mechanisms to handle hazards without compromising correctness.

3.3 **Branch Prediction**:

- Introduce a simple branch prediction mechanism to reduce pipeline stalls caused by branching instructions.
