Mini Project: Design and Implementation of a Minimal 8-Bit CPU

Project Overview

In this mini project for the Computer Architecture and Organization course, students will design and implement a minimal 8-bit CPU using the "Digital" circuit simulation software developed by Hneemann (available at https://github.com/hneemann/Digital). The goal is to apply concepts of CPU architecture, instruction set architecture (ISA), pipelining, memory management, and interrupt handling by building a functional CPU from basic logic gates and components provided in the simulator.

The CPU must adhere to a pipelined architecture with five stages: Fetch, Decode, Execute, Memory, and Write Back. It should include basic I/O ports, separate ROM and RAM, support for arithmetic/logical operations, branching, stack operations, flags, and interrupts. Optional extensions include L1 cache and basic branch prediction for extra credit.

This project emphasizes hands-on understanding of digital logic, CPU internals, and simulation-based verification. Students are encouraged to modularize their design for clarity and ease of debugging.

Minimum Requirements

The CPU must meet the following specifications:

1. Team of 2 students

2. ISA and CPU Specifications

- **8-Bit CPU Architecture**: All data paths, registers, ALU operations, and memory addresses must be 8 bits wide.
- **Instruction Set Architecture (ISA)**: Define a custom ISA with opcodes for the required operations. Instructions should be 8 bits or multiples thereof (e.g., 16 bits for instructions with immediate operands or addresses). Document the opcode format, including fields for opcode, registers, immediates, etc.
- **Pipeline Stages**: Implement a 5-stage pipeline:
    - **Fetch**: Retrieve the instruction from ROM based on the program counter (PC).
    - **Decode**: Decode the instruction and read operands from registers.
    - **Execute**: Perform ALU operations or compute effective addresses.
    - **Memory**: Handle memory reads/writes to RAM (if applicable for the instruction).
    - **Write Back**: Write results back to registers or update flags.
- Handle pipeline hazards (e.g., data dependencies) minimally, such as through stalling or forwarding where necessary.

3. I/O Ports

- **8-Bit Input Port**: A dedicated 4-bit input interface for external data (e.g., from switches or simulated inputs).
- **8-Bit Output Port**: A dedicated 4-bit output interface for displaying results (e.g., to LEDs or simulated outputs).

4. Memory Organization

- **Separate ROM and RAM**:
    - **ROM**: For storing the program code (instructions). Minimum size: 256 bytes (8-bit addressable).
    - **RAM**: For data storage, including the stack. Minimum size: 256 bytes (8-bit addressable).
- Memory access should be handled in the Memory stage of the pipeline.

5. Supported Operations

The CPU must support at least the following instructions:

- **LD** Load.
- **ST** Store.
- **RD** Read from input.
- **WR** Write to output.
- **Arithmetic**: ADD (addition), SUB (subtraction), MUL (multiplication).

- **Logical**: AND, OR, XOR, NOT (bitwise negation).
- Instructions should support register-register and register-immediate modes where applicable.

6. Branching Instructions

- **Unconditional Branch**: Jump to a specified address.
- **Conditional Branches**: Branch if Zero (BZ), Branch if Non-Zero (BNZ), Branch if Carry (BC).
- Branches should update the PC accordingly, with handling for pipeline flushes if needed.

7. Flags

- **Status Flags**: Maintain at least the following flags, updated after relevant operations (e.g., ALU instructions):
  - Z (Zero): Set if result is zero.
  - NZ (Non-Zero): Set if result is non-zero (complement of Z).
  - Carry: Set on carry-out from arithmetic operations.
- Flags should be stored in a dedicated status register and used for conditional branching.

9. Registers

- **General-Purpose Registers**: At least 4-8 registers (e.g., R0-R7) for data manipulation.
- **Special Registers**: PC (Program Counter) and Status Register (for flags).

10. Simulation and Verification

- Use the "Digital" simulator to build the CPU circuit.
- **Test Program**: Write and load at least one assembly program into ROM that demonstrates all required features (e.g., a simple loop with arithmetic, branching, stack usage, and interrupt handling).
- Verify functionality through simulation waveforms, stepping through pipeline stages, and checking outputs.

Deliverables

1. **Circuit Design File**: The .dig file from the "Digital" simulator containing the complete CPU design.
2. **Documentation Report** (PDF, 5-10 pages):
   - Detailed description of the ISA (opcode table, instruction formats).
   - Block diagram of the CPU architecture, including pipeline stages.
   - Explanation of each module (e.g., ALU, control unit, memory interfaces).
   - Assembly code for the test program(s) with comments.
   - Simulation screenshots or waveforms showing key operations (e.g., instruction execution, interrupt triggering).
   - Discussion of challenges faced and solutions.
3. **Video Demo**: A 5-10 minute screen recording demonstrating the CPU in simulation, stepping through the test program, and handling an interrupt.

Evaluation Criteria

- **Functionality (60%)**: Does the CPU correctly implement all minimum requirements? Verified through test programs.
- **Design Quality (20%)**: Modularity, clarity of circuit (use of sub-circuits), efficiency.
- **Documentation (15%)**: Completeness, clarity, and accuracy.
- **Innovation (5%)**: Effective use of optional features or creative optimizations.

Timeline and Submission

- **Duration**: 5 weeks:
- **Submission**:

  21st September Submit Team member and draft CPU specification
  21st October Upload all deliverables
  2nd November Online presentation
- **Resources**: Refer to "Digital" documentation for simulator usage. No external hardware required—pure simulation.

Team and Idea submission:
https://docs.google.com/forms/d/e/1FAIpQLSfSpFgH_2MaplfVrwq6u7tX9F3J53dnoHv_Ygxl9ArIZmhhrw/viewform?usp=header