Mini CPU Design Concept Documentation

1. Overview

This project involves designing a simple CPU using the Digital tool, simulating basic CPU functionalities with a focus on 8-bit calculations. The CPU is built to execute instructions stored in a 256 x 13-bit program RAM (pRAM) and output results to a 256 x 8-bit result RAM (rRAM). The instruction set is designed for basic arithmetic, logical, and control operations, making the CPU capable of executing various test cases efficiently.

Objectives

- Develop a CPU capable of handling 13-bit instructions, split into 5-bit opcodes and 8-bit operands.
- Implement memory structures (pRAM and rRAM) to store program instructions and results.
- Utilize registers for temporary data storage and arithmetic operations.
- Support a variety of instruction types, including arithmetic, logical, and control-flow operations.
- Design the CPU with a synchronous clock, using a positive edge for state changes.

2. CPU Architecture

2.1 Block Diagram

Provide a visual block diagram showing:

- **Control Unit**: Manages instruction decoding and control signals.
- **Data Path**: Handles data operations, arithmetic, and logical computations.
- **Memory Units**: pRAM and rRAM for program storage and results.
- **Input/Output Units**: Interactions with external inputs (M, N) and displaying outputs.
- **Registers**: Temporary storage for computation, including `accA`, `accB`, and `regC`.

2.2 Components

- **Accumulator A (accA)**: 8-bit register for primary arithmetic and logical operations.
- **Accumulator B (accB)**: 8-bit register for storing secondary data.
- **Register C (regC)**: 8-bit register used for specific operations, including input storage.
- **Flag Registers**: Indicators for equal, greater, and lesser conditions.
- **Program Counter (PC)**: Keeps track of the instruction currently being executed.

3. Instruction Set

3.1 Instruction Format

The CPU operates using a 13-bit instruction format:

- **Opcode**: 5 bits to identify the operation.
- **Operand**: 8 bits, used for data or memory addresses.

3.2 Supported Instructions

Include a table that details all the supported opcodes and their descriptions:

3.3 Control Flow Instructions

- **Jump (unconditional)** and conditional jumps (`eq`, `gr`, `le`) are supported based on flag registers.
- Instruction decoding will require branching logic in the Control Unit.

4. CPU Design Approach

4.1 Data Path Design

4.1.1 Register Transfers

- The CPU will utilize register-to-register operations with a central data bus to allow efficient data transfer between the Accumulators and `regC`.
- The ALU (Arithmetic Logic Unit) performs arithmetic and logic operations based on the opcode.

4.1.2 Memory Operations

- Instructions to load and store data from/to pRAM and rRAM will utilize dedicated control lines for addressing and reading/writing data.

4.2 Control Unit Design

4.2.1 Instruction Decoding

- The control unit will decode the opcode to determine the operation type.
- Generates control signals for ALU, register transfer, memory access, and flags setting.

4.2.2 Control Logic

- The CPU operates on a synchronous clock, with operations triggered on the positive clock edge.
- Sequential FSM (Finite State Machine) will manage instruction fetch, decode, execute, and write-back stages.

4.3 Memory Design

- **Program RAM (pRAM)**: 256 x 13-bit memory, storing instructions.
- **Result RAM (rRAM)**: 256 x 8-bit memory, used for storing computation results.

4.4 Input/Output Design

- **Inputs**: Parameters `M` and `N`, `progIN` for program loading, and control signals (`reset`, `progLoad`, `start`, `result`).
- **Outputs**: `valid`, `done`, 8-bit `output`, and two 7-segment displays showing the result in hexadecimal.

5. Finite State Machine (FSM) Design

5.1 State Diagram

Include a state diagram showing:

- **Idle**: Waiting for the `progLoad` signal.
- **Program Load**: Load instructions into pRAM.
- **Execution**: Fetch-decode-execute cycle for instructions.
- **Output Result**: Display the result when the `result` signal is triggered.
- **Halt**: Stop state when `STOP` instruction is encountered.

5.2 State Descriptions

- **Fetch**: Retrieve the instruction from pRAM using the PC.
- **Decode**: Determine the instruction type using the opcode.
- **Execute**: Perform the appropriate operation based on the decoded instruction.
- **Write Back**: Update registers or memory with the result.

6. Arithmetic and Logic Unit (ALU)

6.1 Design Considerations

- Support basic arithmetic: addition, subtraction, multiplication, and division.
- Implement bitwise operations: AND, OR, XOR, NOT.
- Comparison operations for setting flag registers.
- Special instructions like `isPrime` for prime number checking and `LCM` calculation.

7. Test Plan

7.1 Test Cases

- Various test cases to validate arithmetic, logical, and memory operations.
- Conditional branching tests to verify jump instructions.
- Stress test with edge cases, ensuring the CPU handles the full range of instruction set operations.

7.2 Expected Outputs

- Each test case will provide expected values for comparison.
- Validation through rRAM and 7-segment displays.

8. Conclusion

The project is a foundational CPU simulation, introducing digital logic, instruction sets, and control flow. This CPU model will be capable of executing multiple arithmetic and logical operations while managing memory and displaying outputs. The design emphasizes a multi-cycle or pipeline structure for efficiency and better performance under test scenarios.