16-Bit Fine-Grained Interleaved Multithreaded Processor

1. Introduction

This project implements a soft-core 16-bit processor designed to maximize instruction throughput by utilizing **Fine-Grained Interleaved Multithreading (IMT)**. Unlike traditional single-threaded pipelines that rely on complex forwarding units or pipeline stalls to handle data hazards, this architecture switches execution context every clock cycle. This technique effectively "hides" latency, allowing one thread to execute while the other is resolving dependencies.

2. Architecture Overview

The processor features a 4-stage pipeline based on a modified Harvard Architecture. It supports two concurrent hardware threads (Thread 0 and Thread 1) with strict context isolation.

2.1 Pipeline Stages

1. **Fetch (IF):** Retrieves the instruction from the Instruction Memory (ROM) using the Program Counter (PC) of the *active* thread. The thread ID toggles every cycle (Round-Robin Arbitration).
2. **Decode (ID):** Decodes the 16-bit instruction, reads operands from the thread-specific Register File bank, and handles "Flush" logic for control hazards.
3. **Execute (EX):** Performs arithmetic/logical operations via the ALU. Evaluates branch conditions (`BNZ`) and signals the Context Manager if a jump is required.
4. **Writeback (WB):** Writes the result back to the specific Register Bank of the thread that issued the instruction.

2.2 Design Techniques & Methodology

The "Barrel Processor" Approach

The core design philosophy is **Latency Hiding via Interleaving**.

- **Standard Pipeline:** Instruction N depends on $N-1$. If $N-1$ hasn't written back, N must stall.
- **Interleaved Pipeline:**
  - Cycle T: Fetch Thread 0 (Instr A)
  - Cycle $T+1$: Fetch Thread 1 (Instr X)
  - Cycle $T+2$: Fetch Thread 0 (Instr B)
  - When Thread 0 reads registers for Instr B, Instr A has already completed Writeback.
  - **Result:** Zero data hazard stalls without forwarding logic.

Context Isolation

To support multithreading, the architecture duplicates the architectural state:

- **Program Counters:** A `Context Manager` module maintains a table of PCs, indexed by Thread ID.
- **Register File:** A Dual-Banked Register File (`regs[0:1][0:15]`) ensures that Thread 0 cannot corrupt Thread 1's data.

Hazard Resolution

- **Data Hazards:** Eliminated architecturally by the 1-cycle interleave gap.
- **Control Hazards:** Managed via a **Pipeline Flush**. If a branch is taken in the EX stage, the instruction currently in the ID stage (which belongs to the same thread) is invalidated (converted to NOP) to prevent incorrect execution.

3. Instruction Set Architecture (ISA)

The processor uses a custom 16-bit fixed-width ISA.

Opcode	Mnemonic	Format	Description
`0000`	**ADD**	`ADD Rd, Rs1, Rs2`	`Rd = Rs1 + Rs2`
`0001`	**SUB**	`SUB Rd, Rs1, Rs2`	`Rd = Rs1 - Rs2`
`0010`	**AND**	`AND Rd, Rs1, Rs2`	`Rd = Rs1 & Rs2`
`0011`	**OR**	`OR Rd, Rs1, Rs2`	`Rd = Rs1
`0100`	**LDI**	`LDI Rd, Imm8`	Load 8-bit Immediate into Rd
`0101`	**BNZ**	`BNZ Rs, Target`	Branch to Target if Rs != 0
`1111`	**NOP**	`NOP`	No Operation

4. Implementation Details

Hardware Modules

- `mt_core.v` : Top-level entity. Instantiates stages and handles the round-robin scheduler.
- `context_manager.v` : Maintains the PC state machine. It prioritizes Branch Targets over sequential increments.
- `regfile.v` : A parameterized 3-port register file (2 Read, 1 Write) with bank selection based on Thread ID.
- `imem.v` : Combinational ROM storing the machine code.

Synthesis & Resource Utilization

The design is fully synthesizable and optimized for FPGA implementation (e.g., Xilinx Artix-7).

- **LUT Utilization:** < 1% (Extremely lightweight).
- **Max Frequency:** > 100 MHz (Critical path located in ALU carry chain).
- **Logic Type:** Purely synchronous core logic with asynchronous memory reads (Distributed RAM).

5. Verification & Results

The design was verified using behavioral simulation with two concurrent workloads:

1. **Thread 0:** Arithmetic Summation ($\sum_{i=1}^{10} i$).
2. **Thread 1:** Fibonacci Sequence Generator ($N = 10$).

Observed Performance

- **Functional Correctness:** Both threads converged to correct values (`55` and `89`) simultaneously.
- **Speedup:** The interleaved design demonstrated a **1.375x speedup** (37.5% improvement) over a sequential single-threaded execution of the same workloads.
- **Efficiency:** Achieved an effective CPI (Cycles Per Instruction) of ~1.0 for useful instructions, significantly outperforming the ~1.4 CPI of a stalled scalar pipeline.

6. Future Scope

- **Data Memory Integration:** Adding a Load/Store unit for RAM access.
- **Scalability:** Expanding the Context Manager to support N threads (4 or 8) to hide longer latencies (e.g., main memory access).
- **Interrupt Handling:** Adding support for precise exceptions and hardware interrupts.