Rawisara Chairat

U65053623

Lab Report

Introduction

The goal of this lab was to design, implement, and test a 4-stage 32-bit pipelined datapath using behavioral Verilog. The purpose is to deepen understanding of digital design and computer architecture principles by building a pipelined processor model that efficiently executes instructions across multiple stages.

Hierarchical Design

Stage 1: Instruction Fetch and Decode

The S1_Register module is responsible for the initial stages of the pipeline, handling the fetching and decoding of instructions. It extracts and decodes key fields from the instruction such as register select fields (ReadSelect1, ReadSelect2), immediate data (Imm), the instruction type for the ALU (DataSrc), the operation type (ALUOp), the destination register (WriteSelect) and the write enable (WriteEnable). These decoded signals are used to control operations and route data in subsequent stages. This stage sets up all necessary control signals based on the instruction fetched.

Stage 2: Register Read and Setup for Execution

Following the fetch and decode operations, the S2_Register module receives decoded data from the S1_Register. It reads the actual operand values from the register file and uses the immediate value decoded in Stage 1. The module prepares all data paths for execution by configuring inputs to the ALU according to the operation type specified.

Stage 3: Execution

The execution is done by the ALU, which computes the result based on the operation specified and the inputs prepared from Stage 2 and MUX. The result of this computation is subsequently made ready to be passed onto the next stage of the pipeline.

Stage 4: Write Back

In the final stage, the S3_Register handles the write-back process. It receives the computed ALUResult from the execution stage and writes this result back to the register file, depending on the WriteEnable signal. This module finalizes the processing cycle for an instruction by updating the register file with the new data computed by the ALU. The S3_Register ensures that the results of computations are stored back into the designated registers, thus completing the pipeline.

Main Modules

1. **Pipeline:** The pipeline module is the top-level design that coordinates all stages of the pipelined processor. It manages the sequential flow of the instructions through each stage, including instruction fetch, decode, execute, and write-back.
    - **1.1. S1_Register:** This module handles the first stage of the pipeline. It receives the input instruction and separates its components (Opcode, registers, and immediate values) into signals to use in the subsequent stages.
    - **1.2. Register_File:** The register file acts as the storage for the registers. It allows multiple reads and a write operation, synchronized with the pipeline stages.
    - **1.3. S2_Register:** This module receives the outputs from S1_Register and the Register File and prepares them for execution in the ALU.
    - **1.4. MUX:** The multiplexer selects between immediate data from the instruction and data read from the register file based on the instruction type.
    - **1.5. ALU:** The Arithmetic Logic Unit performs all arithmetic and logical operations.
    - **1.6. S3_Register:** This module collects the results from the ALU and prepares them to be written back to the register file.

Rawisara Chairat

U65053623

Instruction Format

Type	Format (bits)				
R	Opcode (6)	R1(5)	R2(5)	R3(5)	
I	Opcode (6)	R1(5)	R2(5)	Immediate (16)	

ALU Functions

The ALU performs the following operations, controlled by the ALUOp signal:

ALUOp	Operation
3'b000	R1 = R2 (MOV)
3'b001	R1 = ~R2 (NOT)
3'b010	R1 = R2 + R3 (ADD)
3'b011	R1 = R2 – R3 (SUB)
3'b100	R1 = R2 | R3 (OR)
3'b101	R1 = R2 & R3 (AND)
3'b110	R1 = 1 if R2 < R3 (SLT)

Implementation

Steps in the design process:

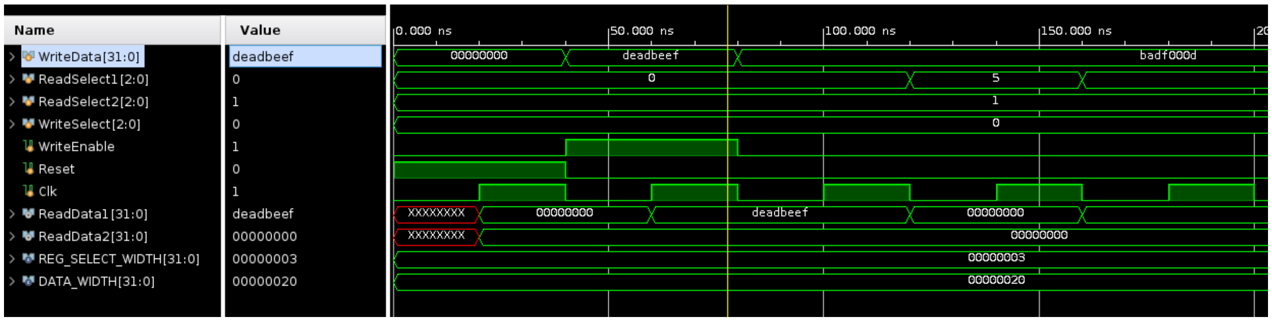
1. **Design the Pipeline Module:** Developed the top-level module to control the flow through the fetch, decode, execute, and write-back stages.
2. **Create Module:**
   - **S1_Register:** Decodes instructions and prepares control signals.
   - **Register File:** Enables multiple reads and a single write per cycle.
   - **S2_Register and MUX:** Set up data for ALU execution.
   - **ALU:** Performs computations.
   - **S3_Register:** Manages the write-back process.
3. **Connect all components:** Wired all stages correctly and synchronized them to maintain accurate data flow and timing across the pipeline.
4. **Testing:** Each module was individually tested to validate its functionality before full system integration.

Testing and Results

Register File Testing:

To verify the functionality of the register file, we use the provided testbench that simulates writing and reading operations. The primary objective was to confirm that data is correctly written to the registers when enabled and to validate the behavior of the write enable signal.

During the test, the WriteData input was set to 0xDEADBEEF, intended to be stored in the register. As illustrated in the waveform, this value is successfully written to the register at the rising edge of the clk signal when the WriteEnable signal is set to 1. This confirms that the register file correctly captures the input data on the clock edge where writing is enabled. Immediately following this operation, the WriteEnable is reset to 0, causing no further data to be written to the register. After resetting the WriteEnable signal, another data value 0xBADF000D was applied to the WriteData input. The waveform clearly shows that despite the presence of the new data on the WriteData line, it is not written to the register due to the WriteEnable being 0.



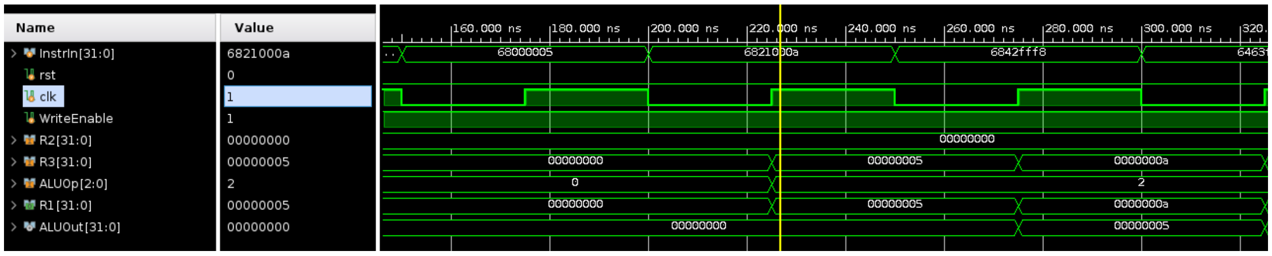
Name	Value
WriteData[31:0]	deadbeef
ReadSelect1[2:0]	0
ReadSelect2[2:0]	1
WriteSelect[2:0]	0
WriteEnable	1
Reset	0
Clk	1
ReadData1[31:0]	deadbeef
ReadData2[31:0]	00000000
REG_SELECT_WIDTH[31:0]	00000003
DATA_WIDTH[31:0]	00000020

Waveform 1: Register File Testing

ALU Testing:

To confirm the ALU's functionality across all operations, the pipeline testbench was utilized. A range of test cases was selected to cover all different operations and instruction types. The simulation waveforms verified that the ALU executed all operations correctly.

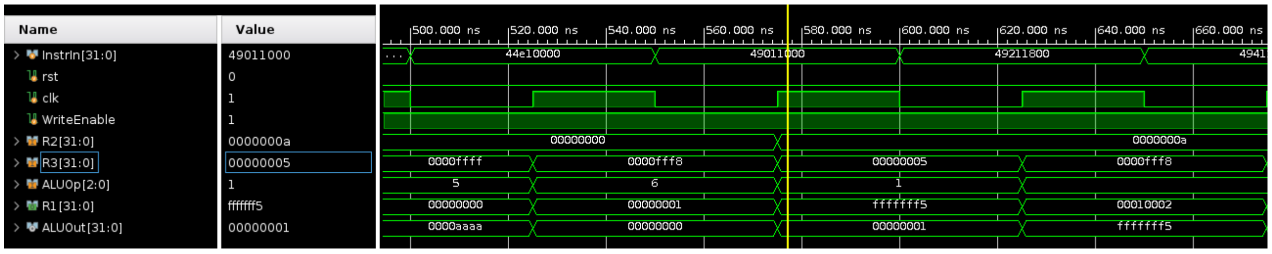
The first test instruction, 0x68000005 or 32'b011010_00000_00000_0000000000000101, involves adding an immediate value to Register 0 and storing the result back in Register 0. As shown in the waveform, one clock cycle after fetching the instruction, the ALU receives 0 (from Register 0) and 5 (the immediate value) as inputs. The operation code 2, which corresponds to the ADD operation, directs the ALU to add these operands. Consequently, the output from the ALU is 5, which, after one clock cycle, is forwarded to the register file as ALUOut.



Name	Value
InstrIn[31:0]	6821000a
rst	0
clk	1
WriteEnable	1
R2[31:0]	00000000
R3[31:0]	00000005
ALUOp[2:0]	2
R1[31:0]	00000005
ALUOut[31:0]	00000000

Waveform 2: ALU Addition with Immediate Value

The second test instruction or 32'b010001_00111_00001_00000_00000000000, performs a NOT operation on the value from Register 1, which contains 0x0000000A. This instruction specifies that the NOT operation's result should be stored back into Register 7. Upon receiving the instruction, the ALU executes the NOT operation, from operation code 1, by inverting all bits of the input value from Register 1. The original value 0x0000000A undergoes bit inversion, resulting in 0xFFFFFFF5. After the operation, the resulting value 0xFFFFFFF5 is latched into the ALUOut register file.

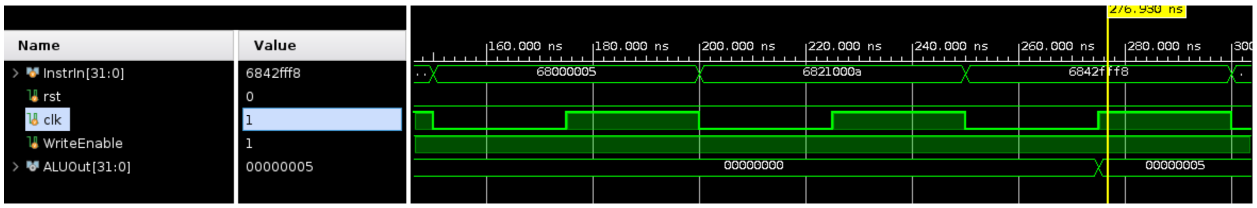


Name	Value
InstrIn[31:0]	49011000
rst	0
clk	1
WriteEnable	1
R2[31:0]	0000000a
R3[31:0]	00000005
ALUOp[2:0]	1
R1[31:0]	fffffff5
ALUOut[31:0]	00000001

Waveform 3: ALU NOT Operation on Register Value

Datapath Testing:

To ensure accurate data movement through the pipeline's various stages, the provided pipeline testbench is utilized. The testbench allows us to perform a timing analysis of the instructions as they progress through the pipeline stages. Initially, in the first clock cycle, the instruction is fetched from memory and then decoded. By the second clock cycle, this instruction reaches the execution stage where the ALU computes the necessary operations. Following this, the results of these computations are captured in ALUOut during the third clock cycle. This sequential progression through the stages results in a two-clock-cycle delay from when the instruction is fetched to when the computed result is visible in ALUOut. The observed timing in the waveform, which shows ALUOut indicating a result of 5 at the end of the second clock cycle post-execution, aligns perfectly with the expected behavior of such a pipeline model, confirming the pipeline's efficiency and the accuracy of our timing analysis.



Waveform 4: Datapath Testing

Conclusion

This lab successfully implemented and tested a 4-stage 32-bit pipelined datapath using behavioral Verilog, demonstrating effective design and functional verification of a complex digital system.