University of Engineering and Technology Lahore

Department of Electrical Engineering

Computer Architecture Lab Manual

Course Code: EE-475L

Spring 2026

Contents

Introduction

The goal of this lab is to familiarize students with the process of designing a RISC-V processor, beginning with a single-cycle implementation and then extending it into a pipelined architecture capable of handling control and data hazards. This course provides practical exposure to designing digital systems using RTL descriptions, resolving hazards in a simple pipeline, and building functional interfaces. Additionally, students will learn how to approach system-level optimization, bridging the gap between individual components and a fully integrated processor.

In tackling these challenges, your first step will be to map the high-level specification to a design that can be translated into a hardware implementation. Following that, you will produce and debug the implementation. These steps can take a significant amount of time if the design is not carefully planned before implementation.

By the end of this lab, students will have gained hands-on experience in processor design, hazard resolution, and system-level thinking, equipping them with essential skills for real-world digital system development. **We will basically follow this EECS151_SP24 document created by University of Berkeley to design the pipelined processor.**

Experiment 1

Design and Implementation of Arithmetic Logic Unit (ALU)

1.1 Overview

In this experiment, students will design and implement an ALU and its controller using SystemVerilog. The design will support RISC-V instructions and will be verified through simulation. This experiment serves as a foundational step toward building a complete RISC-V processor, as the ALU and its controller are critical components used throughout the datapath in later experiments.

1.2 ALU

The Arithmetic Logic Unit (ALU) is a fundamental component of a processor responsible for performing arithmetic and logical operations. These operations include addition, subtraction, logical AND/OR/XOR, comparison, and shift operations.

1.2.1 Block Diagram

Task 1: Draw the schematic / block diagram of the ALU shown in Figure 1.1 using draw.io.

The input and output port names are as shown in Figure 1.1. The ALU module takes two input operands and performs an operation specified by the control signal **alu_operation** and produces the **result**. In addition to the result, the ALU generates a **zero** flag, which is asserted when the result of a subtraction operation is zero.

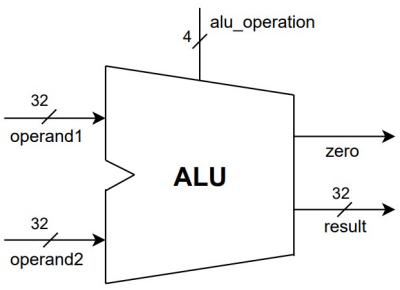

Figure 1.1: ALU

1.2.2 Understanding the RISCV ISA Cheat Sheet (Reference Manual)

Read and understand the first page of this Riscv Cheat Sheet. For more details you can refer to this reference manual.

1.3 Controller Design

The ALU controller determines which operation the ALU should perform for a given instruction. It takes different fields of the RISCV instruction such as **OPCODE, func3** etc. and translates them into a specific control code that drives the ALU. Before designing the controller, you should decide the names and size of the inputs and outputs of the controller. Figure 1.2 shows the controller you will be implementing. The output **alu_operation** depends on three inputs **func3**, **func7** and **ALUOp**.

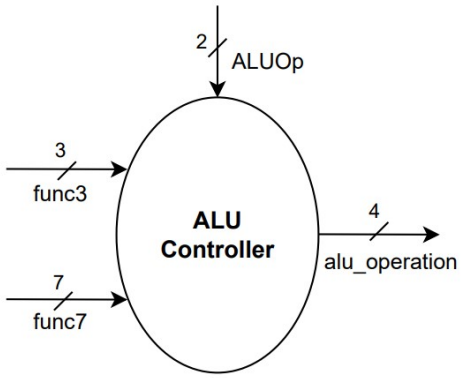

Figure 1.2: ALU Controller

Task 2: Look at the ISA cheat sheet and list down all the arithmetic operations in the first column of table 1.1, to be performed by the ALU. Decide the values and size of the output **alu_operation** in binary yourself. Use WORD or EXCEL to complete the table. The values given in the table is just an example.

Arithmetic Operations	alu_operation (Controller Output)
and	0000
or	0001
add	0010

Table 1.1:

The next task is to decide the values of the **ALUOp** signal. In figure 1.3 observe that **ALUOp** = 10 when the instruction is determined using **Opcode, func3 and func7** fields. **ALUOp** has a different value when all the fields are not used to determine the instruction. Use this strategy to complete table 1.3 for all the instructions.

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

Figure 1.3: ALU Control Signal Generation Based on `ALUOp` and Instruction Fields

Instrcution	func3	func7	ALUOp
and	000	0000000	10
sub			
xor			
or			
and			
slt			
sll			
sltu			
srl			
sra			
addi			
xori			
ori			
andi			
slti			
slli			
sltui			
srli			
srai			
lb/sb			
lh/sh			
lw/sw			

Table 1.2: Table

Acknowledgement

This manual is the result of the work of many including

1. Rehan Naeem (2026)
2. Tayyba Shafiq (2026)