

# Habib University



Dhanani School of Science and Engineering

**Computer Architecture**  
(CS-330 / EE-471)

**Lab Manual**

# Table of Contents

---

ABOUT THE LAB MANUAL .....	1
ABOUT THE LAB EXERCISES.....	3
CONVENTIONS .....	5
LAB 8 – DESIGNING A CONTROL UNIT OF RISC PROCESSOR .....	6
Objectives .....	6
a. Introduction.....	7
b. Implementation.....	7
<i>i. Lab Task 01</i> .....	7
<i>ii. Lab Task 02</i> .....	8
<i>iii. Lab Task 03</i> .....	8

## About the lab manual

---

This lab manual has been created with the help of practical experiments, several supporting documents and presentations listed in the Bibliography section.

The creation process of this manual is started during the summer 2018 by Dr. Hasan Baig, and this manual is continuously being updated.

For questions, comments, or suggestions, please contact Dr. Hasan Baig at the following email address: [hasan.baig@sse.habib.edu.pk](mailto:hasan.baig@sse.habib.edu.pk).





## About the lab exercises

---

These laboratory exercises have been designed to get the students acquainted with the hardware design skills. You will learn how to design hardware using hardware description language (HDL); how to simulate your design; and how to test it on a reconfigurable chip. Once you get familiar with the design flow, you will be required to develop processor peripherals in the following labs. A brief summary of all the lab exercises are given below.

In **Lab 1**, you will be introduced to the programmable logic and the Verilog HDL. Furthermore, you will learn how to design a simple hardware and verify its functional behavior using a professional simulation tool, named *ModelSim®*.

In **Lab 2a**, a hardware synthesis flow is discussed targeting the Xilinx FPGA technology. Furthermore, you will get to run your designed hardware on actual FPGA chip.

In **Lab 2b**, you will learn how to integrate the ready-made module (UART) with your custom design. Also, in this session, you will use desktop-based software, designed specifically for this course, to observe the output of on-chip hardware.

In **Lab 3**, you will be developing some intermediate modules of a processor which will be required in next labs. In particular, you will develop a multiplexer, an instruction parser, and immediate field extractor.

In **Lab 4**, In this lab, you will develop a Register File for a processor, and will simulate its behavior in ModelSim.

In **Lab 5**, you will develop a RISC arithmetic and logic unit and verify its functionality using simulation.

In **Lab 6**, you will develop modules for instruction and data memories.

In **Lab 7**, you will develop a datapath for instruction fetch cycle, followed by its verification.

In **Lab 8**, you will design a control unit of RISC processor.

In **Lab 9**, you will develop a single cycle RISC processor for R-type instructions and perform its behavioral simulation.

In **Lab 10**, you will design and test components for RISC processor that can handle branch instructions.

In **Lab 11**, you will design and test components for RISC processor that can handle memory reference instructions such as load and store.

In **Lab 12**, you will integrate the previously designed modules to form a single datapath for executing any type of instructions.





# Conventions

---

The following conventions appear in this lab manual.



This icon denotes a “pre-lab exercise”, which a student should complete before coming into the respective lab.



This icon denotes a “lab exercise”, which a student should complete during the lab hours.



This icon denotes a “post-lab exercise”, which a student should complete outside the lab hours.



This icon indicates the expected time (in minutes) to complete the specific exercise.



This icon denotes a tip, which notifies you to advisory information.



This icon denotes an alert, which notifies you to important information.

**Bold** or  
*Italic*

The text written in this font is used specifically for the syntax of HDL.

**bold**

Bold text denotes items that you must select or click or enter the value in the software, such as open file option or running the simulation button or entering the command in the transcript window. The bold text is also used to refer to the specific options in the software tools.

*italic*

Italic text denotes the name of a folder or a file path.




***bold and italic***

Bold and italic text denotes the name of a file.

# Lab 8 – Designing a Control Unit of a RISC Processor

## Objectives

In this lab, we will develop an instruction-fetch datapath and verify its functionality.

Section	
a) <u>Introduction</u>  A brief overview of this lab.	10
b) <u>Implementation</u>  In this section, you will implement a control unit of a RISC-V processor.	90







## a. Introduction

The last module we are left with is the control unit, which we have to design before we proceed further to integrate the already-designed processor components. In this lab, we will develop a module for generating control signals for specific instructions. These control signals are used to control the data flow and enabling or disabling the modules which are not needed for specific instructions.

## b. Implementation

Besides control unit, we also have to develop ALU Control unit to set the control signals of ALU.

### i. Lab Task 01

As shown in Fig. 8.1, Write a module, named `Control_Unit`, which takes a 7-bit wide input, named `Opcode`, and generate 7 output signals. Out of these seven outputs, one is `ALUOp` which is 2-bits wide, and the remaining six are 1-bit wide, which are `Branch`, `MemRead`, `MemtoReg`, `MemWrite`, `ALUSrc`, and `RegWrite`.

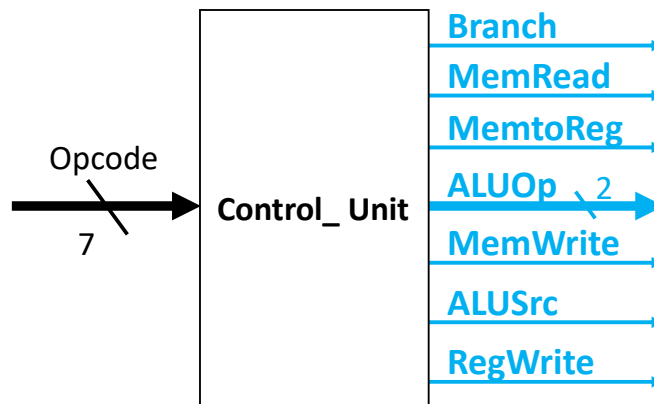


Fig. 8.1. I/O diagram of Control Unit.

The behavior of `Control_Unit` module should be designed according to the Table 8.1. The outputs depend only on the input, `Opcode`.

Table. 8.1. This table shows how the control signals are set based on the input values of `Opcode`.

Instruction Type	Opcode	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp [1:0]
R-Type	0110011	0	0	1	0	0	0	10
I-Type (ld)	0000011	1	1	1	1	0	0	00
I-Type (sd)	0100011	1	X	0	0	1	0	00
SB-Type (Beq)	1100011	0	X	0	0	0	1	01





### ii. Lab Task 02

Write a module, named `ALU_Control`, which takes a 2-bit input, named `ALUOp` and a 4-bit input, named `Funct`, and produces a 4-bit output, named `Operation`, as shown in Figure. 8.2.

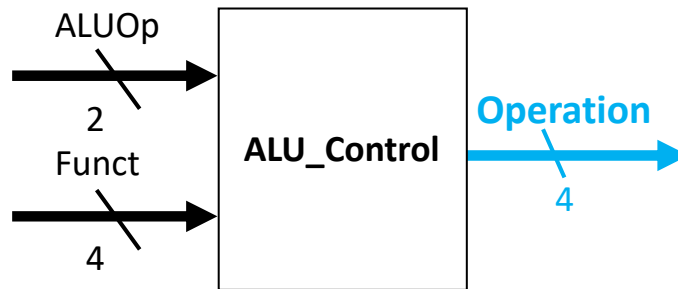


Fig. 8.2. I/O diagram of `ALU_Control` module.

The values of the output, `Operation`, should be set based on the input signals, `ALUOp`, as shown in Table 8.2.

Table. 8.2. The values of the output, `Operation`, based on the corresponding input, `ALUOp`, signals.

Instruction Type	ALUOp [1:0]	Funct	Operation
I/S-Type (ld, sd)	00	xxxx	0010
SB-Type (Beq)	01	xxxx	0110
R-Type	10	0000	0010
		1000	0110
		0111	0000
		0110	0001

### iii. Lab Task 03

Integrate above two modules in a top module, named `top_control`. Write a testbench and perform verification.