

# Habib University



Dhanani School of Science and Engineering

**Computer Architecture**  
(EE-371 / CS-330)

**Lab Manual**

# Table of Contents

---

ABOUT THE LAB MANUAL .....	1
ABOUT THE LAB EXERCISES.....	3
CONVENTIONS .....	5
LAB 7 -REGISTER FILE.....	7
Objectives .....	7
a. Introduction.....	8
b. Implementation.....	8
c. Design Requirements .....	9
Exercise.....	10

## 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).





# 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 7 – Register File

## Objectives

In this lab, we will develop a Register File for a processor, and will simulate its behavior in ModelSim.

Section	
a) <u>Introduction</u>  A brief overview of a Register File.	05
b) <u>Implementation</u>  In this section, you will implement the Register File module.	60
<u>Exercise</u>  Small exercise to practice further.	60



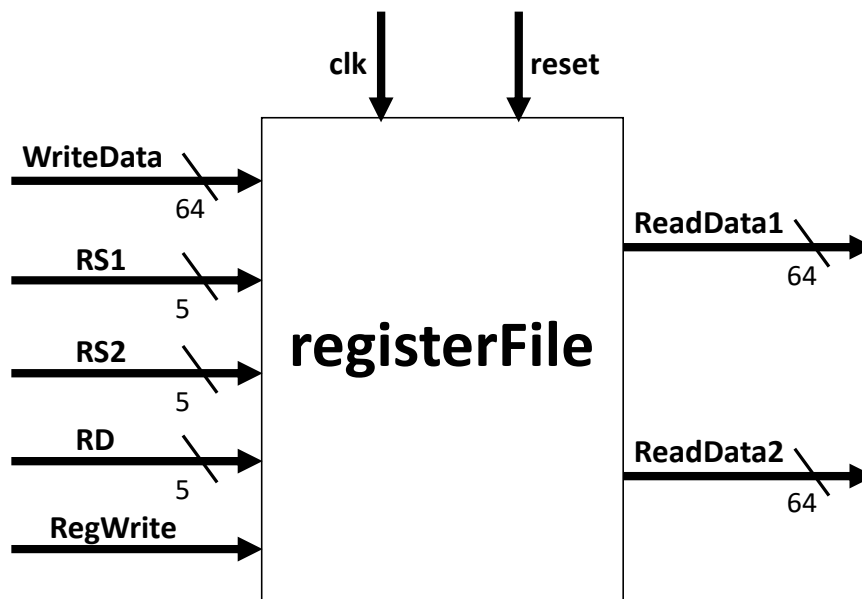
## a. Introduction

Register files are necessary for computer memory. For a computer to be able to function, it needs to have some form of memory. There are two different forms of memory devices: external memory devices and local memory devices. External memory devices are items such as RAM, ROM, disk drives, etc. Registers provide local memory, which allows for temporary storage of data that is about to be processed. For this lab, registers will be used to save and store numbers.

## b. Implementation

Create a module named, *registerFile*, which should have input address ports for reading register 1, register 2, and for writing data in a destination register. Furthermore, it should have a 64-bits data input port, and two 64-bits data output ports to read the values from two different registers. Finally, it should have a *clk*, *reset*, and a *RegWrite* signal which controls when the data should be written to a register. You need to define 32 64-bit internal registers.

The top-level diagram of this module is shown in the figure below.



As an example, the following command creates an Array of type `reg` containing 10 elements, each of which are 5 bits wide.



```
reg [4:0] Array [9:0]
```

where `Array` is the name of a variable. In your case, it should be `Registers`.





### c. Design Requirements

- Initialize Registers with random values.



Use `initial` block to initialize the Registers with any random value.



To verify correct functioning of your design using test bench, make sure that different values are initialized in different Registers.

- The operation of writing data into a Registers should always be done when there is a positive edge of `clk` and `RegWrite` signal is asserted (or set i.e. HIGH).
- On `reset`, place a value 0 at both `ReadData` output ports.
- Reading a data from the register file should be made independent of `clk` signal. Reading should rather be sensitive to the change in inputs `RS1`, `RS2`, or `reset`.

Create a test bench to verify the correct functioning of designed module in ModelSim.



## Exercise

Create a top module having 32-bit input, named `instruction`, and two 64-bit outputs, named `ReadData1` and `ReadData2`. Now, instantiate `instruction parser` module developed in lab 3 and `registerFile` module developed in this lab. Make appropriate connections.

Create a testbench and pass any 32-bit `instruction` in top module, and observe if the data of correct source registers are obtained.