Habib University

Dhanani School of Science and Engineering

Computer Architecture

(CS-330 / EE-471)

Lab Manual

Table of Contents

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.

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 develop a single cycle RISC processor for R-type instructions and perform its behavioral simulation.

In **Lab 9**, you will design and test components for RISC processor that can handle branch instructions.

In **Lab 10**, you will design and test components for RISC processor that can handle memory reference instructions such as load and store.

In **Lab 11**, you will integrate the previously designed modules to form a single datapath for executing any type of instructions.

In **Lab 12**, you will design a control unit of RISC processor and then integrate it with the previously developed complete datapath.

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 – Design of an Instruction-Fetch Datapath

Objectives

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

Section	⏱
a) Introduction A brief overview of this lab.	**10**
b) Implementation In this section, you will implement a single-cycle RISC processor for R-type instructions and simulate its behavior.	**80**

a. Introduction

A reasonable way to start a datapath design is to examine the major components required to execute each class of RISC-V instructions. First, an instruction has to be fetched, which requires the components shown in the following figure.

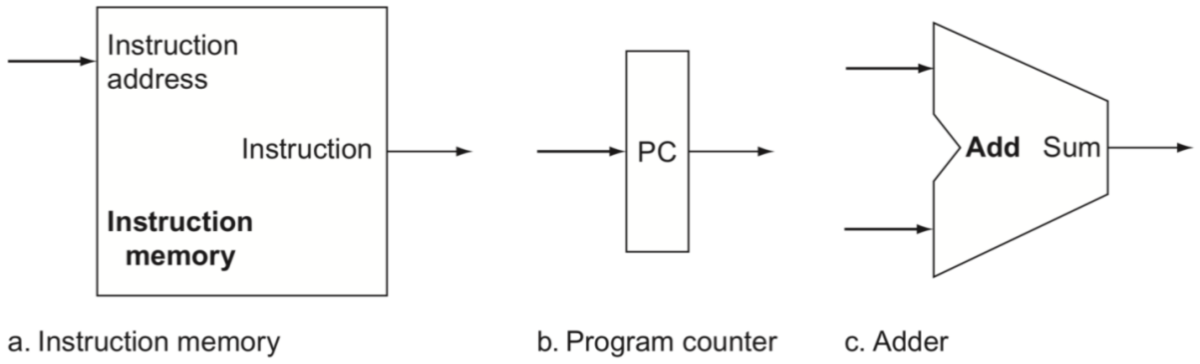

Fig. 6.1. Required components for fetching a processor instruction.

As shown in Fig. 6.1., two state elements are needed to store and access instructions, and an adder is needed to compute the next instruction address. The state elements are the instruction memory and the program counter. The instruction memory need only provide read access because the datapath does not write instructions. Since the instruction memory only reads, we treat it as combinational logic: the output at any time reflects the contents of the location specified by the address input, and no read control signal is needed. The program counter is a 64-bit register that is written at the end of every clock cycle and thus does not need a write control signal. The adder is an ALU wired to always add its two 64-bit inputs and place the sum on its output.

a. Implementation

We already have developed an instruction memory in Lab05. Now, we will start off with developing the separate modules of a program counter (PC) and a two-input adder.

i. *Lab Task 01*

PC Write a module, named `Program_Counter`, which takes three inputs – `clk`, `reset`, a 64-bit input, `PC_In`; and a 64-bit output, `PC_Out`. Initialize `PC_Out` to 0 if reset signal is high, else reflect the value of `PC_In` to `PC_Out`, at the positive edge of clock.

ii. *Lab Task 02*

Adder takes two 64-bits inputs, `a` and `b`; add them, and reflect the results at the 64-bit output port, `out`.

iii. Lab Task 03

Now connect the above two modules and an `Instruction_Memory` (developed in Lab06) to construct an instruction fetch datapath as shown below. Name the module `Instruction_Fetch`; instantiate all three modules and make necessary connections as shown in Fig. 6.2.

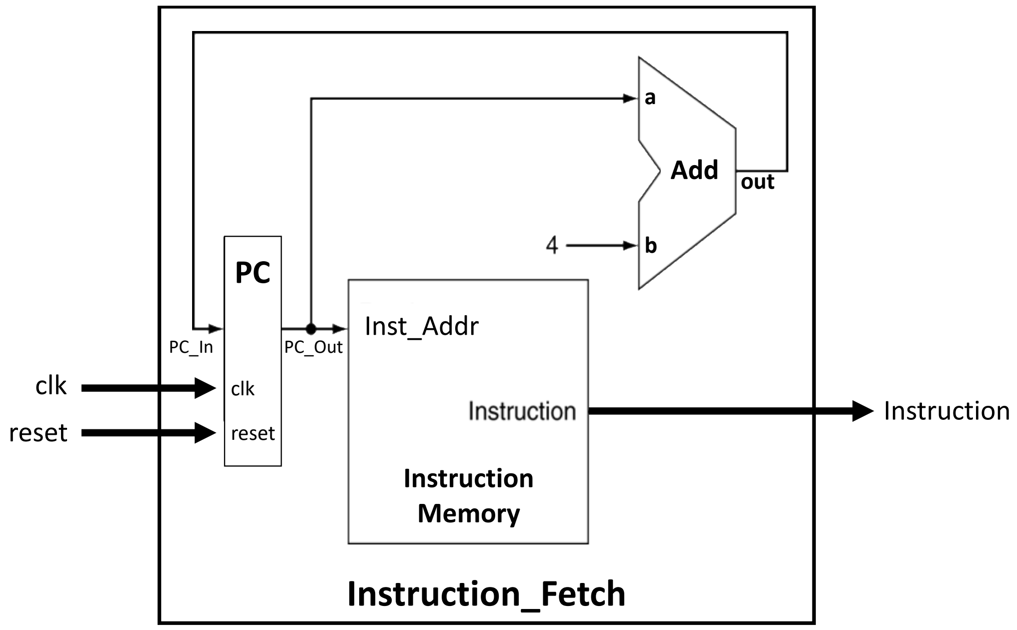

Fig. 6.2. Instruction fetch datapath.

Write a testbench and verify the functionality of instruction fetch datapath.