Lab 9 – 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.

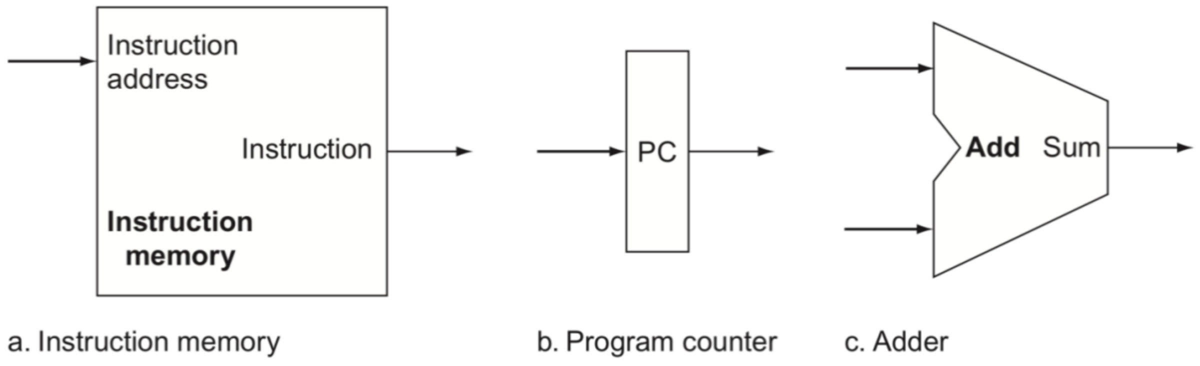

Fig. 9.1. Required components for fetching a processor instruction.

As shown in Fig. 9.1., 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 Lab08. Now, we will start off with developing the separate modules of a program counter (PC) and a two-input adder.

i. *Lab Task 01*

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 Lab08) 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. 9.2.

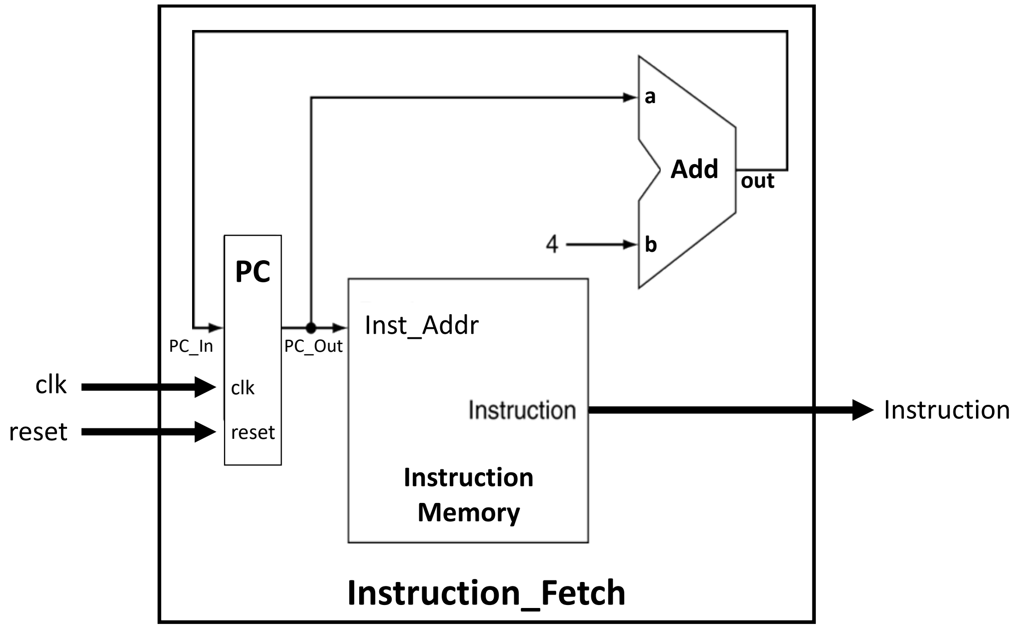

Fig. 9.2. Instruction fetch datapath.

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

Computer Architecture

Spring 20

Lab 09

Marks Distribution:

Task	**LR2: Design/Code**	**LR5: Results**
Task 1 (Program Counter)	10 Points	-
Task 2 (Adder)	10 Points	-
Task 3 (Instruction Fetch)	20 points	10 Points
	Total	50 Points

Marks Obtained:

Task	**LR2: Design/Code**	**LR5: Results**
Task 1 (Instruction Memory)		
Task 2 (Data Memory)		
	Total	