

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 6 – DESIGN OF INSTRUCTION AND DATA MEMORIES	7
Objectives	7
a. Introduction.....	8
b. Implementation.....	9
i. <i>Lab Task 01</i>	9
Exercise.....	11

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

In **Lab 8**, you will design and test components for RISC processor that can handle branch instructions.

In **Lab 9**, you will design and test components for RISC processor that can handle memory reference instructions such as load and store.

In **Lab 10**, you will integrate the previously designed modules to form a single datapath for executing any type of instructions.

In **Lab 11**, 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 6 – Design of Instruction and Data Memories

Objectives

In this lab, we will develop modules for instruction and data memories.

Section	
a) <u>Introduction</u>  A brief overview of this lab and some background about the instruction and data memories	05
b) <u>Implementation</u>  In this section, you will develop a module for instruction memory and verify its functionality.	60
<u>Exercise</u>  In this section, you will develop and verify the module of Data Memory.	60





a. Introduction

Instruction and Data memories are essential components of a processor. We will develop modules for the Instruction and Data memory separately. The instruction memory is used to store instructions. During the processor execution, it is required to only read the instruction from the instruction memory and not to write any data or instruction in it. Hence, we could say that the instruction memory is the read-only memory. However, the data memory is used to either read or write data.

Recall that the size of each instruction is 32-bits, and the data (to be written in memory or to obtain from memory) is 64-bits wide. Therefore, the size of each location in an instruction memory is 32-bits and that of data memory is 64-bits.

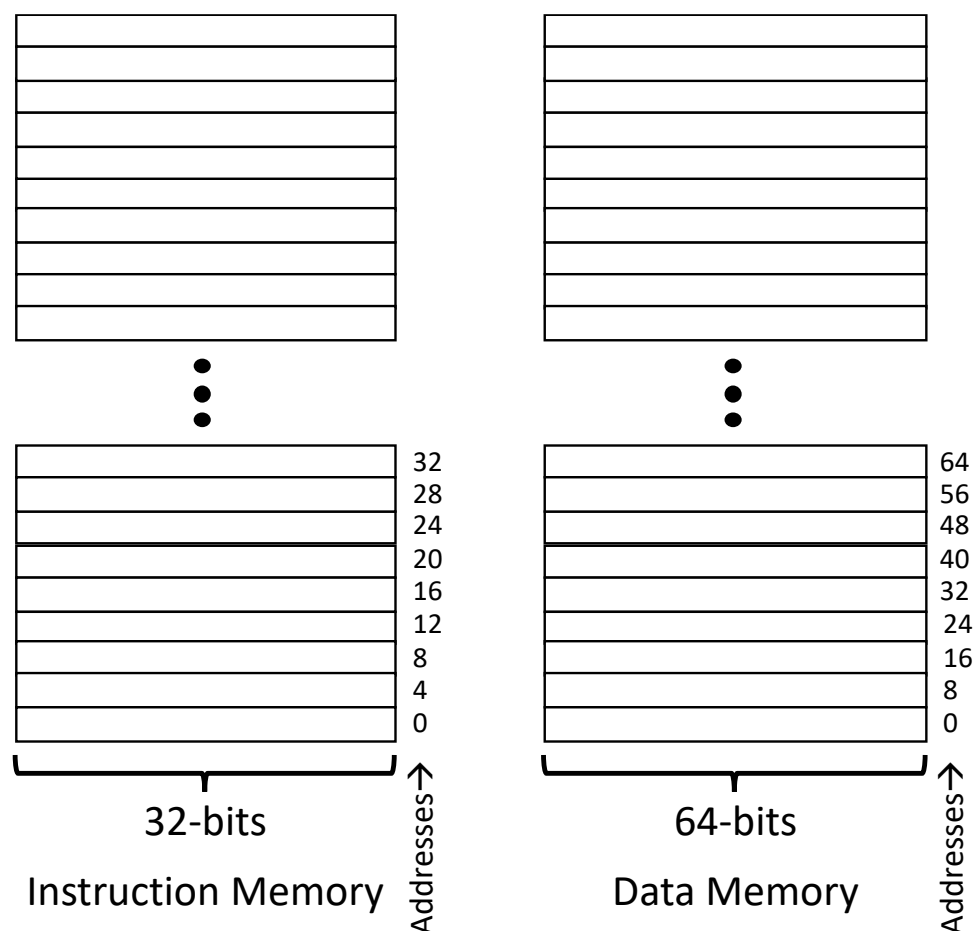


Fig. 6.1. Instruction and Data memories.

In lectures, we have been conceptually visualizing a memory as a stack of memory locations, as shown in Fig. 6.1. Each location in instruction memory is 32-bits (4 bytes) wide and that of data memory is 64-bits (8 bytes) wide.



b. Implementation

In this lab, we will design an instruction memory with each location 8-bits wide. Hence, four memory locations will be required to store a 32-bit instruction in an instruction memory, as shown in Fig. 6.2.

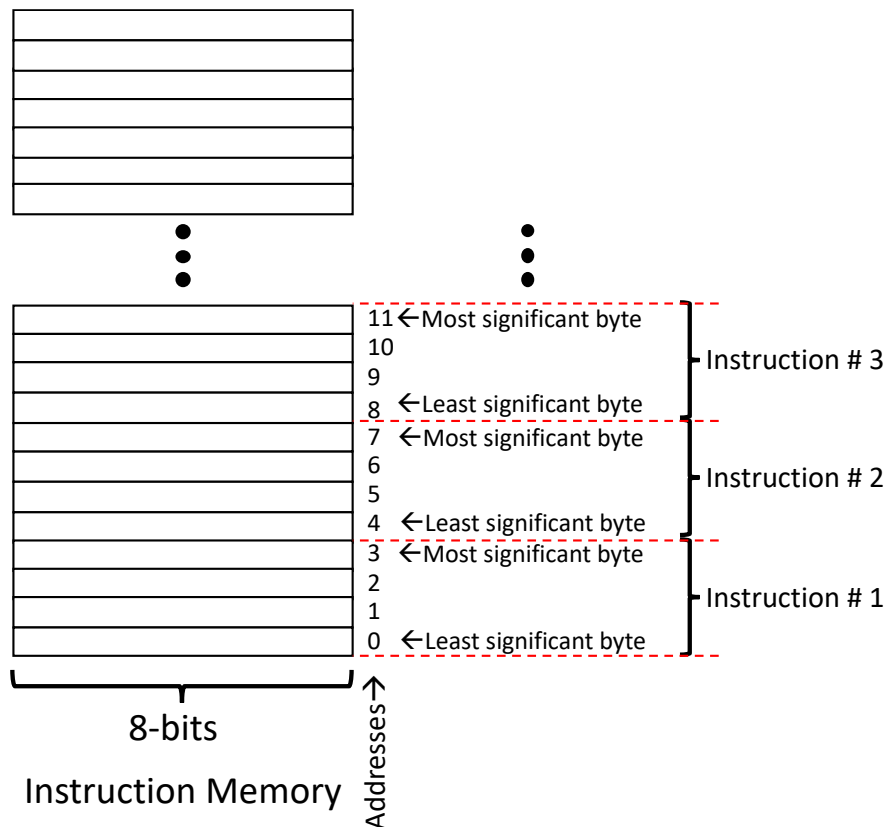


Fig. 6.2. 8xN bytes instruction memory, where N specify the number of n memory locations each of which are 8-bits (or 1-byte) wide.

i. Lab Task 01

Design a module named, `Instruction_Memory`, having a 64-bit input, `Inst_Address`, and a 32-bit output, `Instruction`. Now, declare a 1x16 bytes instruction memory internally in the module and initialized its contents as shown in Fig. 6.3.



Similar procedure is required to declare an instruction memory that you followed while constructing a register file in lab 04.

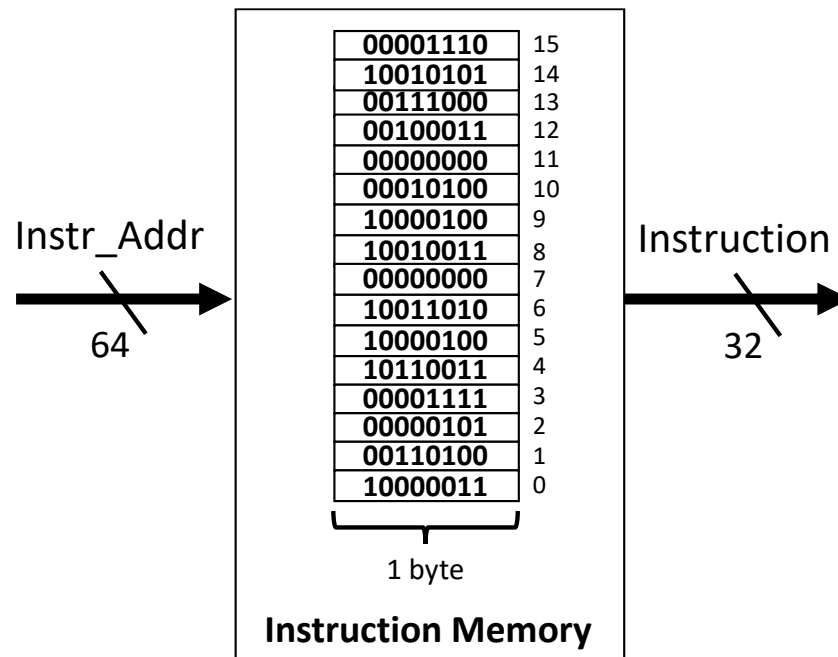


Fig. 6.3. 1x16 bytes (or 8x16 bits) instruction memory.



It should be noted that the 64-bits width of `Inst_Addr` input can access 2^{64} memory locations. Although, we need to have this input 4 bits wide only (to access 16 memory locations), but we are keeping its width to 64 bits just to make synchronization with other modules, which we will develop in upcoming labs.

The behavior of this module should be designed as such that whenever an `Inst_Address` field is changed, the 32-bit instruction corresponding to the `Inst_Address` should appear at the 32-bit output port, `Instruction`.

For example, if the `Inst_Address` is 0, the consecutive 4 bytes (byte no. 0 to 3) should appear at the 32-bit `Instruction` output. Similarly, when the value of `Inst_Address` is 8, the corresponding next four bytes (8 to 11) should be placed at the 32-bit output port, `Instruction`.



While placing four bytes on the 32-bit `Instruction` port, make sure that the bytes are in correct order. That is, the least-significant byte should be the right most byte and that the most-significant byte should be the left most byte in the 32-bit `Instruction` output.

Write a testbench and verify the functionality of this module.



Exercise

The top-level diagram of a Data Memory is shown in Fig. 6.4. Write a module named, `Data_Memory`, with inputs and outputs as shown in Fig. 6.4. The signals shown in blue color are the control signals. We will design the control unit in upcoming labs.

The data at the input port, `Write_Data`, should only be written at the positive edge of `clk` signal and when `MemWrite` signal is asserted (i.e. High). The data can be read from memory at any instant whenever the `Mem_Addr` value changes and when `MemRead` signal is asserted.

In contrast to instruction memory, here 8 consecutive bytes should be placed at the output. For example, if the value of `Mem_Addr` is 0, the data placed at memory locations 0 to 7 should be placed at the output port `Read_Data`. Similarly, if `Mem_Addr` is 16, the data of memory locations 16 to 23 should be placed at the output port `Read_Data`.

Initialized the memory with random data, write a testbench and verify its functionality.

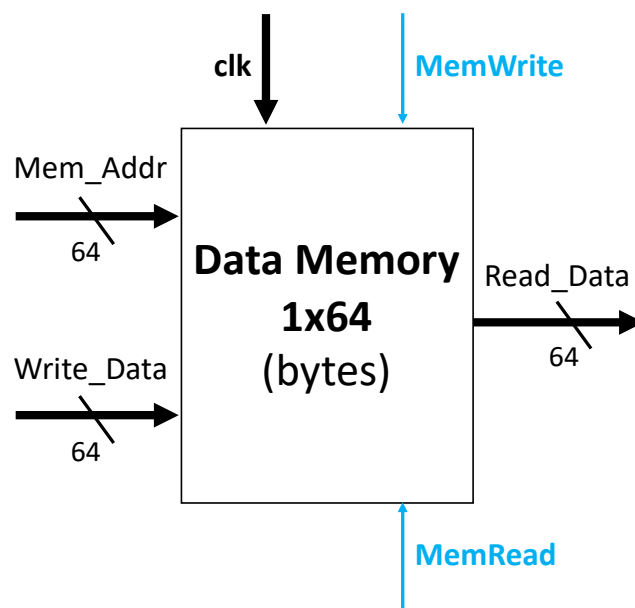


Fig. 6.4. IOs of Data Memory module. There are 64 memory locations, each of which is 1 byte wide.