






# Lab 5 – Basic Processor Modules

## Objectives

In this lab, we will be developing some intermediate modules of a processor which will be required in next labs. In particular, we will develop a multiplexer, an Instruction parser, and an Immediate field extractor.

Section	
a) <u>Introduction</u> 	05
A brief overview of the lab exercises and some additional Verilog elements are discussed which will be required to complete this lab.	
b) <u>Task 1: Multiplexer</u> 	25
In this section, you will develop a Verilog module of 2x1 multiplexer, its testbench and simulate it.	
c) <u>Task 2: Instruction Parser</u> 	25
In this section, you will develop a module to parse 32-bit instruction into 6 different output fields.	
d) <u>Task 3: Immediate Data Generator</u> 	60
In this section, you will develop a module to obtain the 12-bits <i>immediate</i> data from the 32-bit input for different instruction format and then sign-extend it to 64 bits output.	





## a. Introduction

From now on, you will mostly be developing modules and simulating the results using Verilog HDL. In this lab you are required to develop a multiplexer, an instruction parser, and an immediate data extractor.



To complete this lab, you may need to use *if/else structure*, *case structure*, *concatenation operator* or *assignment operator* in Verilog. If you are already familiar with the syntax, jump directly to section b.

### i. Case Structure

We already have seen the syntax and usage of if-else structure in Lab01 (D\_FF implementation). The if-else structure can also be nested similar to traditional programming languages. However, the nested if-else-if can become unwieldy if there are too many alternatives. A shortcut to achieve the same result is to use the case statement.

The syntax of Case conditional structure, in Verilog, is shown below. The keywords `case`, `endcase`, and `default` are used in the case statement.

```
case (expression)
    alternative1: statement1;
    alternative2: statement2;
    alternative3: statement3;
    ...
    default: default_statement;
endcase
```

Each of `statement1`, `statement2`, `default_statement` can be a single statement or a block of multiple statements. A block of multiple statements must be grouped by keywords `begin` and `end`. The expression is compared to the alternatives in the order they are written. For the first alternative that matches, the corresponding statement or block is executed. If none of the alternatives matches, the `default_statement` is executed.



Placing of multiple default statements in one case statement is not allowed.



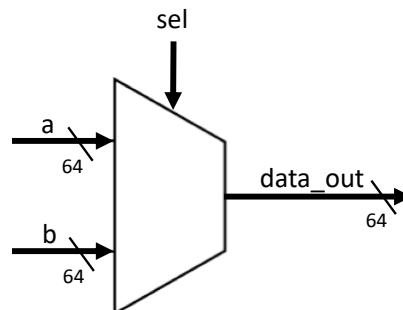
The `default_statement` is optional.





### b. Task 1: Multiplexer

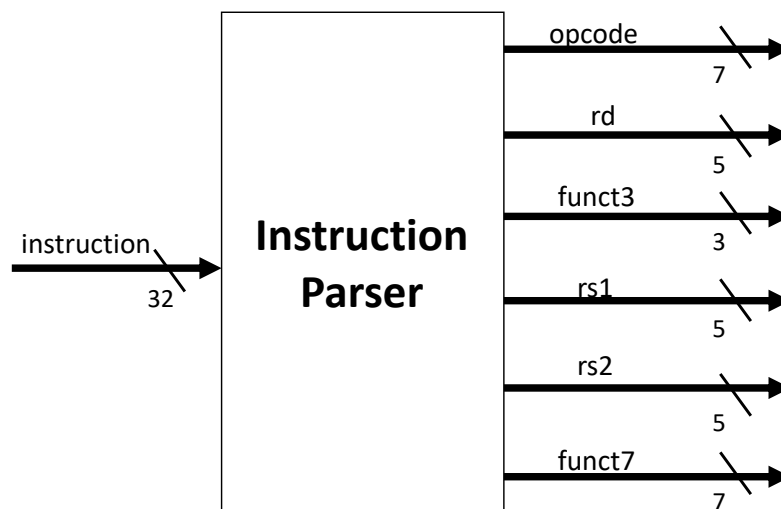
Develop a 2x1 multiplexer in which the two inputs are `a` and `b`, and each of them are 64-bits wide, as shown in the following figure.



Write a testbench to simulate its behavior.

### c. Task 2: Instruction Parser

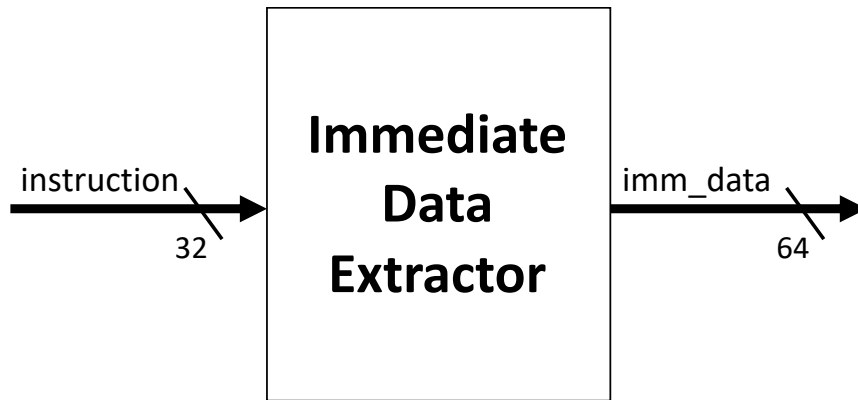
Develop a module which takes a 32-bit input, named `instruction`, and generates different outputs as shown in the following figure.



Assign the specific bits of `instruction` to the corresponding output field. Write a testbench to simulate its behavior.

### d. Task 3: Immediate Data Generator

Develop a module which takes the 32-bit input `instruction` and extracts the 12-bit immediate data field depending on the type of instruction. Then sign-extend these 12-bits to 64-bits output `imm_data`, as shown in the following figure.



The immediate generation logic must choose between sign-extending a 12-bit field in instruction bits 31:20 for load instructions, bits 31:25 and 11:7 for store instructions, or bits 31, 7, 30:25, and 11:8 for the conditional branch. Since the input is all 32 bits of the instruction, it can use the opcode bits of the instruction to select the proper field.

RISC-V opcode bit 6 happens to be 0 for data transfer instructions and 1 for conditional branches, and RISC-V opcode bit 5 happens to be 0 for load instructions and 1 for store instructions. Thus, bits 5 and 6 can control a 3:1 multiplexor inside the immediate generation logic that selects the appropriate 12-bit field for load, store, and conditional branch instructions.

Write a testbench for this module and verify its functionality.

**Assessment Rubric**  
**Computer Architecture Lab**  
**Lab 05**  
**Basic Processor Modules**

<b>Name:</b>	<b>Student ID:</b>
--------------	--------------------

**Points Distribution**

Task No.	LR 2 Code	LR 5 Results	AR 7 Report Submission
Task 1 (Mux)	/10	/10	/20
Task 2 (Instruction Parser)	/20	/10	
Task 3 (Immediate Data Generator)	/20	/10	
Total Points	/100 Points		
CLO Mapped	CLO 1		

*For description of different levels of the mapped rubrics, please refer to the provided Lab Evaluation Assessment Rubrics.*

#	Assessment Elements	Level 1: Unsatisfactory Points 0-1	Level 2: Developing Points 2	Level 3: Good Points 3	Level 4: Exemplary Points 4
LR2	<b>Program/Code/ Simulation Model/ Network Model</b>	Program/code/simulation model/network model does not implement the required functionality and has several errors. The student is not able to utilize even the basic tools of the software.	Program/code/simulation model/network model has some errors and does not produce completely accurate results. Student has limited command on the basic tools of the software.	Program/code/simulation model/network model gives correct output but not efficiently implemented or implemented by computationally complex routine.	Program/code/simulation /network model is efficiently implemented and gives correct output. Student has full command on the basic tools of the software.
LR5	<b>Results &amp; Plots</b>	Figures/ graphs / tables are not developed or are poorly constructed with erroneous results. Titles, captions, units are not mentioned. Data is presented in an obscure manner.	Figures, graphs and tables are drawn but contain errors. Titles, captions, units are not accurate. Data presentation is not too clear.	All figures, graphs, tables are correctly drawn but contain minor errors or some of the details are missing.	Figures / graphs / tables are correctly drawn and appropriate titles/captions and proper units are mentioned. Data presentation is systematic.
AR7	<b>Report Content/Code Comments</b>	Most of the questions are not answered / figures are not labelled/ titles are not mentioned / units are not mentioned. No comments are present in the code.	Some of the questions are answered, figures are labelled, titles are mentioned and units are mentioned. Few comments are stated in the code.	Majority of the questions are answered, figures are labelled, titles are mentioned and units are mentioned. Comments are stated in the code.	All the questions are answered, figures are labelled, titles are mentioned and units are properly mentioned. Proper comments are stated in the code.

