# Introduction

Introduction with problem overview, your design procedure, and rationale.

# Interface

This section explains the input and output relationships of the design, so it can be treated as a black box. I have essentially provided you the inputs, outputs, and inouts, but I want to know you understand why. What inputs are used and how? What outputs are used and how? What sequence of commands need to be performed in order to operate the module? What form does the data get sent in (binary, two’s compliment, excess code, etc.)? How are they grouped?

Control.v

|  |  |
| --- | --- |
| *Name* | *Function* |
|  | **Input** |
| opcode | Instructions of what the control unit should perform. |
|  | **Output** |
| EX | 4-bit control bits that corresponds to the execution/address calculation stage. |
| M | 3-bit control bits that corresponds to the memory access stage. |
| WB | 2-bit control bits that corresponds to the write-back stage. |

Each EX, M, and WB contain the R-format, lw, sw, and beq instructions. These signals form a 9-bit Control bit output.

Sign\_Extend.v

|  |  |
| --- | --- |
| *Name* | *Function* |
|  | **Input** |
| nextend | 16-bit immediate value coming from IF\_ID\_Latch.v (Instruction fetch decoder). |
|  | **Output** |
| extend | Extend the 16-bit immediate value to 32-bits. Sent to the Instruction Register. |

Register.v

|  |  |
| --- | --- |
| *Name* | *Function* |
|  | **Input** |
| rs | 5-bit read register 1. Possible input field of values from the Instruction Fetch Decoder instructions. |
| rt | 5-bit read register 2. Possible input field of values from the IF\_ID\_instr. |
| rd | 5-bit register write from the MEM/WB latch. |
| writedata | 32-bit data write from the WB Multiplexer. |
| regwrite | Comes from the control signal MEM/WB in I\_Decode.v |
|  | **Output** |
| A | 32-bit data 1 that is sent to the Execution ALU. |
| B | 32-bit data 2 that is sent to the Execution Multiplexer and Execution/Memory Latch. |

ID\_Extend.v (Instruction Decoder Extend Pipeline)

|  |  |
| --- | --- |
| *Name* | *Function* |
|  | **Input** |
| ctlwb\_out | Input the output from the Control Unit’s write-back signal. |
| ctlm\_out | Input the output from the Control Unit’s memory signal. |
| ctlex\_out | Input the output from the Control Unit’s Execution signal. |
| npc | Takes the new program control from the Instruction Fetch Decoder. |
| readdat1 | Takes the output of the data read from register 1. |
| readdat2 | Takes the output of the data read from register 2. |
| signext\_out | Takes the 32-bit sign extended output from the sign extend module. |
| instr\_2016 | 4-bit instruction of indexes [20-16] (reverse bits in the bus) from the IF\_ID latch. |
| instr\_1511 | 4-bit instruction of indexes [15-11] from the IF\_ID latch |
|  | **Output** |
| wb\_ctlout | Act as pipeline and sends the Control Unit output to the write-back stage. |
| m\_ctlout | Act as pipeline and sends the Control Unit output to the memory stage. |
| regdst | ??? |
| alusrc | ??? |
| aluop | ??? |
| npcout |  |
| rdata1out |  |
| rdata2out |  |
| s\_extendout |  |
| instrout\_2016 |  |
| instrout\_1511 |  |

This module acts as a pipeline between

I\_Decode.v (Instruction Decode)

|  |  |
| --- | --- |
| *Name* | *Function* |
|  | **Input** |
| IF\_ID\_instrout | Instruction opcode passed to the Control Unit that specifies what operation the CU should perform. |
| IF\_ID\_npcout | Passed to the pipeline ID\_Extend.v npc variable. |
| MEM\_WB\_rd | Wire to the 5-bit write register input in the Register module. |
| MEM\_WB\_regwrite | Wire to the 1-bit register write in the Register module. |
| WB\_mux5\_writedata | Wire to the 32-bit writedata input in the Register module. |
|  | **Output** |
| wb\_ctlout |  |
| m\_ctlout |  |
| regdst |  |
| alusrc |  |
| aluop |  |
| npcout |  |
| rdata1out |  |
| rdata2out |  |
| s\_extendout |  |
| instrout\_2016 |  |
| instrout\_1511 |  |

# Design

This is the internal design of the item. Design description and explanation, including any pictures, charts, etc.

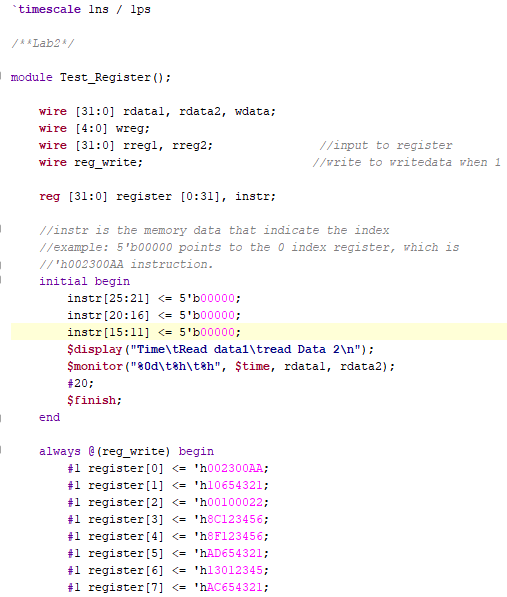
# Implementation

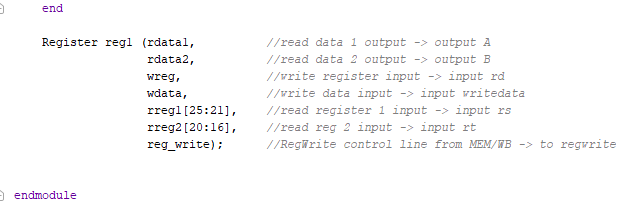
The Verilog code and explanations of why you implemented this way. There are many ways to implement a given design in verilog. For instance why choose a case statement or ifs? Why did you trigger on a negedge verses any signal change?

# Test Bench Design

This is where you discuss the test benches you wrote, and what they were designed to test. You should discuss expected errors as well as random errors. Be sure to include your Verilog code.

Test\_Register



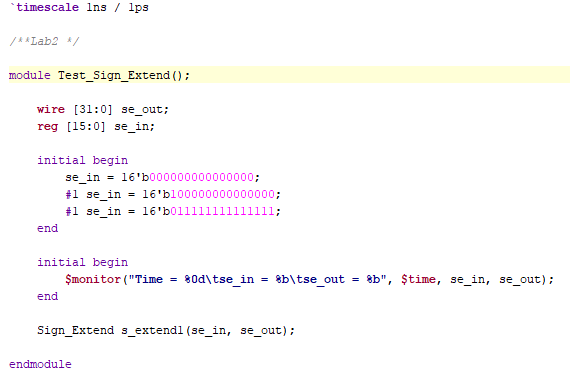


The instr is initialize with the index that would be given to the register. The register holds the MIPS instructions.

Test\_Control



Test\_Sign\_Extend

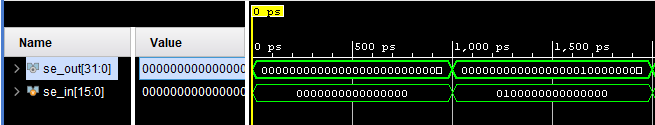


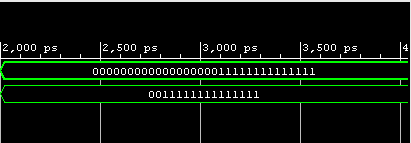
This test the se\_in 16-bits values which will then be extended to 32-bits with the most significant bit to the left.

# Simulation

In this section you should show the results of your simulation, such as timing diagrams and explain any design issues you had to deal with.

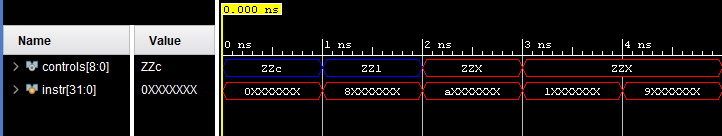
*Sign Extend test simulation:*

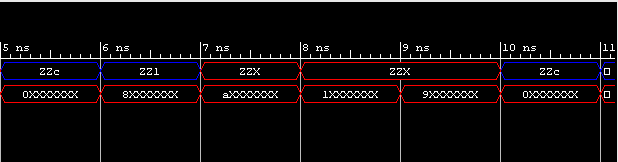


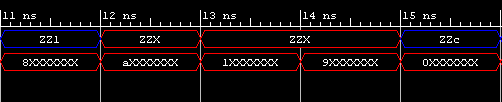


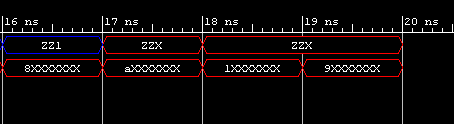
There is a problem in the second test where the 1 is not placed on the outside left (where the most significant bit should be). We compared our test and module code with other groups, which both were the same to the other groups. However, our simulation showed a different result.

*Control test simulation:*









The results show unknown/don’t care conditions and high impedance. We don’t know if this is intentional or an issue.

*Register test simulation:*

# Conclusions

Overview the main points you want to stick in people’s minds and answer key questions you want to stick in people’s minds. Did it work? How well? What would you have done differently? What did you learn?