Laboratory 7

Validating CPU Building Blocks

Due Date: Beginning of class July 13

Objectives:

- Validate common building blocks for use in a 32-bit MIPS CPU.

Assignment

It is common practice to incorporate existing Verilog files into a new design. These may belong to the company, available as free, open source material online, or purchased from a vendor (such places do exist). In all cases, a wise hardware designer would still validate the files to make sure the functionality is correct and matches the intended use. This lab will simulate that process.

Several common MIPS32 data path components are specified below. Pay attention to the module declarations. When instantiating the modules, the test benches must match the module names and port names ***exactly***. Files will be provided on Canvas that implement the components that you may use to validate your test bench. To be compliant with the requirements, you may not alter the provided files, and original copies of the modules will be used when validated the lab submission.

For each component described below, develop a self-checking test bench that meets the following parameters:

- It verifies the correct operation of the component according to the operational descriptions provided.
- The test bench must use a sufficient number of input combinations from a test vector file. *Note: In most cases, there will be too many input combinations to explicitly check all of them, so "sufficient" means a subset of possible inputs that checks for each case described. For example, the extender has 262,144 possible inputs, but it can be reasonably verified with 8 carefully selected inputs, i.e. two different 16-bit inputs for the 4 possible combinations of sign and byte.*
- The test bench must print a message to the command window clearly stating that the module passes, or it must print a line to the command window describing each error (if any) which includes the inputs applied, the expected output, and the actual incorrect output with appropriate formatting and labeling. It ***must not*** print anything for each successful test vector.
- The Verilog test bench and the test vector file must be commented, and the test benches must be named as specified.

- The Verilog test bench ***must not*** use an *'include* statement in the submitted version. Instead of using an *`include* statement to reference the file that has a module, you can enter multiple Verilog files in the iverilog command. For example:

  ```
  >>iverilog -o test mux2to1_tb.v muxes_a.v
  ```

  This avoids having `include "muxes_a.v" in the file mux2to1_tb.v, and testing different versions of modules can now be done without manually editing the test bench files.

Component 1 – 1 bit 2-to-1 Multiplexer

- Module: mux21
- Test bench file name: mux21_tb.v
- Inputs: S, D0, D1
- Output: Y
- Operation: When S=0, the output is input D0. When S=1, the output is input D1.
- Test Files: muxes_a.v, muxes_b.v, muxes_c.v

  Hints:
  - Since there are only three 1-bit inputs and a 1-bit output, you can generate the complete 8-row truth table and determine the output for each row.
  - The code for mux21 in the Verilog files uses a parameter to determine the bit width of the inputs and the output. You will need to provide this parameter when you instantiate the mux21 in your test bench. The format is

    ```
    mux21 #(1) uut(   // the 1 gets passed as the 1st parameter, the bit width
       .S     (wS),       // nothing else changes.
       ...
    ```

Component 2 – 5 bit 4-to-1 Multiplexer

- Module: mux41
- Test bench file name: mux41_tb.v
- Inputs: S[1:0], D0[4:0], D1[4:0], D2[4:0], D3[4:0]
- Output: Y[4:0]
- Operation: When S=00_2, the output is input D0. When S=01_2, the output is input D1. When S=10_2, the output is input D2. When S=11_2, the output is input D3.
- Test Files: muxes_a.v, muxes_b.v, muxes_c.v
  Hints:
    - For each test vector, you must provide values for all four data inputs and the select lines. Test at least two different values per each data input when selected (i.e. 8 test vectors would suffice.) Note that each input should have a different value in order to show that the output matches the correct input.
    - The code for a mux41 in the Verilog files uses a parameter to determine the bit width of the inputs and the output. You will need to provide this parameter when you instantiate the mux41 in your test bench. The format is

      mux41 #(5) uut(// the 5 gets sent at the 1st parameter, which determines width

Component 3 – Signed/Unsigned Extender

- Module: extender
- Test bench file name: extender_tb.v
- Inputs: D[15:0], Sign, Byte
- Output: Y[31:0]
- Operation: The 32-bit output is generated by extending the input as signed or unsigned, and extending the full 16-bit input or just the low 8 bits as shown in the following table.

Sign	Byte	Output (32 bits)	Description
0	0	0...0:data[15:0]	Unsigned halfword extension
0	1	0...0:data[7:0]	Unsigned byte extension
1	0	data[15]...data[15]:data[15:0]	Signed halfword extension
1	1	data[7]...data[7]:data[7:0]	Signed byte extension

- Test Files: extender_a.v, extender_b.v
  Hints: Make sure that there are 16-bit values in the test that would be extended differently if it was interpreted as signed versus unsigned, and the same goes for 8-bit inputs.

Component 4 – Program Counter

- Module: pc
- Test bench file name: pc_tb.v
- Inputs: NextPC[31:0], Clk, Reset
- Output: PC[31:0]
- Operation: If the Reset input is true when the Clk transitions from false to true, the output PC is 0x00400000. If the Reset input is false when Clk transitions from false to true, the output PC is NextPC.
- Test Files: pc_a.v, pc_b.v

  Hints:
  - This is like a state machine. You must toggle the clock and check the result. However, since the output does not depend on anything stored inside the PC when the clock edge happens, there are no internal signals that need to be forced.
  - The pc module has two parameters that allow the module to be customized. The first is WIDTH, which determines how many bits are used for NectPC and PC, and the second is RESET_ADDR, which allows the default starting address to be specified. To instantiate the pc module to meet the above requirements, use

    ```
    pc #(32,32'h00400000) uut(     // sets two parameters in order
    ```

    Note that using this notation, the parameters must be passed in the same order in which they appear in the module's code.

Deliverables

Submit all deliverables according to the posted lab submission guidelines.

- 4 points: Compliance with lab submission guidelines
- 24 points for each component: 8 for each of the three areas in the program rubric.