Laboratory 8

CPU Building Blocks, Part 2

Due Date: Beginning of class July 27

Objectives:

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

Assignment

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

- It verifies the correct operation of the component per the operational descriptions provided.
- The test bench must use sufficient input combinations from a test vector file. (i.e. at least two different sets of data for each type of operation)
- 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 a successful test vector.
- The Verilog test bench and the test vector file must be commented, and the test bench must be named per the requirements.
- The Verilog test bench must ***not*** use an *'include* statement in the submitted version.

Component 1

- Module Name: instmem
- Test bench file name: instmem_tb.v
- Input: addr[31:0]
- Output: data[31:0]
- Mandatory internal signals: *memory* – 8-bit wide array of 4096 values from indices 0x00400000 to 0x00400FFF

- Operation:
  - The instruction memory contains 1024 4-byte words from address 0x00400000 to 0x00400FFF stored in little-endian format, or 4096 bytes (4kiB) total. The output *data* must display the 4-byte value stored at the location specified by *addr*.
  - The expected value supplied to *addr* should always be evenly divisible by 4 (instructions are memory aligned), therefore the module will ignore the rightmost two bits and assume that they were 00_2 when accessing the internal array.

Note:The test bench must manually initialize the internal array first, since there is no ability to write to it using the defined inputs. This is why the requirements include an internal variable name – the test bench must directly write to the array like earlier sequential test benches wrote to state variables. An example of how to do this is shown in instmem_tb_ex.v.

Component 2

- Module Name: regfile
- Test bench file name: regfile_tb.v
- Inputs: input1[4:0]. input2[4:0], writeReg[4:0], writeData[31:0], writeEN, clk
- Outputs: data1[31:0], data2[31:0]
- Operation:
  - The register file contains thirty-two 32-bit registers. Output *data1* must be the value of the register specified by the binary number supplied to *input1*, and *data2* must be the value of the register specified by the binary number supplied to *input2*. For example, if *input1* is 01101_2, then *data1* is the 32-bit value stored in register 13_{10}.
  - Both outputs work asynchronously, meaning that they are updated as soon as *input1* or *input2* change, and they do not wait for the clock edge.
  - If *writeEN*, or write enable, is true when *clk* transitions from 0 to 1 (i.e. positive edge), the 32-bit value supplied to *writeData* is stored into the register specified by the 5-bit number supplied to *writeReg*.

Note: There are no requirements for known internal signals, so the test bench ***cannot*** initialize the register file like the instruction memory. It needs to store values to registers first, then look at the stored values by requesting the registers using input1 and input2 to see if the values were stored correctly.

Since this is a sequential component, you should use a test bench like those in Lab 6, in which the test vector is read, there is a slight pause, the clock is switched to true, another pause, then the outputs are checked.

Component 3

- Module Name: datamem
- Test bench file name: datamem_tb.v
- Input: address[31:0], WE, clk, writebyte, writehalfword, datain[31:0]
- Output: data[31:0]
- Operation:
  - The output *data* must be the 32-bit little-endian word stored at the memory location supplied by the 32-bit input *address*. Only addresses in the range 0x10010000 to 0x10010FFF are valid. Any request for data outside that range may result in undetermined behavior.
  - If WE, or write enable, is set to true when a positive clock edge occurs, then:
    - dataIn[31:0] is written to the four bytes in memory specified by address if writebyte and writehalfword are both false. Data is stored little-endian.
    - dataIn[15:0] is written to the two bytes in memory specified by address if writebyte is false and writehalfword is true. Data is stored little-endian.
    - dataIn[7:0] is written to the byte in memory specified by address if writebyte is true and writehalfword is false.
    - If writebyte and writehalfword are both true, the operation is undefined. (this is the standard way of saying that this case will never occur during correct use of your module, so the module does not need to check for it and may do whatever it wants in this case.)

Notes:

The datamem module has asynchronous (i.e. not based off of a clock signal) and synchronous (coordinated with a clock signal) behavior. The *data* output is asynchronous, so when a value is applied to *address*, the output *data* should immediately change to the value currently stored in that location. The write operation is synchronous, so the value stored in a location is updated only on a positive clock edge. Therefore, to correctly validate the write operation, the test bench would need to supply an address and verify the correct value for data before a clock transition, then verify the new correct value after the transition.

Component 4

- Module Name: alu
- Test bench file name: alu_tb
- Inputs: inA[31:0], inB[31:0], operation[3:0]
- Outputs: result[31:0], zero
- Operation:

  The ALU must calculate the result based on the mathematical/logical operation specified by the operation bits. The table below shows what operation corresponds to each operation input.

operation[3:0]	result	Description
0000	inA + inB	Addition
0001	inA – inB	Subtraction
0010	inA AND inB	Bitwise AND
0011	inA OR inB	Bitwise OR
0100	inA XOR inB	Bitwise Exclusive-OR
0101	inA NOR inB	Bitwise NOR
0110	1 if inA < inB, 0 otherwise, signed	Set bit if less than (signed)
0111	1 if inA < inB, 0 otherwise, unsigned	Set bit if less than (unsigned)
1000	Shift inB logically left by inA[4:0]	Shift logical/arithmetic left
1001	Shift inB logically right by inA[4:0]	Shift logical right
1010	Shift inB arithmetically right by inA[4:0]	Shift arithmetically right
1011	inA	Pass through
1100	inB[15:0],16'b0	Load upper immediate
1101	Not implemented	Not implemented
1110	Not implemented	Not implemented
1111	Not implemented	Not implemented

The output zero must be 1 (i.e. true) when the result is 0, and 0 (i.e. false) otherwise.

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.

  Please DO NOT upload the supplied module Verilog files.