LSU EE 3755 Homework 6 Solution Due: 26 November 2012

This assignment is to be completed on the ECE Linux workstations, please follow the instructions at `http://www.ece.lsu.edu/ee3755/proc.html`.

The Verilog code showing the solution is available at `http://www.ece.lsu.edu/ee3755/2012f/hw06-sol.v`
The unmodified code is at `http://www.ece.lsu.edu/ee3755/2012f/hw06.v`

Problem 0: Follow the instructions for MIPS Homework Workflow in the procedures page, `http://www.ece.lsu.edu/ee3755/proc.html`, but substitute `hw06` for the directory and `hw06.v` for the name of the file to copy. (The extension is `.v` (Verilog), not `.s` (assembler).)

Load the file into an Emacs buffer. If things are set up correctly the buffer should show most comments in red (though titles will be a larger black font) and Verilog code should by syntax-highlighted, for example showing `input` and `output` in light blue and `reg` in purple. There should also be a Verilog pull-down menu.

Run the Verilog simulation without modification by pressing `F9` or by selecting Verilog Compile. (Don't select synthesize and compile, it does not work yet.) The window should be split into two panes, with the lower pane starting with something like:

```
-*- mode: compilation; default-directory: "~/teach/co12f/s/" -*-
Compilation started at Mon Nov 19 08:05:19

irun -batch -exit hw06.v
irun(64): 10.20-s120: (c) Copyright 1995-2012 Cadence Design Systems, Inc.
file: hw06.v
module worklib.cpu:v
errors: 0, warnings: 0
```

and ending with something like

```
 PC 0x00400010:   10: li $a1, 10
 Register $5 (  a1): 0x00000032 (         50) -> 0x0000000a (         10)
 PC 0x00400014:   12: addi $t6, $0, -2
 Register $14 (  t6): 0x0000008c (        140) -> 0xfffffffe (         -2)
 PC 0x00400018:   13: addi $t5, $0, 2

 Register $13 (  t5): 0x00000082 (        130) -> 0x00000002 (          2)
 PC 0x0040001c:   15: bgez $t5, SKIP1
*** Illegal instruction exception at address 0x0040001c ***
Executed 8 instructions, average time 2.88 CPI.
End of testbench run.
```

The text "Illegal instruction exception" indicates that the simulated MIPS processor, in `hw06.v`, did not recognize an instruction. That will be fixed in Problem 2.

Problem 1: Change implementation of `lui` so that it uses the shift unit in the ALU rather than using a shifted version of the immediate, `limmed`. There is no way to tell if this is solved correctly just by looking at simulator output. But, if the modified implementation does not execute `lui` correctly there will be an error message. For example,

```
 PC 0x00400000:   6: lui $t1, 0x1234;
 Register $9 (  t1): 0x0000005a (         90) -> 0x00000000 (          0)
*** FAIL: Wrong value written.  0x12340000 (correct) != 0x00000000 ***
```

Hint: Look at the way the `sll` *instruction is implemented.*

Use the ALU in a manner similar to type I instructions such as `addi`. For the operation choose `OP_sll`, for the `alu_a` input (the shift amount) use the constant 16, `alu_b` is set to `uimmed` and the destination is `rt`. With this change one input to the `alu_b` multiplexor has been eliminated, the one for `limmed` (which is no longer being used), but one input to the `alu_b` mux has been added. If both mux inputs cost as much as a typical 32-bit mux input then there is only a small benefit. However, with the right logic design (or good synthesis optimization) the input to `alu_a` can be shrunk to 5 bits (since `OP_sll` only examines 5 bits). Another design alternative is to use a fixed `OP_sll16` operation in the ALU, eliminating the need even for an extra input to `alu_a` (with `OP_sll16` the value of `alu_a` would be ignored).

Here is the relevant change:

```
O_lui:  bndl = {rt, rs_val, OP_or,  limmed }; // Before change.

O_lui:  bndl = {rt, 32'd16, OP_sll, uimmed }; // After change.
```

Problem 2: The illegal instruction exception encountered above is due to the `bgez` instruction not being implemented by the MIPS module. Implement `bgez` and `bltz`, the simulation should run to completion (without showing an illegal instruction exception or other error messages) finishing with a "PASS" message.

For detailed descriptions of these instructions see
`http://www.ece.lsu.edu/ee4720/mips32v2.pdf`.

Whether these branches are taken depends on the value of the `rs` register. Instruction `bgez` is taken if the value is not negative and `bltz` is taken if the value is negative. To check if a value is negative one only needs to look at the most significant bit. In the solution this test is done in the `ID` state. If the branch is not taken the next state is `ST_if` and if the branch is taken it is `ST_ex_targ`. Notice that the ALU is not needed to test the branch condition, for this reason the ID state can set the ALU to compute the branch condition:

```
O_br:   bndl = {R0, npc, OP_add, simmed << 2 };     // SOLUTION
```

The code for testing the branch condition and setting the next state is here:

```
      case ( opcode )
        O_lbu, O_sb  : state = ST_ex_addr;
        O_bne, O_beq : state = ST_ex_cond;
        /// SOLUTION BELOW
        O_br:
          // Use the rt field to determine the exact branch
          // condition.
          case ( rt )
            B_bgez: state = rs_val[31] ? ST_if : ST_ex_targ;
            B_bltz: state = rs_val[31] ? ST_ex_targ : ST_if;
          endcase
        /// SOLUTION ABOVE
        O_j          : state = ST_if;
        default      : state = ST_ex;
      endcase
    end
```