ECS 154A **Homework #3 (77 points)** **Winter 2014**

Due Monday, February 24. Written to the ECS 154A box in 2131 Kemper at 4pm. Submit authors.txt, 1.sv, 2.sv, 3.sv, alu.sv, cpu.sv from the Programs section to cs154a p3 by 11:59pm using handin.

Written (16 points)

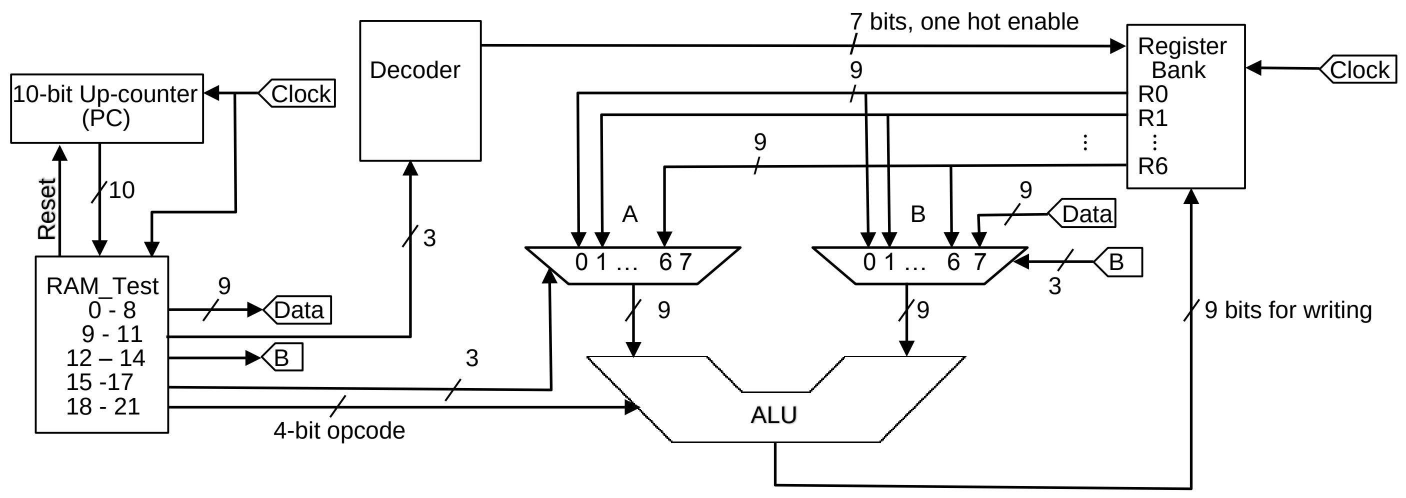
1. Exercise 4.2 (2 points)
2. Exercise 4.16 (2 points)
3. Exercise 4.22 (2 points)
4. Exercise 4.24 (2 points)
5. Exercise 4.48 (2 points)
6. Exercise 4.50 (6 points)

Programs (61 points)

The modules in the programs must have the signatures specified so that our testbenches can test them properly. The testbenches will be available Thursday night in ~ssdavis/154/p3. To avoid revealing implementation details, some testbenches rely on accompanying data files that must be copied too.

1. Exercise 4.8 (3 points), named 1.sv with signature: mux8 #(parameter width = 4) (input logic [width-1:0] d0, d1, d2, d3,d4, d5, d6, d7, input logic [2:0] s, output logic [width-1:0] y);
2. Exercise 4.26 (3 points), named 2.sv with signature: srlatch(input logic s, r, output logic q, qbar);
3. Exercise 4.38 (5 points), named 3.sv with signature: ex4_38(input logic clk, reset, up, output logic [2:0] q);
4. alu.sv (15 points) Your assignment is to use SystemVerilog to create a simple processor that is 9 bits wide and can do various register transfers and ALU operations over a common bus. The entire design will be available on Wednesday. For now, you can start on a 9-bit ALU that utilizes three 3-bit carry look-ahead units to implement the following operations. The CLA module should have the signature: CLA(output logic [2:0] carries, output logic cout, input logic [2:0] ps, gs, input cin). The ALU module should have a signature: ALU(output logic [8:0] out, input [8:0] a, b, input[3:0] opcode). All modules having to do with the ALU, including the ALU module itself, should be in alu.sv. Note that there are testbenches for just the ALU and just the CLA modules in the ram.sv file.
5. cpu.sv (35 points) Below is an outline of the overall CPU design. It does not address the circuitry that you may need to implement the HALT, NOP, and three immediate mode instructions. It also does not show the register bus leading from the Register Bank to the RAM_Test module. The signature of the cpu module should be cpu(input clock);

   Development suggestions:
   - The first lines of the cpu module are instantiations of all the buses needed to connect the components.
   - I had the following modules: cpu, Decoder, PC, Mux8, RegisterBank, Register, Decoder.
   - As with a normal programming project, create module stubs so that the cpu module compiles, and then add to the code to each module in succession.
   - I compiled often so I could learn from my lessons early, and not repeat them.



CPU Components
You can break the CPU circuit down into the following components:
1. ALU
As described above for alu.sv.

2. Register Bank (aka Register File)
The register bank will contain seven 9-bit registers. On the falling edge of Clock, if Write Enable is asserted, the register corresponding to the appropriate one-hot input will be written with the input data value. Although a CPU would normally store output in computer memory, we will not be dealing with memory in this lab. We will be treating the register values as the "output" of this CPU. Though not shown in the above diagram, there is a 6 x 9 bus that runs from the register bank to the RAM_Test module. This bus is read by that module to check if the registers have the correct values after each operation.

3. Decoder
The decoder will determine the destination register of any output from the ALU by specifying a single high value on one of seven decoder outputs.

4. Multiplexers
Multiplexers are used to select between the registers or immediate data input to the ALU. Note that you must implement circuits to deal with the three immediate mode instructions selecting input #7 on muxB.

5. RAM_Test
The RAM_Test module will serve as both the RAM and the testbench for the whole CPU. It will supply instructions and also check to see if the registers have correct values. It will be located in a file named ram.sv that will be available in ~ssdavis/154/p3 on Thursday. The RAM_Test module will have the following signature RAM_Test(output logic [21:0] RAMdata, output PC_Reset, input [9:0] address, input [6:0][8:0] registerBus, input clock);

6. PC (up counter)
Associated with the RAM will be a PC register that will have a reset input pin. When the reset pin is set to one, the PC should be reset to zero. The first thing the RAM_Test module does is set the PC reset pin to one to reset it, and then sets the pin to zero to start the program running.

Instruction Format

Name	Bits	Function in CPU	Programming Function
OpCode	21-18	ALU Control	Determines which operation to perform, and immediate mode sourcing
A	17-15	Mux A Control	Primary source register specification
B	14-12	Mux B Control	Secondary source register specification, ignored for ADDI, SUBI and MOVI
D	11-9	Reg File Register Select	Destination register specification
Data	8-0	Input Data (when necessary)	Unsigned input data (when necessary)

Operation description
All operations, except NOP and HLT, put their results in the destination register (D in the instruction).

Operation	OpCode [21..18]	Description
AND	0000	Bitwise AND of registers A and B
OR	0001	Bitwise OR of registers A and B
NOT	0010	Negate register A (do this for SLL and SRL too)
ADD	0011	Add registers A and B. You must use carry look-ahead unit(s)
MOV	0100	Copy register A
SLL	0101	Shift Left Logical. Shift all bits to the left by 1; discard leftmost bit and make the least significant bit 0. Example: 1011 -> 0110 (operate on mux A input)
SRL	0110	Shift Right Logical. Shift all bits to the right by 1; discard rightmost bit and make the most significant bit 0. Example: 1011 ->0101 (operate on mux A input)
SUB	0111	Subtract the register B from register A.
ADDI	1000	Add the instruction data to register A
SUBI	1001	Subtract the instruction data from register A
MOVI	1010	Copy the instruction data.
NOP	1011	No registers, other than the PC, should change during this instruction cycle.
HALT	1111	Stop the CPU from executing any further instructions.

Testing your CPU.
The RAM_Test module reads 1024 lines from the file cpu_test.txt. Each line contains an instruction, a register number, and the value that that register should hold. cpu_test.txt will be generated using a random number generator and a simulated CPU written in C++. The file given you for your testing will not be the one used for grading.