CPEN 211 Introduction to Microcomputers, 2021
Lab Proficiency Test #3

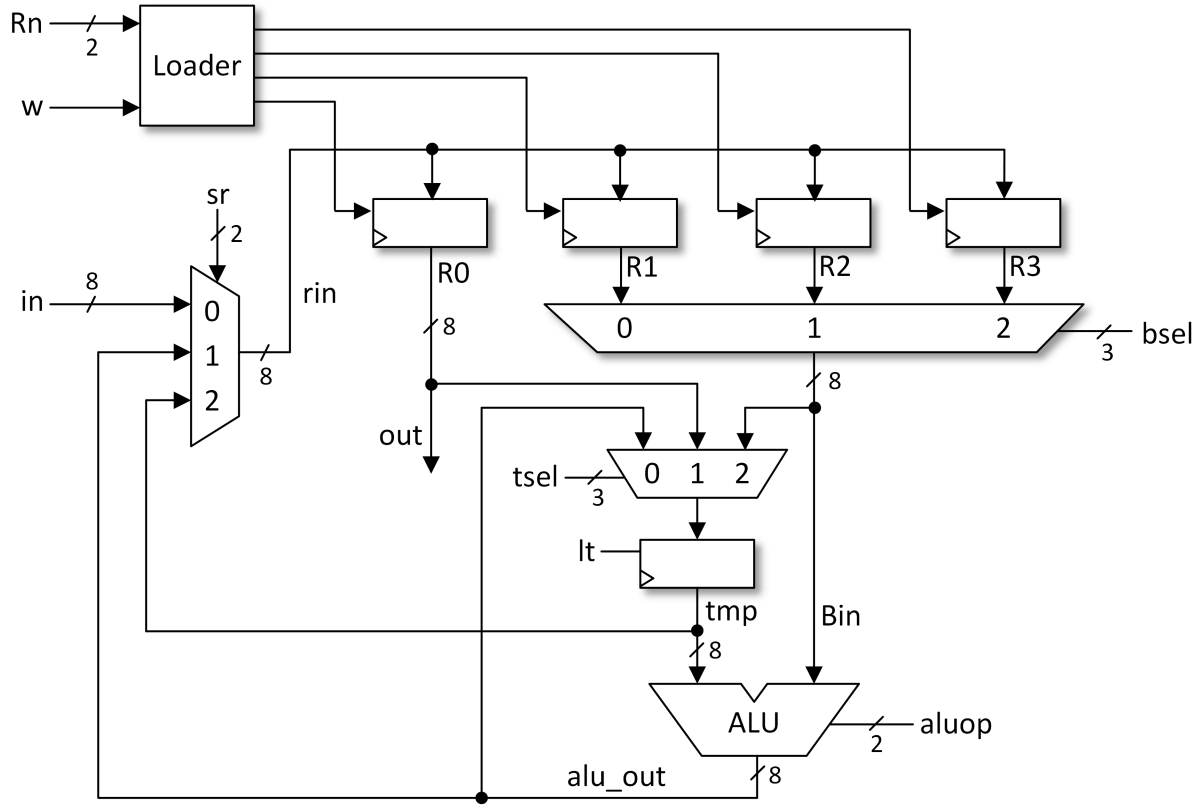
Question 1 [2.5 marks, part marks possible]: In a file "**q1.v**" write **synthesizable** Verilog implementing the datapath in the figure below. Your top-level module must be declared as:

```
module datapath(clk,in,sr,Rn,w,aluop,lt,tsel,bsel,out);
  input clk, w, lt;
  input [7:0] in;
  input [1:0] sr, Rn, aluop;
  input [2:0] bsel, tsel;
  output [7:0] out;
```

Use the signal names as shown in the figure. ALU is combinational logic defined as follows:

aluop	ALU Operation
2'b00	alu_out = tmp ^ Bin
2'b01	alu_out = tmp & Bin
2'b10	alu_out = tmp << 1
2'b11	alu_out = Bin

The datapath contains five 8-bit registers with load enable and clock input `clk`, connected to 8-bit signals `R0`, `R1`, `R2`, `R3`, and `tmp`. The block "Loader" is combinational logic that enables loading of `R0` only when `w` is 1 and `Rn` is `2'b00`, `R1` only when `w` is 1 and `Rn` is `2'b01`, `R2` only when `w` is 1 and `Rn` is `2'b10`, and `R3` only when `w` is 1 and Rn is `2'b11`. The input "`sr`" is 2-bit binary select, 3-bit inputs "`tsel`" and "`bsel`" are one-hot select, and input "`lt`" is a load enable. Your q1.v **must** include definitions for any modules instantiated. You can include testbenches in q1.v. The testbench `tb_check_q1` in tb.v inside https://cpen211.ece.ubc.ca/2021/lpt-3-checker-G3iW.zip should print "`INTERFACE OK`" and generate no ModelSim warnings if your interface is compatible with the autograder (cut and paste link if clicking does not work). **Upload your Verilog file named "q1.v" for Question 1 on "Lab Proficiency Test #3" on Canvas before 6:45 pm.**



CPEN 211 Introduction to Microcomputers, 2021
Lab Proficiency Test #3

Question 2 [2.5 marks, part marks possible]: Create a state machine to control the datapath described in Question 1. Make a new file named "**q2.v**" and include the starter code for module `bitwise` below then add **synthesizable** Verilog where says "`// IMPLEMENT YOUR CONTROLLER HERE`". Your q2.v will be autograded using a **reference (i.e., correct)** `datapath` module. To test `bitwise`, create a project with q1.v and q2.v. Do **NOT** include your `datapath` code in q2.v. To avoid synthesis errors during testing use different names for other modules needed in both files.

```
module bitwise(clk,reset,s,op,in,out,done);
  input clk, reset, s;
  input [7:0] in;
  input [3:0] op;
  output [7:0] out;
  output done;
  wire [1:0] sr, Rn, aluop;
  wire w, lt;
  wire [2:0] bsel, tsel;
  datapath DP(clk,in,sr,Rn,w,aluop,lt,tsel,bsel,out);
  // IMPLEMENT YOUR CONTROLLER HERE
endmodule
```

Using the datapath defined in Question 1 module `bitwise` implements the instructions defined in the table below. Input "op" to module `bitwise` is a 4-bit instruction encoded as the column "Encoding". In the column "Operation" in the table, the notation "`in`" refers input "`in`", "`tmp`" refers to "tmp" inside `datapath` (defined in Question 1), and `Ri` refers to the 8-bit contents of register `Ri` inside `datapath`. E.g., if `i` is `2'b10` then `Ri` refers to the 8-bit value `R2`.

Marks	*Assembly*	*Encoding (op)*				*Operation (may require more than one cycle)*
		3	2	1	0	
1	`MOV Ri`	0	0	*i*		`Ri = in`

0.5	`XOR`	0	1	*unused*		`tmp = R1`
						`tmp = tmp ^ R2`
						`R0 = tmp`

0.5	`ASL`	1	0	*unused*		`tmp = R1`
						`tmp = tmp & R2`
						`tmp = tmp << 1`
						`R0 = tmp`

0.5	`SWP Ri`	1	1	*i*		`tmp = R0`
						`R0 = Ri`
						`Ri = tmp`

Your controller should reset to a wait state on the next rising edge of "`clk`" if "`reset`" is 1. In the wait state output "`done`" should be set to 1. The input "`s`" is initially 0 but will be set to 1 to indicate the values of "`in`" and "`op`" are valid and that your module should begin executing the instruction specified by "`op`". When "`s`" is set to 1 the inputs "`in`" and "`op`" will stay the same until your circuit sets "`done`" to 1. Your controller should stay in the wait state until the input "`s`" is 1 and there is a rising edge of "`clk`". While executing an instruction, output "`done`" should be set to 0 until the instruction has finished and "`done`" must be 0 for at least one cycle of `clk` (i.e., the length of time between two rising edges of `clk`). When your controller has finished executing an instruction it must return to the wait state. If your code is compatible with the autograder `tb_check_q2` inside `tb.v` (from URL in Question 1) should print "`INTERFACE OK`" and generate no ModelSim warnings. The testbench `tb_add` found in `tb.v` uses `bitwise` to add 42 and 11.