CPEN 211 Introduction to Microcomputers, 2024
Lab Proficiency Test #3

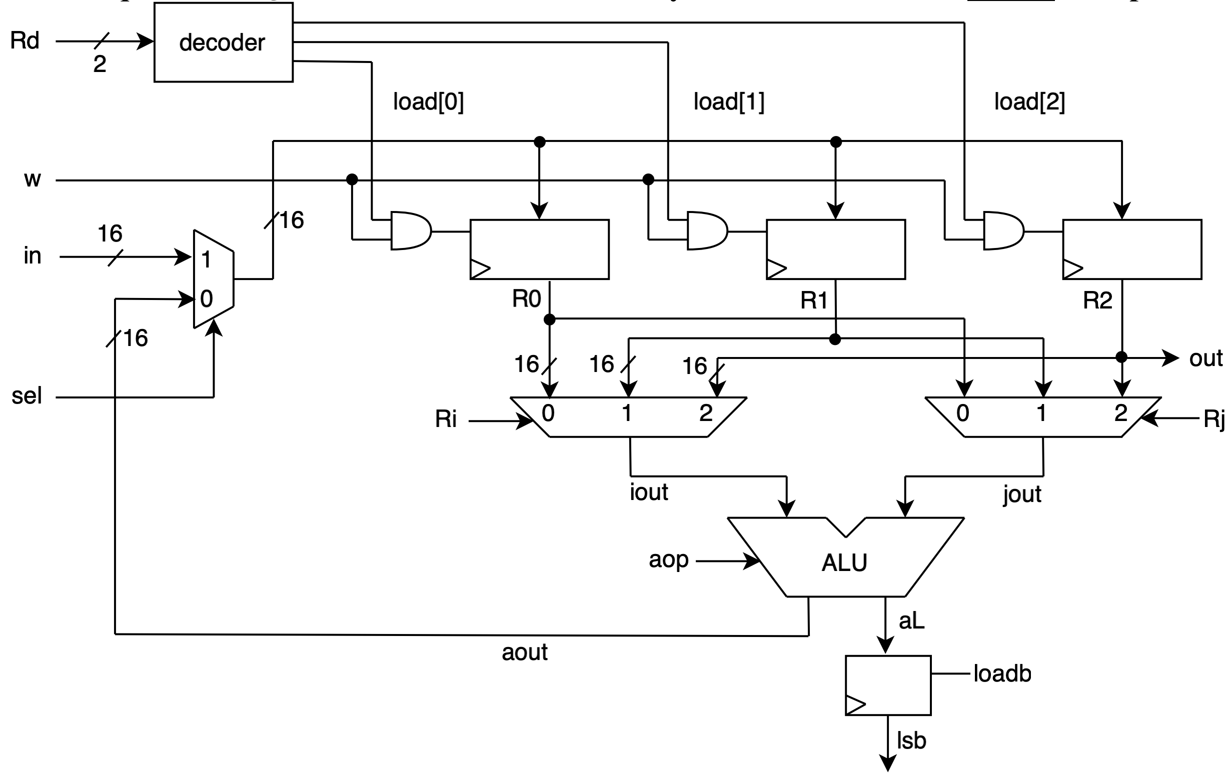
Question 1 [2.5 marks, part marks possible]: In a file "**q1.sv**" write **synthesizable** Verilog implementing the datapath in the figure below. Your top-level module *must* be declared as:

```
module datapath(clk,Rd,w,in,sel,Ri,Rj,lsb,aop,loadb,out);
  input clk, w, sel, loadb;
  input [1:0] Rd, Ri, Rj;
  input [15:0] in;
  input [1:0] aop;
  output lsb;
  output [15:0] out;
```

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

aop	ALU Operation
2'b00	aout = iout >> 1
2'b01	aout = iout << 1
2'b10	aout = iout + jout
2'b11	aout = iout

The output `aL` of the ALU is a 1-bit signal that is 1 when bit 0 of `aout` is 1 and 0 otherwise. The datapath contains three 16-bit registers with load enable and clock input `clk`, connected to 16-bit signals `R0`, `R1`, and `R2`. Datapath output `out` is connected to `R2`. The block "decoder" is a one hot decoder without 2-bit input `Rd` and 3-bit output `load`. The inputs "`Ri`" and "`Rj`" are 2-bit binary selects, input "`loadb`" is a load enable. You can include testbenches in q1.sv. The testbench `tb_check_q1` in `tb.sv` inside https://cpen211.ece.ubc.ca/2024/lpt3-check-T7wBe9Vz.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). Your q1.sv **must** include definitions for any modules instantiated. *Hint:* Tor's solution for q1.sv is 29 lines. **Upload your Verilog file named "q1.sv" for Question 1 on "Lab Proficiency Test #3" on Canvas <u>before</u> 5:45 pm.**



CPEN 211 Introduction to Microcomputers, 2024
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.sv**" and include the starter code for module `mult` below then add **synthesizable** Verilog where says "**`// IMPLEMENT YOUR CONTROLLER HERE`**". Your q2.sv will be autograded using a **reference (i.e., correct)** `datapath` module. To test `mult`, create a project with q1.sv and q2.sv. Do **NOT** include your `datapath` code in `q2.sv`. To avoid synthesis errors during testing use different names for other modules needed in both files.

```
module mult(clk,reset,s,op,in,out,done);
  input clk, reset, s;
  input [15:0] in;
  input [4:0] op;
  output [15:0] out;
  output done;
  wire [1:0] Rd, Ri, Rj, aop;
  wire w, loadb, sel, lsb;
  datapath DP(clk,Rd,w,in,sel,Ri,Rj,lsb,aop,loadb,out);
  // IMPLEMENT YOUR CONTROLLER HERE
endmodule
```

Using the datapath defined in Question 1 module `mult` implements the instructions defined in the table below. Input "op" to module `mult` is a 5-bit instruction encoded as in the column "Encoding". In the column "Operation" in the table, the notation "`in`" refers input "`in`", "`N`" refers to the 1-bit signal `N` inside `datapath`, and `R[`*n*`]` refers to the 16-bit contents of register *Rn* inside `datapath` (defined in Question 1). E.g., if *i* is `2'b10` then `R[`*i*`]` refers to the 16-bit value `R2`.

Marks	***Assembly***	***Encoding (op)***					***Operation (may require more than one cycle)***
		4	3	2	1	0	
1	`MOV i`	0	*i*		0	0	`R[i] = in`
0.5	`LSR i`	0	*i*		0	1	`R[i] = R[i] >> 1`
0.5	`LSL i`	0	*i*		1	0	`R[i] = R[i] << 1`
0.25	`CHK i`	0	*i*		1	1	`lsb = aL ? 1 : 0`
0.25	`ADC i, j`	1	*i*		*j*		`if (lsb == 1)` `R[i] = R[i] + R[j]`

The notation "a << b" means left shift the value "a" by "b" bit positions with zeros shifted into the least significant bit positions; "a >> b" means right shift the value "a" by "b" bit positions with zeros shifted into the most significant bit positions; and "a?b:c" has the meaning taught in class. 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. `tb_check_q2` inside `tb.v` (from URL in Question 1) should print "`INTERFACE OK`" **and generate no ModelSim warnings**. The testbench `tb_mult` found in `tb.sv` uses `mult` to multiply 6 by 7. *Hint*: Tor's solution for q2.sv is 33 lines long.