Your Name (first last) Solution *← Name of person on left (or aisle)*	**UC Berkeley EECS151** **EECS251A** **Fall 2021 Midterm 1** *TA name*	*SID* *Name of person on right (or aisle) →*

Fill in the correct circles & squares completely…like this: ● (select ONE), and ■ (select ALL that apply)

Question	1	2	3	4	5	Total
Minutes	12	15	14	14	20	**75**
Max Points (151)	14	20	16	14	20	**84**
Max Points (251A)	14	20	16	14	24	**88**
Points						

1) *It's all logical…* (14 points, 12 minutes)

You need to design a circuit for the following truth table.

A	B	C	Y
0	0	1	1
1	0	0	1
0	1	1	0
1	1	1	1
0	0	0	1
1	0	1	1
0	1	0	1
1	1	0	0

a) Write the sum-of-products expression for output Y directly from this truth table.

Y = $\bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC + \bar{A}\bar{B}\bar{C} + A\bar{B}C + \bar{A}B\bar{C}$ (minimized expression $\bar{B} + \bar{A}\bar{C} + AC$ also accepted).

1) *It's all logical…* (continued)

b) Use the following 4-input Karnaugh-map (inputs A, B, C, D) to write a simplified product-of-sums expression.

AB \ CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	0	1	0
10	0	0	1	0

Common errors:
- missing single 0
- not grouping properly
- writing SOP instead of POS

Y = $(\bar{A}+C)(B+C)(\bar{A}+D)(A+\bar{B}+\bar{C}+\bar{D})$.

c) Suppose we want to design a 2-bit unsigned comparator circuit which computes if the 2-bit input, {AB}, is greater than or equal to the 2-bit input, {CD} (ie. the output is 1 if {AB} ≥ {CD}. For clarity, A and C are the MSBs of their respective inputs. Fill in the following Karnaugh-map for this function and write a minimized sum-of-products expression.

AB \ CD	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	1	1	1	1
10	1	1	0	1

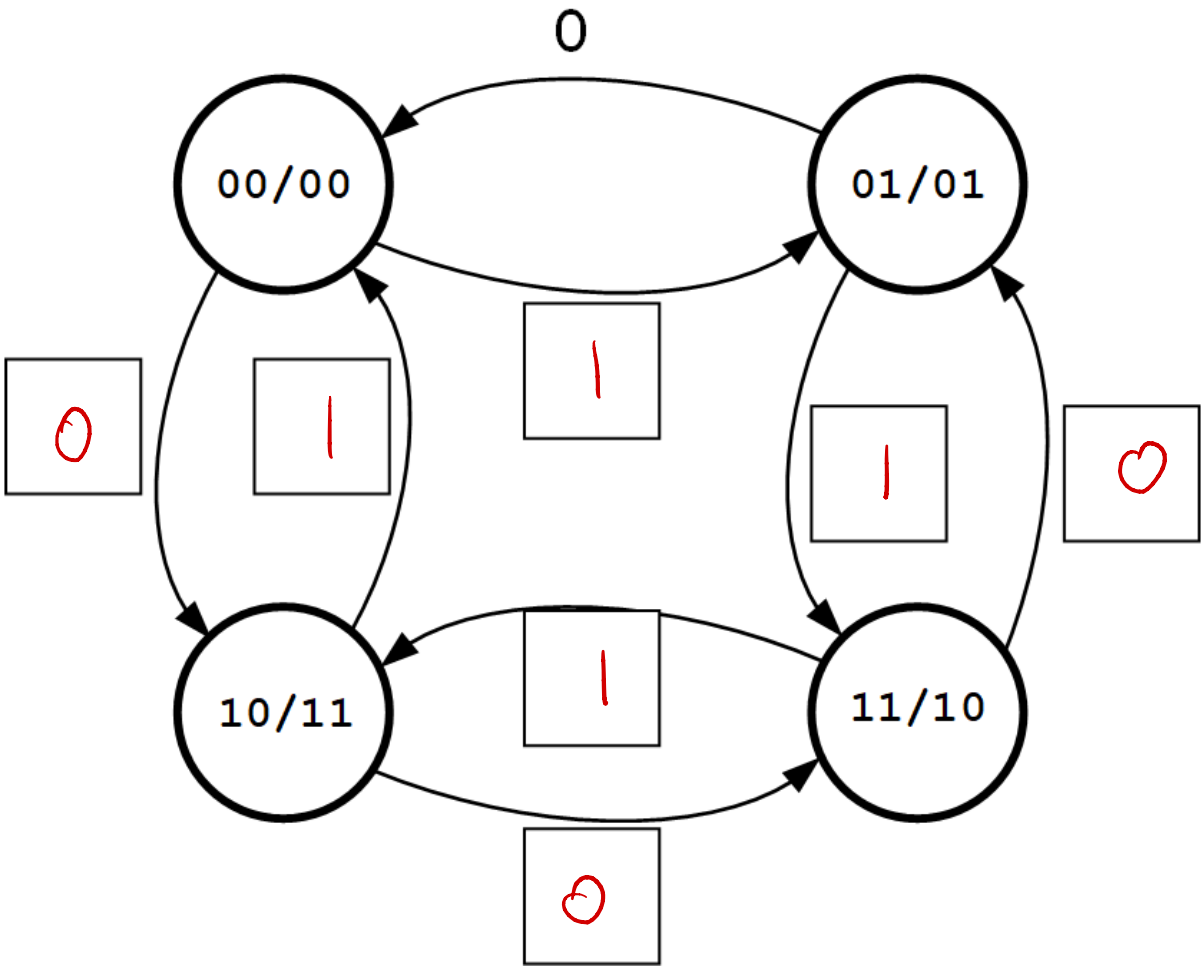
Common errors:
- writing K-map wrong by missing gray code ordering for the comparison logic
- not grouping properly

Y= $AB + A\bar{C} + A\bar{D} + B\bar{C} + \bar{C}\bar{D}$

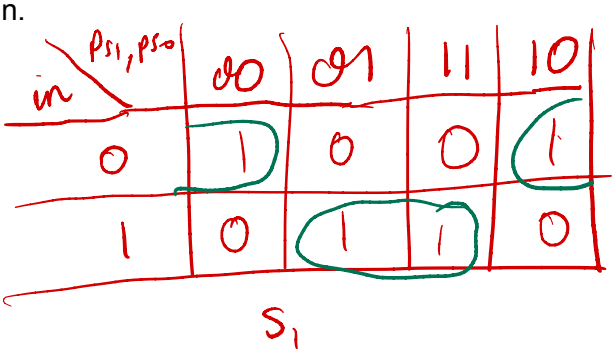
2) Gray Code Counter (20 points, 15 minutes)

Gray codes have a property in that consecutive binary numbers differ in only a single bit position. Design a 2-bit up/down modulo-4 Gray code counter FSM with one input and two outputs. (A modulo-N counter counts from 0 to N − 1, then repeats). When the counter is reset (reset signal is not shown in the picture), the output should be 00. On each clock edge, the output should advance to the next Gray code if the input signal is 1, and decrement if the signal is 0. After reaching 10, it should repeat with 00. The output of the counter should be a binary number b_1b_0.

a) In the state transition diagram below, label the unlabeled inputs that correspond to state transitions.



b) Derive the minimal logic that computes the new state $\{s_1, s_0\}$ from the previous state $\{ps_1, ps_0\}$ and the input, in.



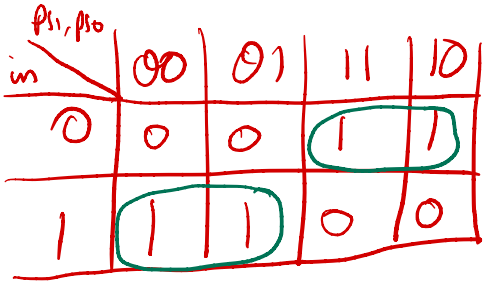
s_1

$s_1 = in \cdot ps_0 + \overline{in} \cdot \overline{ps_0}$

$= in \text{ xnor } ps_0$

$= \overline{in \oplus ps_0}$

all correct



s_0

$s_0 = in \cdot \overline{ps_1} + \overline{in} \cdot ps_1$

$= in \oplus ps_1$

$= in \text{ xor } ps_1$

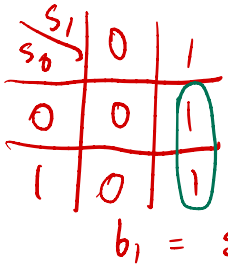
$= in \wedge ps_1$

all correct

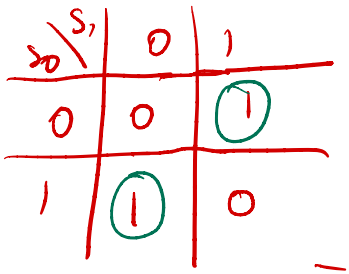
s_1 (ps_1,ps_0, in) = $\overline{in \oplus ps_0}$.

s_0 (ps_1,ps_0, in) = $in \oplus ps_1$

c) Derive the minimal logic that computes the binary outputs b_1, b_0, from the state bits, s_1, s_0.



$b_1 = s_1$



$b_0 = s_1 \overline{s_0} + s_0 \overline{s_1}$

$= s_1 \oplus s_0$

$= s_1 \text{ xor } s_0$

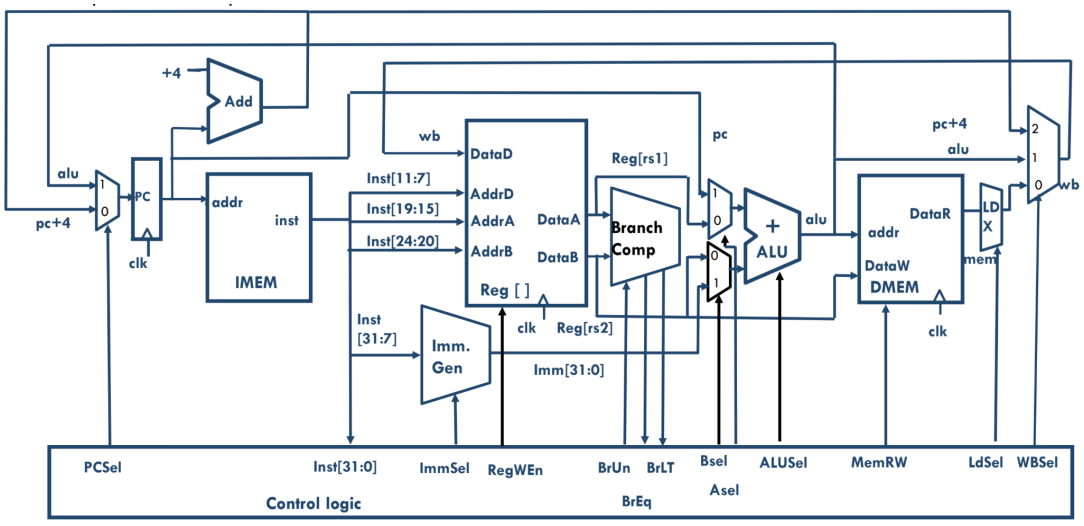
$= s_1 \wedge s_0$

all correct

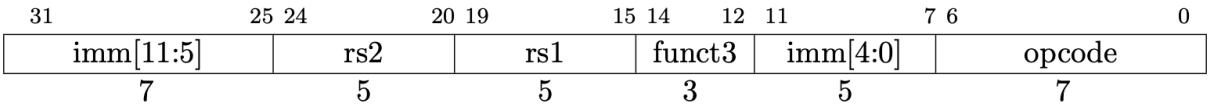
b_1 (s_1,s_0) = s_1 .

b_0 (s_1,s_0) = $s_1 \oplus s_0$

3. Datapathology



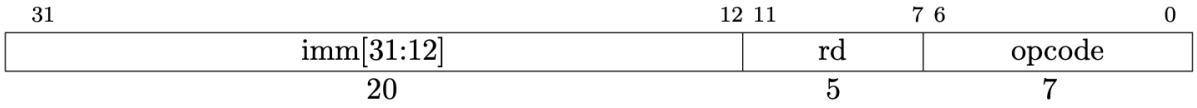
a. What is used by the **lw** instruction (select all that apply)?



Select one per row	**PCSel Mux:** ○ "pc + 4" branch ○ "alu" branch ○ * (don't care) **ASel Mux:** ○ "pc" branch ○ Reg[rs1] branch ○ * (don't care) **BSel Mux:** ○ "imm" branch ○ Reg[rs2] branch ○ * (don't care) **WBSel Mux:** ○ "pc + 4" branch ○ "alu" branch ○ "mem" branch ○ * (don't care)
Select all that apply	**Datapath units:** ☐ Branch Comp ☐ Imm. Gen ☐ Load Extend **RegFile:** ☐ Read Reg[rs1] ☐ Read Reg[rs2] ☐ Write Reg[rd]

PCSel: pc+4 ASel: Reg[rs1] BSel: imm WBSel: mem
Datapath units: Imm. Gen, (Load Extend - optional)
RegFile: Read Reg[rs1], Write Reg[rd]

b. What is used by **lui** instruction (select all that apply)?



Select one per row	**PCSel Mux:** ○ "pc + 4" branch ○ "alu" branch ○ * (don't care) **ASel Mux:** ○ "pc" branch ○ Reg[rs1] branch ○ * (don't care) **BSel Mux:** ○ "imm" branch ○ Reg[rs2] branch ○ * (don't care) **WBSel Mux:** ○ "pc + 4" branch ○ "alu" branch ○ "mem" branch ○ * (don't care)
Select all that apply	**Datapath units:** ☐ Branch Comp ☐ Imm. Gen ☐ Load Extend **RegFile:** ☐ Read Reg[rs1] ☐ Read Reg[rs2] ☐ Write Reg[rd]

PCSel: pc+4 ASel: * BSel: imm WBSel: alu
Datapath units: Imm. Gen
RegFile: Write Reg[rd]

c. Suppose you have the following errors in hardware. Which instructions would be affected.

PCSel stuck at 0.
O LUI O AUIPC O J-type O B-type O S-type O R-type O I-type

J-type, B-type, I-type (JALR)

RegWEn stuck at 1.
O LUI O AUIPC O J-type O B-type O S-type O R-type O I-type

B-type, S-type

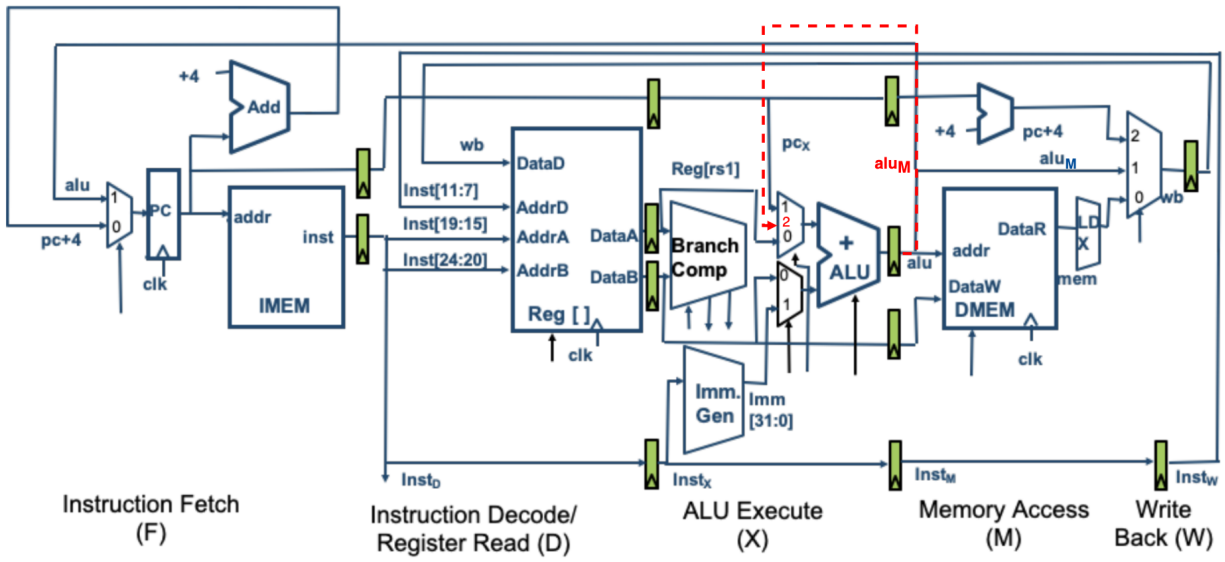
WBSel stuck at 0.
O LUI O AUIPC O J-type O B-type O S-type O R-type O I-type

LUI, AUIPC, J-type, R-type, I-type

4. Pipelining

This question will compare the performance of several different RISCV pipeline designs, referring to the datapath diagram below:

(1) unpipelined: only blue parts
(2) 5-stage pipelined: including green pipeline registers
(3) 5-stage pipelined + forwarding path: including red dashed line



a. For the code sequence below, what is the value at Mem[0x8] at the end of the program?

```
        addi x1, x0, 8          x1 = 8 = 0x08
        addi x2, x0, 128        x2 = 128 = 0x80 = 32'b1000_0000
        sw x2, 0(x1)            Mem[x1] = 0x80
        srai x2, x2, 4          x2 = x2 >> 4 = 0x08
        lb x3, 0(x1)            x3 = 0xff_ff_ff_80
        bge x2, x3, Label       8 >= <some neg #>? goto Label
        and x2, x2, x3
Label:  sw x2, 0(x1)            Mem [x1] = 0x8 = 8
```

Mem[0x8] = _____

Mem [x1] = 8

Variations receiving partial credit:

5/8 points: Forgot to sign extend lb

```
        addi x1, x0, 8          x1 = 8 = 0x08
        addi x2, x0, 128        x2 = 128 = 0x80 = 32'b1000_0000
        sw x2, 0(x1)            Mem[x1] = 0x80
        srai x2, x2, 4          x2 = x2 >> 4 = 0x08
        lb x3, 0(x1)            x3 = 0x00_00_00_80
        bge x2, x3, Label       0x08 = 8 >= 0x80 = 128 ? goto Label
        and x2, x2, x3          x2 = 0x08 & 0x80 = 0x0
Label:  sw x2, 0(x1)            Mem [x1] = 0x0 = 0
```

b. How many cycles would this program take to complete for the…
   i. unpipelined design.
   ii. 5-stage pipelined design
   iii. 5-stage pipelined design + forwarding path (red in diagram)

You may use the pipeline timing charts below, but only your response in the blanks will be graded.

unpipelined: _____
5-stage: _____
5-stage + bypass: _____

unpipelined: 7 cycles
5-stage: 18 cycles
5 stage + bypass: 18 cycles

5-stage

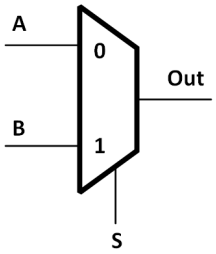
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
addi x1, x0, 8	F	D	E	M	W															
addi x2, x0, 128		F	D	E	M	W														
sw x2, 0(x1)			F	D	D	D	E	M	W											
srai x2, x2, 4				F	F	F	D	E	M	W										
lb x3, 0(x1)							F	D	E	M	W									
bge x2, x3, Label								F	D	D	D	E	M	W						
and x2, x2, x3									F	F	F	D	~~E~~							
Label: sw x2, 0(x1)												F	~~D~~	F	D	E	M	W		

5-stage + bypass

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
addi x1, x0, 8	F	D	E	M	W															
addi x2, x0, 128		F	D	E	M	W														
sw x2, 0(x1)			F	D	D	D	E	M	W											
srai x2, x2, 4				F	F	F	D	E	M	W										
lb x3, 0(x1)							F	D	E	M	W									
bge x2, x3, Label								F	D	D	D	E	M	W						
and x2, x2, x3									F	F	F	D	~~E~~							
Label: sw x2, 0(x1)												F	~~D~~	F	D	E	M	W		

5) *Verilog* (24/20 points, 20 minutes)

a) What Verilog block would synthesize to this circuit (select all that apply)?
(mark all that apply; +2 points for correct answers, -1 for incorrect)



☐
```
wire a,b,s,out;
assign out = s ? a : b;
```

☐
```
wire a,b,s; reg out;
always @(*)
      out = s ? a : b;
```

☐
```
wire a,b,s,clk; reg out;
always @(posedge clk)
      out <= s ? a : b;
```

☐
```
wire a,b,s,out;
always @(*) begin
      if (s==1'b0)
            out=b;
      else
            out=a;
end
```

reversed logic for sel

☒
```
wire a,b,s,out;
always @(*) begin
      case(s)
            0: out=a;
            1: out=b;
      endcase
end
```

Correction during exam: should be "reg out"

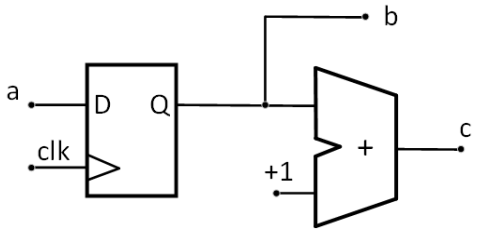
- If you left all boxes blank, you will still get full credit

b) What circuit would this Verilog code synthesize to?
(mark all that apply; +2 points for correct answers, -1 for incorrect)

```
wire[1:0] a;
wire clk;
reg[1:0] b,c;
always @(posedge clk) begin
      b <= a;
      c <= b+2'b1;
end
```

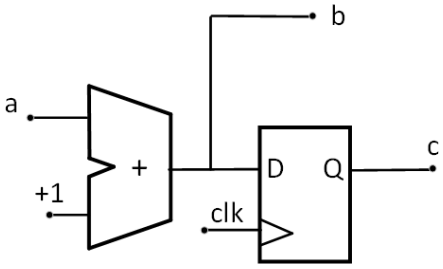
adder should be after FF

☐

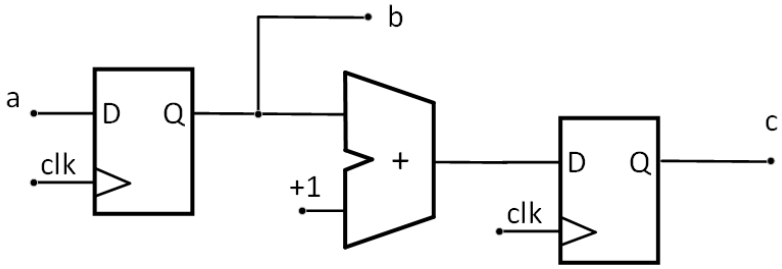


missing FF

☐

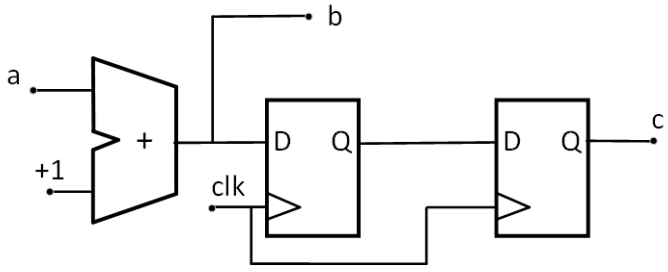


☒



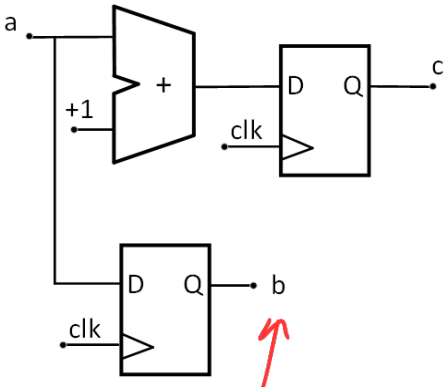
correct

☐



adder should be after FF

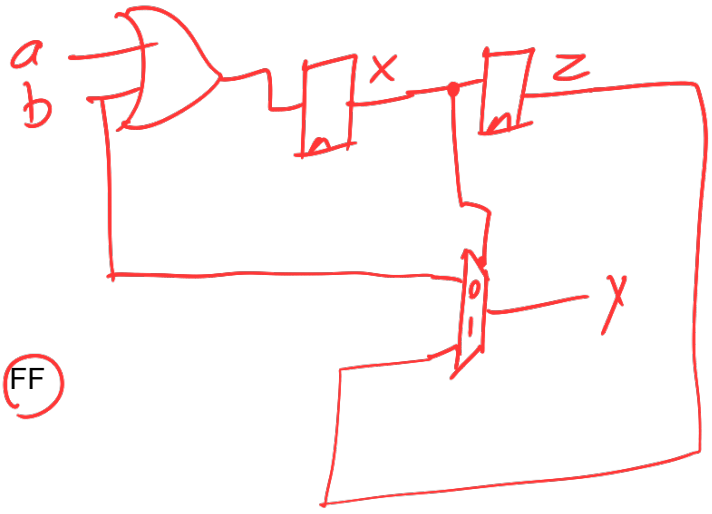
☐



should be connected to the adder

c) Mark the nets as representing the output of combinational logic (CL) or the output of a flip-flop/register (FF).

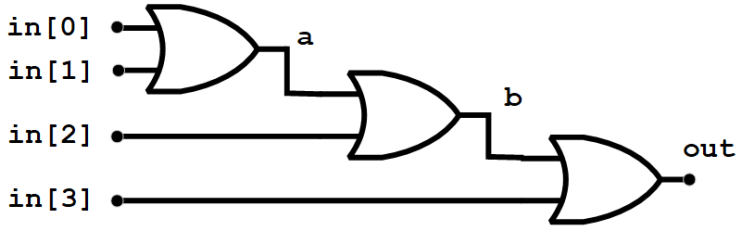
```
wire a,b,c,clk;
reg x,y,z;
always@(posedge clk) begin
      x <= a||b;
      z <= x;
end
always@(*)
      y = x ? z : b;
```



Circle one:

x: CL | FF y: CL | FF z: CL | FF

d) We would like to implement an or-reduction of a multi-bit net efficiently, by using structural Verilog.




```
module chained_or(input [3:0] in, output out);
      wire a, b;
      or (__1__, __2__, __3__);
      or (__4__, __5__, __6__);
      or (__7__, __8__, __9__);
endmodule
```

Write down the expression that should go in each blank in the following table:

1	a	6	in[2]
2	in[0]	7	out
3	in[1]	8	b
4	b	9	in[3]
5	a		

e) Let's make this module parameterized on the width of the input **in**.

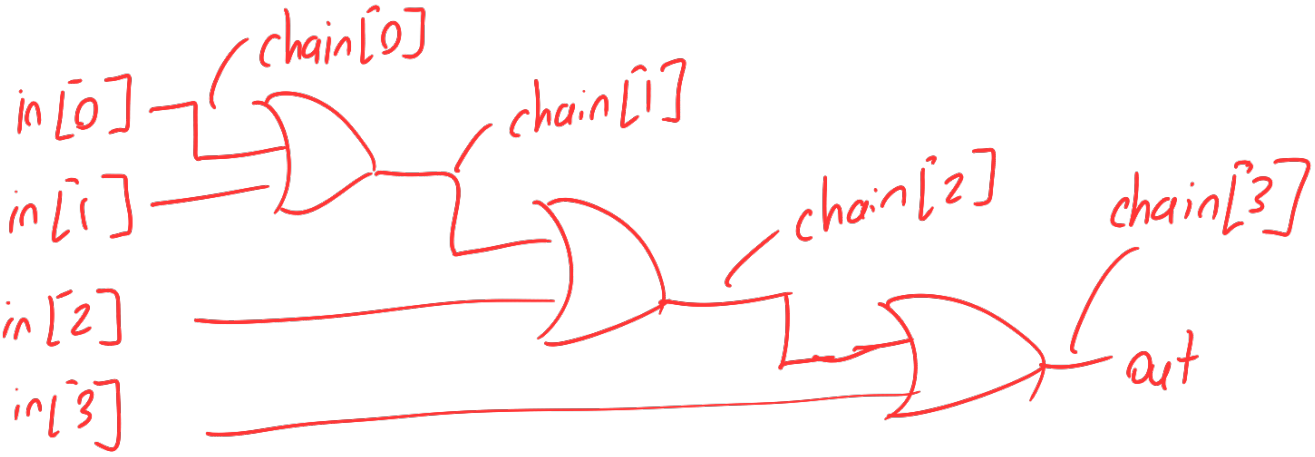
```
module parameterized_chained_or #(WIDTH) (input [WIDTH-1:0] in, output out);
      wire [__1__:0] chain;
      assign chain[0] = in[0];
      genvar i;
      generate for (i = 0; i < __2__; __3__)
            or (__4__, __5__, __6__);
      endgenerate
      assign out = chain[WIDTH];
endmodule
```

should be semicolon

should be WIDTH-1

Write down the expression that should go in each blank in the following table:

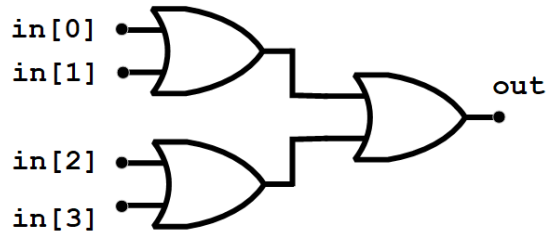
1	WIDTH-1	4	chain[i+1]
2	WIDTH-1	5	chain[i]
3	i = i+1	6	in[i+1]



WIDTH = 4

★ submit a regrade request if you think your answer is still correct (or partially correct) without the corrections given during the exam.

f) **(EECS 251A only!!)** There is a better way to implement or-reduction by using a tree, e.g:




```
module tree_or #(WIDTH) (input [WIDTH-1:0] in, output out);
      generate if (WIDTH == 1)
                  assign out = __1__;
            else if (WIDTH == 2)
                  assign out = __2__;
            else begin
                  wire lower_or, upper_or;
                  tree_or #(.WIDTH(__3__)) lower
                        (.in(in[__4__:0], .out(lower_or));
                  tree_or #(.WIDTH(__5__)) upper
                        (.in(in[WIDTH-1:__6__]), .out(upper_or));
                  assign out = __7__;
            end
      endgenerate
endmodule
```

Write down the expression that should go in each blank in the following table.
Note: WIDTH may not be a power of 2.

1	in	5	WIDTH - WIDTH/2
2	in[0] || in[1] OR |in	6	WIDTH/2
3	WIDTH/2	7	lower_or || upper_or
4	WIDTH/2 - 1		