KEY

Your Name (first last) _____ SID _____

UC Berkeley EECS151
Fall 2019 Midterm 1/A

← Name of person on left (or aisle) _____ TA name _____ 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	Total
Minutes	15	20	25	20	80
Max Points	12	16	24	18	70
Points					

1) *It's all logical…* (12 points, 15 minutes)

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

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

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

$Y = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}cd + a\bar{b}\bar{c}\bar{d} + abc\bar{d}$

1) *It's all logical...* (continued)

b) Draw a Karnaugh map for this expression

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

c) Use the Karnaugh map from part b) to simplify the circuit and write the simplified sum-of-product representation.

Y= $\bar{a}\bar{b} + \bar{b}\bar{c}\bar{d} + abc\bar{d}$

2) *State Machines* (16 points, 20 minutes)

A state transition diagram for a finite state machine (FSM) is shown in Figure 2.

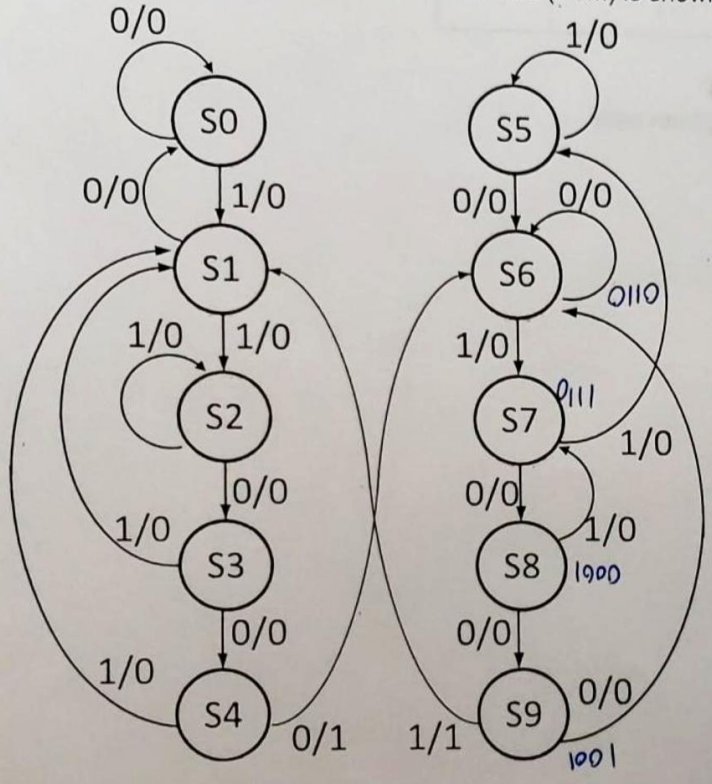


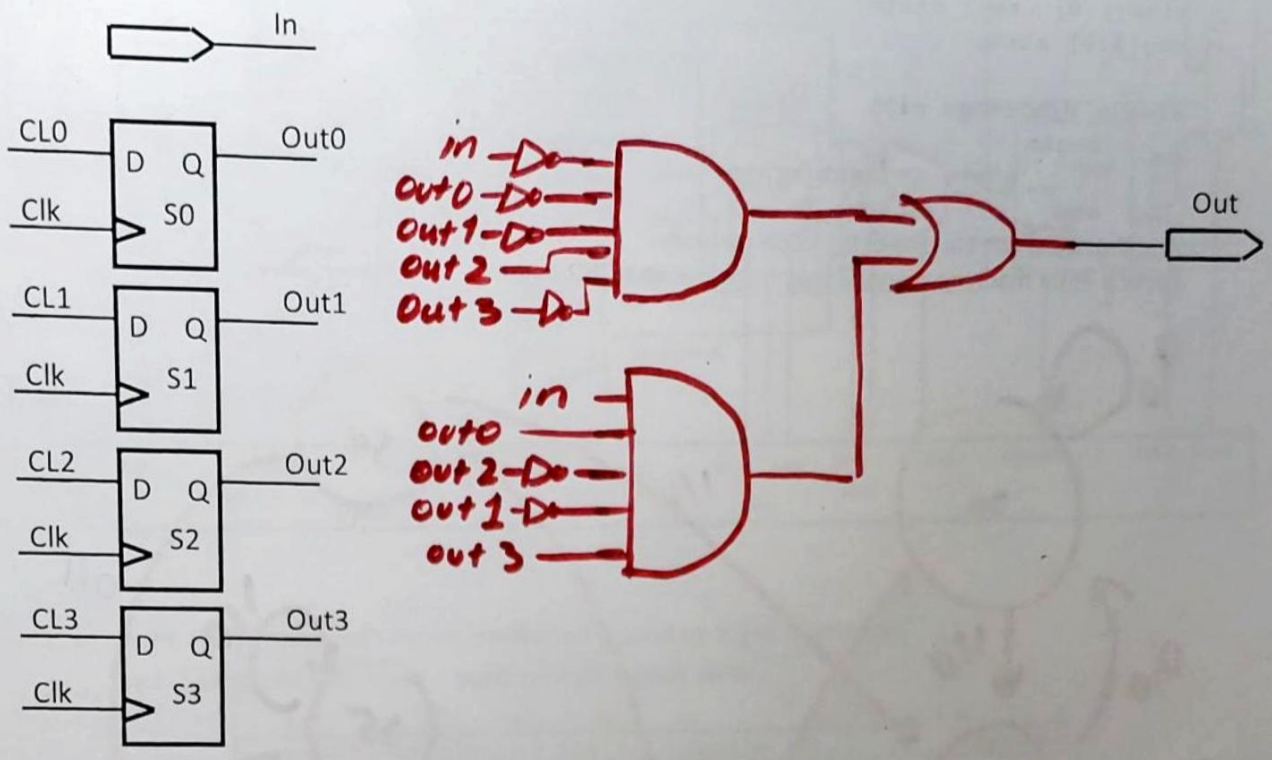
Figure 2.

a) If the FSM starts in state S0, in which state will it be after the input pattern 01011000101?

● S7	○ S0	○ S1	○ S2	○ S8
○ S6	○ S4	○ S3	○ S9	○ S5

2) *State Machines* (continued)

b) If the state is represented by a four-bit register S[3:0] and S4 = 4'b0100 and S9 = 4'b1001, complete the following diagram:



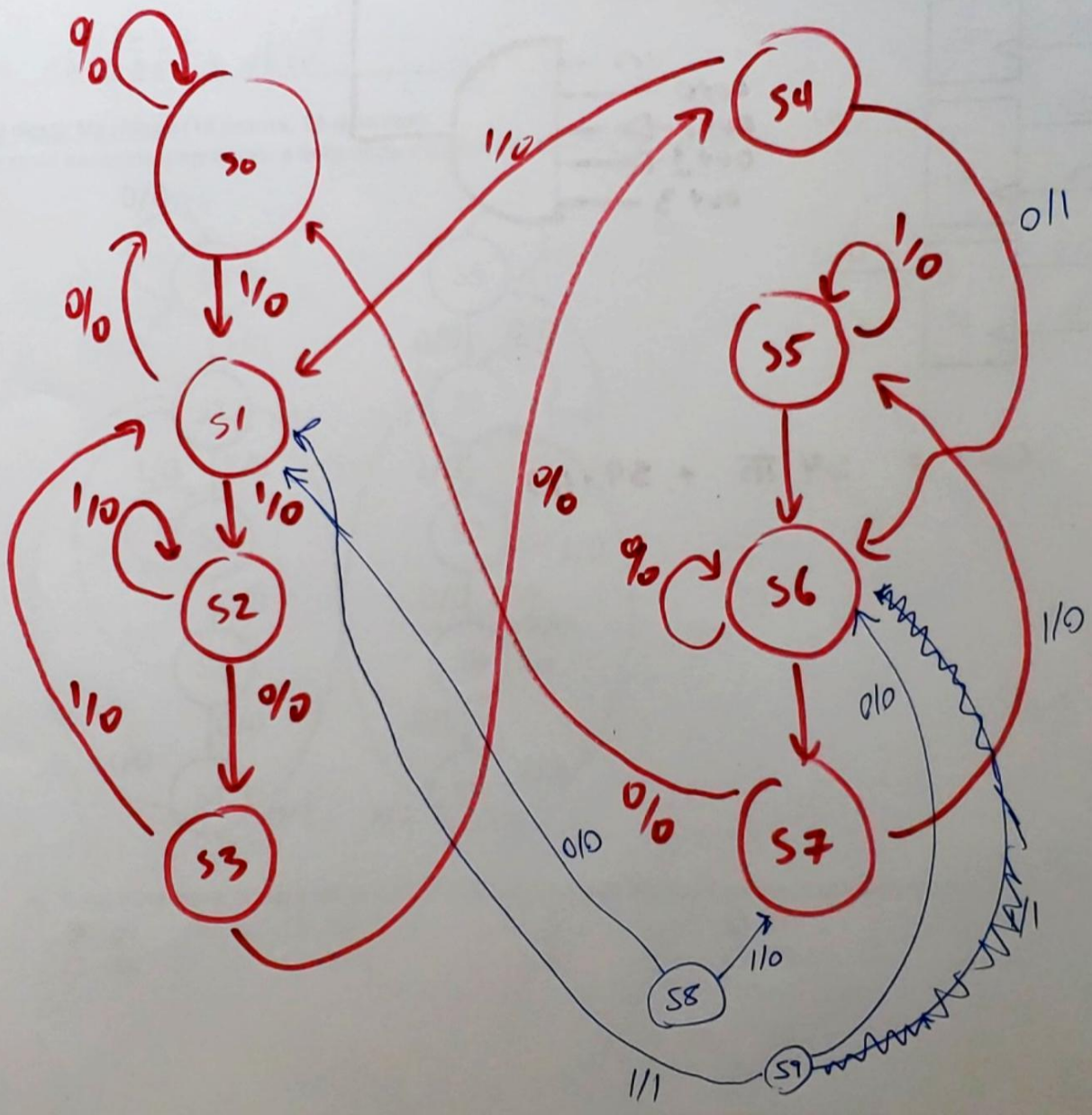
$Out = S4 \cdot \overline{in} + S9 \cdot in$

2) *State Machines* (continued)

c) Your colleague wrote the following code to represent this FSM:

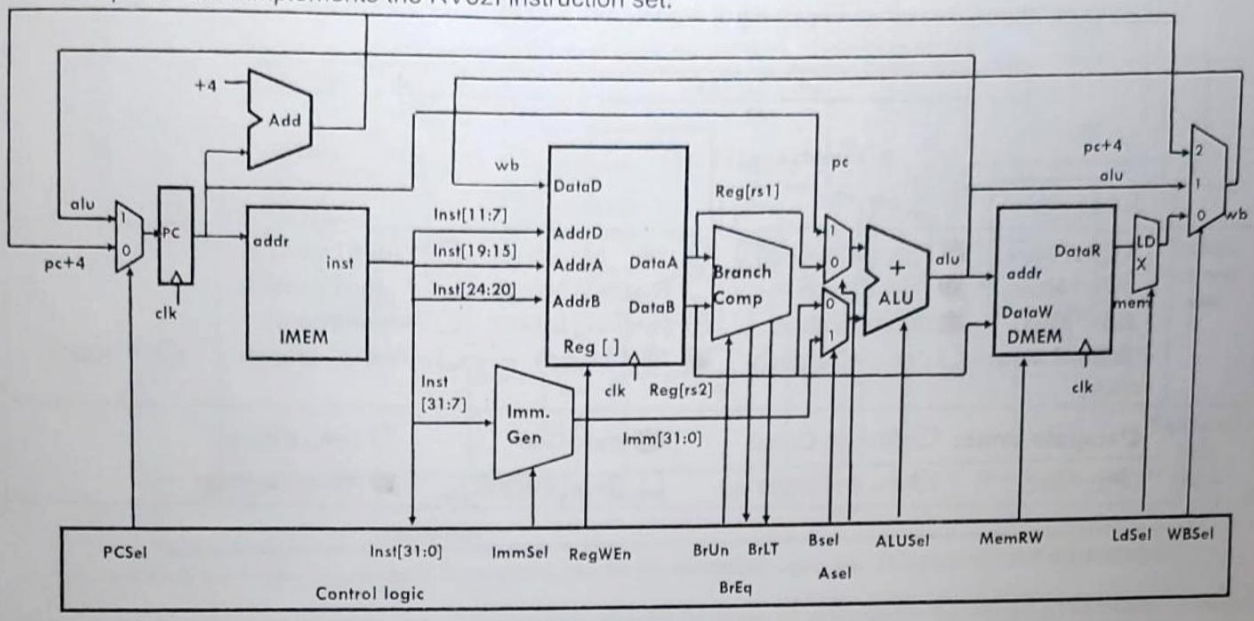
```
wire[2:0]  next_state;
reg[3:0] state;

always @(posedge clk)
      begin
            state <= next_state;
      end
```

And they also got the rest of the code correctly.
Draw a state machine diagram that corresponds to this code.



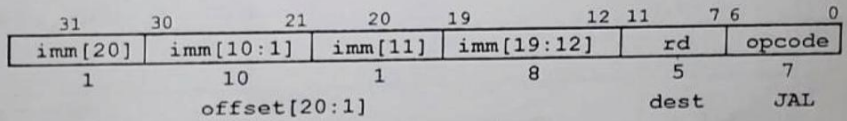
3) *Datapathology* (24 points, 25 minutes)

The datapath below implements the RV32I instruction set.



a) In the RISC-V datapath above, mark what is used by a `jal` instruction.

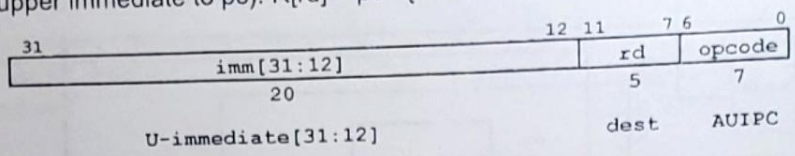
`jal` (Jump and link): R[rd] = pc+4; pc = pc + {imm,1b'0}



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]	

b) In the RISC-V datapath above, mark what is used by a **auipc** instruction.

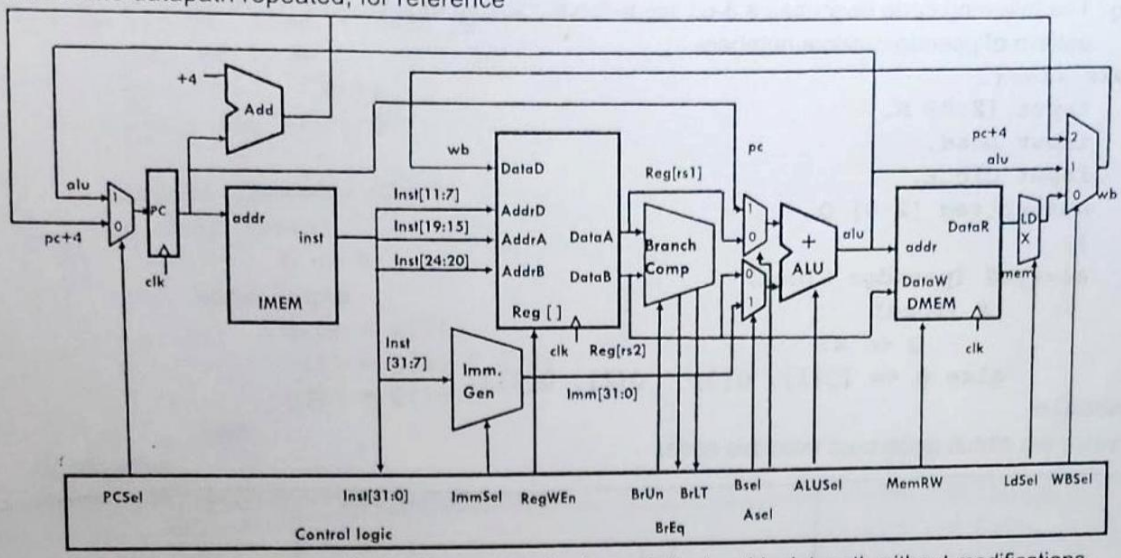
auipc (add upper immediate to pc): R[rd] = pc+ {imm,12b'0}



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]	

3) *Datapathology* (continued)

c) The same datapath repeated, for reference



Specify whether the following proposed instructions can be implemented using this datapath without modifications.

If the instruction can be implemented, specify an expression for the listed control signals, by following the example below.

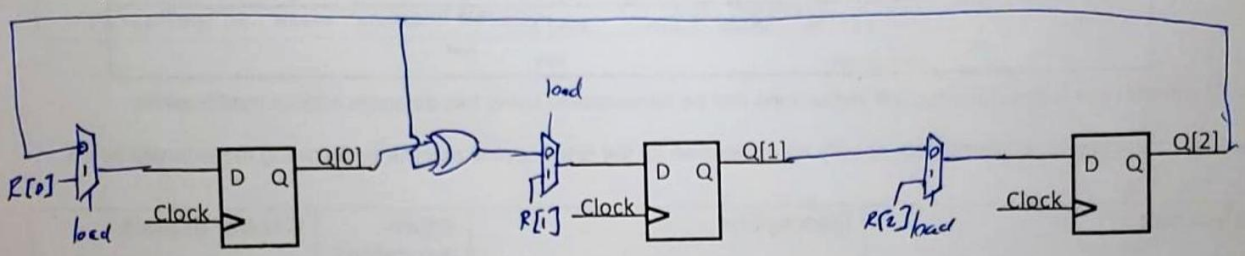
Instruction	Description	Imple-mentable?	Control Signals
Add `add rd, rs1, rs2`	R[rd] = R[rs1] + R[rs2]	Yes	ALUSel = Add WBSel = 1
Load word with add: `lwadd rd, rs1, rs2, imm`	R[rd] = M[R[rs1] + imm] + R[rs2]	NO	RegWEn = X WBSel = X
beq with writeback: `beq rd, rs1, rs2, imm`	R[rd] = R[rs1] + R[rs2] if (R[rs1] == R[rs2]) PC = PC + {imm, 1'b0}	NO	WBSel = X PCSel = Y
PC-relative load: `lwpc rd, imm`	R[rd] = M[PC + imm]	Yes	ASel = 1 BSel = 1
Register offset load: `lwreg rd, rs1, rs2`	R[rd] = M[R[rs1] + R[rs2]]	Yes	ASel = 0 BSel = 0

4) *Verilog* (points, 20 minutes)

a) The following code describes a 3-bit linear-feedback shift register (LFSR), which generates a repeating pattern of pseudo-random numbers.

```
module lfsr(
      input [2:0] R,
      input Load,
      input Clock,
      output reg [2:0] Q
      );
      always@ (posedge Clock)
            if (Load)
                  Q <= R;
            else Q <= {Q[1], Q[0] ^ Q[2], Q[2]};
endmodule
```

Complete the circuit generated from this code:



b) If the initial state of Q[2:0] is 3'b100, write the outputs that correspond to the first 8 cycles:

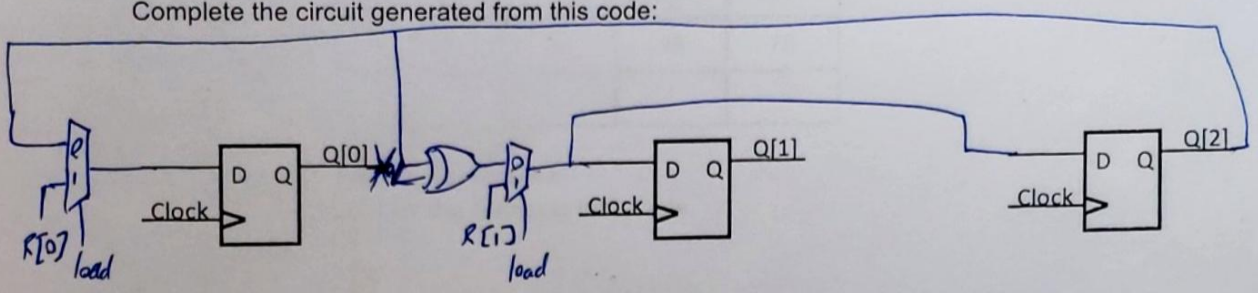
Cycle	Q[2:0]
0	100
1	011
2	110
3	111
4	101
5	001
6	010
7	100

4) *Verilog* (continued)

c) Similar code is shown below:

```
module lfsr(R, Load, Clock, Q) ;
      input [2:0] R;
      input Load, Clock;
      output reg [2:0] Q;

      always@ (posedge Clock)
            if (Load)
                  Q <= R;
            else begin
                  Q[0] = Q[2];
                  Q[1] = Q[0] ^ Q[2] ;
                  Q[2] = Q[1];
            end
endmodule
```

Complete the circuit generated from this code:



Q <= { 0 , 0 , Q[2] }

d) If the R[2:0] value of 3'b100 is loaded initially, write the outputs that correspond to the first 8 cycles:

Cycle	Q[2:0]
0	100
1	001
2	000
3	000
4	000
5	000
6	000
7	000