EECS151/251A Midterm 1

Review Session

Zhenghan Lin & Harrison Liew

Midterm Logistics

- Finish your practice exam ASAP!
  - Remember, everything must emulate the exam day environment/sequence
- Exam: Oct. 6 @ 3:40pm - 5:00pm PST (except for approved accommodations)
  - PDF format, 5 questions + subparts
  - Mostly handwritten (drawing diagrams, multiple choice selections, not the same as previous midterm formats)
  - No long-form written answers, only some short answers
- You will receive an email with a Google Doc link before the exam
  - Links to exam PDF, submission forms, errata

Midterm Scope

5 questions on topics through Lecture 10

1. Boolean logic (translation, simplification)
2. FSMs (construct from specification, waveforms)
3. RISC-V instructions (instr. formats, address ranges, pseudo-insts)
4. RISC-V datapath (Verilog of datapath/control logic, extending RV32I w/ new inst., timing)
5. Verilog (deduce function from code, write some small code snippets)

Logic (FA18, Problem 1)

[PROBLEM 1] Combinational Logic Design and Optimization (10 Pts)

a) Consider the following truth table. Use K-maps to derive a minimized sum-of-products expression for the outputs **C_0 and S_0 only**. (4 Pts)

A_1	A_0	B_1	B_0	C_0	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

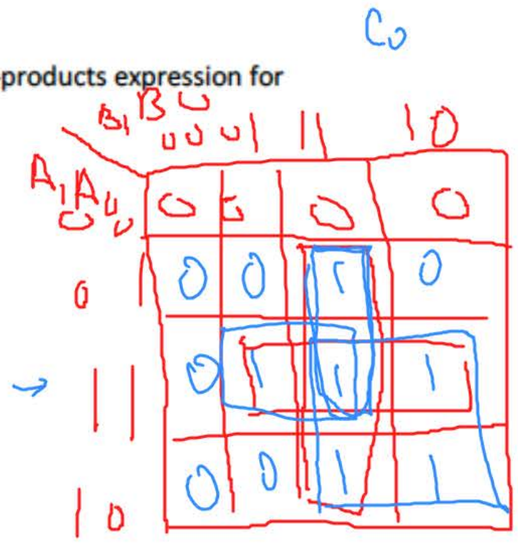
b) What function does this truth table represent? Explain in words – no gate-level drawings are required. (1 Pts)

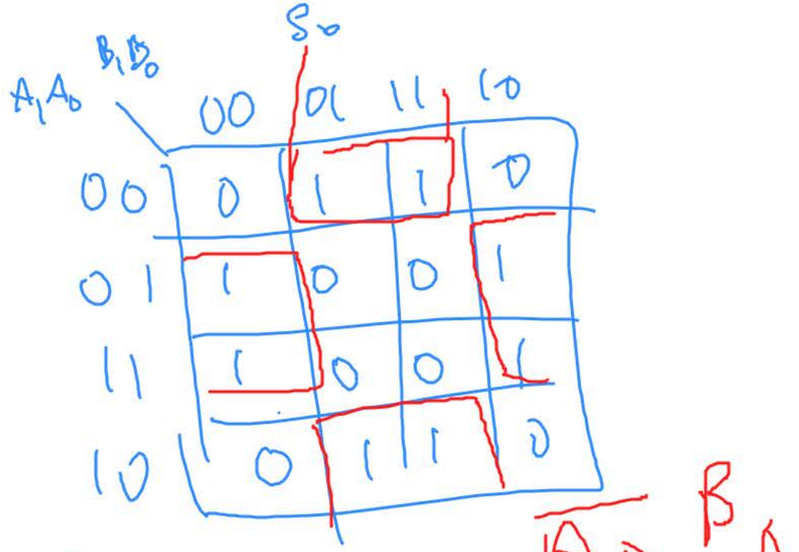
Logic (FA18, Problem 1)

[PROBLEM 1] Combinational Logic Design and Optimization (10 Pts)

a) Consider the following truth table. Use K-maps to derive a minimized sum-of-products expression for the outputs **C_0 and S_0 only**. (4 Pts)

A_1	A_0	B_1	B_0	C_0	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0





$C_0 = A_1B_1 + A_1A_0B_0 + A_0B_1B_0$

$S = \bar{A}_0B_0 + A_0\bar{B}_0 = A_0 \oplus B_0$

b) What function does this truth table represent? Explain in words – no gate-level drawings are required. (1 Pts)

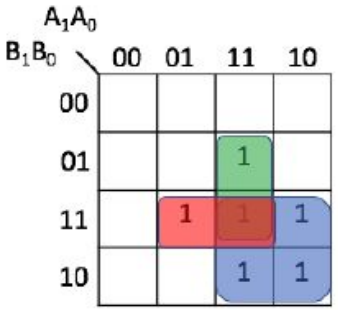
Logic (FA18, Problem 1)

[PROBLEM 1] Combinational Logic Design and Optimization (10 Pts)

a) Consider the following truth table. Use K-maps to derive a minimized sum-of-products expression for the outputs **C_0 and S_0 only**. (4 Pts)

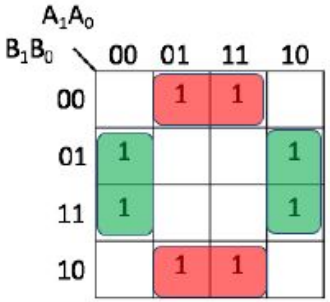
A_1	A_0	B_1	B_0	C_0	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

C_0



$C_0 = A_1B_1 + A_1A_0B_0 + A_0B_1B_0$

S_0



$S_0 = A_0\overline{B_0} + \overline{A_0}B_0 = A_0 \oplus B_0$

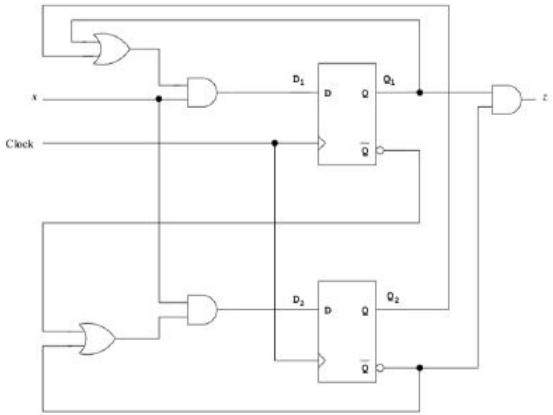
b) What function does this truth table represent? Explain in words – no gate-level drawings are required. (1 Pts)

This is a 2-bit adder. It adds A and B which are each 2 bits wide (without a carry-in) and produces a 2 bit sum S and a single carry out bit C_0.

FSM (FA18, Problem 2)

[PROBLEM 2] Finite State Machines (10 Pts)

Consider the following circuit with *x* the input and *z* the output.



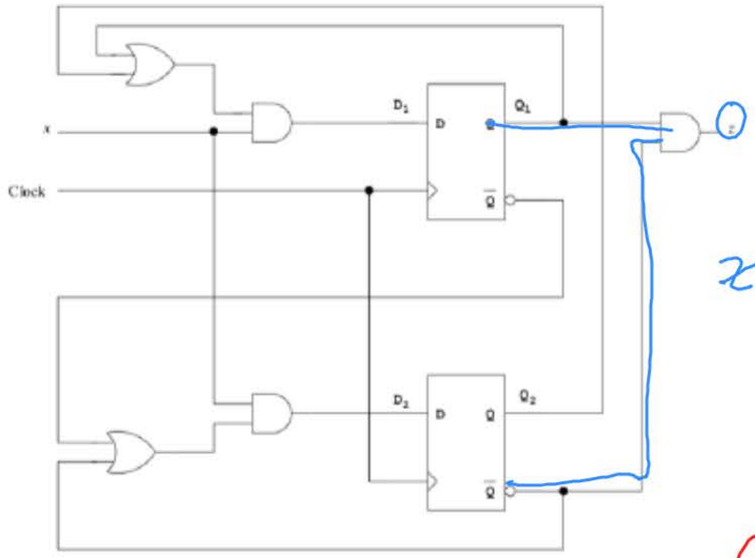
a) Is this a Mealy or a Moore machine? Why? (2 Pts)

b) Write the Boolean expressions for Flip-Flop inputs D_1 and D_2, and the system output z. (2 Pts)

FSM (FA18, Problem 2)

[PROBLEM 2] Finite State Machines (10 Pts)

Consider the following circuit with *x* the input and *z* the output.



a) Is this a Mealy or a Moore machine? Why? (2 Pts)

b) Write the Boolean expressions for Flip-Flop inputs D_1 and D_2, and the system output z. (2 Pts)

$z = Q_1 \bar{Q}_2$

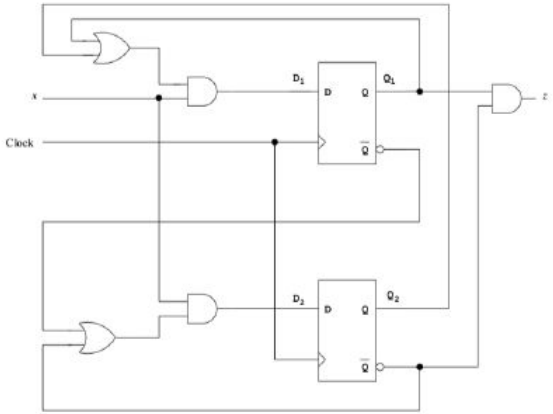
$D_1 = (Q_1 + Q_2)\, x$

$D_2 = (\bar{Q}_1 + \bar{Q}_2)\, x$

FSM (FA18, Problem 2)

[PROBLEM 2] Finite State Machines (10 Pts)

Consider the following circuit with *x* the input and *z* the output.



a) Is this a Mealy or a Moore machine? Why? (2 Pts)

This is a Moore machine. The output z depends only on the state registers Q_1 and Q_2, and not on the input x.

b) Write the Boolean expressions for Flip-Flop inputs D_1 and D_2, and the system output z. (2 Pts)

$$D_1 = x \cdot (Q_1 + Q_2)$$

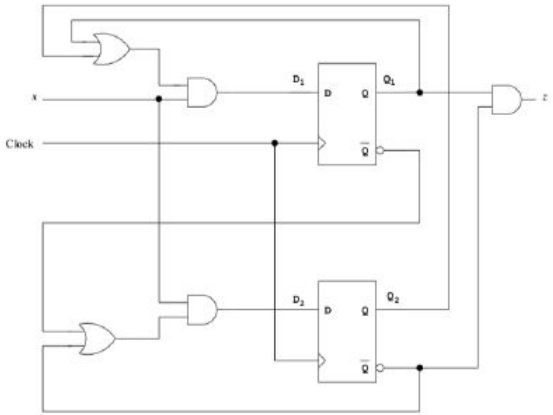
$$D_2 = x \cdot (Q'_1 + Q'_2)$$

$$z = Q_1 Q'_2$$

FSM (FA18, Problem 2 continued)

[PROBLEM 2] Finite State Machines (10 Pts)

Consider the following circuit with *x* the input and *z* the output.



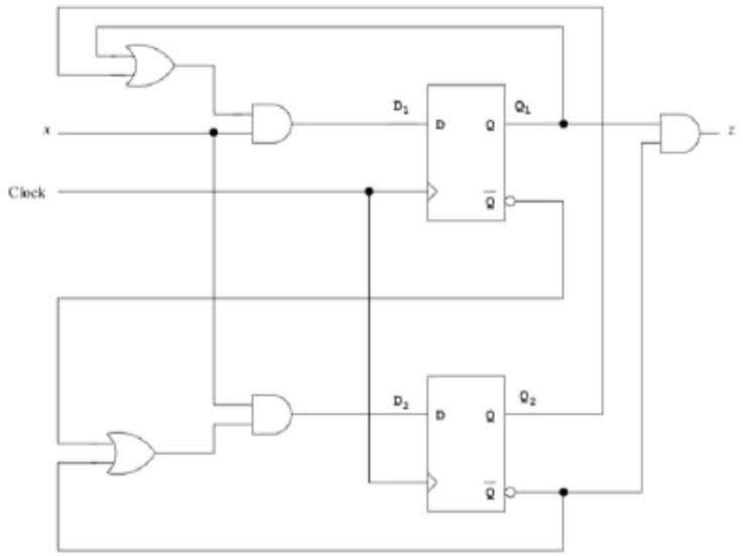
c) Derive the state transition table for the circuit. (3 Pts)

d) Draw a state diagram for the system. (3 Pts)

FSM (FA18, Problem 2 continued)

[PROBLEM 2] Finite State Machines (10 Pts)

Consider the following circuit with *x* the input and *z* the output.



c) Derive the state transition table for the circuit. (3 Pts)

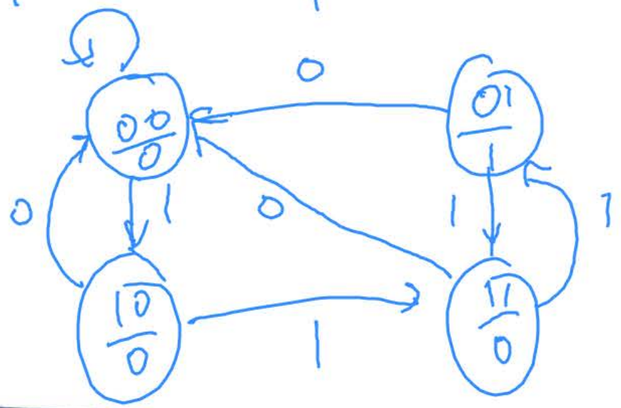
$D_1 = x(Q_1 + Q_2)$

$D_2 = x(\bar{Q}_1 + \bar{Q}_2)$

$z = Q_1\bar{Q}_2$

Present State Q_2 Q_1	Input x	Next State Q_2 Q_1	Output z
0 0	0	0 0	0
0 0	1	1 0	0
0 1	0	0 0	1
0 1	1	1 1	1
1 0	0	0 0	0
1 0	1	1 1	0
1 1	0	1 0	0
1 1	1	0 1	0

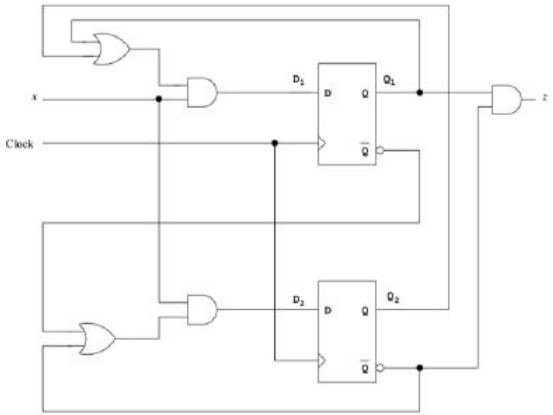
d) Draw a state diagram for the system. (3 Pts)



FSM (FA18, Problem 2)

[PROBLEM 2] Finite State Machines (10 Pts)

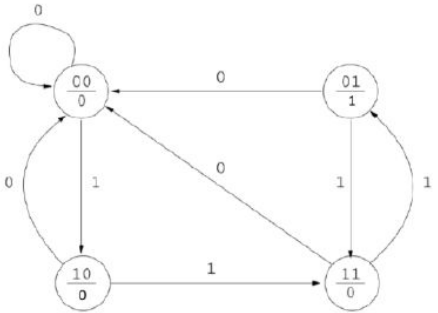
Consider the following circuit with x the input and z the output.



Present State		Input	Next State		Output
Q_2	Q_1	x	Q_2	Q_1	z
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	0	1	0

c) Derive the state transition table for the circuit. (3 Pts)

d) Draw a state diagram for the system. (3 Pts)



FSM (FA19, Problem 2)

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

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

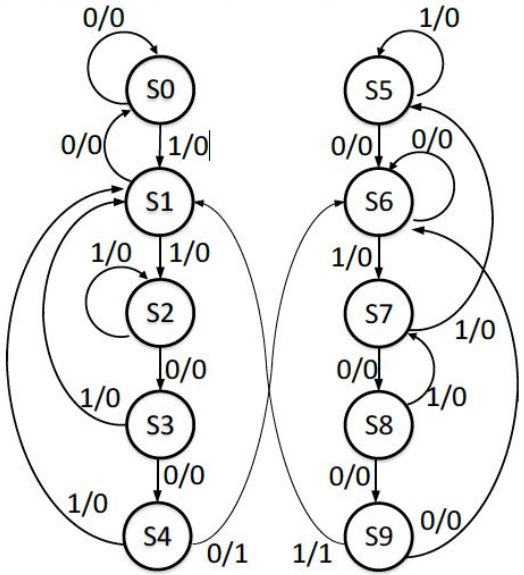


Figure 2.

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

FSM (FA19, Problem 2)

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

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

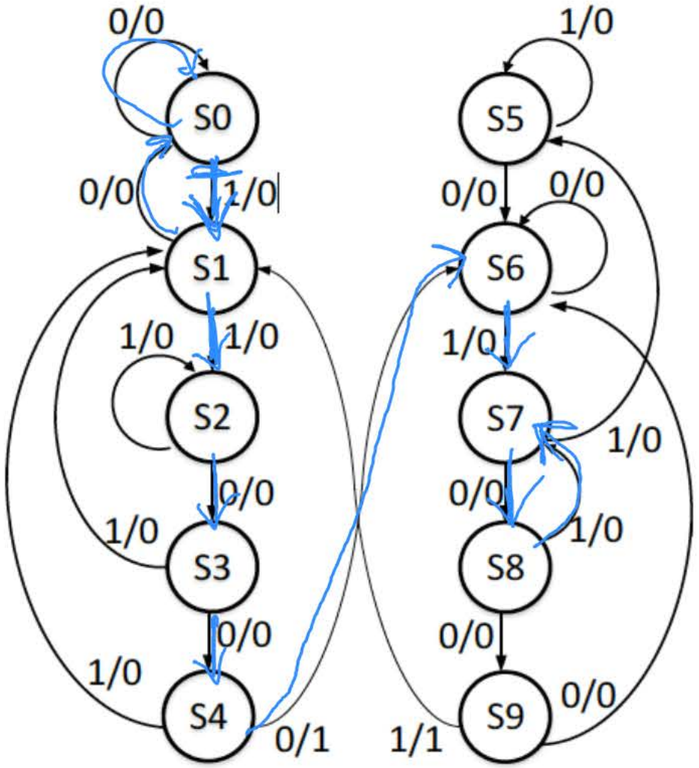


Figure 2.

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

FSM (FA19, Problem 2 continued)

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

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

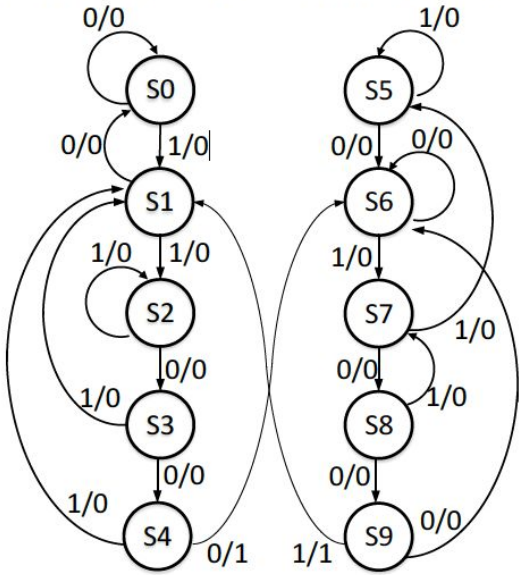
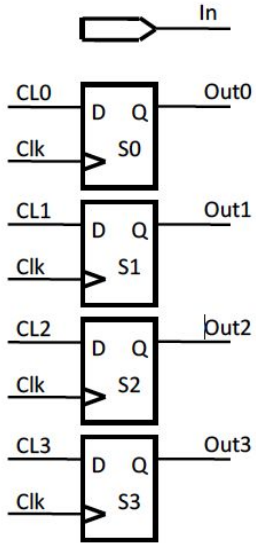
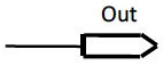


Figure 2.

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:





FSM (FA19, Problem 2 continued)

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

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

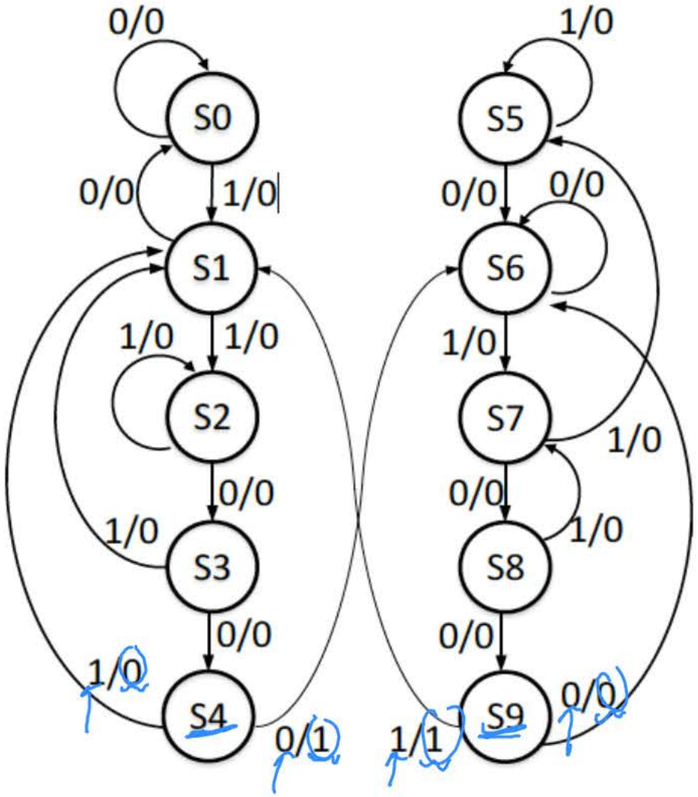
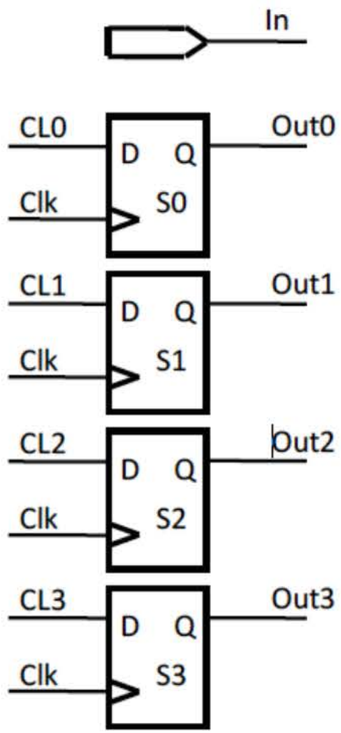


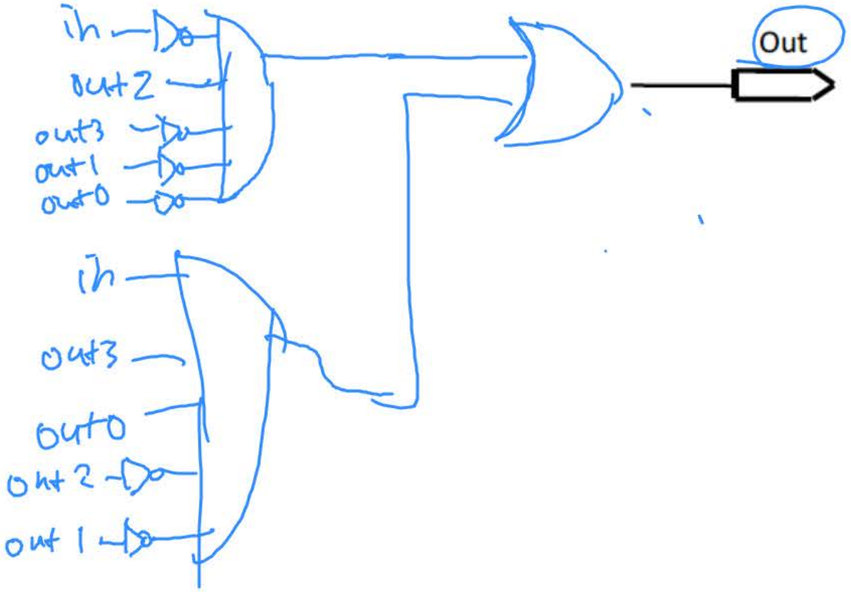
Figure 2.

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:



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



FSM (FA19, Problem 2 continued)

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

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

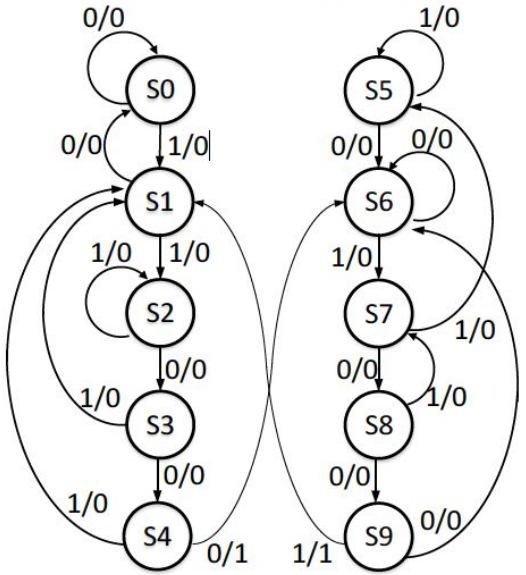
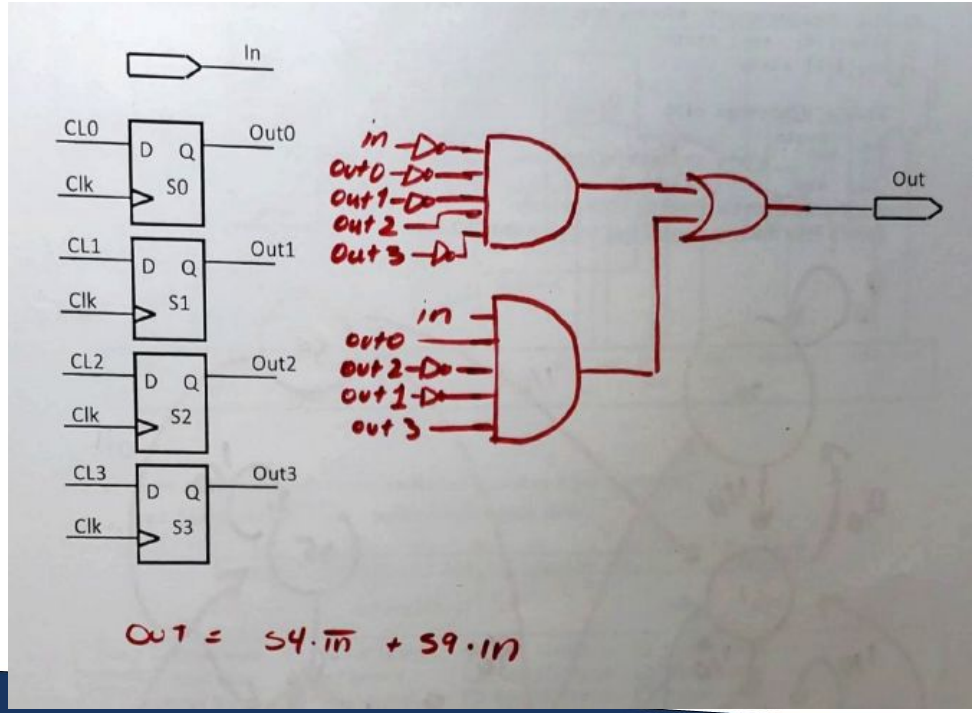


Figure 2.

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:



FSM (FA19, Problem 2 continued)

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

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

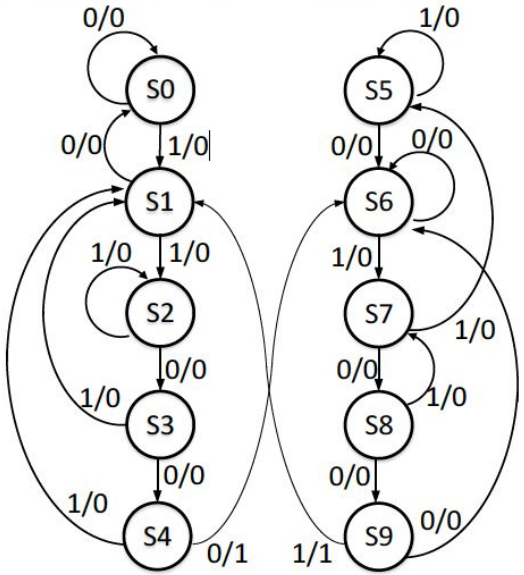


Figure 2.

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.

FSM (FA19, Problem 2 continued)

3) *Datapathology* (24 points, 25 minutes)
The datapath below implements the RV32I instruction set.

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save

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

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

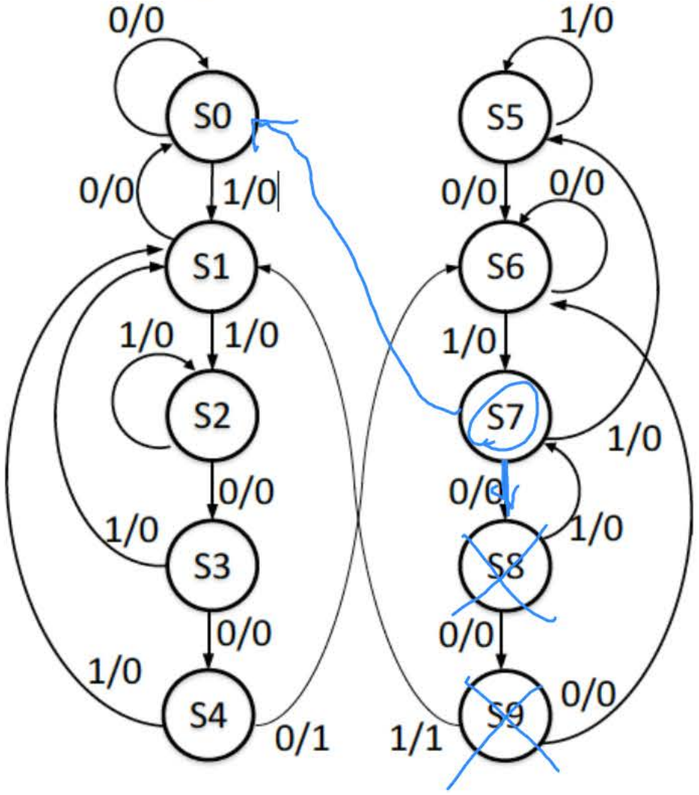


Figure 2.

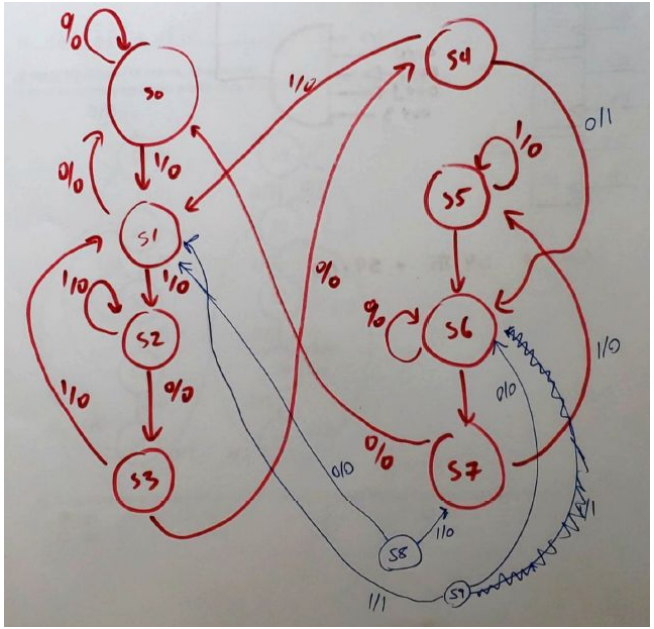
2) *State Machines* (continued)

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.

FSM (FA19, Problem 2 continued)



2) *State Machines* (continued)

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

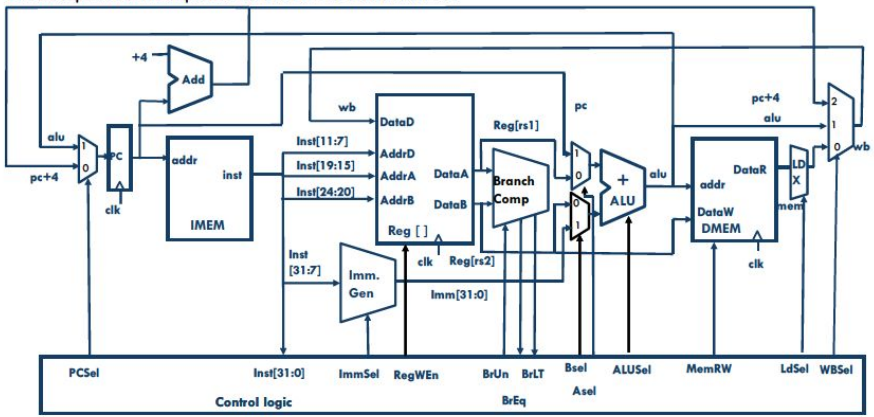
```
wire[2:0]  next_state;
reg[3:0] state;

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.

RISC-V Datapath (FA19, Problem 3)

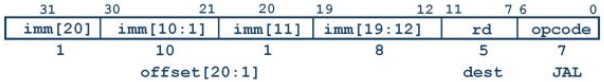
3) *Datapathology* (24 points, 25 minutes)

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}

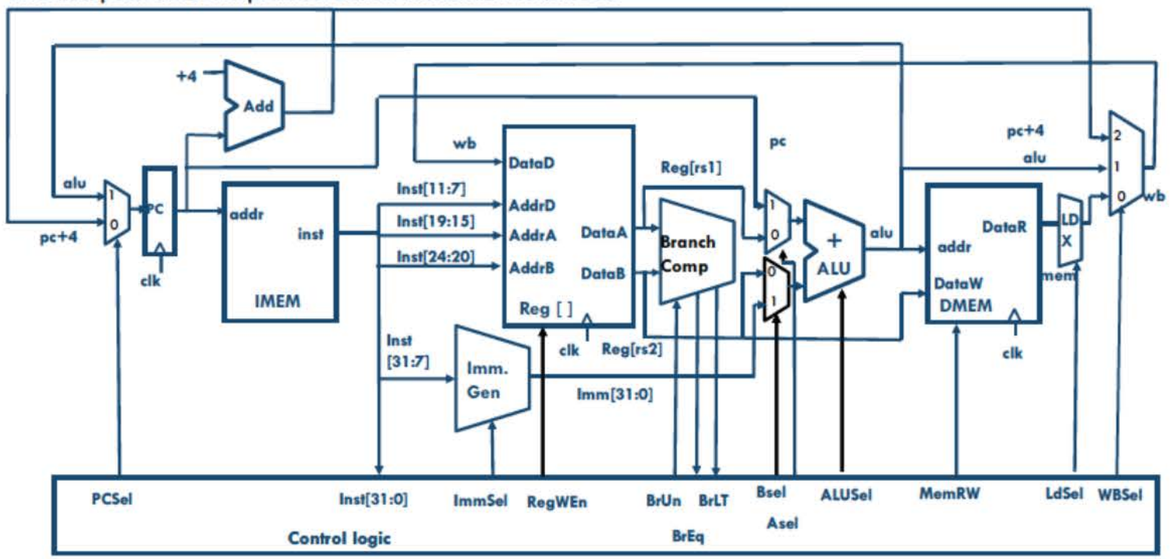


Select one per row	**PCSel Mux:** ○ "pc + 4" branch	○ "alu" branch	○ * (don't care)		
	ASel Mux: ○ "pc" branch	○ Reg[rs1] branch	○ * (don't care)		
	BSel Mux: ○ "imm" branch	○ Reg[rs2] branch	○ * (don't care)		
	WBSel Mux: ○ "pc + 4" branch	○ "alu" branch	○ "mem" branch	○ * (don't care)	
Select all that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

RISC-V Datapath (FA19, Problem 3)

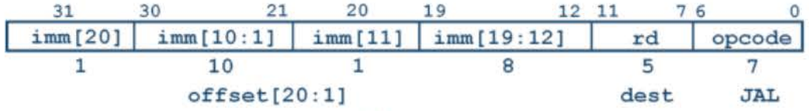
3) *Datapathology* (24 points, 25 minutes)

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}

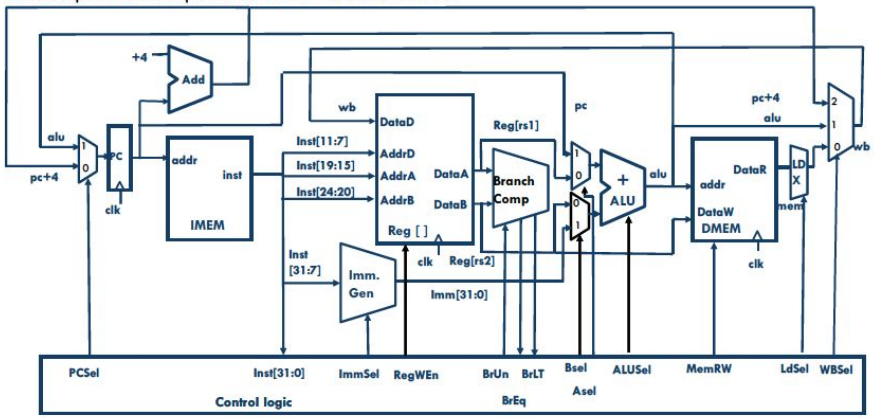


Select one per row	**PCSel Mux:** ○ "pc + 4" branch	● "alu" branch	○ * (don't care)		
	ASel Mux: ● "pc" branch	○ Reg[rs1] branch	○ * (don't care)		
	BSel Mux: ○ "imm" branch	○ Reg[rs2] branch	○ * (don't care)		
	WBSel Mux: ○ "pc + 4" branch	○ "alu" branch	○ "mem" branch	○ * (don't care)	
Select all that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

RISC-V Datapath (FA19, Problem 3 continued)

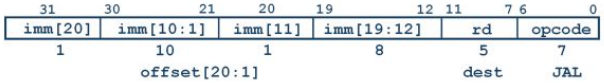
3) *Datapathology* (24 points, 25 minutes)

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.

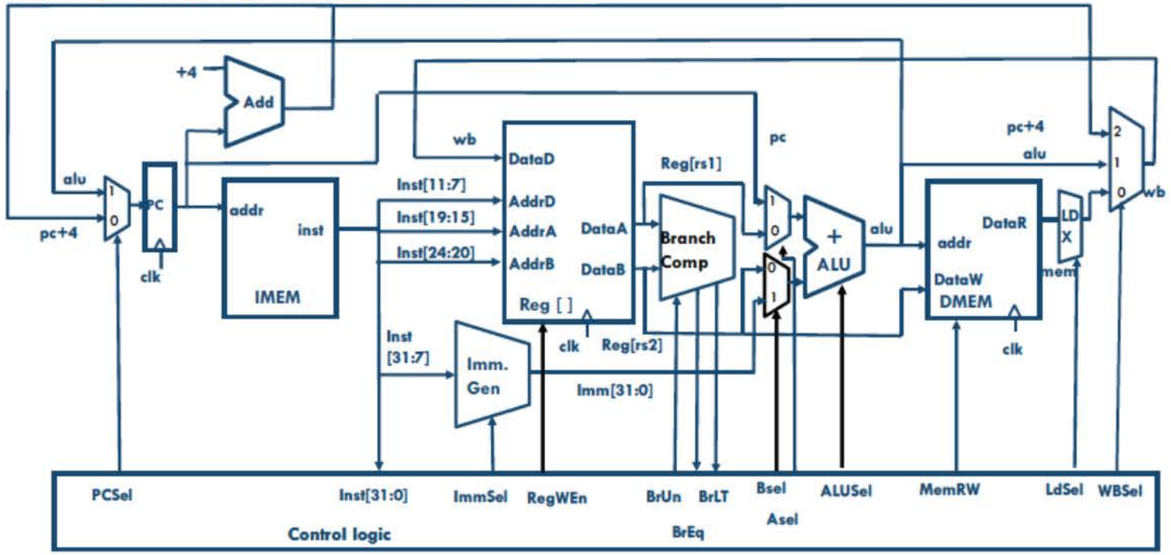
`auipc` (add upper immediate to pc): R[rd] = pc+ {imm,12b'0}

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]		

RISC-V Datapath (FA19, Problem 3 continued)

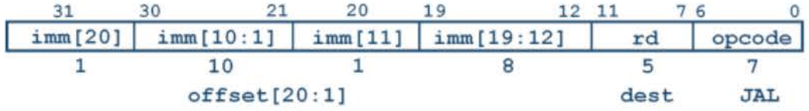
3) *Datapathology* (24 points, 25 minutes)

The datapath below implements the RV32I instruction set.



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

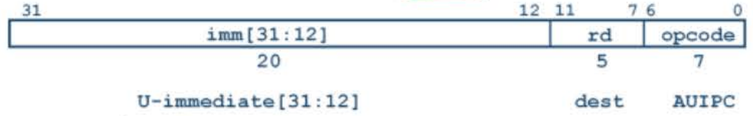
`jal` (Jump and link): R[rd] = pc+4; pc = pc + {imm,1b'0}



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}

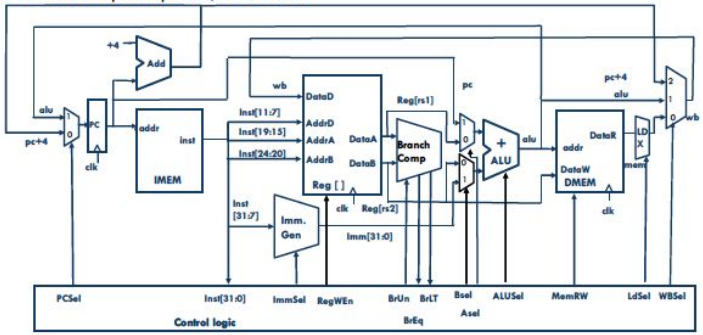


Select one per row	**PCSel Mux:** ○ "pc + 4" branch ○ "alu" branch ○ * (don't care) **ASel Mux:** ○ "pc" branch ○ Reg[rs1] branch ○ * (don't care) **BSel Mux:** ○ "imm" branch ○ Reg[rs2] branch ○ * (don't care) **WBSel Mux:** ○ "pc + 4" branch ○ "alu" branch ○ "mem" branch ○ * (don't care)
Select all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

RISC-V Datapath (FA19, Problem 3 continued)

3) *Datapathology* (continued)

c) The same datapath repeated, for reference



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

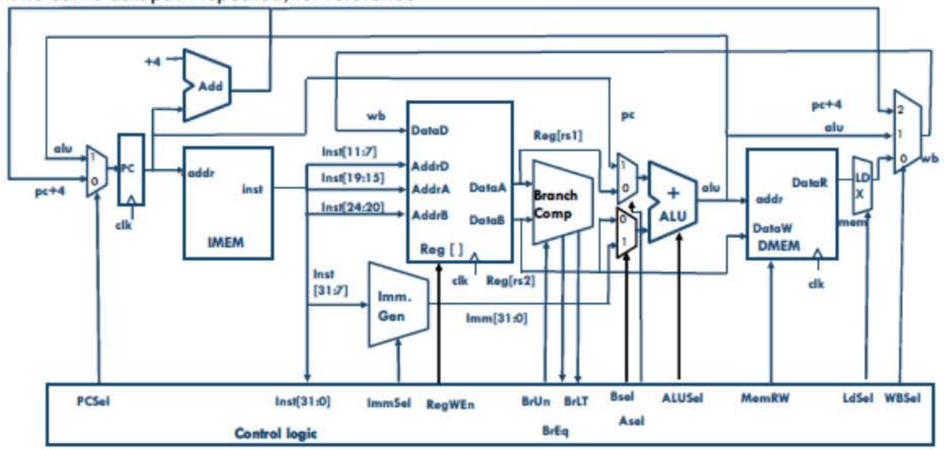
If the instruction can be implemented, specify an expression for the listed control signals, by following the example below.

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]		RegWEn = WBSel =
beq with writeback: `beq rd, rs1, rs2, imm`	R[rd] = R[rs1] + R[rs2] if (R[rs1] == R[rs2]) PC = PC + {imm, 1'b0}		WBSel = PCSel =
PC-relative load: `lwpc rd, imm`	R[rd] = M[PC + imm]		ASel = BSel =
Register offset load: `lwreg rd, rs1, rs2`	R[rd] = M[R[rs1] + R[rs2]]		ASel = BSel =

RISC-V Datapath (FA19, Problem 3 continued)

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]	X	RegWEn = WBSel =
beq with writeback: `beq rd, rs1, rs2, imm`	R[rd] = R[rs1] + R[rs2] if (R[rs1] == R[rs2]) PC = PC + {imm, 1'b0}	X	WBSel = PCSel =
PC-relative load: `lwpc rd, imm`	R[rd] = M[PC + imm]	✓	ASel = 1 (PC BSel = 1 imm
Register offset load: `lwreg rd, rs1, rs2`	R[rd] = M[R[rs1] + R[rs2]]	✓	ASel = 0 BSel = 0

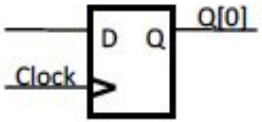
Verilog (FA19, Problem 4)

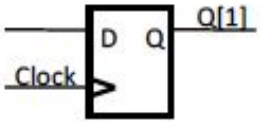
4) *Verilog* (points, 20 minutes)

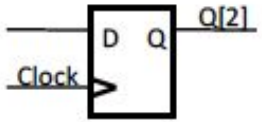
a) The following code describes a 3-bit linear-feedback shift register (LFSR), which generates a repeating pattern of pseudo-random numbers.

```
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:







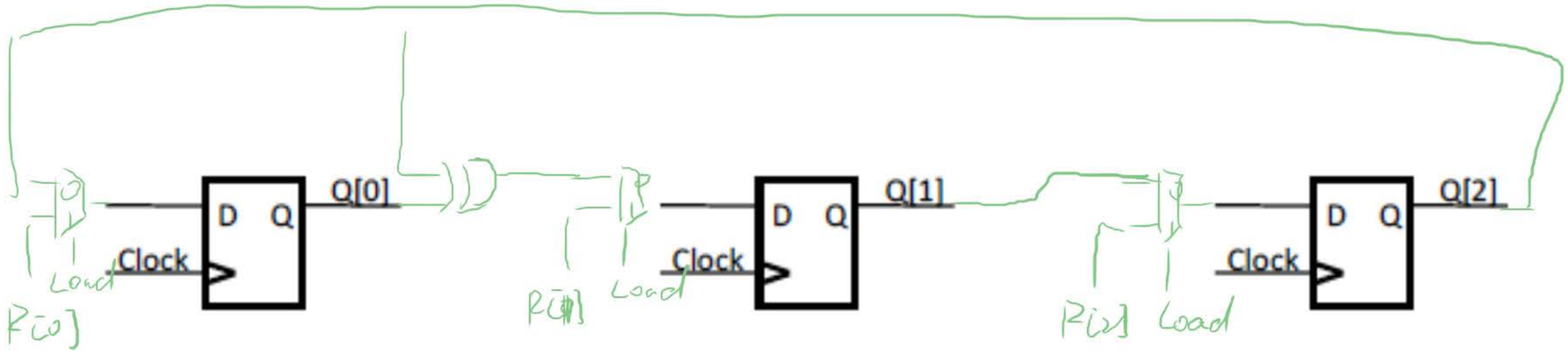
Verilog (FA19, Problem 4)

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:

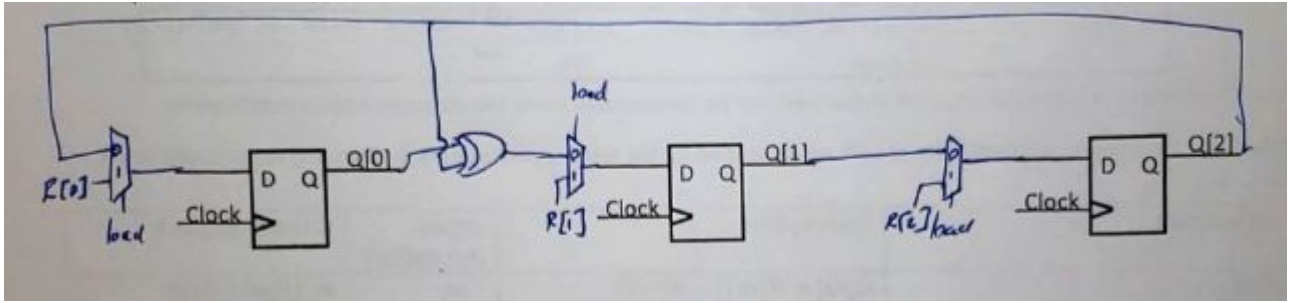


Verilog (FA19, Problem 4)

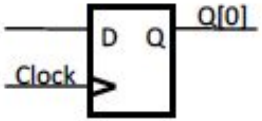
4) *Verilog* (points, 20 minutes)

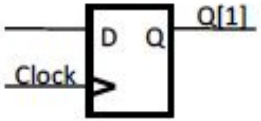
a) The following code describes a 3-bit linear-feedback shift register (LFSR), which generates a repeating pattern of pseudo-random numbers.

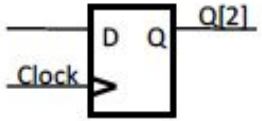
```
module lfsr(
      input [2:0] R,
      input Load,
      input Clock,
      output reg [2:0] Q
      );
      always@ (posedge Clock)
            if (Load)
                  Q <= R;
            else Q <= {Q[1], Q[0] ^ Q[2], Q[2]};
endmodule
```



Complete the circuit generated from this code:







Verilog (FA19, Problem 4 continued)

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
```

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	
2	
3	
4	
5	
6	
7	

Verilog (FA19, Problem 4 continued)

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
```

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

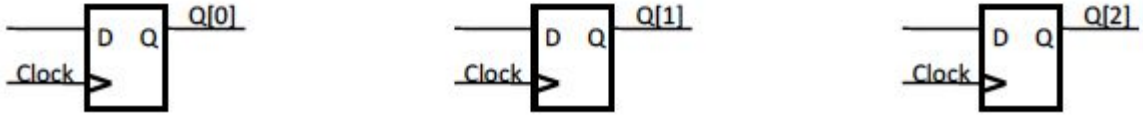
Verilog (FA19, Problem 4 continued)

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:



Verilog (FA19, Problem 4 continued)

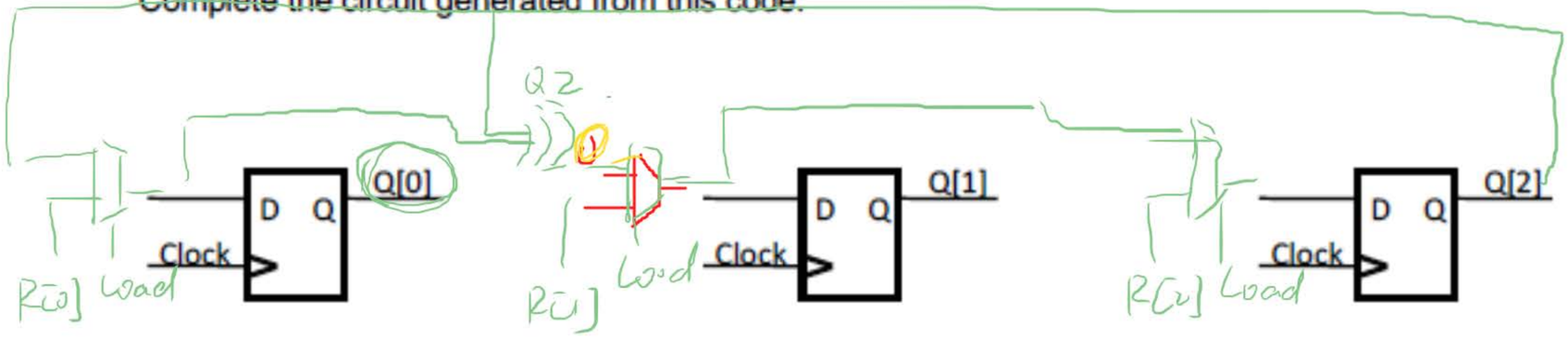
4) *Verilog* (continued)

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
```

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

Complete the circuit generated from this code:



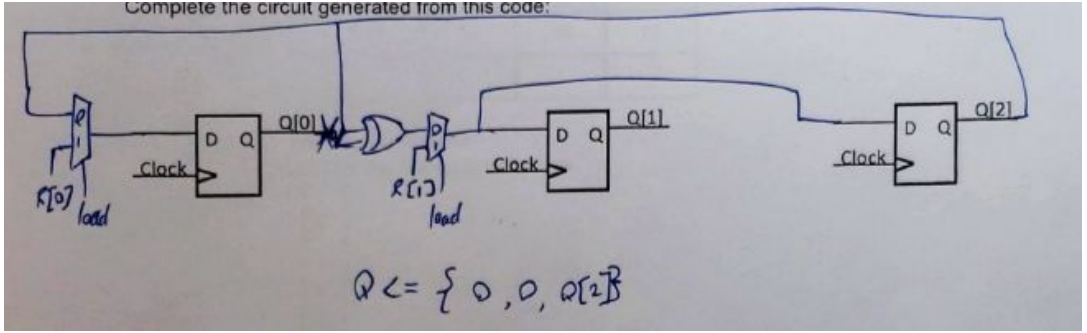
Verilog (FA19, Problem 4 continued)

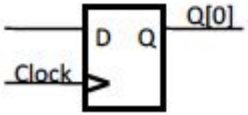
4) *Verilog* (continued)

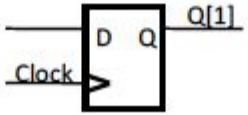
c) Similar code is shown below:

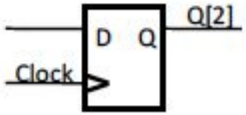
```
module lfsr(R, Load, Clock, Q) ;
      input [2:0] R;
      input Load, Clock;
      output reg [2:0] Q;

      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:









Verilog (FA19, Problem 4 continued)

4) *Verilog* (continued)

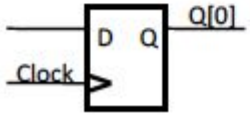
c) Similar code is shown below:

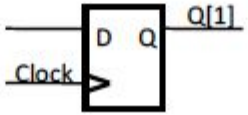
```
module lfsr(R, Load, Clock, Q) ;
      input [2:0] R;
      input Load, Clock;
      output reg [2:0] Q;

      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
```

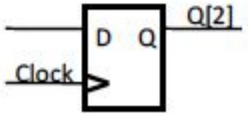
Complete the circuit generated from this code:

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	
2	
3	
4	
5	
6	
7	







Verilog (FA19, Problem 4 continued)

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:

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

$Q \in \{0, 0, Q[2]\}$