1) It's all logical... (12 points, 15 minutes)
You need to design a circuit for the following truth table.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

\[ Y = \]  

b) Draw a Karnaugh map for this expression

\[
\begin{array}{cccc}
00 & 01 & 11 & 10 \\
\hline
00 & & & \\
01 & & & \\
11 & & & \\
10 & & & \\
\end{array}
\]

c) Use the Karnaugh map from part b) to simplify the circuit and write the simplified sum-of-product representation.
Follow-up Question:
What kind of machine is this?

Follow-up Question:
How many registers do we need to maintain state?
FA19, MT1 (FSM)

2) State Machines (continued)
   b) If the state is represented by a four-bit register $S[3:0]$ and $S_4 = 4'b0100$ and $S_9 = 4'b1001$, complete the following diagram:

   ![Diagram of state machines](image)

   Clarification:
   Only draw the logic relating to out, you may ignore CL0-3.
2) **Verilog** (34 points, 35 minutes)

a) (4 pts) Are the circuits inferred by the two blocks always equivalent? Mark Yes or No. Incorrect answers carry negative credit.

i)

```verilog
wire [9:0] a, b;
wire [10:0] c;
assign c = a + b;
```

```verilog
wire [9:0] a, b;
wire [10:0] c;
assign c = $signed(a) + $signed(b)
```

ii)

```verilog
wire [7:0] a, c;
wire [2:0] b;
assign c = a >>> b;
```

```verilog
wire [7:0] a, c;
wire [2:0] b;
assign c = $signed(a) >>> b;
```
FA21, MT1 (Verilog)

iii)

```verilog
wire a;
assign a = 1'b0;
```

iv)

```verilog
wire [1:0] a;
reg [3:0] b;
always @(*) begin
    case(a)
        2'b1x: b = 4;
        2'b01: b = 3;
        2'b0x: b = 2;
    endcase
end
```

```verilog
wire [1:0] a;
reg [3:0] b;
always @(*) begin
    b = 3;
    if (a == 2'b01) b = 2;
    else if (a == 2'b00) b = 2;
    else if (a == 2'b10) b = 4;
end
```

Assume that a case item with an "x" is matched when the case expression is 0 or 1. E.g. 3'b11x matches 3'b110 and 3'b111.
b) (2 pts) What circuit does this Verilog synthesize to?

```verilog
wire [1:0] a;
wire clk;
reg [1:0] b, c, d;

always @(posedge clk) begin
    b <= a;
    c = b + 2'b1;
    d <= c + 2'b2;
end
```
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}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>offset</td>
<td>dest</td>
<td>JAL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Select one per row:
- PCSel Mux: ○ “pc + 4” branch ○ “alu” branch ○ * (don’t care)
- ASel Mux: ○ “pc” branch ○ Reg[r1] branch ○ * (don’t care)
- BSel Mux: ○ “imm” branch ○ Reg[r2] 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[r1] ☐ Read Reg[r2] ☐ Write Reg[rd]
3) Datapathology (24 points, 25 minutes)

The datapath below implements the RV32I instruction set.

b) In the RISC-V datapath above, mark what is used by a \texttt{auipc} instruction.

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

\begin{align*}
\text{imm}[31:12] & \quad 31 \quad 30 \quad 29 \quad 28 \quad 27 \quad 26 \quad 25 \quad 24 \quad 23 \quad 22 \quad 21 \quad 20 \\
\text{rd} & \quad 19 \quad 18 \quad 17 \quad 16 \quad 15 \quad 14 \quad 13 \quad 12 \quad 11 \quad 10 \quad 9 \quad 8 \quad 7 \quad 6 \quad 0 \\
\text{dest} & \quad 7 \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \quad 0 \\
\text{opcode} \quad & \quad 31 \quad 30 \quad 29 \quad 28 \quad 27 \quad 26 \quad 25 \quad 24 \quad 23 \quad 22 \quad 21 \quad 20 \\
\text{U-immediate}[31:12] & \quad 20 \quad 19 \quad 18 \quad 17 \quad 16 \quad 15 \quad 14 \quad 13 \quad 12 \quad 11 \quad 10 \quad 9 \quad 8 \quad 7 \quad 6 \quad 0 \\
\end{align*}

<table>
<thead>
<tr>
<th>Select one per row</th>
<th>PCSel Mux:</th>
<th>○ “pc + 4” branch</th>
<th>○ “alu” branch</th>
<th>○ * (don’t care)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASel Mux:</td>
<td>○ “pc” branch</td>
<td>○ Reg[rs1] branch</td>
<td>○ * (don’t care)</td>
<td></td>
</tr>
<tr>
<td>BSEL Mux:</td>
<td>○ “imm” branch</td>
<td>○ Reg[rs2] branch</td>
<td>○ * (don’t care)</td>
<td></td>
</tr>
<tr>
<td>WBSEL Mux:</td>
<td>○ “pc + 4” branch</td>
<td>○ “alu” branch</td>
<td>○ “mem” branch</td>
<td>○ * (don’t care)</td>
</tr>
</tbody>
</table>

Select all that apply

Datapath units: □ Branch Comp □ Imm. Gen □ Load Extend

RegFile: □ Read Reg[rs1] □ Read Reg[rs2] □ Write Reg[rd]
Problem 2: VERILOG LOGIC [15 points, 15 minutes]

Consider the following Verilog module:

```verilog
define my_module(input clk, load, input [2:0] in,
 output reg [2:0] out);
    always @(posedge clk) begin
        if (load) out <= in;
        else begin
            out[0] <= out[0];
            if (out[0]) out[1] <= ~out[1];
            if (out[1]) out[2] <= ~out[2];
            end
    endmodule
```

(a) Draw the circuit diagram for this design. You may use the module inputs (e.g. in[0]),
constants, muxes, inverters, and/or two-input logic gates.

(b) Say we load 3'b011 using our load and in input signals. We then deassert load. What is the
value of out for the first 5 cycles?

<table>
<thead>
<tr>
<th>Cycle</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>011</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Now, consider this similar module:

```verilog
module my_module(
    input clk, load,
    input [2:0] in,
    output reg [2:0] out
);

always @(posedge clk) begin
    if (load) out = in;
    else begin
        out[0] = ~out[0];
        if (out[1]) out[1] = ~out[1];
        if (out[0] & out[1])
            out[2] = ~out[2];
    end
end
endmodule
```

(c) Say we load 0b1011 again using our load and in input signals. What is the value of out for the first 6 cycles?

<table>
<thead>
<tr>
<th>Cycle</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>011</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>