1.) Answer the following questions as True or False

a.) A 4-to-1 multiplexer requires at least 4 select lines: true / false

b.) An 8-to-1 mux and no other logi can be used to implement any combinational logic function of 4 input variables: true / false

c.) A 3-to-8 decoder along with (4) 8-input OR gates can be used to implement any combinational logic involving 3 input variables and 4 output variables: true / false

d.) 3 separate 2-to-1 muxes can be used to build a single 4-to-1 mux: true / false

e.) A 4-to-16 decoder with an enable can be used as a 1-to-16 demultiplexer: true / false

f.) 5 flip-flops are required to implement a state machine with 5 states: true / false

g.) In a state machine with Moore style outputs, a change in the external inputs can independently and immediately cause the outputs to change: true / false

h.) In binary, performing X-Y can be performed by adding X to the 2’s complement of Y: true / false

i.) The characteristic equation of a JK Flip-flop is \( Q^* = JQ' + K'Q \): true / false

j.) The passive inputs (hold state) of an active-hi set and reset SR-Latch are \( S=0, R=0 \): true / false
2.) State Machine Design (Down Counter w/ Restart): Design a synchronous state machine circuit that implements a 2-bit down counter (i.e. counts 11, 10, 01, 00, 11…). The circuit has an external input, R (RESTART), that when ‘1’ should force the counter back to the 11 state no matter what the current state is. As long as R stays ‘1’, the counter should stay in the 11 state. The circuit should also have one output Z. Z=1 when in the 00 state and Z = 0 otherwise.

Let us use 4 states:

- **S3** (initial state on reset)  
  The count should be $11_2 = 3_{10}$
- **S2**  
  The count should be $10_2 = 2_{10}$
- **S1**  
  The count should be $01_2 = 1_{10}$
- **S0**  
  The count should be $00_2 = 0_{10}$

a.) Complete the state diagram below by filling in all necessary transitions and the values of Z.

b.) What is the minimum number of flip-flops required to implement this state machine?

c.) Complete the state transition/output table given below. (Note: We have provided the state assignment already). We have ordered the states in such a way to use gray code ordering, so take care when translating your state diagram to the transition/output table.

<table>
<thead>
<tr>
<th>Current State</th>
<th>R=0</th>
<th>Next State</th>
<th>R=1</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>Q1Q0</td>
<td>State</td>
<td>Q1<em>Q0</em></td>
<td>State</td>
</tr>
<tr>
<td>S0</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
d.) Assume we will implement our circuit using D Flip-Flops. Use the K-Maps below to find minimal expressions for D1, D0, and Z.

\[ D_1 = \]

\[ D_0 = \]

\[ Z = \]

e.) Show how to implement the initial state (power-on/reset state) by connecting the PRE (PRESReset) and CLR inputs of the FF’s appropriately. Assume the signal /RESET is available to you. You do not need to implement the next-state or output-function logic.
f.) Using your design above draw the waveform for the sequence of states that the machine will go through and what the output will be for the given input sequence of X. Remember you are using positive edge-triggered devices.
3.) **Sequential Circuit Analysis:** For the Mealy-style state machine below, find the state diagram that it is implementing. Show all your work, including the intermediate steps you took to arrive at your solution. You do not need to concern yourself with which state we start in on reset. You may use the table provided below:

![State Machine Diagram](image)

Area for equations:

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State / Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
<td><strong>Q1, Q0</strong></td>
</tr>
<tr>
<td>SA</td>
<td>0 0</td>
</tr>
<tr>
<td>SB</td>
<td>0 1</td>
</tr>
<tr>
<td>SC</td>
<td>1 0</td>
</tr>
<tr>
<td>SD</td>
<td>1 1</td>
</tr>
</tbody>
</table>

State Diagram:

![State Diagram](image)
4.) Implement the following function using (2) 2-to-4 decoders.
   a.) Cascade the individual decoders to build a 3-to-8 decoder. You may use an
       **inverter**
   b.) Then use a 3-input NAND to implement the function $G$.

\[
G = \sum_{x,y,z} (2,5,6)
\]
6.) We need to create a circuit to perform the following conditional operation:

\[
\begin{align*}
\text{if}(x \geq 10) \text{ then } \\
\quad z &= x + y; \\
\text{else } \\
\quad z &= x - y;
\end{align*}
\]

or put another way…

\[
z = \begin{cases} 
  x+y, & x \geq 10 \\
  x-y, & \text{otherwise}
\end{cases}
\]

- X, Y, and Z are all unsigned 4-bit numbers
- You can assume that X > Y so that Z will never be negative.
- Don’t worry about overflow when performing X+Y.

To implement this design you will use the following:

- (1) 4-bit adder
- 74LS85 4-bit comparator (see logic diagram on the next page)
- A few other logic gates

a.) Finish the design below. Your inputs should be clearly labeled \(X_3 – X_0, Y_3 – Y_0,\) and your output should be labeled \(Z_3 – Z_0\). You may use temporary variable labels rather than wires to connect your devices.

b.) When you have completed your design, go back annotate your design with the test case of \(X=9, Y=5\) and show the intermediate outputs and the final output Z.
7.) During your summer vacation Internet music company, MP3 Worldwide, hires you for selling MP3 players. Apart from the hourly salary they offer you a performance bonus of $1 for each MP3 player you sell for the week. Assume you can only sell between 0 and 99 MP3 players a week (never more than 99). If you sell 75 or more MP3 players in a week you get an additional bonus of $25. So if you sold 10 MP3 players in a week you would get a $10 bonus. If you sold 90 MP3 players you would get a $115 bonus (90 + 25). Build a circuit to calculate the bonus. The input is an unsigned number, \( X[6:0] \), for how many MP3 players you’ve sold. Your output should be an unsigned number, \( Y[6:0] \).

a.) Using only (2) 4-bit 74LS85 comparators, produce a 1 bit output, \( S \), to detect whether the number of MP3 players you sold is 75 or more.

b.) Using your output, \( S \), from part a, and two 4-bit adders, produce the output \( Y \).