The following DFA $D_1$ recognizes the language $L_1 = \{ w \in \{0, 1\}^* \mid w \text{ starts with } 111 \}$. 

The following DFA $D_2$ recognizes the language $L_2 = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 001 \}$. 

We can easily recognize the union of $L_1$ and $L_2$ by replacing $D_1$'s "sink"/"trap" state with $D_2$.

- $Q_0$: Start state only
- $Q_1$: read in 1 only
- $Q_2$: started with 11
- $Z_1$: last char is 0 and did not start with 11
- $Z_0$: last char is 1 and did not start with 11
- $Z_2$: last 2 chars were 00 and did not start with 11
b) Consider how the decimal value of our input changes as the DFA reads in digits:
   - If we read in a 0, the decimal value doubles (i.e., a left shift occurs.)
   - Likewise, if we read in a 1, the decimal value doubles and increases by one.

Next, we consider how right-appending a 0/1 affects the remainder (modulus 6) of our original number. Any non-negative integer $z$ can be written via division, as $z = 6q + r$ (where $q, r$ are non-negative integers).

Appending a 0 to $z$ gives:
$$z_0 = 2(6q + r) \equiv 2r \mod 6.$$

Appending a 1 to $z$ gives:
$$z_1 = 2(6q + r) + 1 \equiv (2r + 1) \mod 6.$$

The following table outlines the new remainder (modulus 6) in all 12 cases:

<table>
<thead>
<tr>
<th>Old Remainder</th>
<th>New Digit</th>
<th>New Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r = 0$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$r = 1$</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$r = 2$</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$r = 3$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$r = 4$</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$r = 5$</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
We can use the previous table as a transition function for a DFA that recognizes the language:

General state strategy: keep states for each prefix of 0, 11, and 010, and one extra state which states that none of those is possible.

\( q_0 \): Start state only
\( q_1 \): read in 0
\( q_2 \): read in 01
\( q_3 \): read in 010
\( q_4 \): read in 1
\( q_5 \): read in 11
\( q_6 \): Can not be 0, 11, or 010.

\( q_0 \): Start state only
\( q_1 \): Start ends w/o
\( q_2 \): Start w/o, end w/1.
\( q_3 \): Start ends w/1.
\( q_4 \): Start w/1, end w/o.
\( Q_0 \): last char is 0, and doesn't have 110.
\( Q_1 \): last char is 1, but last 2, if they exist, are not 01.
\( Q_2 \): last 2 chars are 11.
\( Q_3 \): contains 110.

For strings that do not yet have "110", we just need to keep track how close we are to forming it, using the last couple characters. Namely, we just need to know which prefix of 110 is a suffix of the string read in. Updates based on the next character are made accordingly.