CMSC 311
Solutions to Midterm II

Problem 1.

```assembly
add $s0, $zero, $zero  # $s0 = 0; counts number of 28's
lw  $s2, 240($zero)   # $s2 = length of array
beq $s2, $zero, done  # if array length = 0 then done
sll $s2, $s2, 2      # multiply array length by 4 to obtain number of bytes
lw  $s1, 236($zero)   # $s1 = starting address of array
add $s3, $s0, $s1    # $s3 is address after last number in array
addi $s4, $zero, 28   # $s4 = 28 is number being searched for

loop: lw  $t1, 0($s0)   # load next number in array
       bne $t1, $s4, skip  # skip if number ≠ 28
       addi $s0, $s0, 1    # increment number of 28's

skip:  addi $s1, $s1, 4   # address of next number in array
       bne $s1, $s4, loop  # loop if next address still in bounds

done:  sw  $s0, 244($zero)  # store number of 28's
```

NOTE: Instead of
```assembly
bne  $t1, $s4, skip  # skip if number ≠ 28
addi $s0, $s0, 1    # increment number of 28's
```

we could use the **pseudo-instruction**
```assembly
seq  $t2, $t1, $s4    # $t2 is set to 1 if number is 28; 0 otherwise
add  $s0, $s0, $t2   # increment number of 28's if number equals 28
```

The **seq** pseudo-instruction substitutes two MIPS instruction for the one bne instruction, but avoids a potential branch.

Problem 2. The routine splits $s2$ into left and right halves. A pair of bits are peeled off the right ends of the two halves, one from each. The pair of bits are pasted into the result $s1$. In preparation for the next pair of bits, the two halves are each shifted right one bit position and $s1$ is rotated right two bit positions.

```assembly
andi  $s1, $zero, $zero  # initialize $s1 to 0
andi  $t1, $s2, 2¹⁶ - 1  # $t1 is right half of $s2
srl  $t0, $s2, 15       # $t0 is left half of $s2 shifted left 1 bit
addi $t5, $zero, 16    # counter for number of passes

loop: andi $t2, $t1, 2   # get next bit of left side (from right to left)
or  $s1, $s1, $t2       # put bit into result
andi $t2, $t0, 1       # get next bit of right side (from right to left)
or  $s1, $s1, $t2       # put bit into result
srl  $t1, $t1, 1       # shift down bits of left side
srl  $t0, $t0, 1       # shift down bits of right side
ror  $s1, $s1, 2       # prepare for next two bits; **pseudo-instruction**
addi $t5, $t5, -1     # decrement counter
bne  $t5, $zero, loop
```
Problem 3. Form the three values
\[
\begin{array}{c}
\overline{x_0} \oplus x_1 \\
x_1 \oplus x_2 \\
\overline{x_2} \oplus x_3
\end{array}
\]

Use them as inputs to a 3 \times 1 multiplexer selected by \(i_1i_0\).
\[
a \oplus b = a \overline{b} \lor \overline{a} \overline{b}
\]

So \(a \oplus b\) needs two AND gates and a NOR gate.

A 3 \times 1 multiplexer needs three AND gates and an OR gate.

Problem 4. We need two bits to control the inputs to the shift register. We can use the leftmost bit, call it \(L\), in the shift register as one control bit: The output \(L\) will be 0 until the leftmost 1 shifts into the leftmost position at which point it will be 1 for exactly one clock cycle. The final right shift will make \(L\) 0 again (forever).

A T flip-flop with input from \(L\) will stay 0 until one clock cycle after \(L\) flips to 1, and then stay 1 (forever).

The following table gives the necessary inputs \(i_1i_0\) for the shift register given the outputs of \(L\) and of the T flip-flop, which we call \(Q\).

<table>
<thead>
<tr>
<th>(L)</th>
<th>(Q)</th>
<th>(i_1)</th>
<th>(i_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Using minterms, we get
\[
i_1 = \overline{L} \overline{Q} \quad i_0 = L \overline{Q}
\]
Problem 5.

(a)

![Diagram showing a state transition graph with states labeled 00, 01, 10, 11, and transitions labeled with 00, 01, 01, 11, and 11.]

(b)

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_2$</td>
<td>$v_1$</td>
<td>$v_0$</td>
<td>$x_1$</td>
<td>$x_0$</td>
<td>$v'_2$</td>
<td>$v'_1$</td>
<td>$v'_0$</td>
<td>$T_2$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(c)

$$T_2 = x_0 v_0$$
$$D_1 = \overline{v_2} v_1 v_0 x_1 x_0 \lor v_2 v_1 v_0 x_1 x_0 \lor v_2 v_1 v_0 x_1 x_0$$
$$D_2 = \overline{v_2} v_1 v_0 x_1 x_0 \lor v_2 v_1 x_0 x_0 \lor v_2 v_1 x_0 x_0$$
$$z_1 = v_2$$
$$z_0 = \overline{v_2} \quad \text{(or} \quad z_0 = v_0 v_1)$$