COMP370 Review

Signed Magnitude

- Negative number are the same as positive with the sign bit set
  
  Three bit example
  
  |   000  001  010  011  100  101  110  111 |
  |-----|-----|-----|-----|-----|-----|-----|-----|
  |  0  |  1  |  2  |  3  | -0  | -1  | -2  | -3  |

- There are two zeroes, positive and negative.

Negative Integers

- Almost all systems for storing negative binary numbers set the left most bit (MSB) to indicate the sign of a number.

  Common formats:
  - Signed Magnitude
  - Ones Complement
  - Twos Complement

Ones Complement

- Negative number are the logical inverse of positive numbers

  Three bit example
  
  |   000  001  010  011  100  101  110  111 |
  |-----|-----|-----|-----|-----|-----|-----|-----|
  |  0  |  1  |  2  |  3  | -3  | -2  | -1  | -0  |

- Mathematically positive and negative zero are the same, but they are different bit patterns.
Twos Complement

• Negative number are the logical inverse of positive numbers plus 1.

Three bit example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

Normal binary arithmetic works for positive and negative numbers.

Two’s Complement Representation

• If number is positive or zero,
  – normal binary representation, zero in upper bit
• If number is negative,
  – start with positive number
  – flip every bit (i.e., take the one’s complement)
  – then add one

\[
\begin{align*}
00101 & \quad (5) \\
11010 & \quad (1's \ comp) \\
+ 1 & \\
11011 & \quad (-5)
\end{align*}
\]

\[
\begin{align*}
010101 & \quad (10) \\
10101 & \quad (1's \ comp) \\
+ 1 & \\
10110 & \quad (-10)
\end{align*}
\]

Single-Precision Floating-point Numbers

• `float` variables in C++ or Java

Signed Magnitude

• For positive numbers, the sign bit is zero
• For negative numbers, the sign bit is one and everything else is the same

Boolean Expression Forms

• You can express a Boolean equation in many ways.
• The Sum of Products form ORs together sub-expressions that are ANDed
  \[ F = ABC + A'B'C + AB' \]
• The Product of Sums form ANDs together sub-expressions that are ORed
  \[ F = (A+B+C) * (B' + C') \]
Karnaugh Example

Copy the output values into the Karnaugh map.

\[
\begin{array}{c|ccc|c}
A & B & C & \text{F} \\
\hline
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc|c}
 & B'C' & B'C & BC & BC' \\
\hline
A' & 1 & 1 & 0 & 0 \\
A & 0 & 1 & 0 & 0 \\
\end{array}
\]

\[F = A'B' + B'C\]

Simplified Truth Table

\[
\begin{array}{c|ccc|c}
A & B & C & \text{F} \\
\hline
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc|c}
 & B'C' & B'C & BC & BC' \\
\hline
A' & 0 & 1 & 1 & 1 \\
A & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[F = B + A'C\]

Truth Table with “Don’t Care”

Note that one “d” was included to make a big group while the other was ignored.

\[
\begin{array}{c|ccc|c}
A & B & C & \text{F} \\
\hline
0 & 0 & 0 & d \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & d \\
1 & 1 & 1 & d \\
\end{array}
\]

\[
\begin{array}{cccc|c}
 & B'C' & B'C & BC & BC' \\
\hline
A' & d & 1 & 0 & 0 \\
A & 1 & 1 & 0 & d \\
\end{array}
\]

\[F = B'\]

Unnecessary Groups

- Avoid groups that do not contain any squares that are not already in a group.
### Karnaugh Map Process

- Fill the Karnaugh map with 0’s, 1’s and d’s
- Group the 1’s into as big as group as possible. Include d’s only if necessary.
- Do not include 0’s in the groups.
- Write the Boolean expression for each group with the groups ORed together.

### Segment “b” of Display

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>b</th>
<th>C’D’</th>
<th>C’D</th>
<th>CD</th>
<th>CD’</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<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>
<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>
<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>
<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>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \text{seg b} = B’ + C’D’ + CD \]

### Logic Gates & Symbols

- (a) Inverter
- (b) AND gate
- (c) OR gate
- (d) NAND gate
- (e) NOR gate

Note that gates can have more than 2 inputs.

### Creating Logic Gate Circuits

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
SR Flip-Flops

- An SR flip-flop can be constructed from two NOR gates

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Keep state</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q = 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Q = 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Restricted combination</td>
</tr>
</tbody>
</table>

SR Action
0 0 Keep state
0 1 Q = 0
1 0 Q = 1
1 1 Restricted combination

D Flip-Flop

- A D flip-flop has only one data input, D, plus enable, C

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>Q_{next}</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>Q_{prev}</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Set</td>
</tr>
</tbody>
</table>

D Flip-Flop

Memory

- The output of an SR flip-flop depends on the previous output value.

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Old Q</th>
<th>New Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Register

- A register stores a multi-bit value.
  - We use a collection of D-latches, all controlled by a common enable.
  - When enable=1, n-bit value D is written to register.