Today’s Lecture

- Homework #2 Due
  - Written today
  - Programming extended to Thursday midnight.
- Homework #3 Due Thursday October 3
- Project Groups of Three, will tolerate some 2-member
- Building the building blocks...

Outline
- Review
- Digital building blocks
- An Arithmetic Logic Unit (ALU)

Reading
- Appendix B, Chapter 4
Review: Digital Design

- Logic Design, Switching Circuits, Digital Logic
- Recall: Everything is built from transistors
- A transistor is a switch
- It is either on or off
- On or off can represent True or False

Given a bunch of bits (0 or 1)...
- Is this instruction a lw or a beq?
- What register do I read?
- How do I add two numbers?
- Need a method to reason about complex expressions

Review: Boolean Functions

- Boolean functions have arguments that take two values (\{T,F\} or \{0,1\}) and they return a single or a set of (\{T,F\} or \{0,1\}) value(s).
- Boolean functions can always be represented by a table called a “Truth Table”
- Example: \( F: \{0,1\}^3 \to \{0,1\}^2 \)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>( f_1 )</th>
<th>( f_2 )</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>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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Review: Boolean Functions and Expressions

\[ F(A, B, C) = (A \cdot B) + (\neg A \cdot C) \]

<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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>0</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Review: Boolean Gates

- **Gates** are electronics devices that implement simple Boolean functions.

**Examples**

- AND gate
- OR gate
- NOT gate
- XOR gate
- NAND gate
- NOR gate
- XNOR gate
### Boolean Functions, Gates and Circuits

- **Circuits** are made from a network of gates. (function compositions).

\[
F = \sim a \cdot b + \sim b \cdot a
\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>XOR (a, b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Digital Design Examples

**Input:** 2 bits representing an unsigned number (n)
**Output:** \( n^2 \) as unsigned binary number

**Input:** 2 bits representing an unsigned number (n)
**Output:** \( 3-n \) as unsigned binary number
Design Example

- Consider machine with 4 registers
- Given 2-bit input (register specifier, \( I_1, I_0 \))
- Want one of 4 output bits (\( O_3-O_0 \)) to be 1
  \( \Leftrightarrow \) E.g., allows a single register to be accessed
- What is the circuit for this?

Circuit Example: Decoder

```
<table>
<thead>
<tr>
<th>( I_1 )</th>
<th>( I_0 )</th>
<th>( Q_0 )</th>
<th>( Q_1 )</th>
<th>( Q_2 )</th>
<th>( Q_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 0 0 0 0</td>
</tr>
<tr>
<td>0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 1 0 1 0 0</td>
</tr>
<tr>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 0 0 0 1 0</td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 1 0 0 0 1</td>
</tr>
</tbody>
</table>
```

Circuit Example: 2x1 MUX

Multiplexor (MUX) selects from one of many inputs

\[ MUX(A, B, S) = (A \cdot S) + (B \cdot \sim S) \]

Example 4x1 MUX

\[ S_0 = 0, S_1 = 1 \]

\[ a \]
\[ b \]
\[ c \]
\[ d \]
\[ y \]
Arithmetic and Logical Operations in ISA

- What operations are there?
- How do we implement them?

Consider a 1-bit Adder

Truth Table for 1-bit Addition

```
+ 00101100
10011001
```

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>C_in</th>
<th>Sum</th>
<th>C_out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

What is the circuit for Sum and for Cout?
A 1-bit Full Adder

\[
\begin{array}{c|c|c|c}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

Example: 4-bit adder
Subtraction

- How do we perform integer subtraction?
- What is the HW?
ALU Slice

<table>
<thead>
<tr>
<th>( B_{\text{inv}} )</th>
<th>( F )</th>
<th>( Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( a + b )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( a - b )</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>( \text{NOT} \ b )</td>
</tr>
<tr>
<td>-2</td>
<td>1</td>
<td>( a \text{ OR} \ b )</td>
</tr>
<tr>
<td>-3</td>
<td>1</td>
<td>( a \text{ AND} \ b )</td>
</tr>
</tbody>
</table>

Example: Adder/Subtractor

Add/Sub = 0 => Addition
Add/Sub = 1 => Subtraction
Note: Flips A & B (does \( A_{\text{invert}} \))
**Overflow**

**Example 1:**

\[
\begin{array}{c}
0100000_2 \\
0110101_2 \\
+0101010_2 \\
\hline
1011111_2
\end{array}
\]

\[
(= 53_{10}) \\
(= 42_{10}) \\
(= -33_{10})
\]

**Example 2:**

\[
\begin{array}{c}
1000000_2 \\
1010101_2 \\
+1001010_2 \\
\hline
00110111_2
\end{array}
\]

\[
(= -43_{10}) \\
(= -54_{10}) \\
(= 31_{10})
\]

**Example 3:**

\[
\begin{array}{c}
1100000_2 \\
0110101_2 \\
+1101010_2 \\
\hline
0001111_2
\end{array}
\]

\[
(= 53_{10}) \\
(= -22_{10}) \\
(= 31_{10})
\]

**Example 4:**

\[
\begin{array}{c}
0000000_2 \\
0010101_2 \\
+0101010_2 \\
\hline
0111111_2
\end{array}
\]

\[
(= 21_{10}) \\
(= 42_{10}) \\
(= 63_{10})
\]
The new ALU Slice

<table>
<thead>
<tr>
<th>A</th>
<th>F</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>a + b</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>a - b</td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td>NOT b</td>
</tr>
<tr>
<td>-2</td>
<td></td>
<td>a OR b</td>
</tr>
<tr>
<td>-3</td>
<td></td>
<td>a AND b</td>
</tr>
</tbody>
</table>

The ALU
Abstraction: The ALU

- General structure
- Two operand inputs
- Control inputs

The Shift Operation

- Consider an 8-bit machine
- How do I implement the shift operation?
Summary

• Given Boolean function, generate a circuit that “realizes” the function.
• Constructed circuits that can add and subtract.
• The ALU: a circuit that can add, subtract, detect overflow, compare, and do bit-wise operations (AND, OR, NOT)
• Shifter

Administrative
• Homework #2 Due Today/Thursday
• Homework #3 Due Thursday October 3
• Read Appendix B, Chapter 4
Next up: Storage Elements: Registers, Latches, Buses