Basics of Logic Design
Arithmetic Logic Unit (ALU)

CPS 104
Lecture 9

Today’s Lecture

• Homework #3 Due Tuesday
• Midterm in class next Thursday, closed book/notes
• 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: \( F: \{0,1\}^3 \rightarrow \{0,1\}^2 \)

\[
\begin{array}{ccc|cc}
 a & b & c & f_1 & f_2 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 \\
\end{array}
\]
Review: Boolean Functions and Expressions

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<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 (a, b)**
- **OR (a, b)**
- **NAND (a, b)**
- **NOR (a, b)**
- **XOR (a, b)**
- **XNOR (a, b)**
- **NOT (a)**
**Boolean Functions, Gates and Circuits**

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

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</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, I₁, I₀)
- Want one of 4 output bits (O₃-O₀) to be 1
  - E.g., allows a single register to be accessed
- What is the circuit for this?

---

**Circuit Example: Decoder**

<table>
<thead>
<tr>
<th>I₁ I₀</th>
<th>Q₀ Q₁ Q₂ Q₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

© Alvin R. Lebeck

CPS 104
Circuit Example: 2x1 MUX

Multiplexor (MUX) selects from one of many inputs

MUX(A, B, S) = (A * S) + (B * ~S)

Example 4x1 MUX
Arithmetic and Logical Operations in ISA

• What operations are there?
• How do we implement them?
  ➢ Consider a 1-bit Adder

Example: 4-bit adder
ALU Slice (Almost)

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

Subtraction

- How do we perform integer subtraction?
- What is the HW?
Example: Adder/Subtractor

\[ \text{Add/Sub} = 0 \implies \text{Addition} \]
\[ \text{Add/Sub} = 1 \implies \text{Subtraction} \]

Note: Flips A & B (does \( A_{\text{invert}} \))

\[ \begin{array}{cccccc}
    & a0 & a1 & a2 & a3 & b0 & b1 & b2 & b3 \\
    \text{Add/Sub} & \downarrow & \downarrow & \downarrow & \downarrow & \text{Full Adder} & \text{Full Adder} & \text{Full Adder} & \text{Full Adder} \\
    \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
    \text{C}_{\text{in}} & \text{C}_{\text{out}} & \text{S}_0 & \text{S}_1 & \text{S}_2 & \text{S}_3 \\
\end{array} \]

Overflow

Example 1:
\[
\begin{array}{l}
0100000 \\
0110101_2 \quad (= \quad 53_{10}) \\
+0101010_2 \quad (= \quad 42_{10}) \\
\hline
1011111_2 \quad (= -33_{10})
\end{array}
\]

Example 2:
\[
\begin{array}{l}
1000000 \\
1010101_2 \quad (= -43_{10}) \\
+1001010_2 \quad (= -54_{10}) \\
\hline
0011111_2 \quad (= \quad 31_{10})
\end{array}
\]

Example 3:
\[
\begin{array}{l}
1100000 \\
0110101_2 \quad (= \quad 53_{10}) \\
+1101010_2 \quad (= -22_{10}) \\
\hline
0011111_2 \quad (= \quad 31_{10})
\end{array}
\]

Example 4:
\[
\begin{array}{l}
0000000 \\
0010101_2 \quad (= \quad 21_{10}) \\
+0101010_2 \quad (= \quad 42_{10}) \\
\hline
0111111_2 \quad (= \quad 63_{10})
\end{array}
\]
**Add/Subtract With Overflow detection**

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>1</td>
<td>NOT b</td>
</tr>
<tr>
<td>-2</td>
<td>2</td>
<td>a OR b</td>
</tr>
<tr>
<td>-3</td>
<td>3</td>
<td>a AND b</td>
</tr>
</tbody>
</table>
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 thus far

- 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

Next up: Storage Elements: Registers, Latches, Buses

Memory Elements

- All the circuit we looked at so far are combinational circuits: the output is a Boolean function of the inputs.
- We need circuits that can remember values. (registers)
- The output of the circuit is a function of the input AND a stored value (state).
- Circuits with memory are called sequential circuits.
Set-Reset Latch

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Set-Reset Latch (Continued)

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>
Set-Reset Latch (Continued)

Data Latch (D Latch)

Does not affect Output
On C↑ D is transferred to the first D latch and the second is stable.

On C↓ the output of the first stage is transferred to the second (output), and the first stage is stable.
Tri-State Driver

- The Tri-State driver is like a (one directional) switch:
  - When the Enable is on (E=1) it transfers the input to the output.
  - When the Enable is off (E=0) it disconnects the output.

Z :- High Impedance

Bus Connections

- The Bus: Many to many connections.
- Mutual exclusion: At most one Enable is on!
Register Cells on a bus

One can “source” and “sink” from any cell on the bus by activating the right controls (WE and RE).

3-Port Register Cell

- Stores one bit of a register
- Can Read onto Bus-A & Bus-B and Write from Bus-C Simultaneously
3-Port Register File

Address Decode Circuit

Register address: 01
Summary

- Given Boolean function, generate a circuit that “realize” 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
- Memory Elements: SR-Latch, D Latch, D Flip-Flop
- Tri-state drivers & Bus Communication
- Register Files
- Control Signals modify what circuit does with inputs
  - ALU, Shift, Register Read/Write