Today’s Lecture

• Midterm Wednesday
• Dennis Monday office hours now 5-7pm
• Building the building blocks…

Outline
• Review
• Digital building blocks
• An Arithmetic Logic Unit (ALU)
• Storage Elements
• Register File

Reading
  Chapter 4.2
  start reading 4.1 & 4.3
Review: Boolean Functions and Expressions

F(A, B, C) = (A * B) + (~A * C)

<table>
<thead>
<tr>
<th></th>
<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>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<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>
<td>1</td>
</tr>
<tr>
<td>1</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</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 Gates

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

**Examples**

- AND(a, b)
- OR(a, b)
- NOT(a)
- XOR(a, b)
- NAND(a, b)
- NOR(a, b)
- XNOR(a, b)
Review: Boolean Functions, Gates and Circuits

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

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

\[
\begin{array}{c|c|c}
 a & b & \text{XOR} (a, b) \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

Circuit Example: Decoder

\[
\begin{array}{c|c|c|c|c|c}
 I_1 & I_0 & Q_0 & Q_1 & Q_2 & Q_3 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 \\
\end{array}
\]
Circuit Example: 2x1 MUX

Multiplexor (MUX) selects 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?
• Arithmetic Logic Unit (ALU)
  ➢ Hardware that performs operations
  ➢ Only one operation at a time
• How do we implement the operations?
  ➢ Consider AND, OR, NOT, and ADD
  ➢ Input is two bits, output...

Truth Table for 1-bit Addition

<table>
<thead>
<tr>
<th>01101100</th>
<th>01101101</th>
<th>+00101100</th>
<th>10011001</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b C_{in} Sum C_{out}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</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>0</td>
<td>1</td>
<td>1</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>

Example: 4-bit adder

This is called a Ripple Carry Adder:
There are faster, more complex adder circuits.
Subtraction

- How do we perform integer subtraction?
- What is the hardware (circuit)?
ALU Slice

A | F | Q
---|---|---
0 | 0 | \(a + b\)
1 | 0 | \(a \cdot b\)
- | 1 | \(\text{NOT } b\)
- | 2 | \(a \text{ OR } b\)
- | 3 | \(a \text{ AND } b\)

\(~\text{Add/Sub}\)

Adder

Overflow

Example 1:

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

\(= 53_{10}\)

\(= 42_{10}\)

\(= -33_{10}\)

Example 2:

\[
\begin{array}{c}
\text{1000000} \\
\text{+1010101} \\
\text{0011111}_2
\end{array}
\]

\(= -43_{10}\)

\(= -54_{10}\)

\(= 31_{10}\)

Example 3:

\[
\begin{array}{c}
\text{1100000} \\
\text{+1101010} \\
\text{0011111}_2
\end{array}
\]

\(= 53_{10}\)

\(= -22_{10}\)

\(= 31_{10}\)

Example 4:

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

\(= 21_{10}\)

\(= 42_{10}\)

\(= 63_{10}\)
Overflow Detection for 4-bit adder

The ALU
Abstraction: The ALU

- General structure
- Two operand inputs
- Control inputs

- We can build circuits for
  - Multiplication
  - Division
  - They are more complex

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 circuits 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 function of 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>Don't set both S &amp; R to 1</td>
</tr>
</tbody>
</table>

Don't set both S & R to 1
Set-Reset Latch (Continued)

Data Latch (D Latch)

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>Q</td>
</tr>
</tbody>
</table>

Does not affect Output
D Flip-Flop

- On $C\uparrow$, $D$ is transferred to the first D latch and the second is stable.
- On $C\downarrow$, the output of the first stage is transferred to the second (output), and the first stage is stable.
- Output changes only on the edge of a clock

Register File

- Register File = the set of locations for register values
  - E.g., 32 32-bit registers
- How do I build a Register File using D Flip-Flops?
- What other components do I need?
Register File

• Circuit to determine which of 32 registers?
• Circuit to get just the data from one of 32 registers?

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.

\[
\begin{array}{ccc}
D & E & Q \\
0 & 1 & 0 \\
1 & 1 & 1 \\
- & 0 & Z \\
\end{array}
\]

\[Z :- \text{High Impedance}\]
Bus Connections

- The Bus: Many to many connections.
- Mutual exclusion: At most one Enable is on!
- Control must ensure this!
- Note: Bus sometimes used to denote multiple parallel wires

Register Cells on a bus

One can “source” and “sink” from any cell on the bus by activating the right controls, IE--input enable, and OE--output enable.
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

- Store 2-bit data
- Each bit has an input and output path for Bus-A, Bus-B, and Bus-C

© Alvin R. Lebeck
Address Decode Circuit

Register address: 01

Register File (Four 4-bit Registers)
**Digital Logic Summary**

- Given Boolean function, generate a circuit to “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 vs. MUX**
- **Register Files**
- **Control Signals** modify what circuit does with **inputs**
  - ALU, Shift, Register Read/Write

---

**Summary**

- **The ALU & Shifter**
- **Storage elements**
  - S-R latch, D-Latch, D Flip-Flop
- **Register File**
- **Next time**
  - Finite state machines
  - HCL