ECE 2300
Digital Logic & Computer Organization
Spring 2017

More Binary Arithmetic
ALU
Announcements

• Lab 4 prelab (B) due tomorrow

• Lab 5 to be released tonight

• Final exam schedule and location announced
  – Monday May 15, 9am, HLS B14
Example: Fixed Size 2’C Addition

- White stone = 0
- Black stone = 1

- With each row representing a 4-bit 2’s complement, adding which two rows would cause an overflow?
  - Row 1 + Row 4 : (-3) + (-6)
  - Row 3 + Row 4 : (-5) + (-6)
Review: Ripple Carry Adder (RCA)
Adder Implementations

- Many different adder implementations exist, which differ in speed and circuit complexity

- Ripple carry adder is simple but slow
  - Each 1-bit full adder must wait for the carry bit to be calculated from the previous full adder

- Common techniques to speed up carry propagation
  - Carry lookahead
  - Carry select
  - etc.
Carry Lookahead Adder (CLA)

- Calculation of carry out of MSB is slow for large RCAs
  - Carry has to propagate from LSB to the MSB, which forms the longest path (critical path)

- CLA adder calculates groups of carries in parallel
Carry Generation and Propagation

- $C_{out} = 1$ from a bit position happens for 2 reasons
  - Carry is *generated* from this bit position
  - *Carry in* to this position is *propagated* to the next
Carry Generation and Propagation

• Carry into $i^{th}$ position: $c_i$
  – Also the carry out from $(i-1)^{th}$ position

• Carry generation function for the $i^{th}$ position
  \[ G_i = a_i \cdot b_i \]

• Carry propagation function for the $i^{th}$ position
  \[ P_i = a_i + b_i \]

• Carry out of the $i^{th}$ position
  \[ c_{i+1} = G_i + P_i \cdot c_i \]
Carry Out Equations for 4-bit Adder

\[ c_1 = G_0 + P_0 \cdot c_0 \]

\[ c_2 = G_1 + P_1 \cdot c_1 \]
\[ = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot c_0 \]

\[ c_3 = G_2 + P_2 \cdot c_2 \]
\[ = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot c_0 \]

\[ c_4 = G_3 + P_3 \cdot c_3 \]
\[ = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0 \]

\[ G_i = a_i \cdot b_i \]
\[ P_i = a_i + b_i \]
Carry Out Logic for 4-bit Adder

\[ c_4 = G_3 + P_3 \cdot c_3 \]
\[ = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0 \]
\[ = G_3 + P_3 \cdot (G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0)) + (P_3 \cdot P_2 \cdot P_1 \cdot P_0) \cdot c_0 \]

- \( G_{3:0} \) – Carry generation from bits 3-0
- \( P_{3:0} \) – Carry propagation through bits 3-0

\[ G_i = a_i \cdot b_i \]
\[ P_i = a_i + b_i \]
• Carry out logic gets more complicated beyond 4 bits

• CLAs are often implemented as 4-bit modules and instantiated in a hierarchical way to realize wider adders
32-bit CLA with 4-bit RCAs

P's and G's for all CLA blocks are generated in parallel
Longest Delay (Critical Path)

Bits 31-28

Bits 7-4

Bits 3-0
Longest Delay (Critical Path)

Bits 31-28

Bits 7-4

Bits 3-0
Multiplication

- Form each partial product by multiplying a single digit of the multiplier by the multiplicand
- Add shifted partial products to get result

\[
\begin{array}{c}
\text{multiplicand (}= 5) \\
\text{multiplier (}= 7) \\
\times
\end{array}
\]

\[
\begin{array}{c}
0101 \\
0101 \\
0101 \\
0000
\end{array}
\]

result (\(= 35\))

0100011
**4 × 4 Unsigned Array Multiplier**

\[
\begin{array}{cccc}
A_3 & A_2 & A_1 & A_0 \\
\times & B_3 & B_2 & B_1 & B_0 \\
\hline
A_3 B_0 & A_2 B_0 & A_1 B_0 & A_0 B_0 \\
A_3 B_1 & A_2 B_1 & A_1 B_1 & A_0 B_1 \\
A_3 B_2 & A_2 B_2 & A_1 B_2 & A_0 B_2 \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
\]
Our Microprocessor is Built to Compute

ALU is the computing core of a processor
What is an ALU?

- **Arithmetic Logic Unit (ALU):** Combinational logic circuit that combines a variety of operations into a single unit

- **Common operations may include**
  - Addition and subtraction
  - Logical (OR, AND)
  - Shift
  - Comparisons
A Simple 8-Bit ALU

[Diagram of an 8-bit ALU with inputs and outputs labeled: SI, CI, OP, B, A, Y, SO, CO, O, Z, and 3]
ALU Operations, Inputs & Outputs

• **Operations**
  – Addition and Subtraction
  – Bitwise AND and OR
  – Left Shift and Right Shift

• **Inputs**
  – A, B, Carry In (CI)
  – Shift In (SI)
  – Code indicating operation to be performed (OP)

• **Outputs**
  – Y, Carry Out (CO)
  – Shift Out (SO)
  – Flags regarding the result of the operation
    • Overflow flag (O), Zero flag (Z)
Shift Operations

- **Left Shift:** shifts each bit left by 1 position
  \[
  \begin{array}{cccccccc}
  A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0 \\
  \hline
  SI \\
  \hline
  SO & A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0
  \end{array}
  \]
  - MSB shifted into SO, SI shifted into LSB

- **Right Shift:** shifts each bit right by 1 position
  \[
  \begin{array}{cccccccc}
  A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0 \\
  \hline
  SI \\
  \hline
  SO & A7 & A6 & A5 & A4 & A3 & A2 & A1 & A0
  \end{array}
  \]
  - LSB shifted into SO, SI shifted into MSB
ALU Block Diagram

Control Logic

Adder

Logical

Shifter

NOR
### Example: ALU Operation Encodings

<table>
<thead>
<tr>
<th>OP Name</th>
<th>OP</th>
<th>BSEL</th>
<th>CI</th>
<th>LOP</th>
<th>SOP</th>
<th>OSEL</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>000</td>
<td>00</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>00</td>
<td>Y = A + B + CI</td>
</tr>
<tr>
<td>SUB</td>
<td>001</td>
<td>01</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>00</td>
<td>Y = A + B’ + 1</td>
</tr>
<tr>
<td>AND</td>
<td>011</td>
<td>00</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>01</td>
<td>Y = A AND B</td>
</tr>
<tr>
<td>OR</td>
<td>100</td>
<td>00</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>01</td>
<td>Y = A OR B</td>
</tr>
<tr>
<td>SHL</td>
<td>101</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>10</td>
<td>Y = A[6..0],SI</td>
</tr>
<tr>
<td>SHR</td>
<td>110</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>10</td>
<td>Y = SI,A[7..1]</td>
</tr>
<tr>
<td>PASS A</td>
<td>111</td>
<td>10</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>00</td>
<td>Y = A</td>
</tr>
</tbody>
</table>

**Diagram:**
- **Adder**
  - Inputs: A, B, CI
  - Outputs: Y, CO
- **Logical**
  - Inputs: A, B, LOP
  - Outputs: Y, CO
- **Shifter**
  - Inputs: A, SI, SOP
  - Outputs: Y, SO
- **Control Logic**
  - Inputs: OP, BSEL, LOP, SOP, OSEL
  - Outputs: Y

---

Lecture 13: 23
Comparison Operations (Byproduct of SUB)

- To compare A and B, perform $A - B$
  - If the result is 0, then $A = B$
  - Z flag set to 1 whenever ALU result is 0
  - Can check for $A \geq B$ and $A < B$ by observing the MSB of the result of $A - B$
Before Next Class

- H&H 7.1-7.3.1

Next Time

Memories