Decimal v. binary addition

\[
\begin{align*}
110 & \quad 1111 \\
43582 & \quad 01011 \\
+ 22573 & \quad + 00101 \\
\hline 
66155 & \quad 10000
\end{align*}
\]
Ripple carry adder

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>cin</th>
<th>cout</th>
<th>s</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</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>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Full adder:
\[ \text{Cout} = ab + bc_{\text{in}} + a_{\text{cin}} \]
\[ S = a \oplus b \oplus c_{\text{in}} \]

Half adder:
\[ c = ab \]
\[ S = a \oplus b \]
Subtraction w. twos complement representation

Can be accomplished with a **twos-complementor** and an **adder**
In class exercise: designing an adder-subtractor

Two high-level structures:
Adder/subtractor for #'s in 2's complement form

\[ \begin{array}{c|c|c}
   B_0 & S & X_0 \\
   \hline
   0 & 0 & 0 \\
   0 & 1 & 1 \\
   1 & 0 & 1 \\
   1 & 1 & 0 \\
\end{array} \]

\[ a_3 a_2 a_1 a_0 \]
\[ \overline{b_3} b_2 b_1 b_0 \]
\[ \overline{b} \]
\[ + 0 0 0 1 \]
Handling overflow

![Diagram of twos-complement representation of numbers with overflow examples.](image)
Handling overflow

<table>
<thead>
<tr>
<th></th>
<th>c4</th>
<th>c3</th>
<th>c2</th>
<th>c1</th>
<th>c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a3</td>
<td>a2</td>
<td>a1</td>
<td>a0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b3</td>
<td>b2</td>
<td>b1</td>
<td>b0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>s2</td>
<td>s1</td>
<td>s0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Table of full adder](image)

Overflow: $c_3 \oplus c_4$

Adding two 2's complements:

<table>
<thead>
<tr>
<th></th>
<th>a3</th>
<th>b3</th>
<th>c3</th>
<th>c4</th>
<th>s3</th>
<th>Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</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>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Adding two reg.:

Overflow: $c_3 \oplus c_4$
Overflow computation in adder/subtractor

For 2’s complement, overflow if 2 most significant carries differ
Ripple carry adder delay analysis

- Assume unit delay for all gates
  - Use 2-input gates
  - \( S = A \oplus B \oplus C_{in} \)
  - \( S \) ready \( 2 \) units after \( A, B \) and \( C_{in} \) ready
  - \( C_{out} = A \overline{B} + A\overline{C}_{in} + B\overline{C}_{in} \)
  - \( C_{out} \) ready \( 3 \) units after \( A, B \) and \( C_{in} \) ready

Diagram of ripple carry adder with inputs and outputs labeled.
Carry lookahead adder (CLA)

- Goal: produce an adder of less circuit depth

- Start by rewriting the carry function

\[ c_{i+1} = a_i b_i + a_i c_i + b_i c_i \]
\[ c_{i+1} = a_i b_i + c_i (a_i + b_i) \]
\[ C_i + 1 = g_i + C_i (p_i) \]

- carry generate \( g_i = a_i b_i \)
- carry propagate \( p_i = a_i + b_i \)
Carry lookahead adder (CLA) (2)

- Can recursively define carries in terms of propagate and generate signals
  \[ c_1 = g_0 + \overline{c_0}p_0 \]
  \[ c_2 = g_1 + c_1p_1 \]
  \[ = g_1 + (g_0 + c_0p_0)p_1 \]
  \[ c_3 = g_2 + c_2p_2 \]
  \[ = g_2 + (g_1 + g_0p_1 + c_0p_0p_1)p_2 \]
  \[ = g_2 + g_1p_2 + g_0p_1p_2 + c_0p_0p_1p_2 \]

- \( g_i = a_i \cdot b_i \)
- \( \overline{p_i} = a_i + b_i \)

- ith carry has i+1 product terms, the largest of which has i+1 literals
- If AND, OR gates can take unbounded inputs: total circuit depth is 2 (SoP form)
- If gates take 2 inputs, total circuit depth is \( 1 + \log_2 k \) for k-bit addition
Carry lookahead adder (CLA) (3)

\[ c_0 = 0 \quad \quad s_0 = a_0 \oplus b_0 \oplus c_0 \]
\[ c_1 = g_0 + c_0 p_0 \quad s_1 = a_1 \oplus b_1 \oplus c_1 \]
\[ c_2 = g_1 + g_0 p_1 + c_0 p_0 p_1 \quad s_2 = a_2 \oplus b_2 \oplus c_2 \]
\[ c_3 = g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 p_1 p_2 \quad s_3 = a_3 \oplus b_3 \oplus c_3 \]
Contraction

*Contraction is the simplification of a circuit through constant input values.*
Contraction example: adder to incrementer

- What is the hardware and delay savings of implementing an incrementer using contraction?

Can be reduced to half-adders

Incrementer circuit

<table>
<thead>
<tr>
<th>$\bar{a}_0$</th>
<th>S</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$S_0 = \bar{a}_0$, $C_0 = a_0$