Lecture 13: ALUs, Adders

• Note: HW 6 submission has been moved to 2/29

• Today’s topics:
  ▪ ALU wrap-up
  ▪ Carry-lookahead adder
1-Bit ALU with Add, Or, And

- Multiplexor selects between Add, Or, And operations

Source: H&P textbook
32-bit Ripple Carry Adder

1-bit ALUs are connected “in series” with the carry-out of 1 box going into the carry-in of the next box.
Incorporating Subtraction

Must invert bits of B and add a 1
- Include an inverter
- CarryIn for the first bit is 1
- The CarryIn signal (for the first bit) can be the same as the Binvert signal

Source: H&P textbook
Incorporating NOR and NAND

Source: H&P textbook
Control Lines

What are the values of the control lines and what operations do they correspond to?

<table>
<thead>
<tr>
<th>Ai</th>
<th>Bn</th>
<th>Op</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OR</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Add</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sub</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NAND</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NOR</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Source: H&P textbook
Incorporating slt

- Perform $a - b$ and check the sign
- New signal (Less) that is zero for ALU boxes 1-31
- The 31\textsuperscript{st} box has a unit to detect overflow and sign – the sign bit serves as the Less signal for the 0\textsuperscript{th} box

Source: H&P textbook
Incorporating beq

- Perform $a - b$ and confirm that the result is all zero’s.

Source: H&P textbook
Control Lines

What are the values of the control lines and what operations do they correspond to?
Control Lines

What are the values of the control lines and what operations do they correspond to?

<table>
<thead>
<tr>
<th>Ai</th>
<th>Bn</th>
<th>Op</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OR</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Add</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sub</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NOR</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NAND</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SLT</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BEQ</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Source: H&P textbook
Speed of Ripple Carry

• The carry propagates thru every 1-bit box: each 1-bit box sequentially implements AND and OR – total delay is the time to go through 64 gates!

• We’ve already seen that any logic equation can be expressed as the sum of products – so it should be possible to compute the result by going through only 2 gates!

• Caveat: need many parallel gates and each gate may have a very large number of inputs – it is difficult to efficiently build such large gates, so we’ll find a compromise:
  ▪ moderate number of gates
  ▪ moderate number of inputs to each gate
  ▪ moderate number of sequential gates traversed
Computing CarryOut

\[
\text{Cout} = A \cdot B + A \cdot \text{Cin} + B \cdot \text{Cin}
\]

\[
\text{CarryIn1} = b_0.\text{CarryIn0} + a_0.\text{CarryIn0} + a_0.b_0
\]

\[
\text{CarryIn2} = b_1.\text{CarryIn1} + a_1.\text{CarryIn1} + a_1.b_1
\]

\[
= b_1.b_0.c_0 + b_1.a_0.c_0 + b_1.a_0.b_0 +
\]

\[
+ a_1.b_0.c_0 + a_1.a_0.c_0 + a_1.a_0.b_0 + a_1.b_1
\]

\[
\ldots
\]

\[
\text{CarryIn32} = \text{a really large sum of really large products}
\]

- Potentially fast implementation as the result is computed by going thru just 2 levels of logic – unfortunately, each gate is enormous and slow
Generate and Propagate

Equation re-phrased:
\[ C_{i+1} = a_i b_i + a_i C_i + b_i C_i \]
\[ = (a_i b_i) + (a_i + b_i) C_i \]

Stated verbally, the current pair of bits will generate a carry if they are both 1 and the current pair of bits will propagate a carry if either is 1

Generate signal = \( a_i b_i \)
Propagate signal = \( a_i + b_i \)

Therefore, \( C_{i+1} = G_i + P_i \cdot C_i \)
Generate and Propagate

c1 = g0 + p0.c0
\n\nc2 = g1 + p1.c1
= g1 + p1.g0 + p1.p0.c0
\n\nc3 = g2 + p2.g1 + p2.p1.g0 + p2.p1.p0.c0
\n\nc4 = g3 + p3.g2 + p3.p2.g1 + p3.p2.p1.g0 + p3.p2.p1.p0.c0

Either,
\na carry was just generated, or
\na carry was generated in the last step and was propagated, or
\na carry was generated two steps back and was propagated by both the next two stages, or
\na carry was generated N steps back and was propagated by every single one of the N next stages
Divide and Conquer

• The equations on the previous slide are still difficult to implement as logic functions – for the 32\textsuperscript{nd} bit, we must AND every single propagate bit to determine what becomes of c0 (among other things)

• Hence, the bits are broken into groups (of 4) and each group computes its group-generate and group-propagate

• For example, to add 32 numbers, you can partition the task as a tree

![Tree Diagram](image-url)
P and G for 4-bit Blocks

• Compute $P_0$ and $G_0$ (super-propagate and super-generate) for the first group of 4 bits (and similarly for other groups of 4 bits)
  $P_0 = p_0.p_1.p_2.p_3$
  $G_0 = g_3 + g_2.p_3 + g_1.p_2.p_3 + g_0.p_1.p_2.p_3$

• Carry out of the first group of 4 bits is
  $C_1 = G_0 + P_0.c_0$
  $C_2 = G_1 + P_1.G_0 + P_1.P_0.c_0$
  $C_3 = G_2 + (P_2.G_1) + (P_2.P_1.G_0) + (P_2.P_1.P_0.c_0)$

• By having a tree of sub-computations, each AND, OR gate has few inputs and logic signals have to travel through a modest set of gates (equal to the height of the tree)
### Example

<table>
<thead>
<tr>
<th>Add</th>
<th>A</th>
<th>0001</th>
<th>1010</th>
<th>0011</th>
<th>0011</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1110</td>
<td>0101</td>
<td>1110</td>
<td>1011</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>0000</td>
<td>0000</td>
<td>0010</td>
<td>0011</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1011</td>
<td></td>
</tr>
</tbody>
</table>

\[
P = \begin{bmatrix} 1 & 1 & 1 & 0 \\
G = \begin{bmatrix} 0 & 0 & 1 & 0 \\
C4 = 1
\end{bmatrix}
\]
Trade-Off Curve

- Truth table sum-of-products adder, $(2, 2^{64})$
- gp adder $(3, 33)$
- Carry Lookahead GP adder $(7, 5)$
- Ripple-Carry adder $(64, 2)$
Carry Look-Ahead Adder

- 16-bit Ripple-carry takes 32 steps
- This design takes how many steps? 5 sequential steps

Source: H&P textbook