March 01

- Chapter 4 – Logic Gates

- By Popular demand! The second exam will be 22 March, the SECOND Class after Spring Break.

- Comp 120 Discussion Forum Created. Link from the class home page.
  - ID = COMP120-001
  - PASSWD =
- Don’t give the password to anyone outside the class.
Ideal Switch

When Control == 0
   NC == IN (NC is connected to IN)
When Control == 1
   NO == IN (NO is connected to IN)
Inverter

Out = ~In
IF (In == 0) Out = 1 ELSE Out = 0
AND

Out = A&B
OR

\[ \text{Out} = A \lor B \]
2 Input MUX
4-input MUX
1-bit ALU for Addition

- Not easy to decide the “best” way to build something
  - Don’t want too many inputs to a single gate
  - Don’t want to have to go through too many gates
  - for our purposes, ease of comprehension is important

- Let’s look at a 1-bit ALU for addition:

\[
\begin{align*}
c_{\text{out}} &= a \& b \mid a \& c_{\text{in}} \mid b \& c_{\text{in}} \\
\text{sum} &= a \& \neg b \& \neg c_{\text{in}} \mid \neg a \& b \& \neg c_{\text{in}} \\
&\quad \mid \neg a \& \neg b \& c_{\text{in}} \mid a \& b \& c_{\text{in}}
\end{align*}
\]

<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>
Cout in Gates

\[
c_{\text{out}} = a \& b \mid a \& c_{\text{in}} \mid b \& c_{\text{in}}
\]
Sum in Gates

\[
\text{sum} = a \& \sim b \& \sim c_{\text{in}} \\
\quad | \sim a \& b \& \sim c_{\text{in}} \\
\quad | \sim a \& \sim b \& c_{\text{in}} \\
\quad | a \& b \& c_{\text{in}}
\]

\[
\begin{array}{c}
\sim \\
a \\
& \sim \\
b \\
& \sim \\
Cin \\
& \& \\
& \& \\
& \sim \\
& \& \\
& \& \\
\end{array}
\]

Sum
Building a 32 bit ALU

\[
\begin{array}{c}
\text{Operation} \\
\text{CarryIn}
\end{array}
\]

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{Result} \\
\text{CarryOut}
\end{array}
\]

\[
\begin{array}{ccc}
\text{ALU0} & \text{CarryIn} & \text{CarryOut} \\
\text{a0} & \text{b0} & \text{Result0}
\end{array}
\]

\[
\begin{array}{ccc}
\text{ALU1} & \text{CarryIn} & \text{CarryOut} \\
\text{a1} & \text{b1} & \text{Result1}
\end{array}
\]

\[
\begin{array}{ccc}
\text{ALU2} & \text{CarryIn} & \text{CarryOut} \\
\text{a2} & \text{b2} & \text{Result2}
\end{array}
\]

\[
\begin{array}{ccc}
\text{ALU31} & \text{CarryIn} & \text{CarryOut} \\
\text{a31} & \text{b31} & \text{Result31}
\end{array}
\]
What about subtraction \((a - b)\)?

- Two's complement approach: just negate \(b\) and add.
- How do we negate?

- A very clever solution:
Test for equality

\[\text{Note: zero is a 1 when the result is zero!}\]
How Fast?

What does this do?

How about this?
How many gate delays for Cout?

Cout is ready 2 gate delays after the inputs are ready.
How many gate delays for Sum?

Sum is ready 3 gate delays after the inputs are ready.
How slow can you go?

Cin for ALU31 is ready 62 gate delays after the initial input was ready
Result31 will be ready 3 gate delays later for a total of 65 gate delays

If a gate delay is 1ns what is the fastest clock rate for addition to happen in 1 cycle?

What about AND?
Conclusion

• We can build an ALU to support the MIPS instruction set
  – key idea: use multiplexer to select the output we want
  – we can efficiently perform subtraction using two’s complement
  – we can replicate a 1-bit ALU to produce a 32-bit ALU

• Important points about hardware
  – all of the gates are always working
  – the speed of a gate is affected by the number of inputs to the gate
  – the speed of a circuit is affected by the number of gates in series
    (on the “critical path” or the “deepest level of logic”)

• Our primary focus: comprehension, however,
  – Clever changes to organization can improve performance
    (similar to using better algorithms in software)