Robot rides bike! ⇒ The PRIMER-V2 robot is capable of starting from a stopped position, start riding, follows a path specified by a controller, and can stop without falling! Very very cool...

Review

• Use this table and techniques we learned to transform from 1 to another

Today

• Data Multiplexors
• Arithmetic and Logic Unit
• Adder/Subtractor

Data Multiplexor (here 2-to-1, n-bit-wide)

N instances of 1-bit-wide mux

How do we build a 1-bit-wide mux?

\[
\begin{align*}
c &= \overline{s}a + \overline{b} + s\overline{a} + sb \\
&= \overline{s}(a\overline{b} + ab) + s(\overline{a}b + ab) \\
&= \overline{s}(a\overline{b} + b) + s((\overline{a} + a)b) \\
&= \overline{s}(a(1) + s(1)b) \\
&= \overline{s}a + sb
\end{align*}
\]
4-to-1 Multiplexor?

How many rows in TT?

\[ e = \overline{s_1} \overline{s_0}a + \overline{s_1} s_0 b + s_1 \overline{s_0} c + s_1 s_0 d \]

Is there any other way to do it?

Hint: NCAA tourney!

Ans: Hierarchically!

Arithmetic and Logic Unit

• Most processors contain a special logic block called “Arithmetic and Logic Unit” (ALU)
• We’ll show you an easy one that does ADD, SUB, bitwise AND, bitwise OR

Our simple ALU

Adder/Subtractor Design -- how?

• Truth-table, then determine canonical form, then minimize and implement as we’ve seen before
• Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer

Adder/Subtractor – One-bit adder LSB...
Adder/Subtracter – One-bit adder (1/2)

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c_i</th>
<th>s_i</th>
<th>c_{i+1}</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>1</td>
<td>1</td>
<td>1</td>
<td>0</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>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

\[ s_i = c_{i+1} = \]

Adder/Subtracter – One-bit adder (2/2)

\[ s_i = \text{XOR}(a_i, b_i, c_i) \]

\[ c_{i+1} = \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i \]

N 1-bit adders \(\Rightarrow\) 1 N-bit adder

What about overflow?

- Consider a 2-bit signed # & overflow:
  - \(10 = -2\) + \(-2\) or \(-1\)
  - \(11 = -1\) + \(-1\) only
  - \(00 = 0\) NOTHING!
  - \(01 = 1\) + \(1\) only
- Highest adder
  - \(C_i = \text{Carry-in} = C_{\text{in}}\)
  - \(C_2 = \text{Carry-out} = C_{\text{out}}\)
  - No \(C_{\text{out}}\) or \(C_{\text{in}}\) \(\Rightarrow\) NO overflow!

What \(\cdot C_{\text{in}}\) and \(C_{\text{out}}\) \(\Rightarrow\) NO overflow!

\begin{align*}
\text{op?} & \quad \text{\(C_{\text{in}}\) but no \(C_{\text{out}}\) \(\Rightarrow\) A,B both > 0, overflow!} \\
& \quad \text{\(C_{\text{out}}\) but no \(C_{\text{in}}\) \(\Rightarrow\) A,B both < 0, overflow!}
\end{align*}

Extremely Clever Subtractor

\[ \text{XOR serves as conditional inverter!} \]

\[ a \quad \text{XOR}(a, b) \]

\[
\begin{array}{cccc}
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]
Peer Instruction

1) Truth table for mux with 4-bits of signals has $2^4$ rows
2) We could cascade N 1-bit shifters to make 1 N-bit shifter for sll, srl

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>FF</td>
<td>FT</td>
<td>TF</td>
<td>TT</td>
</tr>
</tbody>
</table>

“And In conclusion…”

• Use muxes to select among input
  • S input bits selects $2^S$ inputs
  • Each input can be n-bits wide, indep of S
• Can implement muxes hierarchically
• ALU can be implemented using a mux
  • Coupled with basic block elements
• N-bit adder-subtractor done using N 1-bit adders with XOR gates on input
  • XOR serves as conditional inverter