Overview

- Multiplexers
Motivation

• At this point, you’ve seen a lot of straightline circuits

• However, this doesn’t quite match up with respect to what a processor does. Why?
Motivation

- At this point, you’ve seen a lot of straightline circuits

- However, this doesn’t quite match up with respect to what a processor does. Why?
  - We don’t always do the same thing - it depends on the instruction
  - What do we need here?
Motivation

• At this point, you’ve seen a lot of straightline circuits

• However, this doesn’t quite match up with respect to what a processor does. Why?
  • We don’t always do the same thing - it depends on the instruction
  • What do we need here?
    • Some form of a conditional
Conditional

• Assume `selector`, `A`, `B`, and `R` all hold a single bit

• How can we implement this using what we have seen so far? (Hint: what does the truth table look like?)

\[ R = (\text{selector}) \ ? A : B \]
R = (selector) ? A : B

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
R = (selector) ? A : B

Unreduced sum-of-products:
R = !S!AB + !SAB + SA!B + SAB
\[ R = (\text{selector}) \ ? \ A : B \]

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Unreduced sum-of-products:**
\[ R = !S!AB + !SAB + SA!B + SAB \]

**Reduced sum-of-products:**
\[ R = !SB + SA \]
Slight Modification

Original

R = (selector) ? A : B

Modified

R = (selector) ? doThis() : doThat()
## Slight Modification

**Original**

\[ R = (\text{selector}) \ ? \ A : B \]

**Modified**

\[ R = (\text{selector}) \ ? \ \text{doThis}() : \text{doThat}() \]

**Intended semantics:** either \( \text{doThis}() \) or \( \text{doThat}() \) is executed. Our formula from before doesn’t satisfy this property:

\[ R = !S\times\text{doThat}() + S\times\text{doThis}() \]
Slight Modification

Original

\[ R = \text{(selector)} ? A : B \]

Modified

\[ R = \text{(selector)} ? \text{doThis()} : \text{doThat()} \]

- Fixing this is hard, but possible
- Involves circuitry we’ll learn later
- Oddly enough, this isn’t as big of a problem as it seems, and it’s ironically faster than doing just one or the other. Why?
Slight Modification

**Original**

\[ R = (\text{selector}) \ ? \ A : B \]

**Modified**

\[ R = (\text{selector}) \ ? \ \text{doThis}() : \text{doThat}() \]

- Oddly enough, this isn’t as big of a problem as it seems, and it’s ironically faster than doing just one or the other. Why? - branches executed in parallel at the hardware level. Faster because extra circuitry is extra.
Multiplexer

- Component that does exactly this:
  \[ R = (\text{selector}) \ ? \ A : B \]
Question

• Recall the arithmetic logic unit (ALU), which is used to add, subtract, shift, perform bitwise operations, etc.

• How might a multiplexer be useful for an ALU?

<table>
<thead>
<tr>
<th>Opcode / Function</th>
<th>Add</th>
<th>Addu</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>And</td>
<td>and</td>
<td></td>
</tr>
<tr>
<td>Add Unsigned</td>
<td>R</td>
<td>R[rd] = R[rs] + R[rt]</td>
<td>0 / 21&lt;sub&gt;hex&lt;/sub&gt;</td>
</tr>
<tr>
<td>And</td>
<td>R</td>
<td>R[rd] = R[rs] &amp; R[rt]</td>
<td>0 / 24&lt;sub&gt;hex&lt;/sub&gt;</td>
</tr>
</tbody>
</table>
Question

• Recall the arithmetic logic unit (ALU), which is used to add, subtract, shift, perform bitwise operations, etc.

• How might a multiplexer be useful for an ALU? - Do all operations at once in parallel, and then use a multiplexer to select the one you want.

Opcode / Function

<table>
<thead>
<tr>
<th>Add Unsigned</th>
<th>addu</th>
<th>$R = R[rs] + R[rt]$</th>
<th>$0 / 21_{\text{hex}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>and</td>
<td>$R = R[rs] &amp; R[rt]$</td>
<td>$0 / 24_{\text{hex}}$</td>
</tr>
</tbody>
</table>
Example

- Let’s design a one-bit ALU that can do bitwise AND and bitwise OR.
- It has three inputs: $A$, $B$, and $S$, along with one output $R$.
- $S$ is a code provided indicating which operation to perform; 0 for AND and 1 for OR.
Example
Bigger Multiplexers

• Can have a multiplexer with more than two inputs
• Need multiple select lines in this case
• Question: how many select lines do we need for a 4 input multiplexer?
Bigger Multiplexers

- Can have a multiplexer with more than two inputs
- Need multiple select lines in this case
- Question: how many select lines do we need for a 4 input multiplexer? - 2. Values of different lines essentially encode different binary integers.
Bigger Multiplexers

• We can build up bigger multiplexers from 2-input multiplexers. How?