Today

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

Review

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

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

N instances of 1-bit-wide mux

How many rows in TT?

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

\[ c = \overline{s}a \overline{b} + \overline{s}ab + \overline{s}a \overline{b} + s \overline{a}b \]

\[ = \overline{s}(\overline{a}b + ab) + s(\overline{a}b + ab) \]

\[ = \overline{s}(a \overline{b} + b) + s((\overline{a} + a)b) \]

\[ = \overline{s}(a1) + s(1)b \]

\[ = \overline{s}a + sb \]
4-to-1 Multiplexor?

How many rows in TT?

\[ e = \overline{s_1}s_0a + \overline{s_1}s_0b + s_1\overline{s_0}c + s_1s_0d \]

Is there any other way to do it?

Hint: NCAA tourney!

Ans: Hierarchically!

Administivia

• HW due wednesday

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

\[ \begin{align*}
&\text{ALU} \\
&\text{Input: } A, B \\
&\text{Output: } S, R \\
&\text{Truth: } \\
&s=00, R=A+B \\
&s=01, R=A-B \\
&s=10, R=A \text{ AND } B \\
&s=11, R=A \text{ OR } B \\
\end{align*} \]

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...

<table>
<thead>
<tr>
<th>a_3</th>
<th>a_2</th>
<th>a_1</th>
<th>a_0</th>
<th>s_0</th>
<th>c_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

s_0 =

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

<table>
<thead>
<tr>
<th>a_3</th>
<th>a_2</th>
<th>a_1</th>
<th>a_0</th>
<th>s_0</th>
<th>c_1</th>
<th>s_1</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>
<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>
<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>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

• Adder/Subtractor – One-bit adder (2/2)...

s_i = XOR(a_i, b_i, c_i)

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

What about overflow?

• Consider a 2-bit signed # & overflow:
  • 10 = -2 + -2 or -1
  • 11 = -1 + -2 only
  • 00 = 0 nothing!
  • 01 = 1 + 1 only

• Highest adder
  - C_1 = Carry-in = C_{in}, C_2 = Carry-out = C_{out}
  - No C_{out} or C_{in} = NO overflow!
  - C_{in} and C_{out} = NO overflow!

• Overflows if...
  - C_{in} but no C_{out} ⇒ A,B both > 0, overflow!
  - C_{out} but no C_{in} ⇒ A,B both < 0, overflow!

Overflow = C_n XOR C_{n-1}

N 1-bit adders ⇒ 1 N-bit adder

What about overflow?

• Consider a 2-bit signed # & overflow:
  • 10 = -2
  • 11 = -1
  • 00 = 0
  • 01 = 1

• Overflows when...
  - C_{in} but no C_{out} ⇒ A,B both > 0, overflow!
  - C_{out} but no C_{in} ⇒ A,B both < 0, overflow!

overflow = C_n XOR C_{n-1}
A. Truth table for mux with 4-bits of signals has $2^4$ rows.

B. We could cascade $N$ 1-bit shifters to make 1 $N$-bit shifter for sll, srl.

C. If 1-bit adder delay is $T$, the $N$-bit adder delay would also be $T$.

Peer Instruction Answer

• Use muxes to select among input
  • S input bits selects $2S$ inputs
  • Each input can be n-bits wide, independant of $S$

• 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