CS 61C: Great Ideas in Computer Architecture (Machine Structures)
*Muxes, Adders, and ALUs*

Instructors:
Randy H. Katz
David A. Patterson
http://inst.eecs.Berkeley.edu/~cs61c/fa10

---

**Agenda**

- Multiplexer
- Administrivia
- Technology Break
- ALU Design

---

**Data Multiplexer**
(e.g., 2-to-1 x n-bit-wide)

![Data Multiplexer Diagram]

**N Instances of 1-bit-Wide Mux**

How many rows in TT?

\[
c = \overline{s}ab + \overline{s}ab + s\overline{a}b + sab \\
= \overline{s}(ab + ab) + s(\overline{a}b + ab) \\
= \overline{s}(a(b + b)) + s(\overline{a} + a)b \\
= \overline{s}(a(1) + s(1)b) \\
= \overline{s}a + sb
\]

---

**How Do We Build a 1-bit-Wide Mux?**

![1-bit-Wide Mux Diagram]
4-to-1 Multiplexer

The 4-to-1 multiplexer is a circuit that selects one of four input signals based on a 2-bit input address. The truth table (TT) for a 4-to-1 multiplexer can be represented as follows:

<table>
<thead>
<tr>
<th>Input Address</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
</tbody>
</table>

The output expression for the multiplexer can be written as:

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

Alternative Hierarchical Approach

In the alternative hierarchical approach, the multiplexer is implemented using a combination of AND, OR, and NOT gates. The output e is a function of the input variables a, b, c, and d.

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.

Simple ALU

The ALU performs basic arithmetic and logic operations. The output is determined by the input variables A, B, and S.

Agenda

- Multiplexers
- Administrivia
- Technology Break
- ALU

Agenda

- Mux + Adder Design
- Administrivia
- Technology Break
- ALU Design
Agenda

• Multiplexer
• Administrivia
• Technology Break
• ALU Design

Adder/Subtractor: One-bit adder
Least Significant Bit

\[
\begin{array}{cccc|ccc}
 & a_3 & a_2 & a_1 & a_0 & b_3 & b_2 & b_1 & b_0 \\
+ & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
\hline
s_3 & s_2 & s_1 & s_0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

\[s_0 = a_0\]
\[c_1 = a_0\]

Overflow Conditions
Add two positive numbers to get a negative number
or two negative numbers to get a positive number

N x 1-bit Adders \(\Rightarrow\) 1 N-bit Adder

Connect Carry Out \(i-1\) to Carry in \(i\):
### Overflow Conditions

<table>
<thead>
<tr>
<th>5</th>
<th>0111</th>
<th>0101</th>
<th>3</th>
<th>0011</th>
<th>1000</th>
<th>#</th>
<th>1000</th>
<th>1001</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0011</td>
<td>0100</td>
<td>#</td>
<td>1011</td>
<td>0111</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overflow

<table>
<thead>
<tr>
<th>5</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0101</td>
</tr>
<tr>
<td>#</td>
<td>1111</td>
</tr>
</tbody>
</table>

No overflow

### Twos Complement Adder/Subtractor

Overflow when carry in to sign does not equal carry out: \( C_n \) xor \( C_{n-1} \)

### Design Hierarchy

- System
  - Datapath
    - Registers
    - Multiplexer
    - Comparator
  - Control
    - Register
    - Logic
    - Switching networks
  - Combinational logic

### Summary

- 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