Building an ALU (Part 2):
State – the central concept of computing

Computer can do 2 things
1) Store state
2) Manipulate state (Combine arithmetic and logical operations into one unit)
Today’s lecture

- We’ll finish the 32-bit ALU today!
  - 32-bit ALU specification

- Complete 1-bit ALU
- Assembling them to make 32-bit ALU
- Handling flags:
  - zero, negative, overflow
A specification for a 32-bit ALU

Did overflow occur?
Is the output equal to zero?
Is the output negative?

module alu32(out, overflow, zero, negative, A, B, control);
  output[31:0] out;
  output overflow, zero, negative;
  input [31:0] A, B;
  input [2:0] control;

  control out=
  0 undefined
  1 undefined
  2 A + B
  3 A – B
  4 A AND B
  5 A OR B
  6 A NOR B
  7 A XOR B
Use a modular 1-bit ALU to build 32-bit ALU

- Previously we showed 1-bit adder/subtractor, 1-bit logic unit
  - Time to put them together.

<table>
<thead>
<tr>
<th>control</th>
<th>( \text{out}_i = )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>undefined</td>
</tr>
<tr>
<td>1</td>
<td>undefined</td>
</tr>
<tr>
<td>2</td>
<td>( A_i + B_i )</td>
</tr>
<tr>
<td>3</td>
<td>( A_i - B_i )</td>
</tr>
<tr>
<td>4</td>
<td>( A_i \text{ AND } B_i )</td>
</tr>
<tr>
<td>5</td>
<td>( A_i \text{ OR } B_i )</td>
</tr>
<tr>
<td>6</td>
<td>( A_i \text{ NOR } B_i )</td>
</tr>
<tr>
<td>7</td>
<td>( A_i \text{ XOR } B_i )</td>
</tr>
</tbody>
</table>

module alu1(out, carryout, A, B, carryin, control);
output out, carryout;
input A, B, carryin;
input [2:0] control;
Addition + Subtraction in one circuit (1-bit Arithmetic Unit)

- When Sub = 0, Y = B and Cin = 0. Result = A + B + 0 = A + B.
- When Sub = 1, Y = \sim B and Cin = 1. Result = A + \sim B + 1 = A - B.

Which parts belong in inside the 1-bit ALU?

A) the Full Adder,  B) the XOR gate,  C) Both,  D) Neither
Addition + Subtraction in one circuit (1-bit Arithmetic Unit)

- When Sub = 0, Y = B and Cin = 0. Result = A + B + 0 = A + B.
- When Sub = 1, Y = ~B and Cin = 1. Result = A + ~B + 1 = A − B.

What should we do with the full adder’s Cin input?

A) Connect to Sub,  B) Connect to 1-bit ALU’s carryin
Addition + Subtraction in one circuit (1-bit Arithmetic Unit)

- When Sub = 0, Y = B and Cin = 0. Result = A + B + 0 = A + B.
- When Sub = 1, Y = ~B and Cin = 1. Result = A + ~B + 1 = A – B.

Where will the “Sub” signal come from?

<table>
<thead>
<tr>
<th>control</th>
<th>out=</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>undefined</td>
</tr>
<tr>
<td>1</td>
<td>undefined</td>
</tr>
<tr>
<td>2</td>
<td>A + B</td>
</tr>
<tr>
<td>3</td>
<td>A – B</td>
</tr>
</tbody>
</table>
Complete 1-bit ALU
Complete 1-bit Logic Unit

- What should the control inputs (R0, R1) connect to?
- How do we select between the adder and the logic unit?
- How do we control the selection?

<table>
<thead>
<tr>
<th>R1</th>
<th>R0</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( G_i = X_i Y_i )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>( G_i = X_i + Y_i )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>( G_i = (X_i + Y_i)' )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>( G_i = X_i \oplus Y_i )</td>
</tr>
</tbody>
</table>
Complete 1-bit ALU
Connecting 1-bit ALUs
Flags (overflow, zero, negative)

- Let’s do negative first; negative evaluates to:
  - 1 when the output is negative, and
  - 0 when the output is positive or zero

- Negative =
  a) carryout [30]
  b) output [30]
  c) carryout [31]
  d) output [31]
  e) control [0]
  f) moar coffee
Flags (overflow, zero, negative)

- zero evaluates to:
  - 1 when the output is equal to zero, else 0

- Zero = \text{nor}(\text{output}, \text{in}1, \text{in}2, \text{in}3, \text{in}4, \text{in}5, \ldots, \text{in}31)
Flags (overflow, zero, negative)

- Overflow (for 2’s complement) evaluates to:
  - 1 when the overflow occurred, else 0
  - adding two positive numbers yields a negative number
  - adding two negative numbers yields a positive number

- Consider the adder for the MSB:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Cin</th>
<th>Cout</th>
<th>S</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
<td>0</td>
<td>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Overflow =
  - a) cin[31] NOR cout[31]
  - b) cin[31] AND cout[31]
  - c) cin[31] OR cout[31]
  - d) cin[31] XOR cout[31]
  - e) cin[31] NAND cout[31]
Overflow examples

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \quad (-3) \\
+ & 1 & 1 & 0 & 0 \quad + (-4) \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 0 & 0 \quad 4 \\
+ & 0 & 1 & 0 & 0 \quad 4 \\
\hline
1 & 1 & 0 & 0 \quad + (-4)
\end{array}
\]

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \quad (-5) \\
+ & 1 & 1 & 0 & 0 \quad + (-4) \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 0 & 0 \quad 4 \\
+ & 1 & 1 & 0 & 0 \quad + (-4) \\
\hline
1 & 1 & 0 & 0 \quad + (-4)
\end{array}
\]