Logic for Arithmetic

adders

Arithmetic Logic Unit
Addition: 1-bit *half* adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Carry out</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>
</tr>
</tbody>
</table>
Addition: 1-bit *full* adder

<table>
<thead>
<tr>
<th>Carry in</th>
<th>A</th>
<th>B</th>
<th>Carry out</th>
<th>Sum</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>1</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</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>
Addition: $n$-bit *ripple-carry* adder

There are faster, more complicated ways too...
Processor Components

Abstraction!
Arithmetic Logic Unit (ALU)

Hardware unit for arithmetic and bitwise operations.
1-bit ALU for bitwise operations

Build an n-bit ALU from n 1-bit ALUs.
Each bit \( i \) in the result is computed from the corresponding bit \( i \) in the two inputs.

<table>
<thead>
<tr>
<th>Op</th>
<th>A</th>
<th>B</th>
<th>Result</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Logic for Arithmetic
1-bit ALU

A

B

Operation

Carry in

MUX

Result

Carry out

Sum

0

1

2

2
n-bit ripple carry adder

\[
\begin{align*}
\text{Carry in} & \\
A_0 & \rightarrow \text{Sum}_0 \\
B_0 & \rightarrow \text{Sum}_0 \\
A_1 & \rightarrow \text{Sum}_1 \\
B_1 & \rightarrow \text{Sum}_1 \\
A_2 & \rightarrow \text{Sum}_2 \\
B_2 & \rightarrow \text{Sum}_2 \\
& \vdots \\
A_{n-1} & \rightarrow \text{Sum}_{n-1} \\
B_{n-1} & \rightarrow \text{Sum}_{n-1} \\
\text{Carry out} & \\
\end{align*}
\]

n-bit ALU

\[
\begin{align*}
\text{Carry in} & \\
A_0 & \rightarrow \text{Sum} \\
B_0 & \rightarrow \text{Sum} \\
A_1 & \rightarrow \text{Sum} \\
B_1 & \rightarrow \text{Sum} \\
& \vdots \\
A_{n-1} & \rightarrow \text{Sum} \\
B_{n-1} & \rightarrow \text{Sum} \\
\text{Carry out} & \\
\end{align*}
\]

Logic for Arithmetic
**ALU conditions**

Extra ALU outputs describing properties of result.

- **Zero Flag:** 1 if result is 00...0 else 0
- **Sign Flag:** 1 if result is negative else 0
- **Carry Flag:** 1 if carry out else 0
- **(Signed) Overflow Flag:** 1 if signed overflow else 0

Implement these.
Add subtraction

How can we control ALU inputs or add minimal new logic to compute \( A - B \)?
A NAND B

A NOR B

A < B

A == B

How can we control ALU inputs or add minimal new logic to compute each?
Controlling the ALU

<table>
<thead>
<tr>
<th>ALU control lines</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>AND</td>
</tr>
<tr>
<td>0001</td>
<td>OR</td>
</tr>
<tr>
<td>0010</td>
<td>add</td>
</tr>
<tr>
<td>0110</td>
<td>subtract</td>
</tr>
<tr>
<td>1100</td>
<td>NOR</td>
</tr>
</tbody>
</table>

Condition Codes

Operand A

Operand B

Result

Control Lines

Abstraction!