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>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Addition: n-bit ripple-carry adder

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

- Instruction Fetch and Decode
- Registers
- ALU
- Memory

**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>

**1-bit ALU**

Operation

Carry in

MUX

Result
**n-bit ripple carry adder**

```
A0 → + → Sum0
B0 → + → Sum0
A1 → + → Sum1
B1 → + → Sum1
A2 → + → Sum2
B2 → + → Sum2
An-1 → + → Sumn-1
Bn-1 → + → Sumn-1
```

**n-bit ALU**

```
A0 → + → Sum0
B0 → + → Sum0
A1 → + → Sum1
B1 → + → Sum1
A2 → + → Sum2
B2 → + → Sum2
An-1 → + → Sumn-1
Bn-1 → + → Sumn-1
```

**Add subtraction**

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

```
A0 → + → Sum0
B0 → + → Sum0
A1 → + → Sum1
B1 → + → Sum1
A2 → + → Sum2
B2 → + → Sum2
An-1 → + → Sumn-1
Bn-1 → + → Sumn-1
```

**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.

```
A0 → + → Sum0
B0 → + → Sum0
A1 → + → Sum1
B1 → + → Sum1
A2 → + → Sum2
B2 → + → Sum2
An-1 → + → Sumn-1
Bn-1 → + → Sumn-1
```

**Negate B**

A NOR B

A<N

A==B

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

## Abstraction!

<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>

### Diagram: ALU Control

- **Condition Codes**
- **Operand A**
- **Operand B**
- **ALU**
- **Result**
- **Control Lines**