Arithmetic Logic

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>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</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>

Carry in

Carry out

A

B

Sum

Carry in

Carry out

Arithmetic Logic
Addition: \textit{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>

Arithmetic Logic
1-bit ALU

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(00)</td>
</tr>
<tr>
<td>1</td>
<td>(01)</td>
</tr>
<tr>
<td>2</td>
<td>(10)</td>
</tr>
</tbody>
</table>

(11 is not used!)

A

B

Carry in

Carry out

Sum

MUX
n-bit ripple carry adder

Carry in

A₀ → Sum₀
B₀ → Sum₀

A₁ → Sum₁
B₁ → Sum₁

A₂ → Sum₂
B₂ → Sum₂

... → ...

Aᵢ → Sumᵢ
Bᵢ → Sumᵢ

Aᵢ₋₁ → Sumᵢ₋₁
Bᵢ₋₁ → Sumᵢ₋₁

Carry out

n-bit ALU

Carry in

A₀ → Sum
B₀ → Sum

A₁ → Sum
B₁ → Sum

A₂ → Sum
B₂ → Sum

... → ...

Aᵢ → Sum
Bᵢ → Sum

Aᵢ₋₁ → Sum
Bᵢ₋₁ → Sum

Operation

Result₀

Result₁

Result₂

... → ...

Resultᵢ

Resultᵢ₋₁

Carry out

Arithmetic Logic
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

You will implement these in Circuits Assignment!
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? You will implement these in Circuits Assignment!
## 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>???</td>
<td>NOR</td>
</tr>
</tbody>
</table>

### Diagram

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

---

Abstraction!