Motivation: how do we go from code to gates?

```
int count_odds(int array[10]) {
    int count = 0;
    for (int i = 0; i < 10; i++) {
        count += array[i] & 0x1;
    }
    return count;
}
```

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>

Hint: the smallest solution uses 2 gates from: AND, OR, XOR, NOT, NAND, NOR
Addition: \textit{n-bit ripple-carry} adder

Processor Components

Arithmetic Logic Unit (ALU)

1-bit ALU for bitwise operations

We will use \( n \) 1-bit ALUs to build an \( n \)-bit ALU. Each bit \( i \) in the result is computed from the corresponding bit \( i \) in the two inputs.

An example (simplified) 1-bit ALU
1-bit ALU: 3 operations

1-bit ALU: 3 operations

Controlling the ALU

<table>
<thead>
<tr>
<th>ALU control lines</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>AND</td>
</tr>
<tr>
<td>01</td>
<td>OR</td>
</tr>
<tr>
<td>10</td>
<td>add</td>
</tr>
</tbody>
</table>

Include subtraction

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

Recall:

$$A - B = A + (-B) = A + (-B + 1)$$

Plan:

Feed bitwise-not $B$ into the adder
Add an extra 1: how?
Include subtraction

Plan to compute $A - B$:
1. Feed bitwise-not $B$ into the adder
2. Add an extra 1

Key insight:
The same selector bit (0 or 1) can be used for both!
1. Feed the selector into a new 2:1
   mux to choose B or ~B
2. Feed the selector in as the carry
   in to the least significant bit

Negate B

Operation

Invert A

Negate B

Operation

ALU control lines | Function
--- | ---
000 | AND
001 | OR
010 | add
110 | subtract
... | ...

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

You will implement some of these in the Arch 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>...</td>
</tr>
</tbody>
</table>

How many different functions (operations) could this ALU theoretically perform?

- 4
- 8
- 16
- 32

None of the above
ALU conditions (additional outputs)

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 the Arch Assignment!