So far

Levels of transformation:

<table>
<thead>
<tr>
<th>Work life</th>
<th>Problem statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS 374</td>
<td>Algorithm</td>
</tr>
<tr>
<td>ECE 220</td>
<td>Program</td>
</tr>
<tr>
<td>120</td>
<td>Instruction Set Architecture</td>
</tr>
<tr>
<td>ECE</td>
<td>Microarchitecture</td>
</tr>
<tr>
<td></td>
<td>Logic circuits</td>
</tr>
<tr>
<td></td>
<td>Devices</td>
</tr>
</tbody>
</table>

Function representation
Binary representation
Binary addition

Let's consider the addition of two N-bit numbers:

\[ \begin{align*}
C_n & C_{n-1} \ldots C_2 \ C_1 \\
A_n & A_{n-1} \ldots A_2 \ A_1 \ A_0 \\
B_n & B_{n-1} \ldots B_2 \ B_1 \ B_0
\end{align*} \]

Carry (C)
First operand (A)
Second operand (B)
Summation (S)

Remember: fixed-width calculations
1-bit half-adder (HA)

Truth table:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>S</th>
<th>Cout</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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Implementation:

Symbol:
1-bit full adder (FA)

Truth table:

<table>
<thead>
<tr>
<th>Cin</th>
<th>a</th>
<th>b</th>
<th>S</th>
<th>Cout</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>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</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>1</td>
</tr>
</tbody>
</table>

Symbol:

Using 3-variable K-maps:

\[
S = \begin{array}{c|c|c|c|c}
    & a & b & S  & Cout \\
  \hline
  \text{Cin} & 0 & 0 & 0  & 0 \\
  & 0 & 1 & 1  & 0 \\
  & 1 & 0 & 1  & 0 \\
  & 1 & 1 & 0  & 1 \\
\end{array}
\]

\[
C_{out} = \begin{array}{c|c|c|c|c}
    & a & b & C_{out} & \text{Cin} \\
  \hline
  \text{Cin} & 0 & 0 & 0  & 0 \\
  & 0 & 1 & 1  & 0 \\
  & 1 & 0 & 1  & 1 \\
  & 1 & 1 & 0  & 1 \\
\end{array}
\]
Implementation:

![Logic circuit diagram](image)
Ripple carry adder

Consider how to add $n$-bit numbers, e.g., 3-bit numbers:

```
  \[ \begin{array}{c}
  C_3 \\
  \text{Cin} \\
  S_0 \\
  S_1 \\
  S_2 \\
  S_3 \\
  \end{array} \]
```

Question: how do we subtract in 2's complement?

Subtraction using ripple carry adder:

```
  \[ \begin{array}{c}
  C_3 \\
  \text{Cin} \\
  S_0 \\
  S_1 \\
  S_2 \\
  \end{array} \]
```
Question: can we combine both designs in a single circuit?

Combined adder/subtractor:

Question: how do we detect overflow in 2's complement addition/subtraction?
Final circuit:

\[ \begin{align*}
\alpha_2 & \quad \rightarrow \\
& \quad \downarrow \\
\alpha_1 & \quad \rightarrow \\
& \quad \downarrow \\
\alpha_0 & \quad \rightarrow \\
& \quad \downarrow \\
b_2 & \quad \rightarrow \\
& \quad \downarrow \\
b_1 & \quad \rightarrow \\
& \quad \downarrow \\
b_0 & \quad \rightarrow \\
& \quad \downarrow \\
K & \quad \rightarrow \\
\end{align*} \]
Arithmetic unit

We can extend the 2’s complement adder/subtractor to perform more functions by adding more control inputs.

Example:

<table>
<thead>
<tr>
<th>$K_1$</th>
<th>$K_0$</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$A + B$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$A - B = A + B' + 1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$A$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$B$</td>
</tr>
</tbody>
</table>
General structure of modified network: