So far

Levels of transformation:

- Problem statement
- Algorithm
- Program
- Instruction Set Architecture
- Microarchitecture
- Logic circuits
- Devices

Function representation
Binary representation
REGISTER TRANSFER LEVEL (RTL)

So far

   Combinational and sequential logic

Now

   Computer architecture

Introduction to RTL

Question: how do we represent data flow without caring about implementation details?
Example: add two $n$-bit quantities $A, B$ and store result in register $R$

\[ R \leftarrow A + B \]

Every statement in RTL implies a hardware construct
Example: when $S=1$, copy contents of register $R_1$ into register $R_2$

RTL notation: $S: R_2 \leftarrow R_1$

Diagram:
Example: if $S_1 = 1$, copy contents of $R1$ to $R0$,
    else if $S_2 = 1$ then copy contents of $R2$ to $R0$

RTL notation:

Diagram:

\[ S_1: R0 \leftarrow R1 \]
\[ S_1, S_2: R0 \leftarrow R2 \]
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letters</td>
<td>Register</td>
<td>AR, R2, PC, IR, SP</td>
</tr>
<tr>
<td>Parentheses</td>
<td>Part of a register</td>
<td>R2(1), R1(5 : 3)</td>
</tr>
<tr>
<td>Arrow</td>
<td>data transfer</td>
<td>R1 ← R2</td>
</tr>
<tr>
<td>Square brackets</td>
<td>memory address</td>
<td>DR ← M[AR]</td>
</tr>
</tbody>
</table>

RTL micro-operations include *transfer, arithmetic, logic, shift* operations.

Symbols include:

- \( R_x ← R_y + R_z \)  addition
- \( R_x ← \overline{R_y} \) 1’s complement
- \( R_x ← R_y + 1 \) 2’s complement
- \( R_x ← R_y + R_z + 1 \) subtraction
- \( R_x ← R_x + 1 \) increment register
- \( R_x ← R_x - 1 \) decrement register
- \( R_x ← \overline{R_y} \) bitwise NOT
- \( R_x ← R_y \land R_z \) bitwise AND
- \( R_x ← R_y \lor R_z \) bitwise OR
- \( R_x ← R_y \oplus R_z \) bitwise XOR
- \( R_x ← l_r R_y \) shift left
- \( R_x ← r_r R_y \) shift right

**Question:** how do we connect multiple registers?
**Bus** Set of interconnection pathways shared by multiple sources and destinations

![Diagram of bus and registers]

**Question:** Can we further simplify the circuit?
Tri-state buffer

Analyze the following circuit:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>IN</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Symbol:

---

Purpose:

Tri-state buffer allows to disconnect output from the rest of the circuit.
We can use a tri-state buffer to implement a bidirectional port:

Symbol:
Bus (revisited)

Using multiple registers we get:

```
LD0
LOAD
RO ▼
ENABLE
EN0

LD1
LOAD
R1 ▼
ENABLE
EN1

LD2
LOAD
R2 ▼
ENABLE
EN2

BUS ▼
```