# Introduction to Digital Logic and CPU Design

Nicholas Miehlbradt

August 12, 2022

# Quick Recap

Combinational logic circuits:

- How we can specify them
  - Truth tables
  - Logical formula
- How we can implement them
  - Inspection
  - Simplification
  - Sum of Products (briefly)

We designed a ripple-carry adder and a multiplexer.

# Recap of ISA

Our very simple ISA:

| Instruction Name | Description |
|---|---|
| put | Put a value directly into a register |
| jnz | Jump if prev result was not zero |
| ldr | Move data from memory to register |
| str | Move data from register to memory |
| add | Add two values |
| orr | Or two values |
| and | And two values |
| not | Not a value |

What else do we need to specify?

# Design Time!

Specification:

- Input A (16 bits)
- Input B (16 bits)
- Input OP (2 bits)
- Output RESULT (16 bits)
- OP tells us what combination of A and B should be displayed on the output.

| **Code** | **Operation** |
|---|---|
| 00 | $A + B$ |
| 01 | $A \vert B$ |
| 10 | $A \& B$ |
| 11 | $\sim A$ |

# ALU

We've built an ALU (Arithmetic and Logic Unit). This is one of the core parts of a CPU and is responsible for doing all the arithmetic and logic operations (hardware designers are pretty straightforward with naming). In modern processors ALU might be able to do many more operations or even multiple operations at the same time.

# Sequential Logic

So far all the circuits have flowed in one direction. What if we connected the output of a gate back to its inputs?

# SR Latch

The simplest of sequential circuits.

| **S** | **R** | $\mathbf{Q}_n$ |
|---|---|---|
| 0 | 0 | latch |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | X |

When S is high then the output is high. When R is high then the output is low. When both S and R are low then the output keeps whatever state is was set to last. When both S and R are high the output is undefined.

# D Latch

The inputs of the SR latch aren't super useful. What we want is the ability to 'remember' the state of a wire.
Sequential logic is very difficult to design from scratch. What we can do is take a simple circuit that we know works and use combinational logic to transform its inputs and outputs into what we want.

| **en** | **D** | $\mathbf{Q}_n$ |
|---|---|---|
| 0 | 0 | latch |
| 0 | 1 | latch |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# D Flip Flop

The D latch by itself is not quite enough because it is **asynchronous**. The output updates whenever the enable input is high. This can cause problems.

We want to introduce an input to synchronise changes to our latch. This synchronisation transforms our D Latch into a D Flip Flop.

# Register File

Specification:

- We want to have 8 registers
- Inputs
  - Clock
  - Read Select 1 (3 bits)
  - Read Select 2 (3 bits)
  - Write Enable (1 bit)
  - Write Select (3 bits)
  - Write Data (16 bits)
- Outputs
  - Read Data 1 (16 bits)
  - Read Data 2 (16 bits)
- Register 0 should always output 0.

# Wrapping Up

Today we've built two major components of our CPU (hopefully).

Next week we will look at memory a bit more and start putting everything together and execute some instructions.

# Useful Links

- Digital (Logic Simulator)
- Digital Design and Computer Architecture (Textbook)