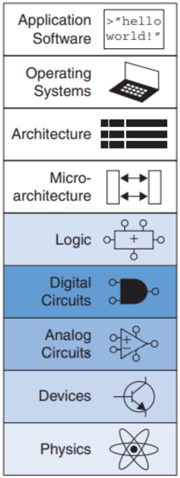
Introduction to Digital Logic and CPU Design

Nicholas Miehlbradt

August 5, 2022

What does modern computing abstraction look like?



What operations does a computer need to be able to do?

What are fundamental operations that are present somewhere in all programming languages. Can we make these operations so basic that we can implement them using logic gates?

What operations does a computer need to be able to do?

What are fundamental operations that are present somewhere in all programming languages. Can we make these operations so basic that we can implement them using logic gates?

- Arithmetic (and Logic) - Add, sub, and, or, etc.
- Control flow (decisions) - jumps, branches, conditional execution
- Data - moving data around the computer e.g. between different bits of memory

An instruction set architecture

An instruction set architecture is a specification of what operations a processor supports and how they behave.

It describes both the syntax (how it looks/is written) and the semantics (what the operations do). The syntax consists of both assembly (human readable encoding) and machine language (binary encoding).

A processors 'implements' a specification. It needs to behave in the way described so that programmers can rely on behaviours and trust that their code will do what is expected.

Our ISA

Modern ISAs are extremely complex, we will design a processor following our own simple custom ISA with just eight instructions.

put	**jnz**	**ldr**	**str**
add	**orr**	**and**	**not**

Is this enough to compute anything possible?

Note that this is not a full ISA, it is missing some crucial parts e.g. how the instructions are represented in machine language. We'll come back to this and fully define everything throughout the sessions.

What are the parts of digital logic

There are two broad families of digital logic circuits

- Combinational - circuits that only depend on their current input
- Sequential - circuits that have a state or memory. Even with the same inputs a sequential circuit might output a different value depending on what the previous inputs were.

How do we specify combinational logic?

Since combinational logic circuits only depend on the current input, it is easy to come up with a table with all possible inputs and their outputs.
There are different conventions:

- True/False
- 1/0
- High/Low

A	**B**	**Q**
0	0	0
0	1	0
1	0	0
1	1	1

Quick refresher on number systems

We normally use base 10, each place value can have 10 possible values before rolling over.

10 is (somewhat arbitrary). Computers tend to use base 2 since it can easily be represented with a high or low voltage.

Sometimes we use base 16 because it is a more compact representation then base 2 and is easy to convert. We use the letters A-F as the 6 additional digits.

Adding numbers

How can we translate adding two numbers into logic gates? Consider the simplest situation, adding two single digit binary numbers (1 bit each). How many outputs do we need? Come up with a truth table for a one bit adder.
Hints:

- Try doing the math manually first.
- Come up with a separate truth table for each output.

Half Adder Truth Table

A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Try implementing this in digital logic.

You might notice this is called a **half** adder. What might go wrong when extending this circuit to add more bits together?

Full Adder Truth Table

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Try implementing this circuit. Hint: You can reuse the half adder.

A proper adder

Congratulations, you've designed a circuit that can add two numbers together.

We can make an adder that adds together numbers as big as we want by just chaining more full adders together, connecting the C_{out} of one adder to the C_{in} of the next.

We won't cover subtraction, but you can look into it yourself if you want. There's a pretty cool way of doing it that reuses the adder we just built with some very minor modifications.

Another important circuit

Another very useful operation is the ability to pick between two inputs.

- Given two inputs A and B, a third input S picks whether A or B is copied to the output.
- If S=0 then whatever value is on A should be copied to the output and B is ignored. If S=1 then B is copied and A is ignored.

Try to come up with the truth table for this circuit.

Multiplexer truth table

S	A	B	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Simplifying specification

We can use a rigorous approach called sum of products to do this or we can simplify the truth table. If S=0 then we don't care what B is, we can combine rows that only differ by B when S=0. Likewise if S=1 we don't care what A is and we can combine rows that only differ by A.

S	**A**	**B**	**Q**
0	0	x	0
0	1	x	1
1	x	0	0
1	x	1	1

This gets us to a case where we are again dealing with 2 inputs which is much easier. Try to implement this in Digital.

Extending the Multiplexer

The multiplexer we designed is called a 2-to-1 one bit multiplexer.

- Can you extend it so that the inputs A and B and be wider (use more bits).
- Can you extend it so that it can select between more different inputs.

Useful Links

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