Computers Aren’t Magic

An Introduction to Computer Architecture
CS 196 – 25 DotStar [Lecture A03]

Let’s Build a Computer

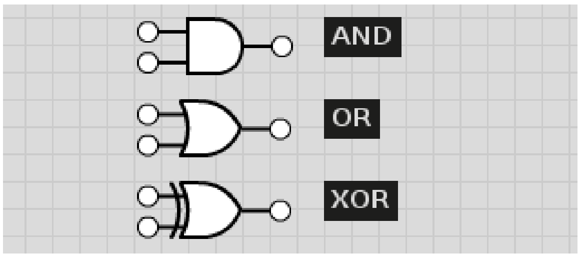
What are we working with?

Anything in digital hardware is either on or off, one or zero. More complicated things, like numbers, are represented by a bunch of ones and zeros stuck together. This is binary representation, much like all the numbers are represented in decimal by a bunch of 0-9 symbols. Hexadecimal (often abbreviated to hex) is a shorter way to represent binary by converting four bits to a single character.

What Can we do with it?

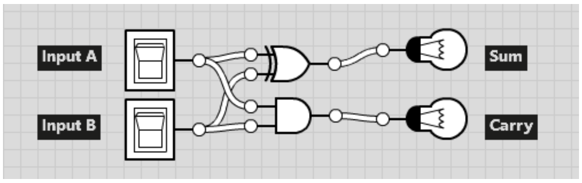
There are logical operations we can mimic with transistors that follow these patterns.

X	Y	X AND Y	X OR Y	X XOR Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

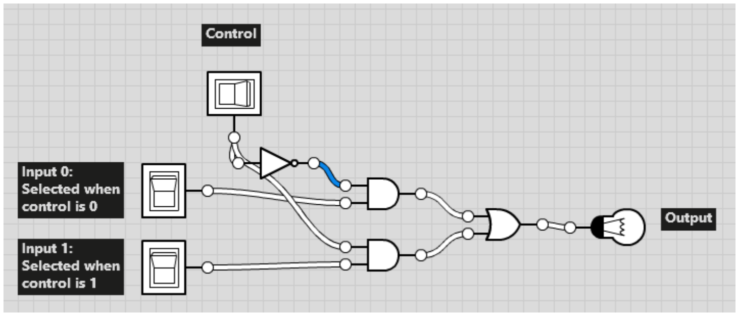


What Are some handy Tools?

A **half adder** takes two inputs and "adds" them, returning a sum bit and a carry bit.



A **simple multiplexer** "chooses" one of two inputs by another signal passed in called the control. There are multiplexers to choose between three, four, eight, and many larger numbers.


Control
Input 0: Selected when control is 0
Input 1: Selected when control is 1
Output

Using these tools, and some more complicated ones, it is possible to add, subtract, and perform logical operations on some given numbers. See today's Piazza post for more information.

Caching

A cache is a component that stores data so future requests for that data can be served faster.

Why Does Caching Work?

Spatial Locality: aka "If you use it, you'll probably use stuff around it."

Temporal Locality: aka "If you use it, you'll probably use it again soon."

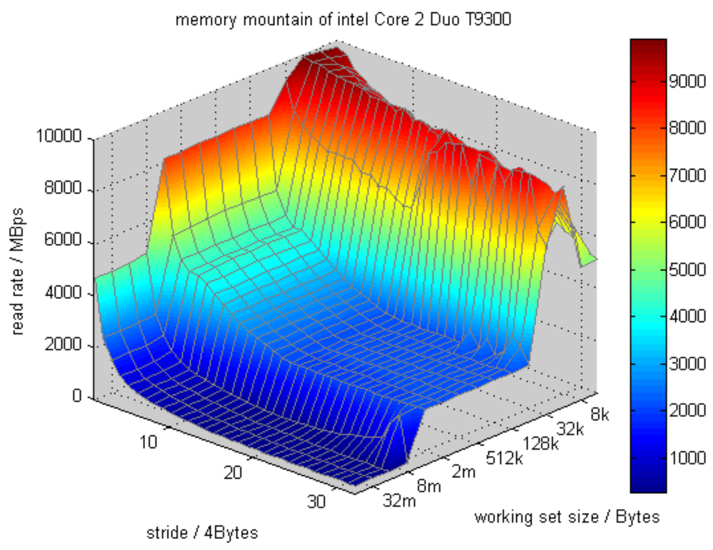
Hardware Implementation?

The main tradeoff in storage hardware is between speed and price. You can get really, really fast access - 3 to 4 clock cycles - but it's not cheap. If the whole cache was this kind of storage, then it'd be prohibitively expensive. Most computers you will encounter have levels of cache, usually about three. The faster levels have less space and the bigger levels are slower.

What happens in one level of a Cache?

The cache has "blocks," which store a lot of data at a time. The block information includes its address in main memory, when it was used last, and if it's changed since it was loaded. When the

CPU asks to load a certain part of memory, the cache checks to see if it has what the CPU is looking for. If it does, it fills that request fast. If not, the computer gets the memory (which takes a while) and replaces the oldest data in the cache with this new data. This design uses spatial locality because the blocks are relatively large - maybe 100 different numbers in an array are in a block. Also, the design uses temporal locality because the cache blocks that are used again and again stay in the cache, so addresses that are commonly accessed are found quickly.



What Are the Effects of Caching?

As a whole, the program is much, much faster than it normally could be. The important themes that you need to consider are all seen in the "Memory Mountain." (credit to Mentao1983 from Wikipedia). This graph plots stride, aka the distance between each part of memory you're using, and working set size, aka the space in memory you're looping over many times, against the read rate (higher is better.)

Starting at the bottom right and moving to the upper left, you can see the slopes of spatial locality. As the memory accesses get closer and closer, the speed of reading improves. The new memory you're looking to read is more likely to be in the cache.

Starting at the bottom left and moving to the upper right, you can see the ridges of temporal locality. As the memory you're working with gets smaller, it can fit into smaller caches, improving the latency. Otherwise, the cache is filled with a bunch of entries you used recently, but won't use again until they get kicked out.

Space for Notes.