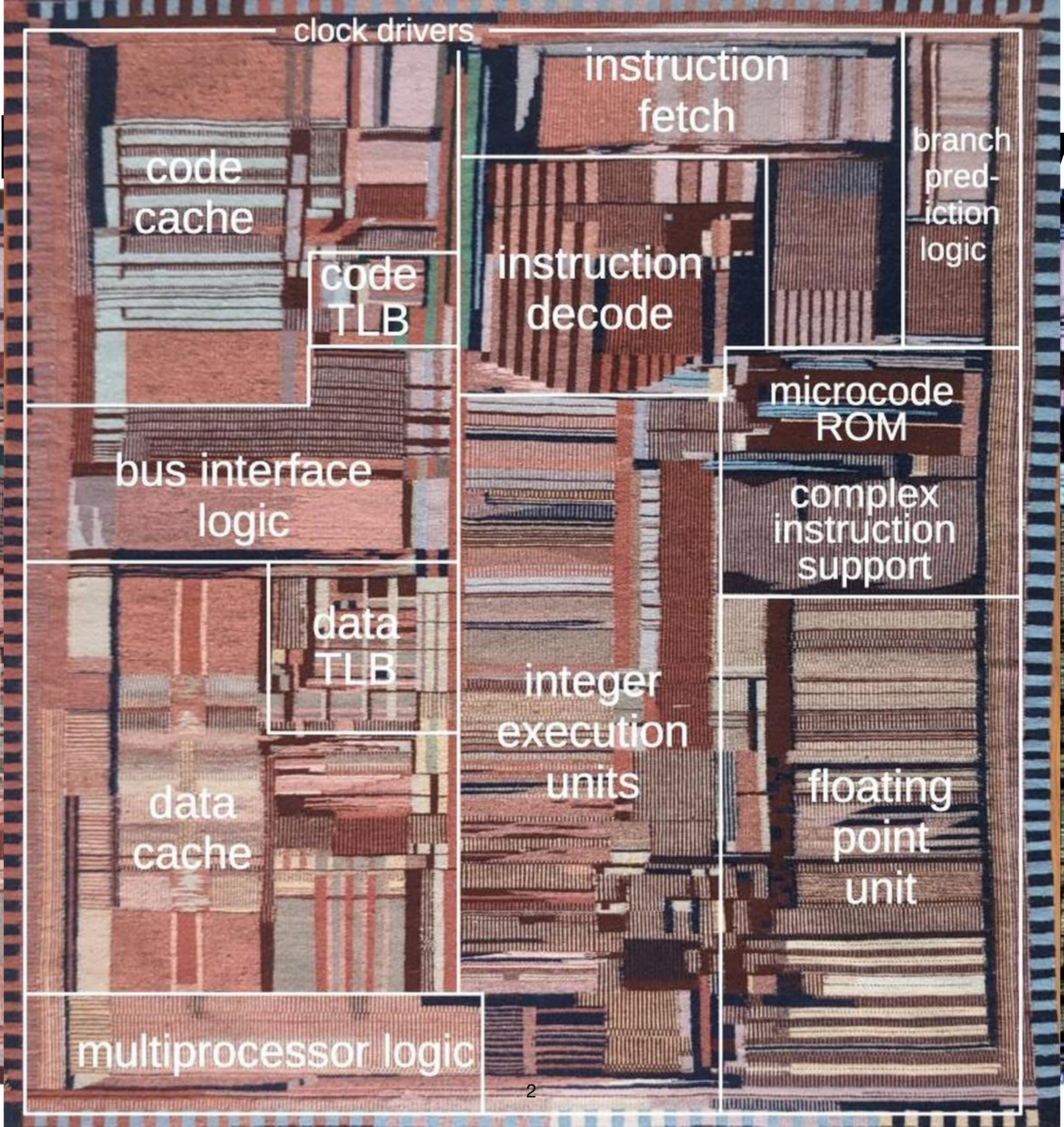
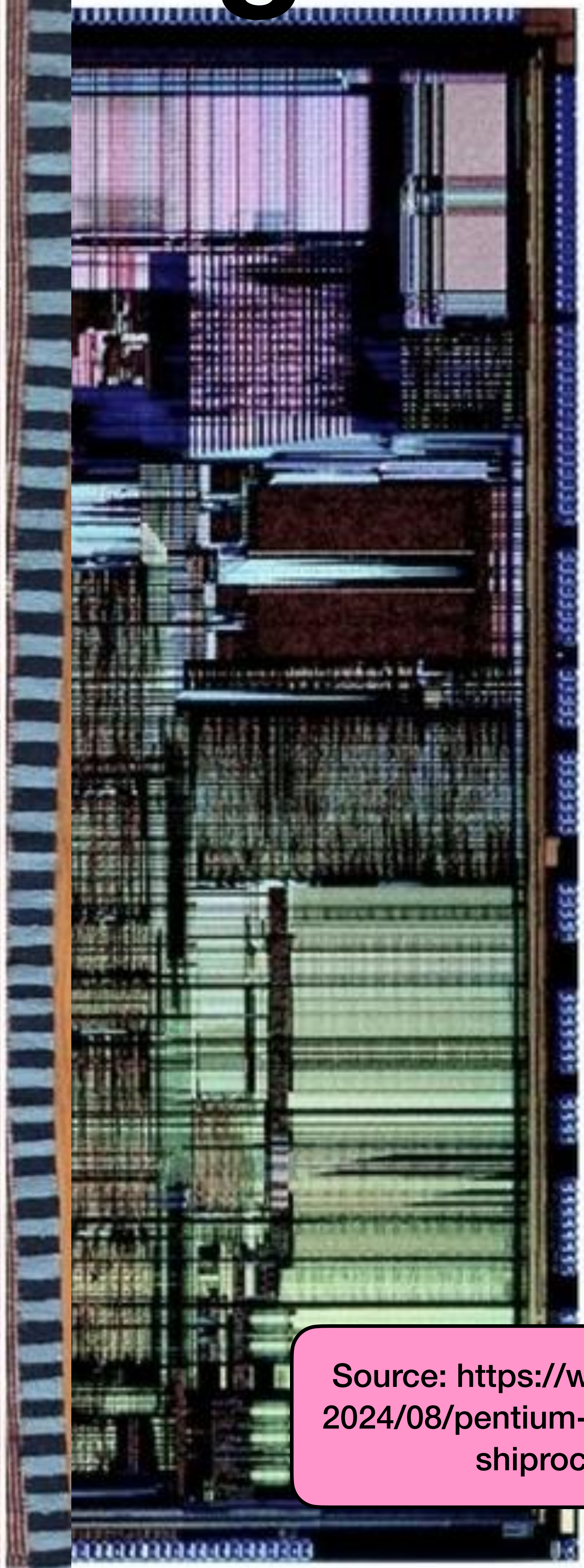
Hardware Overview

Lab Tonight: Homework 1 Gear-Up Session

The IBM ving





Source: https://www.righto.com/2024/08/pentium-navajo-fairchild-shiprock.html

Outline

- Hardware Assumptions
- A Very Brief Introduction to Logical Operations and Combinational Logic
- Constructing Processor Elements from Integrated Logic
- Computer Architecture Design Principles (and whether they are a good idea!)

Today may be split into two lessons!

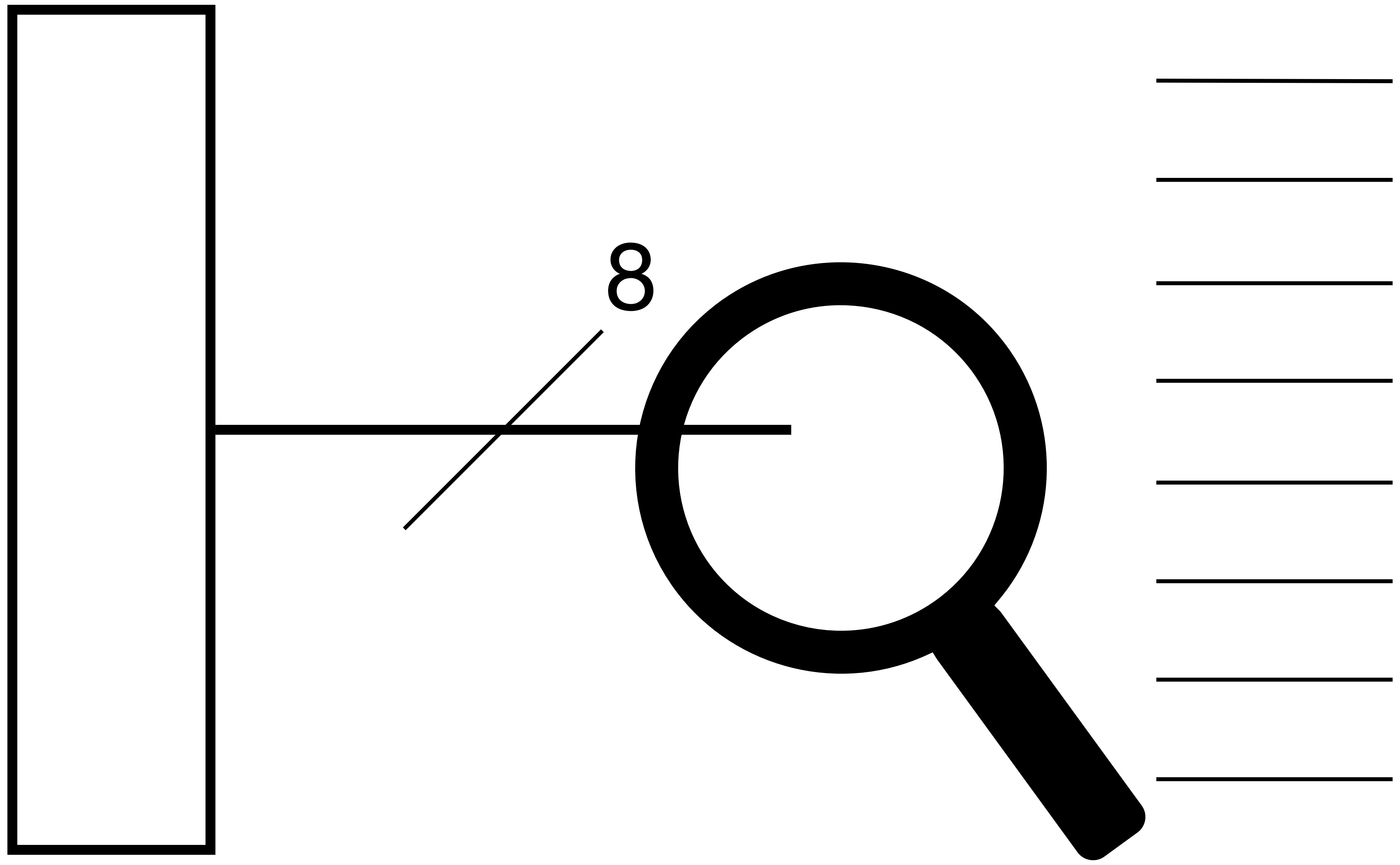
Assumptions

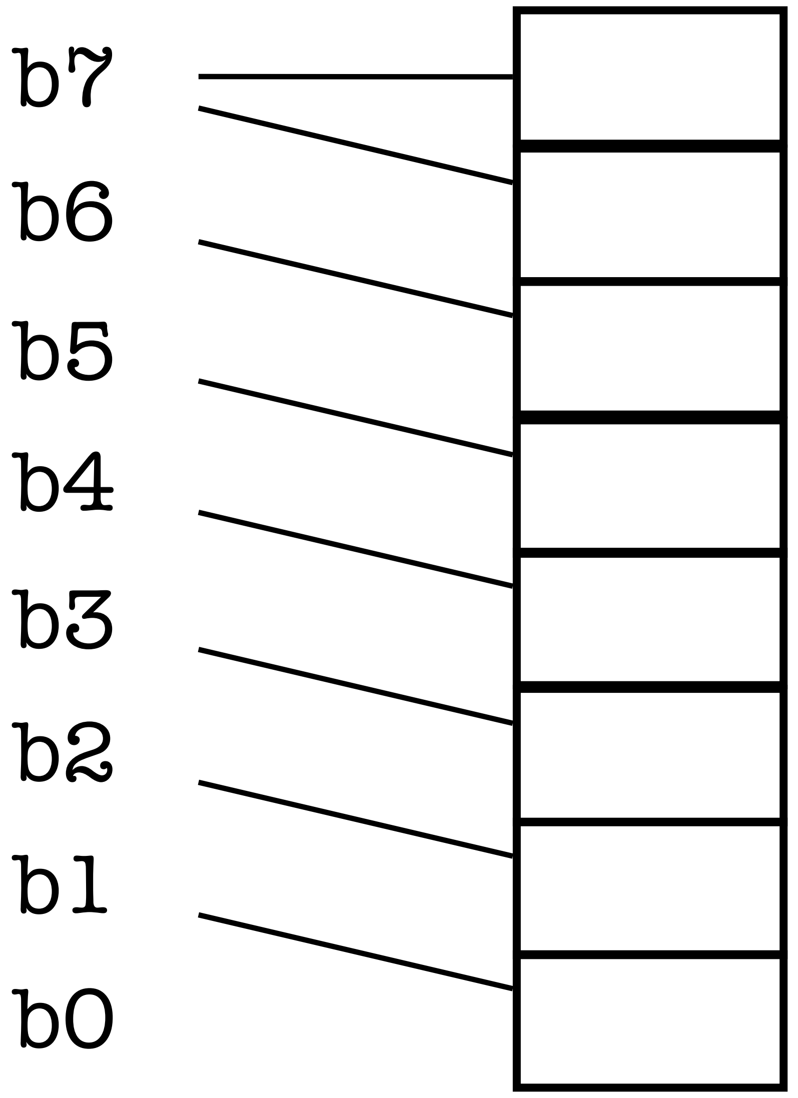
- CPU can read bits from memory as electrical signals (one “wire” per bit)
- Everything is a pure low/high signal, we don’t have to worry about noise or interference
- For now, we aren’t worried about constraints (space, power, complexity, heat, etc)
- Every “step” (to be defined) leaves enough time for a circuit to stabilize

None of these are realistic assumptions in practice, but we have to use a certain degree of abstraction to talk about the cool stuff!

Data as Collections of Wires

- Wire/Data Line: carries a single digit signal (on/off)
- Bus (from textbook): a collection of data lines that is treated as one, multi-bit signal





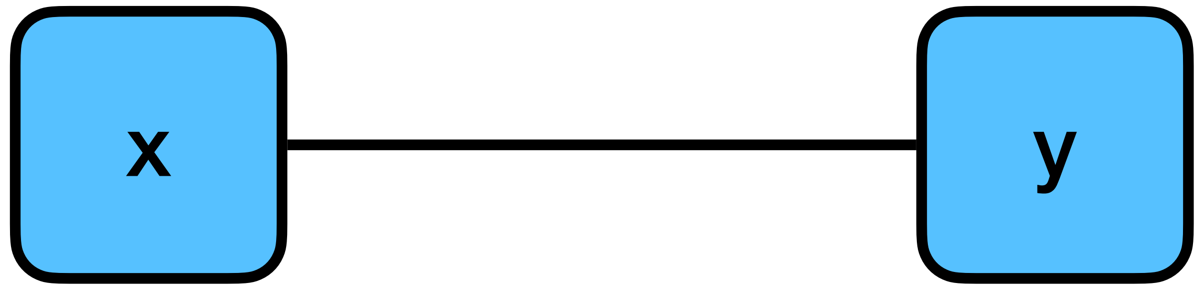
Which operation is this?

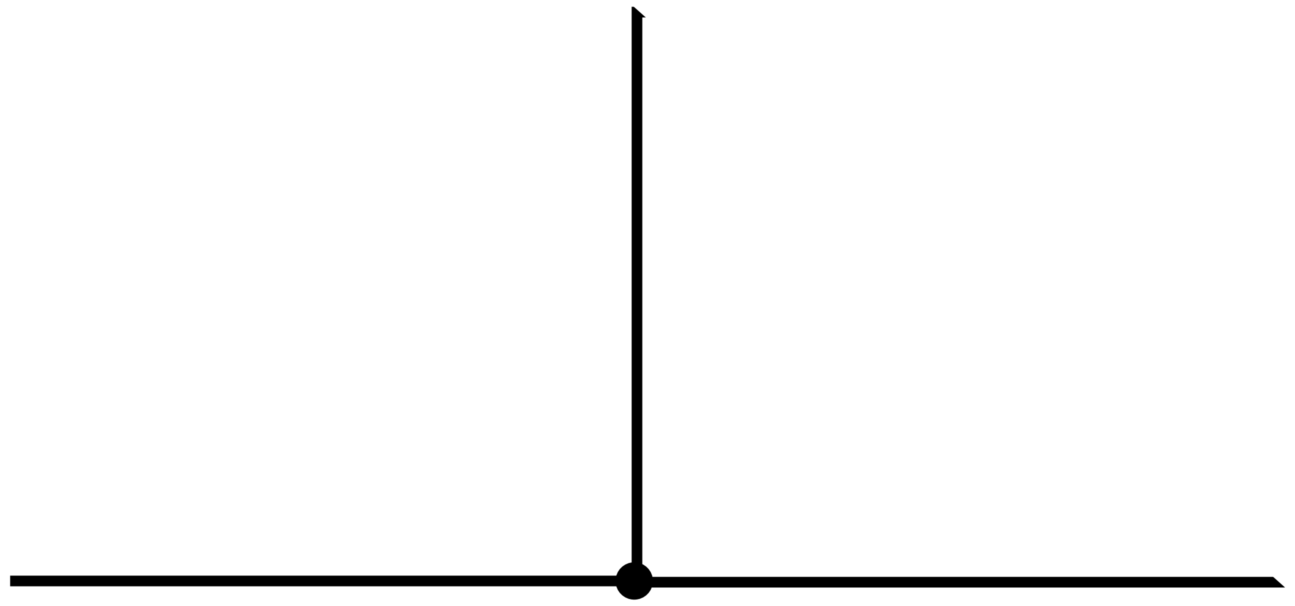
Shift right arithmetic!

Wires

- Can communicate state of charge (high or low) between two gates/storage circuits

- Signal can be split to multiple components





Combinational Logic Circuits

- Take two inputs and produce an output as a “pure function” (no memory involvement)
- Combinatorial expressions can be synthesized to *circuits*
- Physically, logic gates are implemented using *transistors* (electrical switches)
- Examples: adders, logical operators, control signal translations

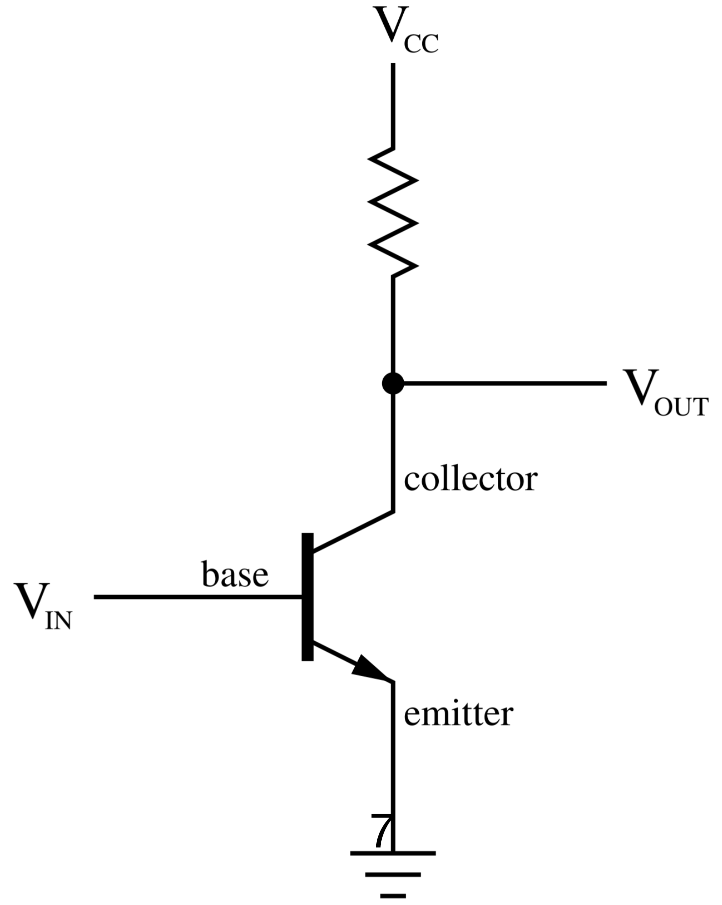
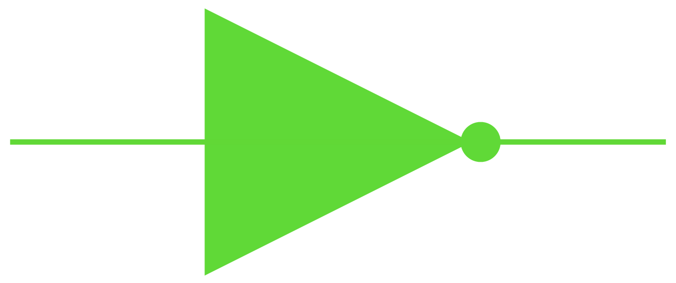


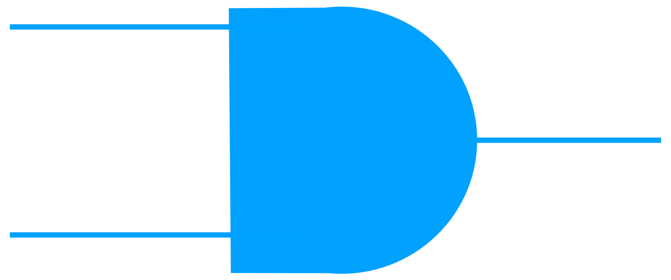
Image credit: https://en.wikipedia.org/wiki/Transistor#/media/File:Transistor_Simple_Circuit_Diagram_with_NPN_Labels.svg

Combinational Logic Circuits



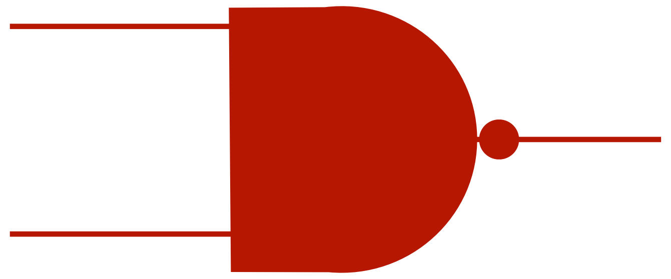
NOT

	in (0)	**in (1)**
out	1	0



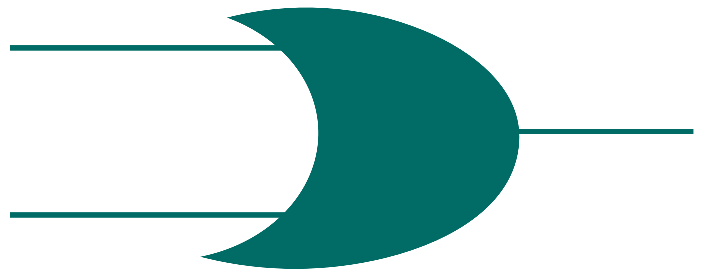
AND

	in (0)	**in (1)**
in (0)	0	0
in (1)	0	1



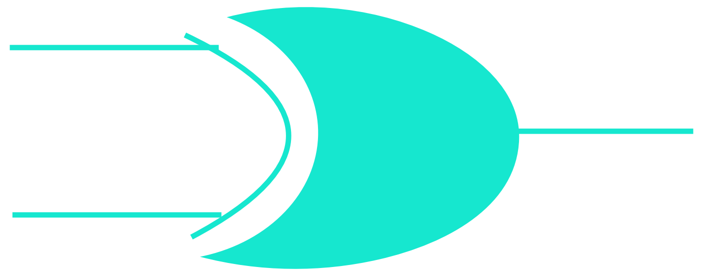
NAND

	in (0)	**in (1)**
in (0)	1	1
in (1)	1	0



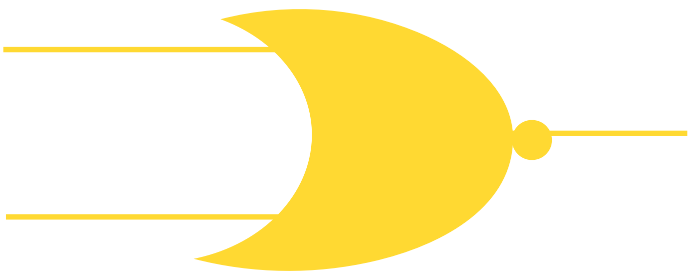
OR

	in (0)	**in (1)**
in (0)	0	1
in (1)	1	1



XOR

	in (0)	**in (1)**
in (0)	0	1
in (1)	1	0

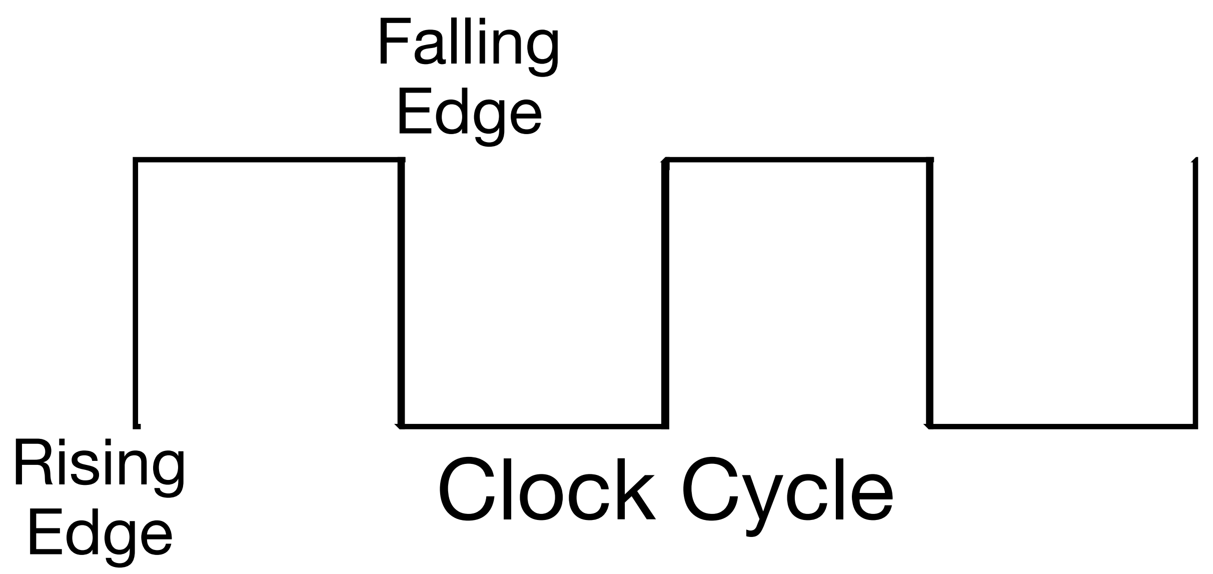


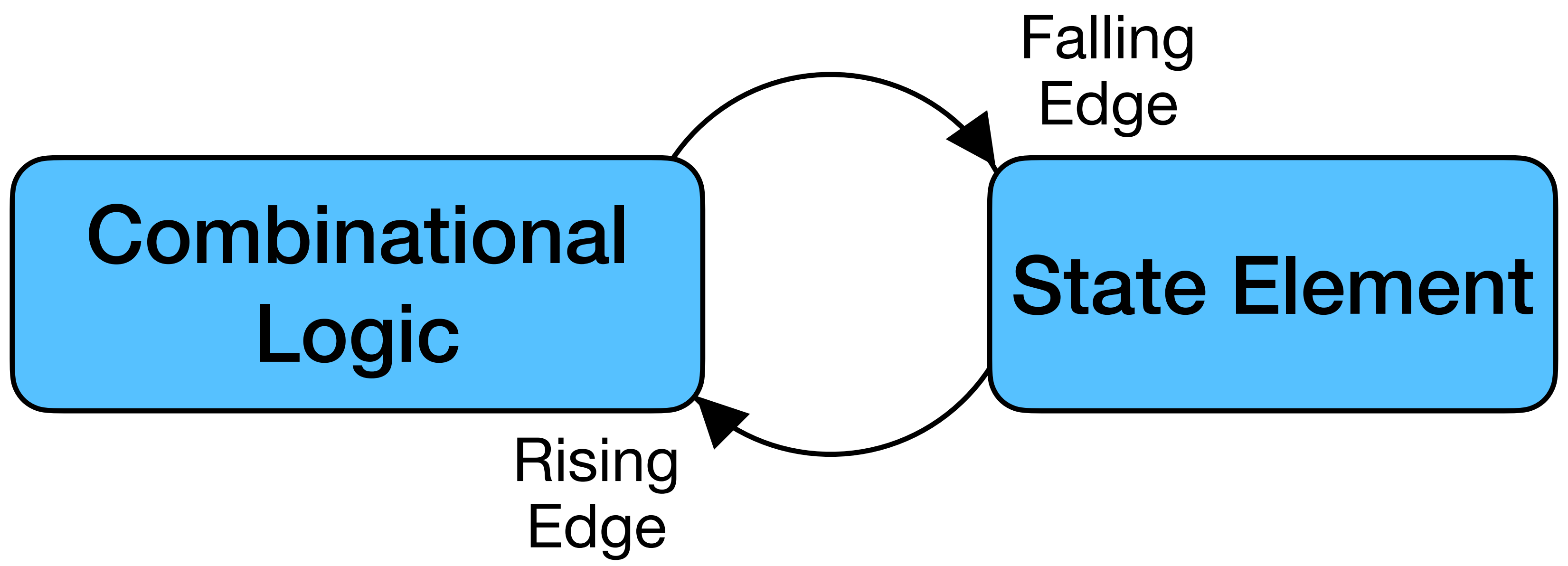
NOR

	in (0)	**in (1)**
in (0)	1	0
in (1)	0	0

Components with State

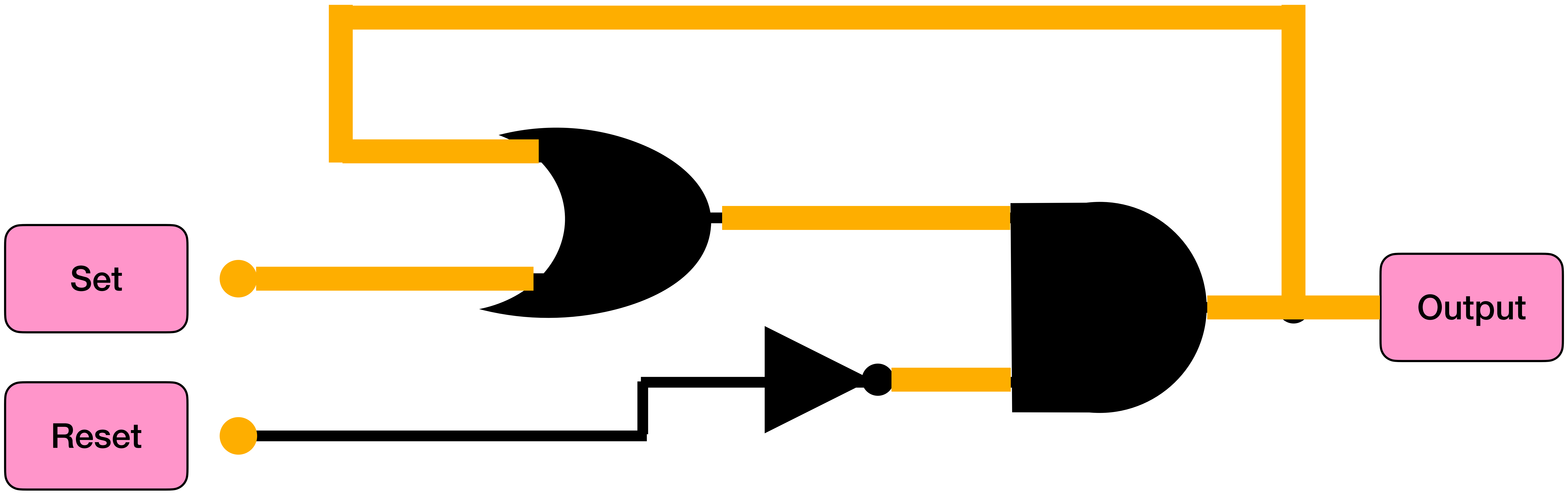
- How do we express “at each step, increment the PC by 4”?
- Need a *clock cycle* to control when state changes
- “Memory elements” (flip flops and latches) have internal state
- Update state on clock downtick





Components with State

Set-Reset Flip-Flop

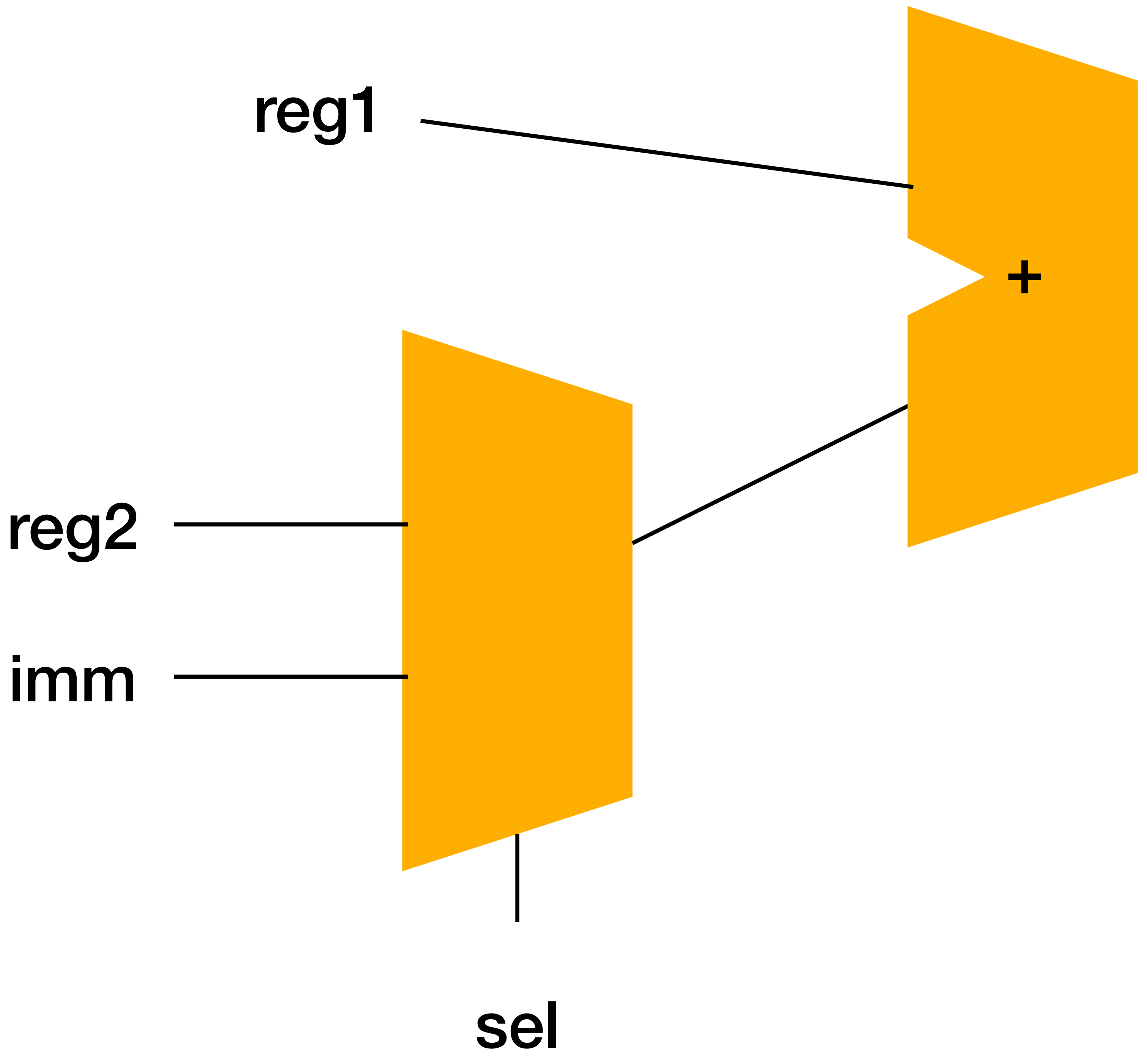


Chat with your neighbor(s)!
What happens next?

Image source: https://en.wikipedia.org/wiki/Flip-flop_(electronics)

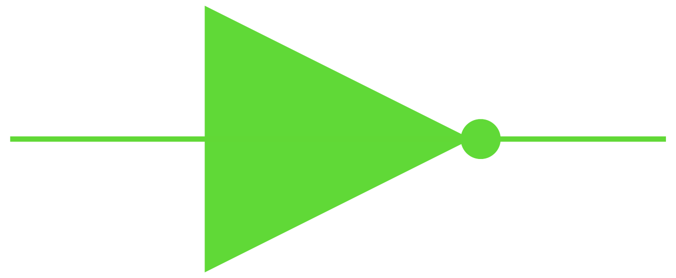
Multiplexers

- “Hardware if-statements”
- Select between multiple inputs
- Example: choose the second operand for `add` and `addi`

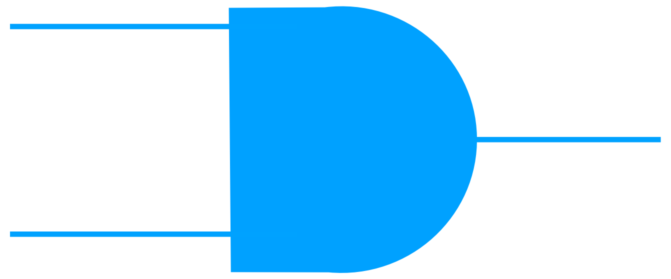


Chat with your neighbor(s)!

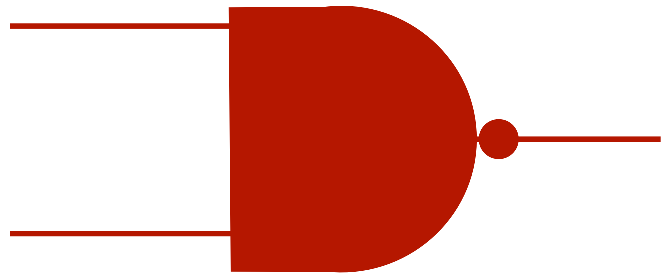
Build a two-input (1-bit selector) MUX out of logic gates!



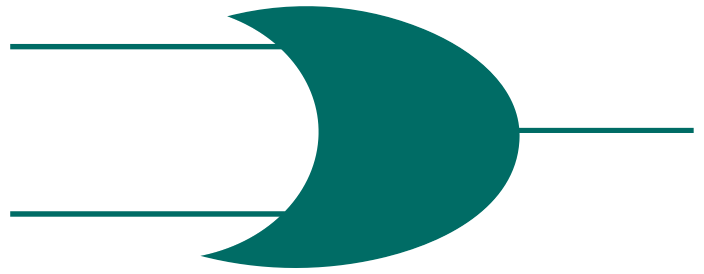
NOT



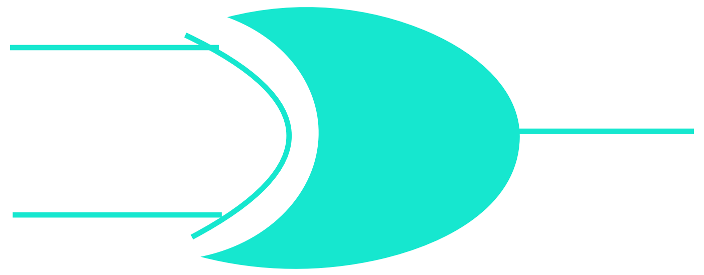
AND



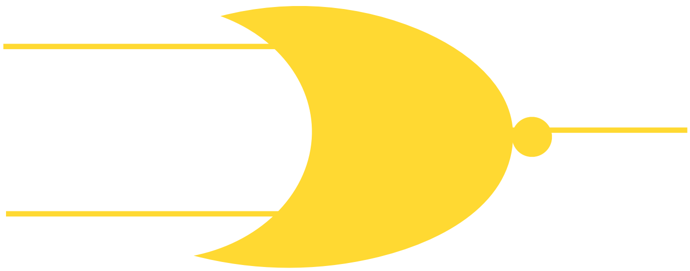
NAND



OR



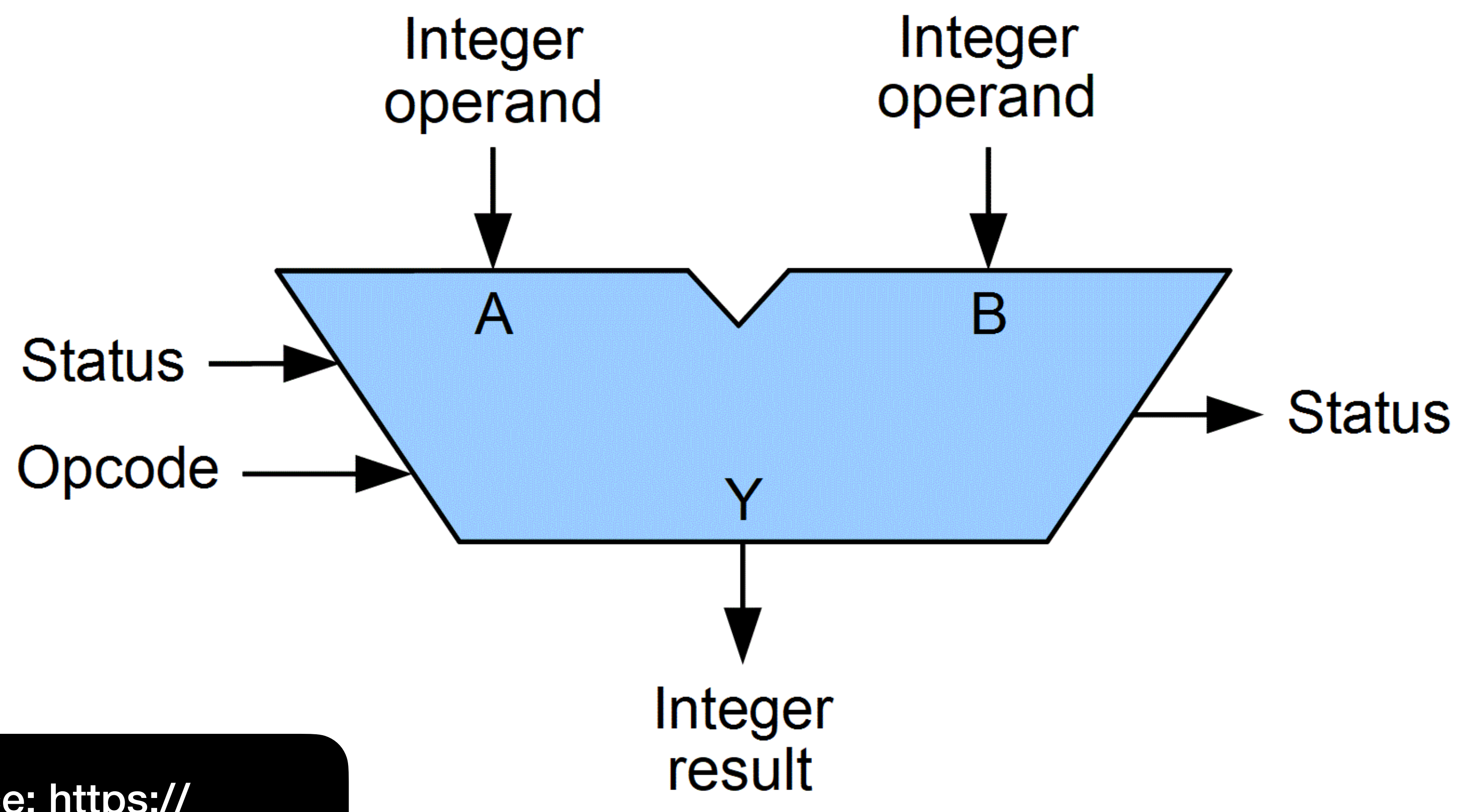
XOR



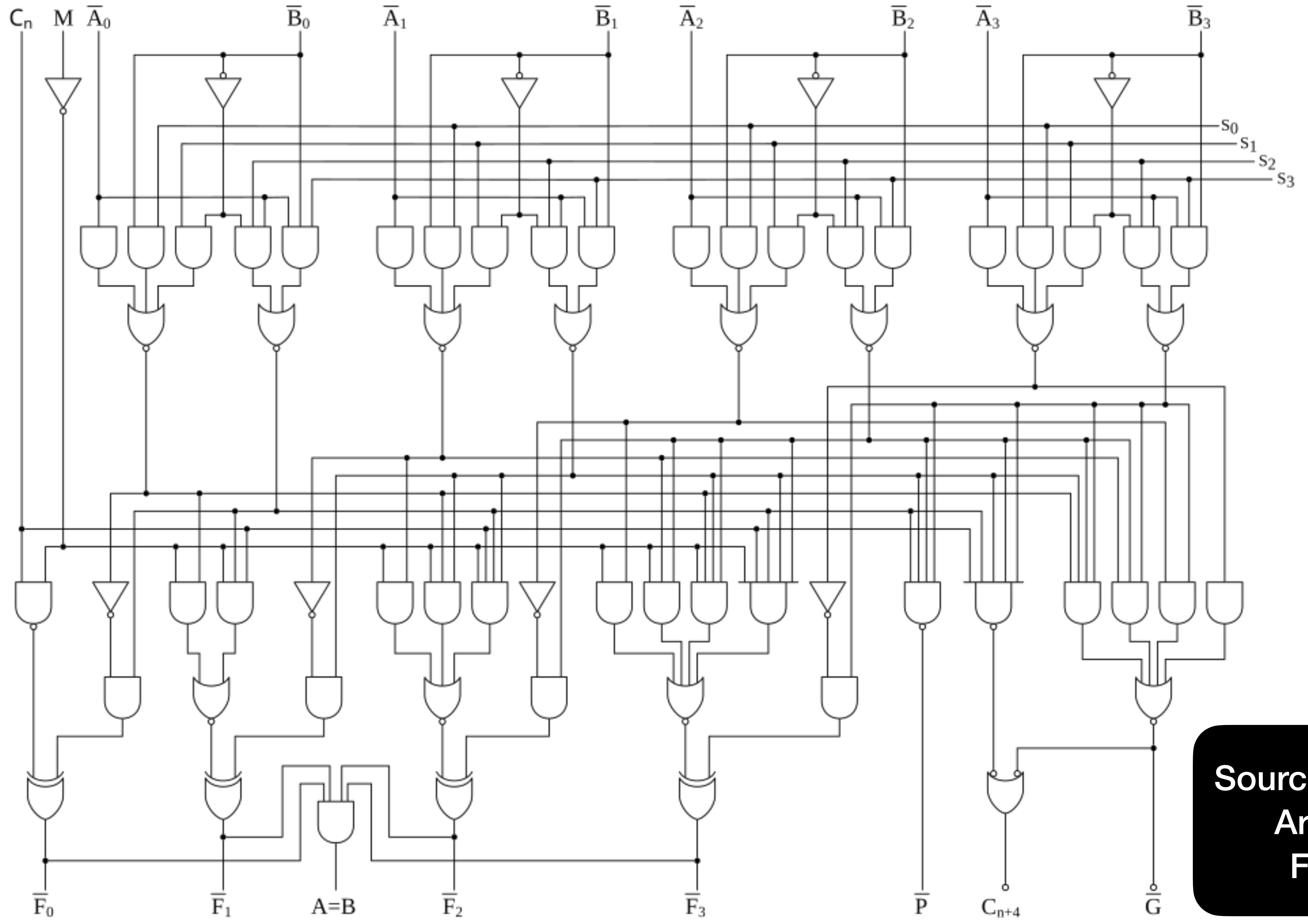
NOR

Arithmetic Logic Unit (ALU)

- Takes in two operands and a control signal for the operation
- Produces result of applying operation on operands (status input/output signals optional)



Source: https://commons.wikimedia.org/w/index.php?curid=36975996



Source: https://en.wikipedia.org/wiki/Arithmetic_logic_unit#/media/File:74181aluschematic.png

Logical Operators Summary

- Bits of information = electrical signals
- CPU is just a (very) big circuit made up of wires, combinational logic elements, and memory elements
- Can implement needed components (multiplexers, ALUs, bit selectors, registers, etc) using these elements
- NAND is Turing Complete!

Takeaway: we have an "existence proof" of the hardware we need, so we can start working one level of abstraction higher to implement a CPU

Design Principles

- "Eight Great Ideas of Computer Architecture" — P&H
  - Design for Moore's Law
  - Use abstraction to simplify design
  - Make common case fast
  - Performance via parallelism
  - Performance via pipelining
  - Performance via prediction
  - Hierarchy of memories
  - Dependability via redundancy

Moore’s Law

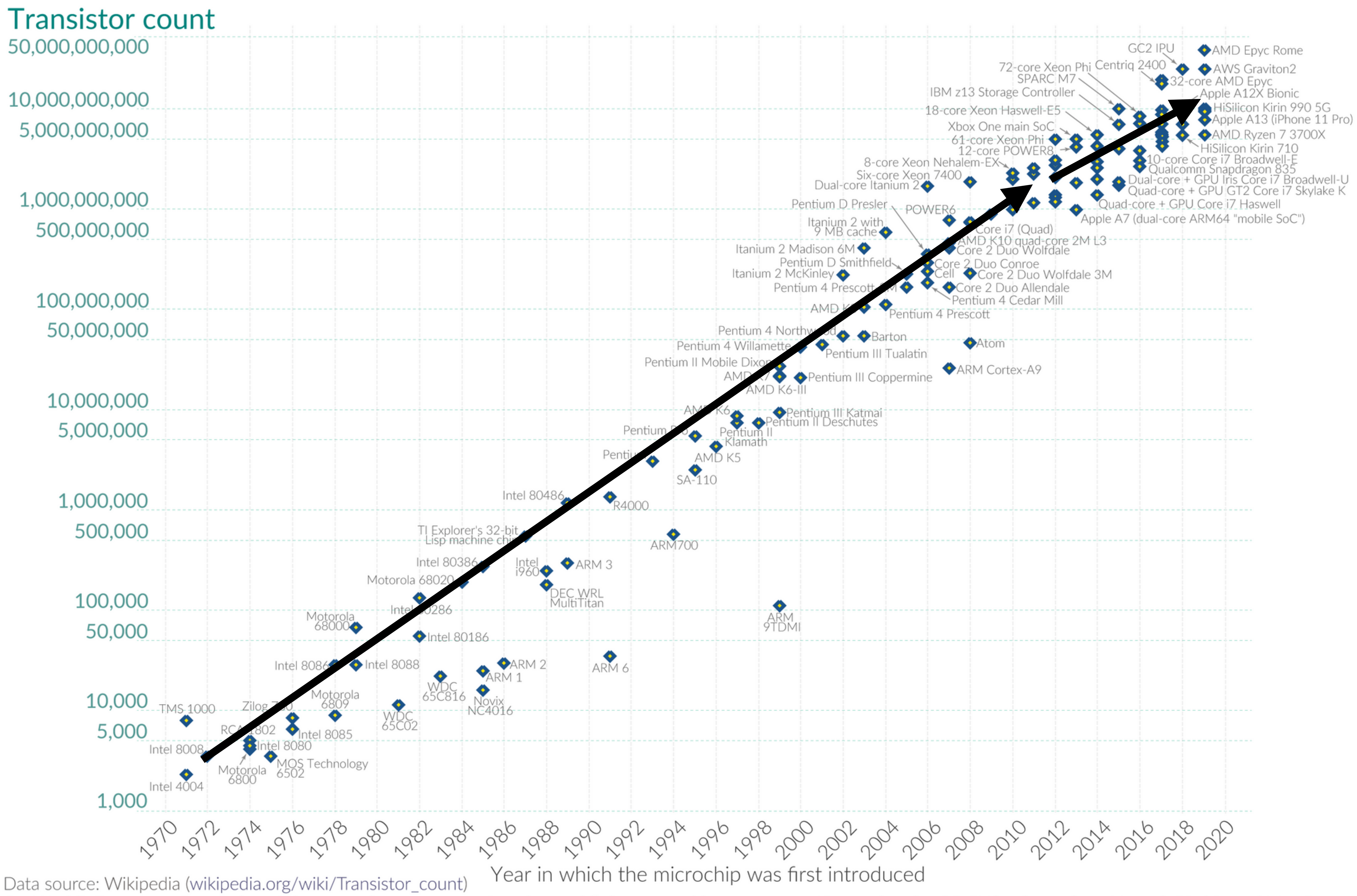
More ICs ➡️ more complex logic on the processor

Denser ICs ➡️ bigger and faster memories

Moore’s Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Our World in Data



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

OurWorldinData.org – Research and data to make progress against the world’s largest problems.

Design Principles

- “Eight Great Ideas of Computer Architecture” — P&H
  - Design for Moore’s Law
  - Use abstraction to simplify design
  - Make common case fast
  - Performance via parallelism
  - Performance via pipelining
  - Performance via prediction
  - Hierarchy of memories
  - Dependability via redundancy

🤔

It is important to challenge the conventions of the well understood assumptions!