From transistor to iPhone… (Hardware)

Plan for next few labs

- Lab 1 Session (1/16): Setup WSL/MacOS/Linux + Python + Jupyter
    - We will check you on Wednesday.
- Lab 2 Assignment:
    - Given out Monday
    - Due Friday 1/30 11:59pm
- Lab 2 Session (1/23):
    - Working session. Lab 3 given out (Due 2/6)
    - I'll give you a tour of Unix Shell.
- Lab 3 Session (1/30): GitHub + Get setup for submission. Submit Lab 2.

Abstraction

- Definitions:
    - the quality of dealing with ideas rather than events
    - the process of considering something independently of its associations, attributes, or concrete accompaniments.
- Usage: (wikipedia)
    - *Computer scientists* use abstraction to make models that can be used and re-used without having to re-write all the program code for each new application on every different type of computer.
    - Abstraction in *mathematics* is the process of extracting the underlying essence of a mathematical concept, removing any dependence on real world objects with which it might originally have been connected, and generalizing it so that it has wider applications or matching among other abstract descriptions of equivalent phenomena.
- Why am I bring this up? Important skill
    - Comfortably think/work above the abstraction barrier… → play!
    - Sufficiently (usually qualitatively) understand the underlying dynamics… → use!
    - Peek below when necessary…
- Examples:
    - Programming
    - Computers, Software, Libraries
    - Algorithms, Machine Learning, Deep Learning
    - Domains

Abstraction Example

- Imagine you wish teach a child to cook mac-and-cheese
    - the recipe on the left would not be sufficient
- Requires how to:
    - Turn on a stove … boil water … melt butter, …
        - → These are tasks that each have several steps that a child would not *a priori* know…
        - and most adults have learned.
- These are levels of abstraction that the recipe assumes.
- In order to teach someone to cook, you start with basic skills and build on them → chef.
- Now imagine trying to teach a robot…

Simple Macaroni and Cheese

allrecipes

Prep 10 m | Cook 20 m | Ready In 30 m

Tom Thumb Food & Pharmacy
1701 W Randol Mill Rd
ARLINGTON, TX 76012

Recipe By: g0dluvsugly

"A very quick and easy fix to a tasty side-dish. Fancy, designer mac and cheese often costs forty or fifty dollars to prepare when you have so many exotic and expensive cheeses, but they aren't always the best tasting. This recipe is cheap and tasty."

Tillamook Farmstyle Cut Shredded Sharp Cheddar Cheese 8 Oz
$6.00 for 2 item - expires in a month

Ingredients

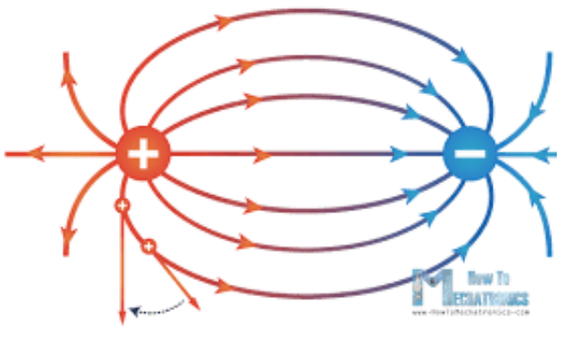
1 (8 ounce) box elbow macaroni
1/4 cup butter
1/4 cup all-purpose flour
1/2 teaspoon salt
ground black pepper to taste
2 cups milk
2 cups shredded Cheddar cheese

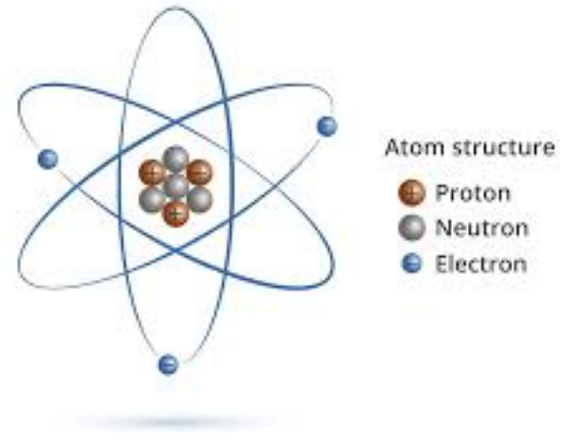
Directions

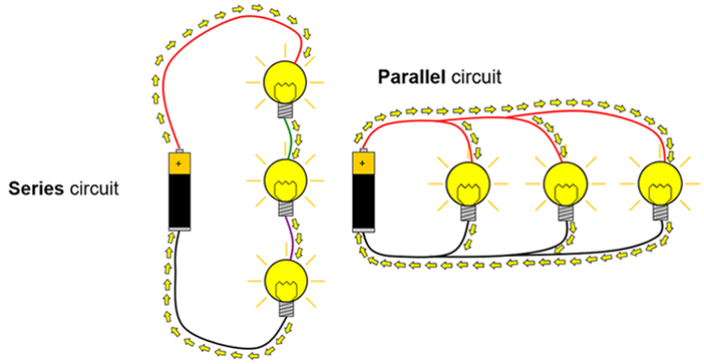
1. Bring a large pot of lightly salted water to a boil. Cook elbow macaroni in the boiling water, stirring occasionally until cooked through but firm to the bite, 8 minutes. Drain.
2. Melt butter in a saucepan over medium heat; stir in flour, salt, and pepper until smooth, about 5 minutes. Slowly pour milk into butter-flour mixture while continuously stirring until mixture is smooth and bubbling, about 5 minutes. Add Cheddar cheese to milk mixture and stir until cheese is melted, 2 to 4 minutes.
3. Fold macaroni into cheese sauce until coated.

Electrodynamics

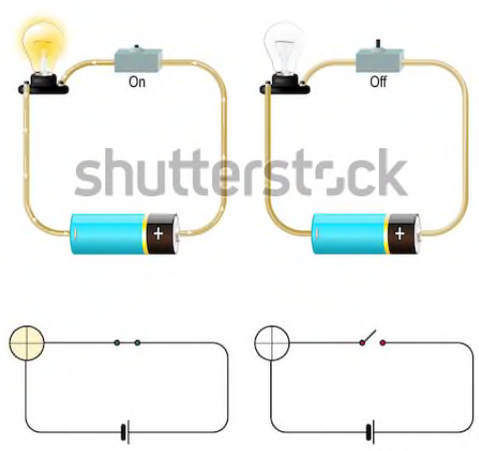
- One of the fundamental forces of nature is ***Electricity and Magnetism***.
    - What keeps Atoms together.
    - Responsible for all ***Chemistry***.
    - Foundation of ***Electronics***.
        - Electrons have an ***electric charge***.
        - Produces an ***Electric Field***.
        - Feel a force when they experience another ***Electric Field.***
- Charges move (a ***Current***) freely in conductors
    - Wires serves as electron pipes… like water pipers.
    - Electric potential (***Voltage***) ~ pressure in pipe. Moves electrons around.
    - Other components: ***capacitors, resistors, inductors, transistors***…
- ***Digital Electronics***: (Abstraction)
    - “0”: No electrons flowing.
    - “1”: Electrons flowing.
    - “Clock”: electrons are pushed through system at a regular rate.







Simple electric circuit



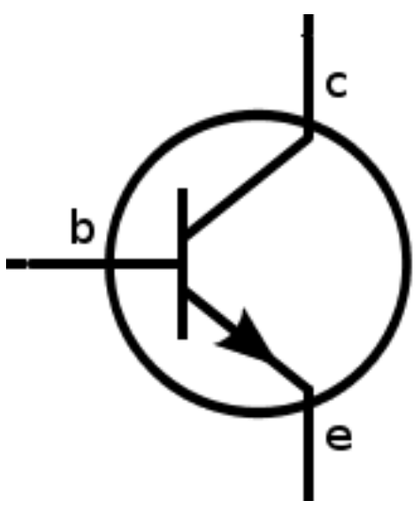
Transistor

- *Base:* The base is responsible for controlling whether current is allowed to flow through the transistor when power is applied.

- *Collector:* When there is power to the base, the collector current is allowed to flow towards the emitter.

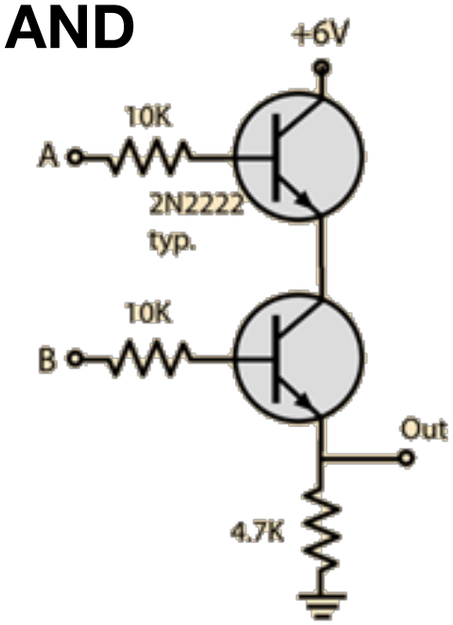
- *Emitter:* The emitter takes the electric current that the collector is allowed to send, to be used on other parts of your circuit.

- → A digital switch

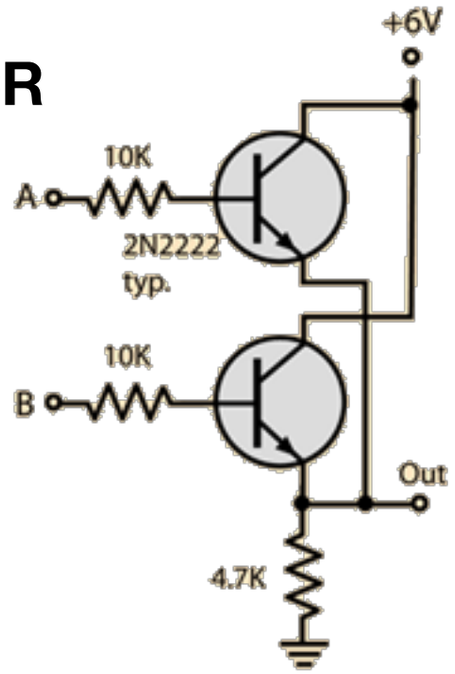


Logic Gates

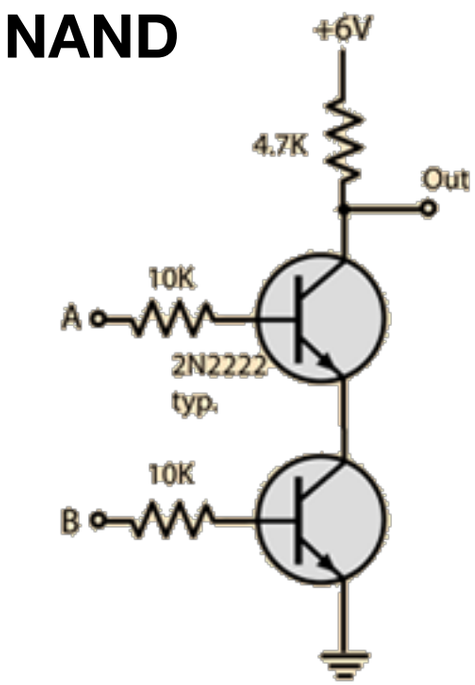
AND



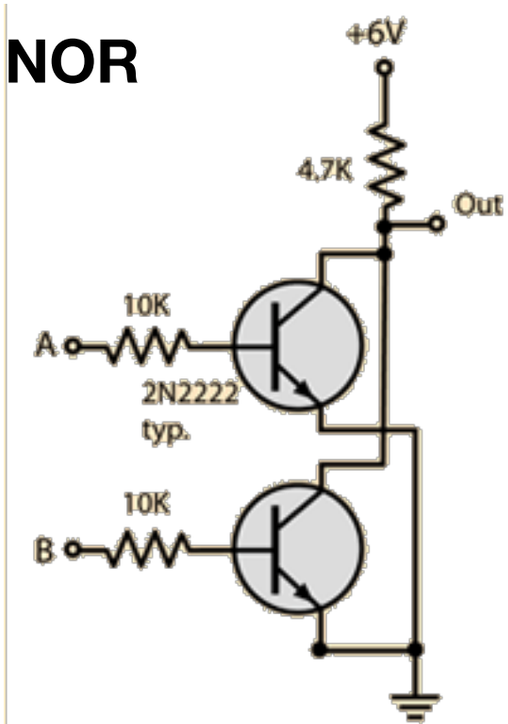
OR



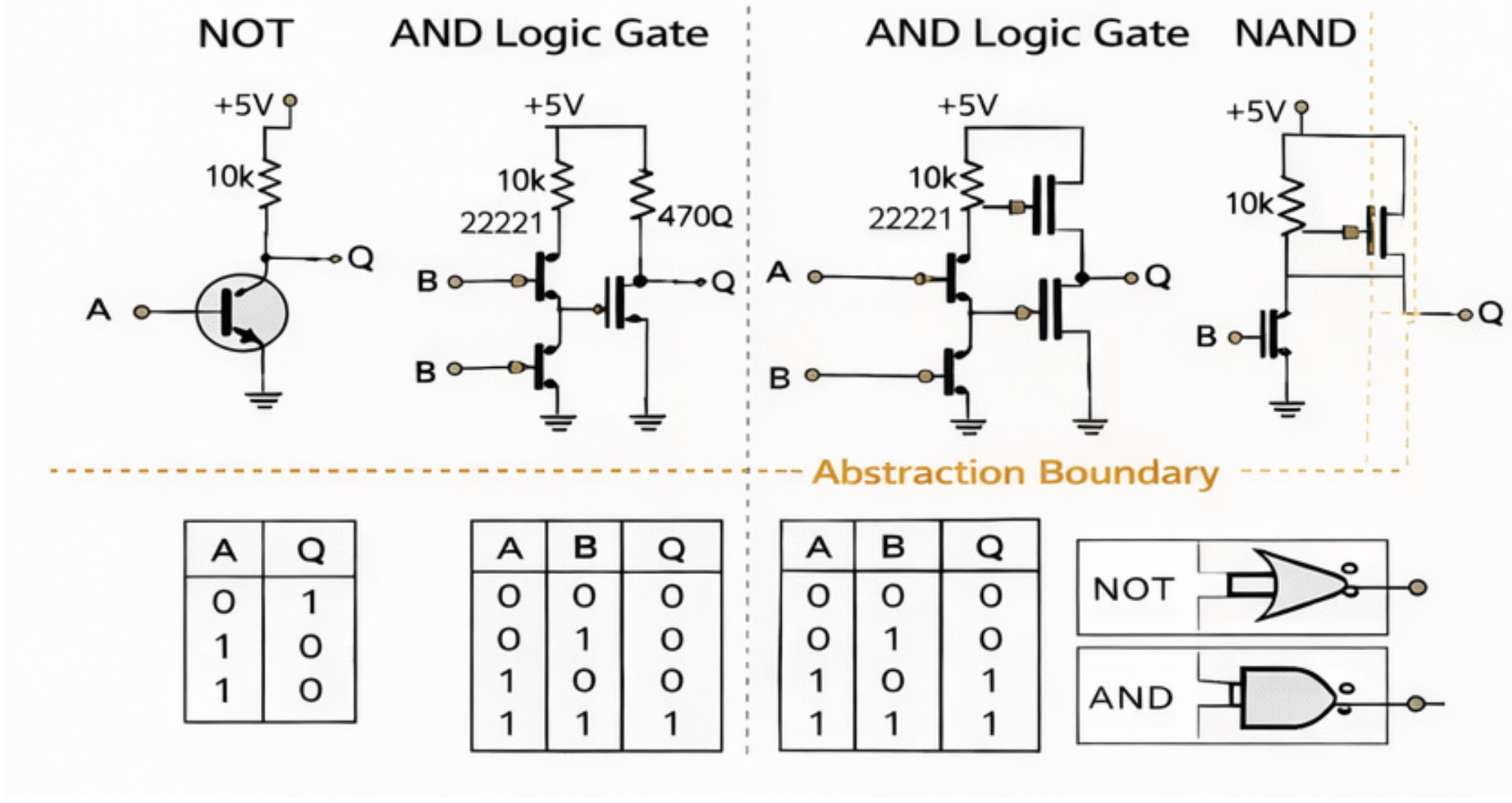
NAND



NOR



Logic Gates and Transistor Abstraction Boundary

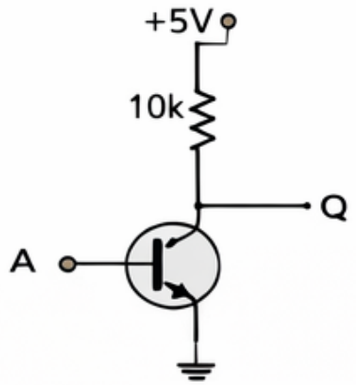


A	Q
0	1
1	0
1	0

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

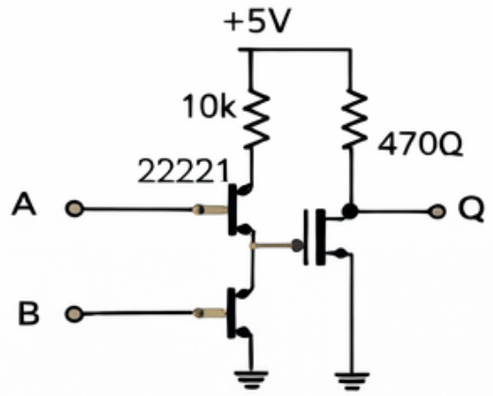
A	B	Q
0	0	0
0	1	0
1	0	1
1	1	1

NOT Logic Gate



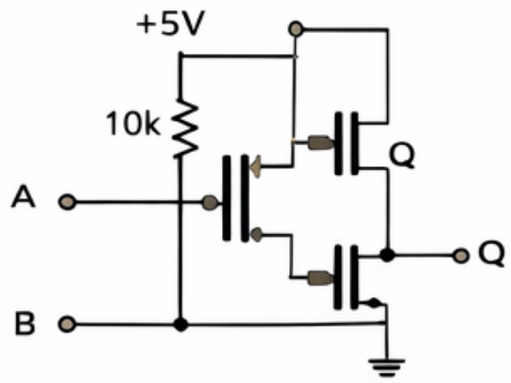
A	Q
0	1
1	0

AND Logic Gate



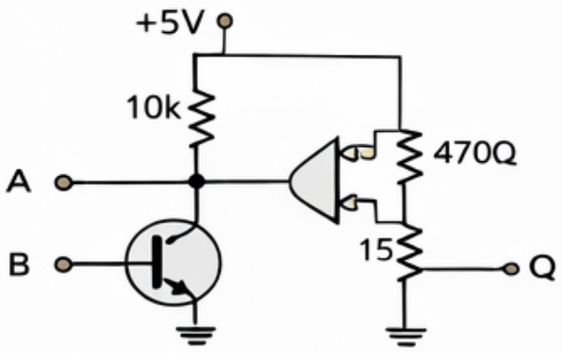
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

OR Logic Gate



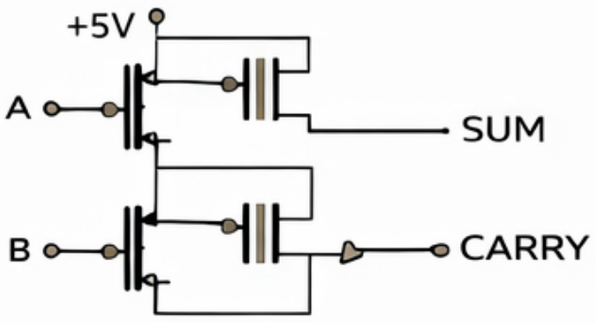
A	B	Q
0	0	0
0	1	0
1	0	1
1	1	1

NAND Logic Gate

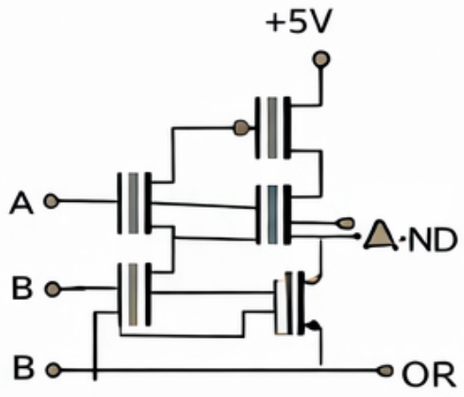


A	B	Q
0	0	1
0	1	0
1	0	1
1	1	0

Half Adder

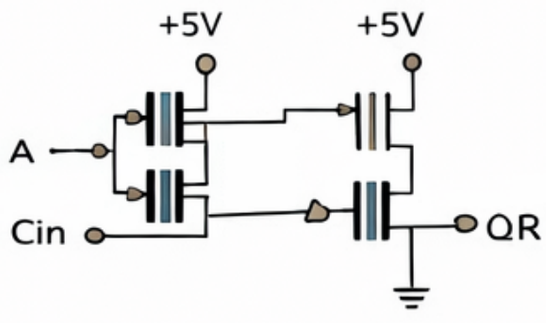


A	B	SUM	SUM	CARRY
0	0	0	0	0
0	1	1	1	0
1	0	1	1	0
1	1	0	0	1

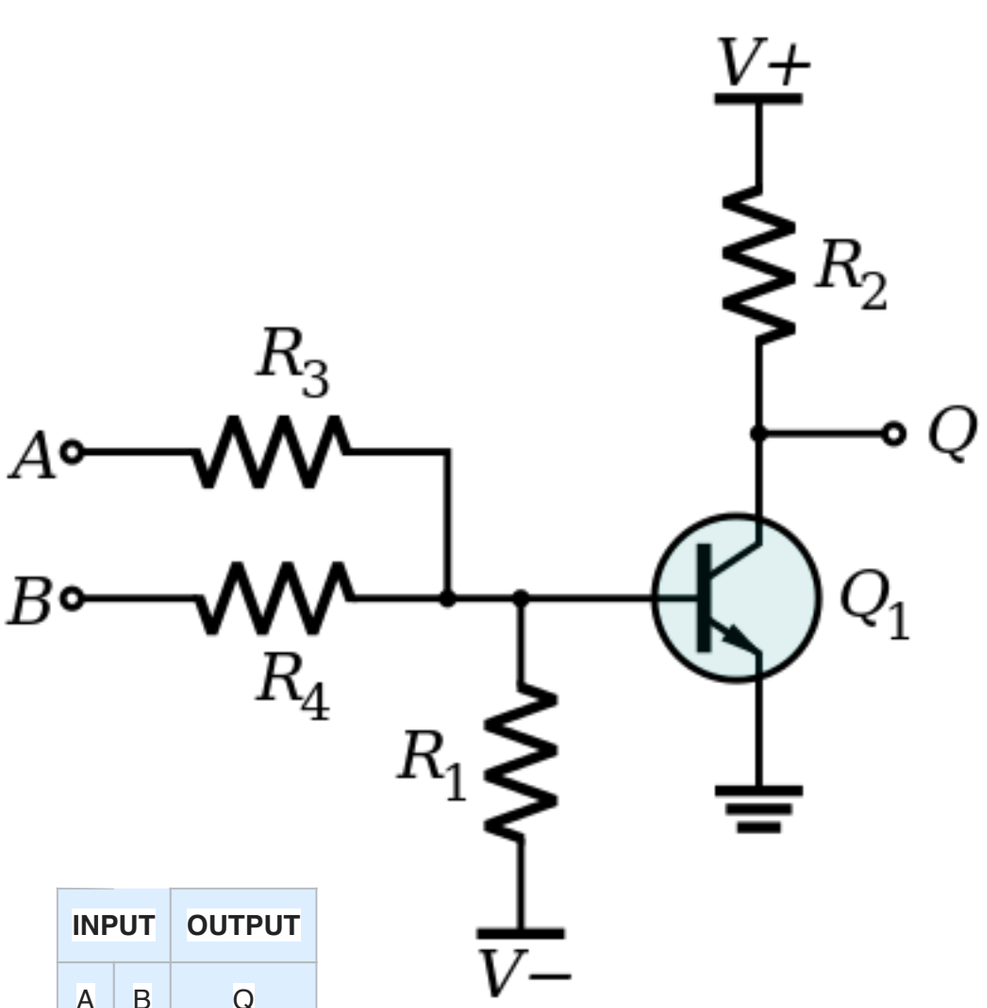


A	B	Cin	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
1	0	0	1	0
1	1	0	1	0
1	1	1	0	1
1	1	1	0	1
1	1	1	1	1

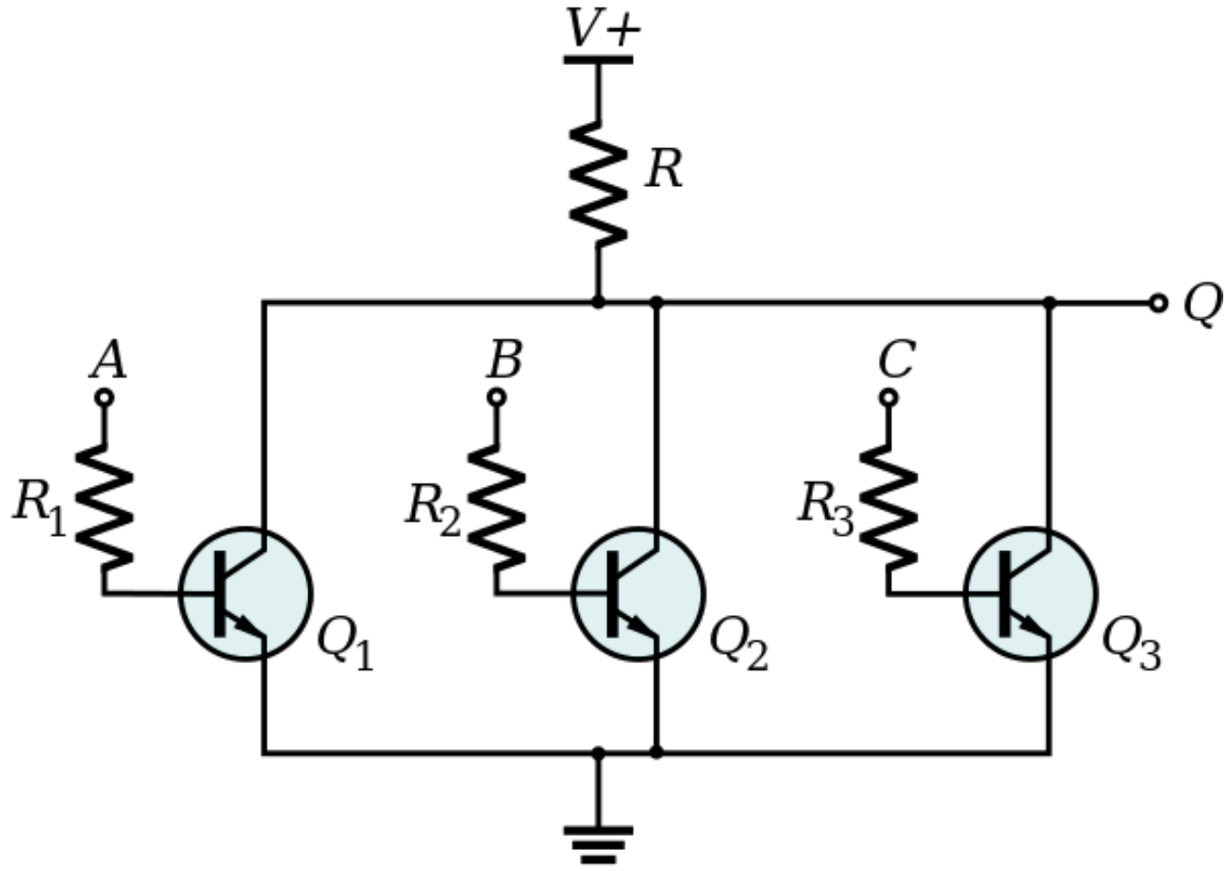
Full Adder



A	B	Cin	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	0	0	1
1	1	1	1	1



INPUT		OUTPUT
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



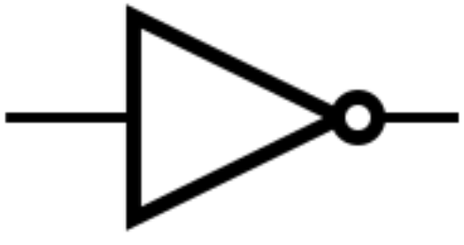
INPUT			OUTPUT
A	B	C	Q
0	0	0	1
0	1	0	0
1	0	0	0
1	1	0	0

INPUT			OUTPUT
A	B	C	Q
0	0	1	0
0	1	1	0
1	0	1	0
1	1	1	0

Boolean Operations

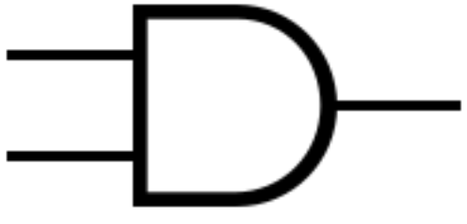
Abstraction

NOT



INPUT	OUTPUT
A	NOT A
0	1
1	0

AND



INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

NAND

NPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

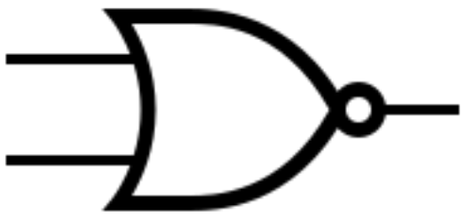
OR

INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

XOR

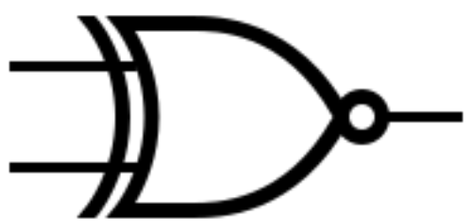
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

NOR



INPUT		OUTPUT
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

XNOR



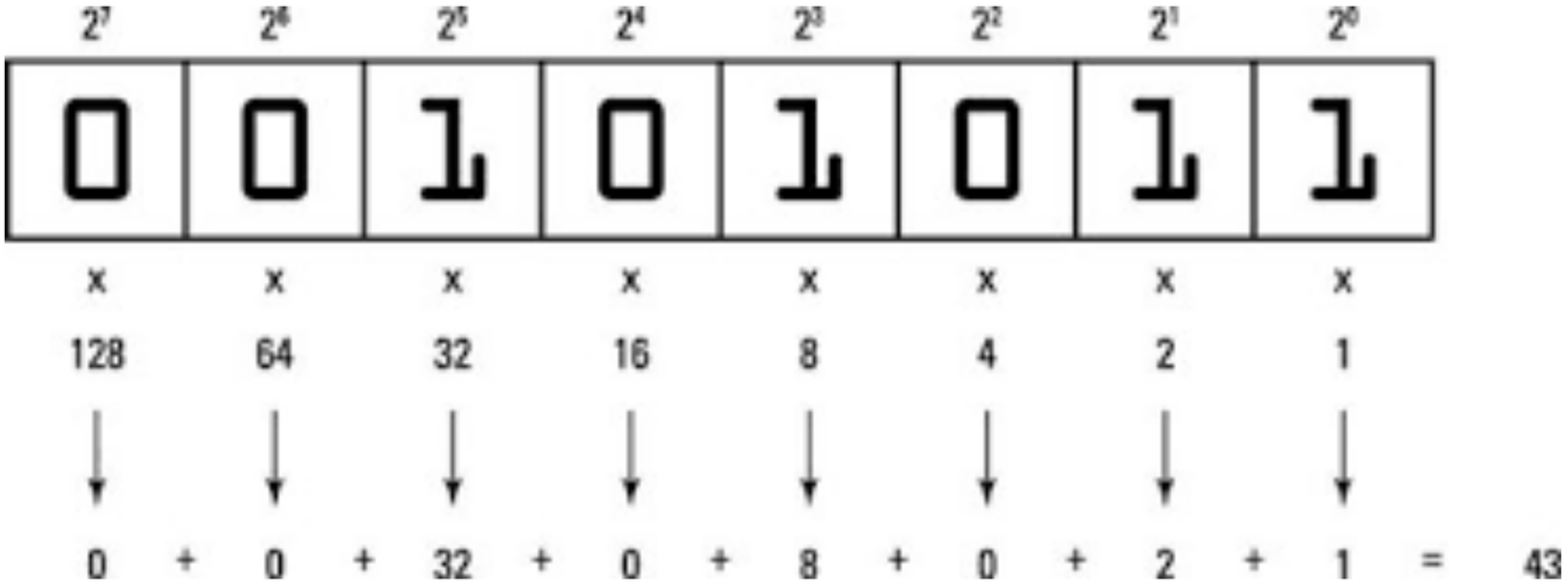
INPUT		OUTPUT
A	B	A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

Representation

- Language is representation of ideas.
- Writing is a representation of language. Speaking is another. So is a recording.
- In mathematics, a representation is a very general relationship that expresses similarities (or equivalences) between mathematical objects or structures.
    - diagrams, number lines, graphs, arrangements of concrete objects or manipulatives, physical models, mathematical expressions, formulas and equations, or depictions on the screen of a computer or calculator – that encode, stand for, or embody mathematical ideas or relationships.
- Data Representation … example pixels in an image.
- Representation Learning…

Binary

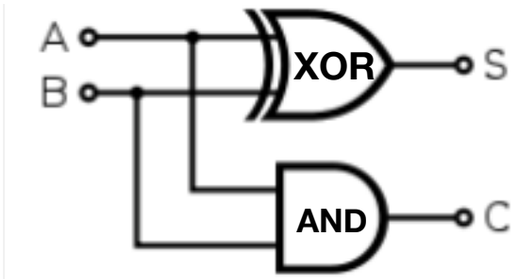
Representation



Number	Binary equivalent
0	000000
1	000001
2	000010
3	000011
4	000100
5	000101
6	000110
7	000111
8	001000
9	001001
10	001010
11	001011
12	001100
13	001101
14	001110
15	001111

Adder

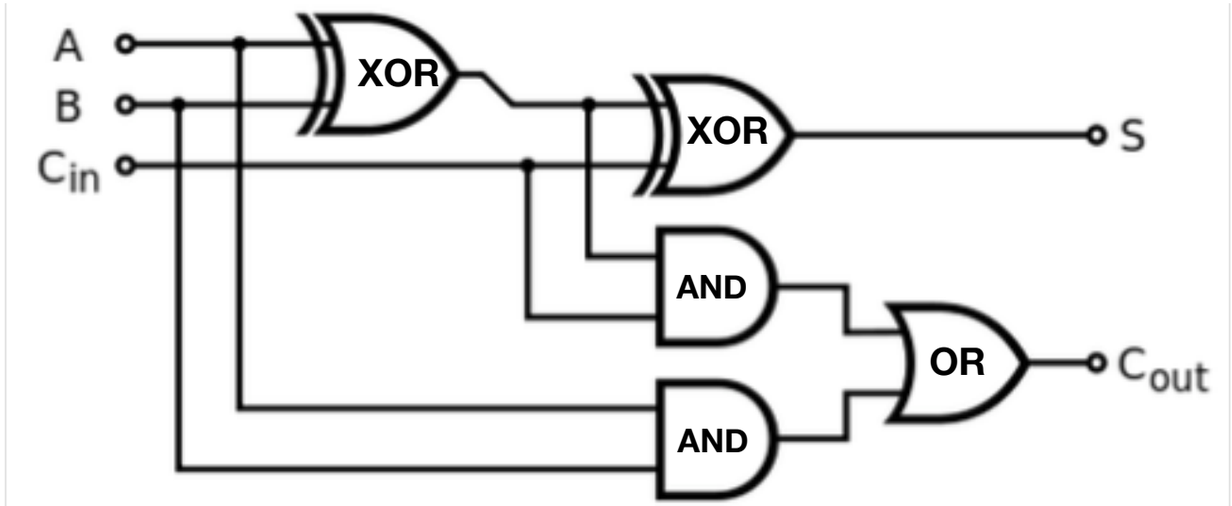
Half Adder



0+0=00
0+1=01
1+0=01
1+1=10

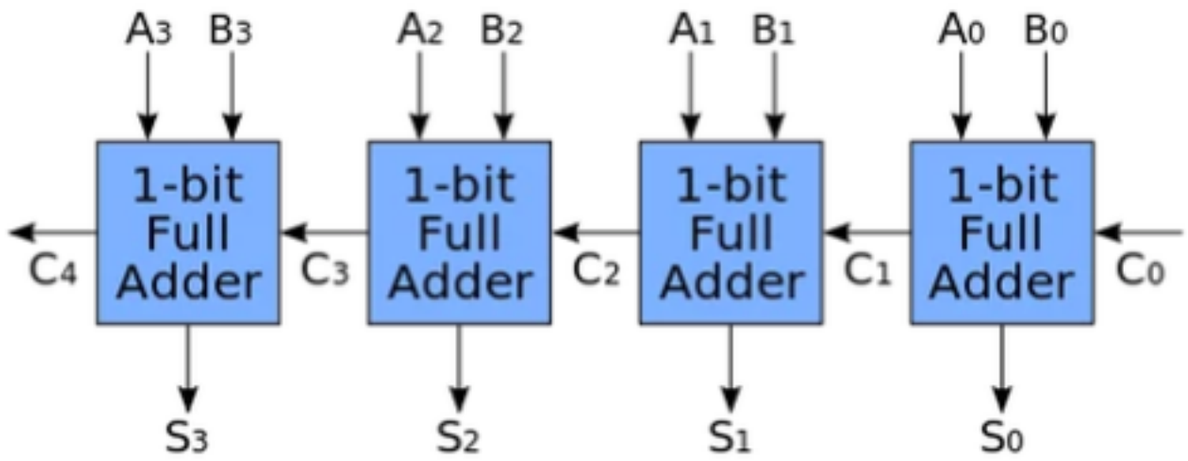
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Full Adder



0+0+0=00
1+0+0=01
0+1+0=01
0+0+1=01
1+1+0=10
0+1+1=10
1+0+1=10
1+1+1=11

N-Bit Adder



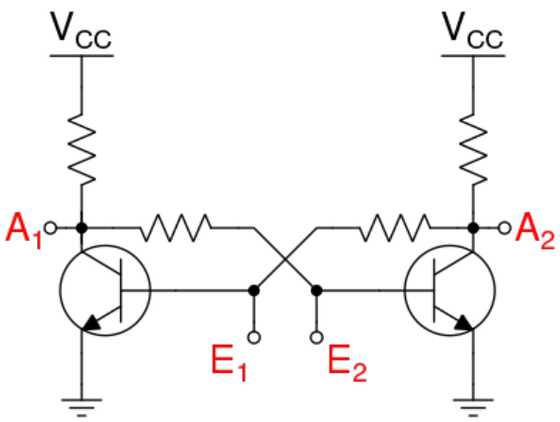
https://gizmodo.com/how-computer-chips-work-1645654993

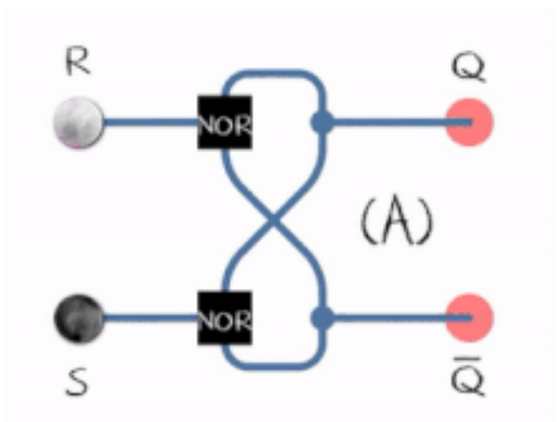
Other Circuits

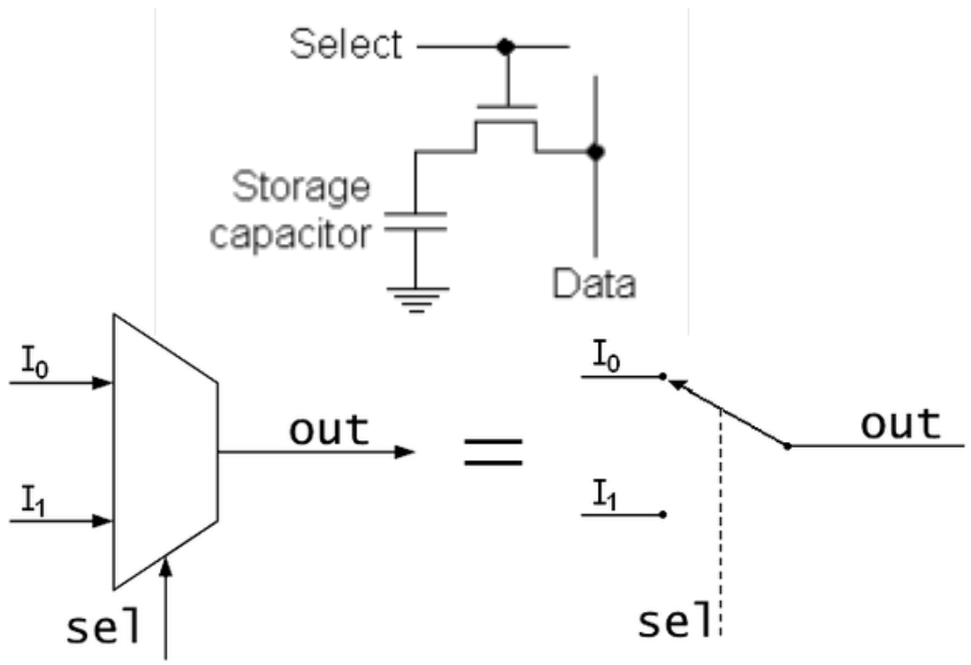
- *Flip-flop:* Circuit that can "store" a value. Its output will depend on what value was written to it earlier.

- *Multiplexer:* selects between two inputs, I_0 and I_1, and outputs I_0 if the command is 0, and I_1 if it is 1
  - Use it to select different instructions (e.g. + or -).

- Random Access Memory: 1024-bit Multiplexer + 1024 Flop-flops = 1 KB





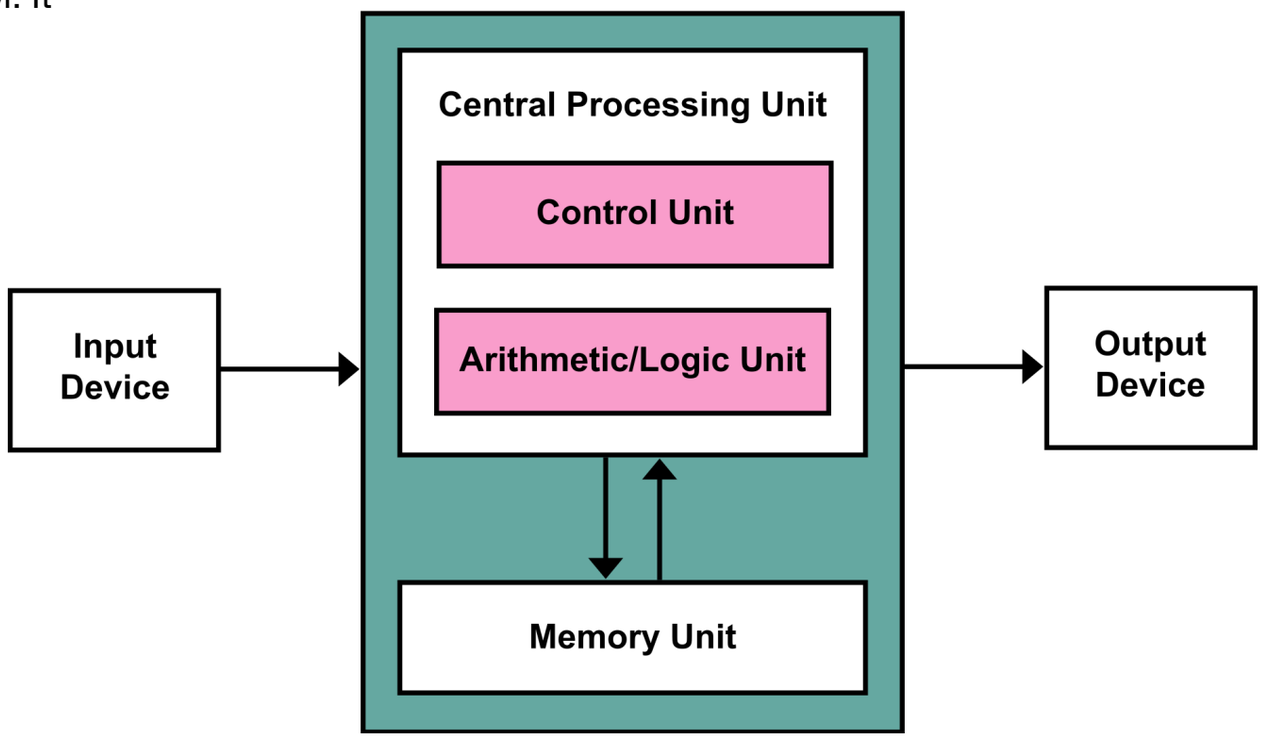


Simple Electronic Calculator

- 3 “Registers” that hold n-bit number.
    - Each is a N-bit flip-flop.
- Each number key -> loads binary representation into a register.
- Operation key (for example + or -) -> takes inputs of two registers and applies operation -> results into 3rd register.

Von Neumann Architecture

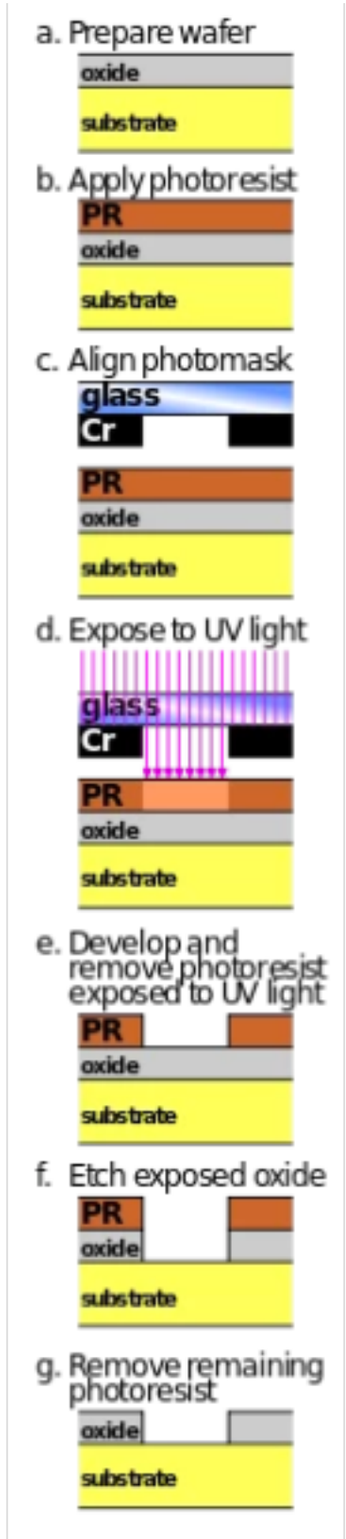
- ***Memory Unit:*** (MEM) holds all the "commands" (instructions) and "numbers" (data).
- ***Control Unit:***
    - ***Program counter***: select which instruction to execute from MEM. It normally just increases by 1 in each step (clock cycle).
- ***Arithmetic block:*** multiplexers connecting to different binary different operations (+, -, …), with input / output
    - Generate inputs to our arithmetic block from MEM.
- [***Registers:*** hold input/output of arithmetic block]
- Instructions
    - Two types: data instructions and control instructions.
    - Each data instruction contains four things:
        - two addresses specifying which two numbers to pick from MEM
        - one command saying what operation to perform
        - and another location saying where to put the result back.
    - The control instructions put another address back into the "program counter."



Scilicon Electronics

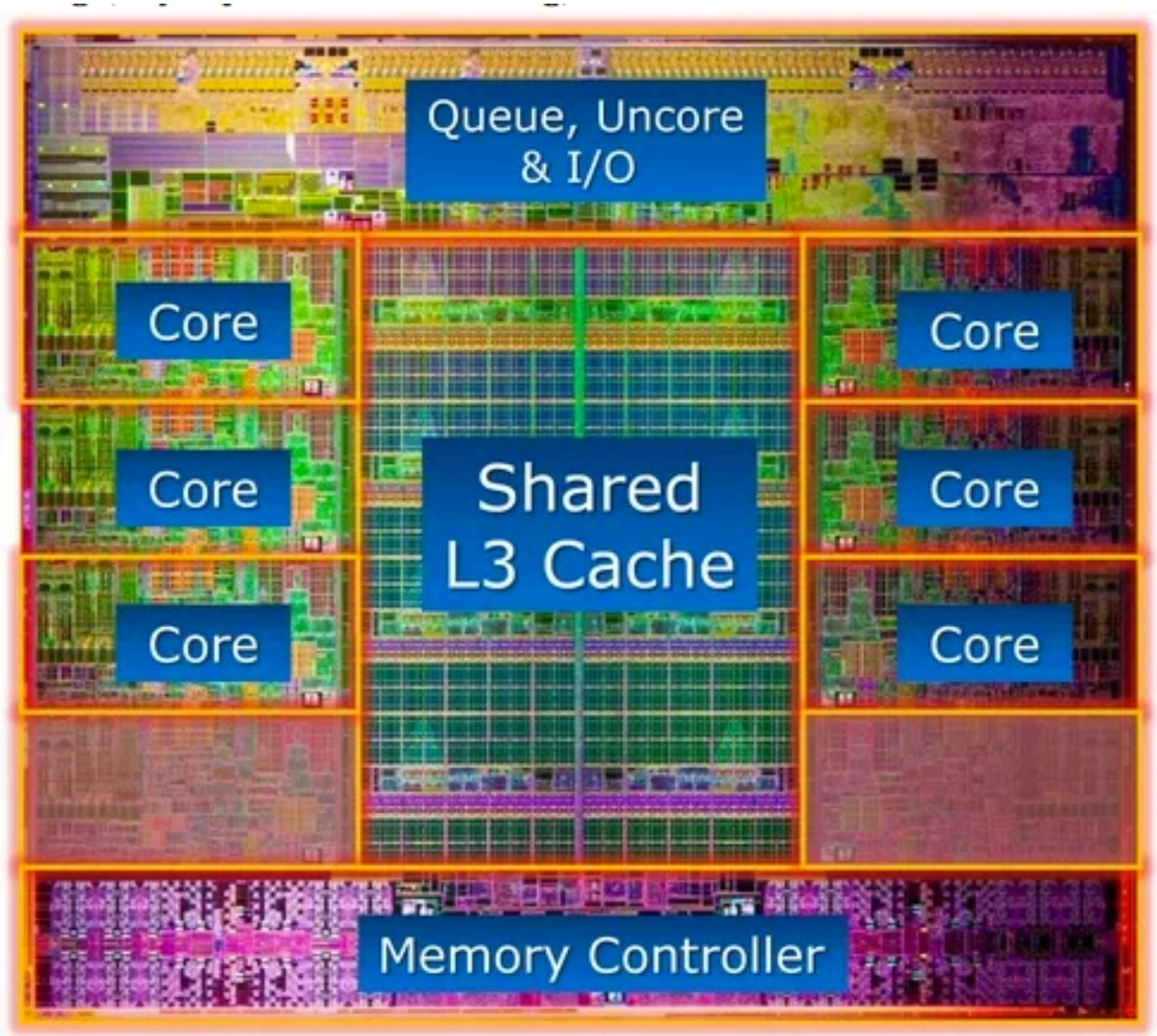
- Modern processors have billions of transistors precisely arranged and connected.

- Photolithography: pattern is an image that is focused on silicon covered with layers photo-sensitive material, that can be subsequently etched away or have metal deposited on, leaving a pattern.

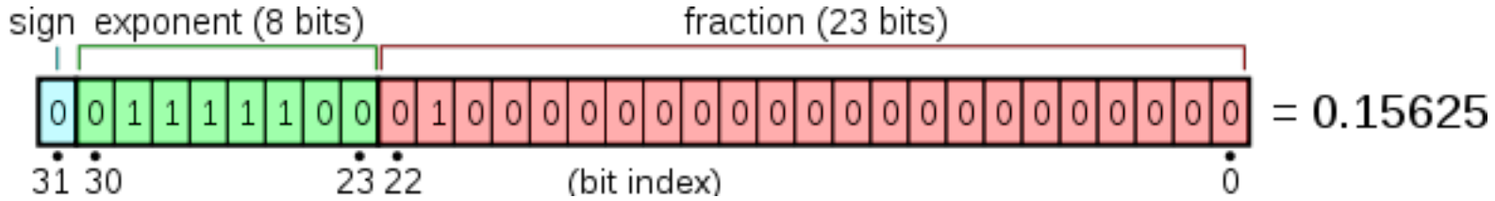


Modern Processors

- Variable clock rates (in GHz).
- Biggest constraints: energy consumption and heat dissipation.
- Large amounts of memory (GBs), usually not on processor.
    - New processors like Apple M1, M2, M3, memory integrated with CPU.
- Cache: small part of memory, mirrored inside/ closer to processor to accelerate memory access.
    - Multi-level: Exchange size vs proximity/speed
- Out-of-order processing
- Many cores



Floating Point



Floating Point Operation per Second (FLOPS) =

cores

x

cycles/second

x

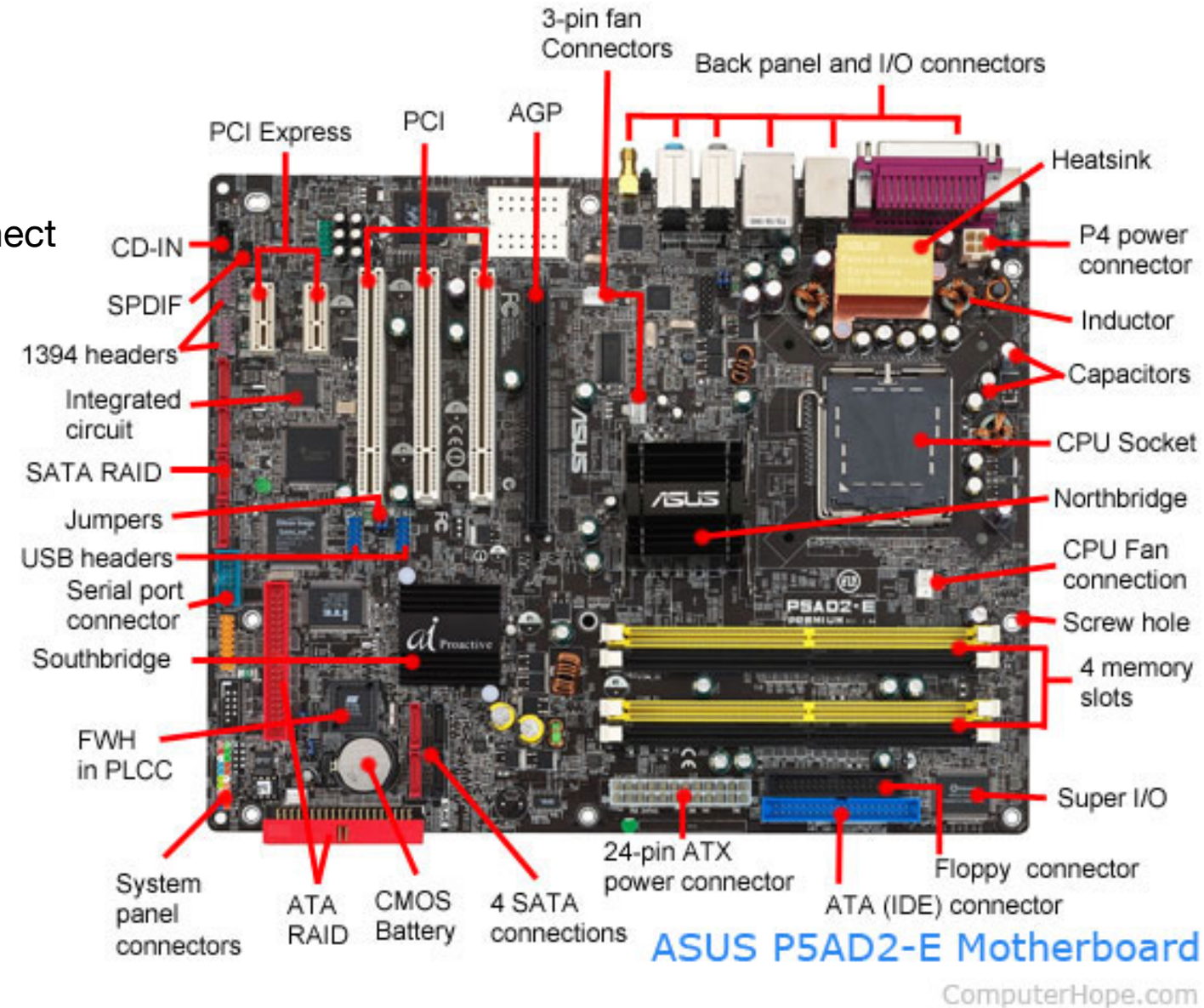
FLOP/cycle

FLOPs per cycle for various processors [edit]

Microarchitecture	ISA	FP64	FP32	FP16
Intel Atom (Bonnell, Saltwell, Silvermont and Goldmont)	SSE3 (64-bit)	2	4	0
Intel Core (Merom, Penryn) Intel Nehalem[6] (Nehalem, Westmere)	SSE4 (128-bit)	4	8	0
Intel Sandy Bridge (Sandy Bridge, Ivy Bridge)	AVX (256-bit)	8	16	0
Intel Haswell[6] (Haswell, Devil's Canyon, Broadwell) Intel Skylake (Skylake, Kaby Lake, Coffee Lake, Whiskey lake, Amber lake)	AVX2 & FMA (256-bit)	16	32	0
Intel Xeon Phi (Knights Corner)	SSE & FMA (256-bit)	16	32	0
Intel Skylake-X Intel Xeon Phi (Knights Landing, Knights Mill)	AVX-512 & FMA (512-bit)	32	64	0
AMD Bobcat	AMD64 (64-bit)	2	4	0
AMD Jaguar AMD Puma	AVX (128-bit)	4	8	0
AMD K10	SSE4/4a (128-bit)	4	8	0
AMD Bulldozer[6] (Piledriver, Steamroller, Excavator)	AVX (128-bit) Bulldozer-Steamroller AVX2 (128-bit) Excavator FMA3 (Bulldozer)[7] FMA3/4 (Piledriver-Excavator)	4	8	0
AMD Zen (Ryzen 1000 series, Threadripper 1000 series, Epyc Naples) AMD Zen+[6][8][9][10] (Ryzen 2000 series, Threadripper 2000 series)	AVX2 & FMA (128-bit, 256-bit decoding)[11]	8	16	0
AMD Zen 2[12] (Ryzen 3000 series, Threadripper 3000 series, Epyc Rome)	AVX2 & FMA (256-bit)	16	32	0
ARM Cortex-A7, A9, A15	ARMv7	1	8	0
ARM Cortex-A32, A35, A53, A55, A72, A73, A75	ARMv8	2	8	0
ARM Cortex-A57[6]	ARMv8	4	8	0
ARM Cortex-A76, A77	ARMv8	8	16	0
Qualcomm Krait	ARMv8	1	8	0
Qualcomm Kryo (1xx - 3xx)	ARMv8	2	8	0
Qualcomm Kryo (4xx)	ARMv8	8	16	0
Samsung Exynos M1 and M2	ARMv8	2	8	0
Samsung Exynos M3 and M4	ARMv8	3	12	0
IBM PowerPC A2 (Blue Gene/Q)	?	8	8 (as FP64)	0
Hitachi SH-4[13][14]	SH-4	1	7	0
Nvidia Fermi (only GeForce GTX 465-480, 560 Ti, 570-590)	PTX	1/4 (locked by driver, 1 in hardware)	2	0
Nvidia Fermi (only Quadro 600-2000)	PTX	1/8	2	0
Nvidia Fermi (only Quadro 4000-7000, Tesla)	PTX	1	2	0
Nvidia Kepler (GeForce (except Titan and Titan Black), Quadro (except K6000), Tesla K10)	PTX	1/12 (for GK110: locked by driver, 2/3 in hardware)	2	0
Nvidia Kepler (GeForce GTX Titan and Titan Black, Quadro K6000, Tesla (except K10))	PTX	2/3	2	0
Nvidia Maxwell Nvidia Pascal (all except Quadro GP100 and Tesla P100)	PTX	1/16	2	1/32

Motherboard

- Central Processing Unit
- Random Access Memory
- Basic Input/Output System (BIOS)
- Off processor Cache
- Expansion Bus: Peripheral Component Interconnect (PCI)
- Chipset: control data flow in/out of CPU
  - Northbridge: CPU / memory
  - Southbridge: CPU / IO
- IO Controllers:
  - Disk: ATA, SATA, RAID
  - Peripherals: USB, …
  - Network: Ethernet, WiFi
- CPU Clock

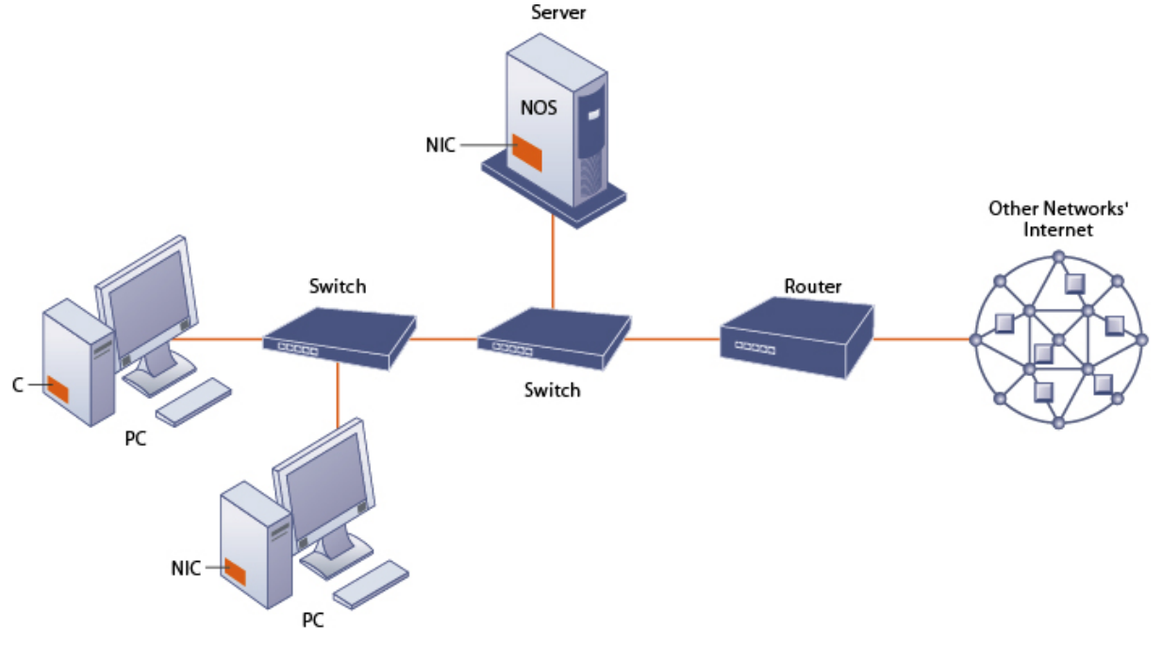


Computer

- Case
- Power supply
- Fan
- Motherboard
    - CPU
        - Heat sink / Fan
    - RAM
    - Graphics Processing Unit (GPU)
    - USB, Ethernet, etc connectors
- Storage: Hard Drive or Silicon Disk Drive

Network

- Local Area Network (LAN) / Wide Area Networks (WAN)
- Signals sent via coaxial cable, twister pair, fiber optics, …
- Components: adapter, switch (connects computers), router (connects networks), wireless…
- Open Systems Interconnection model Layers
    - Physical: bits. Rates now approaching 100 Gb.
    - Ethernet:
        - Every component has a unique address (48-bit MAC address)
        - Data broken into frames, with source/destination address and error checking data.
    - Network: for example Internet Protocol (IP)
        - Packets sent via IP address
        - Addresses kept in Domain Name System (DNS): Match name → address.
    - Transport:
        - How data is exchanged, broken up, transmitted, routed, …
        - Transmission Control Protocol (TCP): Services listen / communicate on ports.



OSI model				
Layer			Protocol data unit (PDU)	Function[6]
Host layers	7	Application	Data	High-level APIs, including resource sharing, remote file access
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1	Physical	Symbol	Transmission and reception of raw bit streams over a physical medium

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
6	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	125.71	7,438
7	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93.01	125.44	15,371
8	**Perlmutter** - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761,856	70.87	93.75	2,589
9	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63.46	79.22	2,646

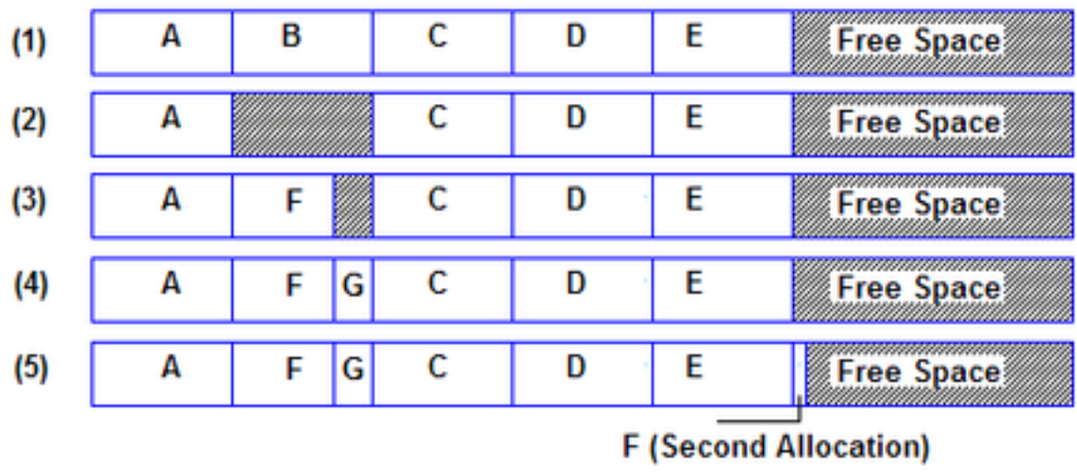
1. Storage, Filesystem
2. Firmware → Operating System → Apps
3. Machine Language → Python

Storage

- Storage devices provide an interface to read / write data into specific locations of some in non-violative media.
    - On traditional ***Hard Drive***, the data is written magnetically on a spinning metal disk.
        - The disk is divided up into sectors, the minimum storage unit.
        - Each sector has an address, which corresponds its physical location on the disk.
        - ***Solid State Drives*** store data on silicon… originally presented same interface as HDs, but new interfaces such as non-volatile memory express (NVMe) are designed for SSDs.
        - In Unix these are referred to as: /dev/hda, /dev/hdb, … /dev/sda, /dev/sdb
- ***Partition***: The disk sectors are partitioned into groups of sectors, each where a different file system can be created.
    - Partition table: keeps track of the locations of the partitions.
    - /dev/hda1, /dev/sdb2
    - Partitions can be further divided into logical partitions.

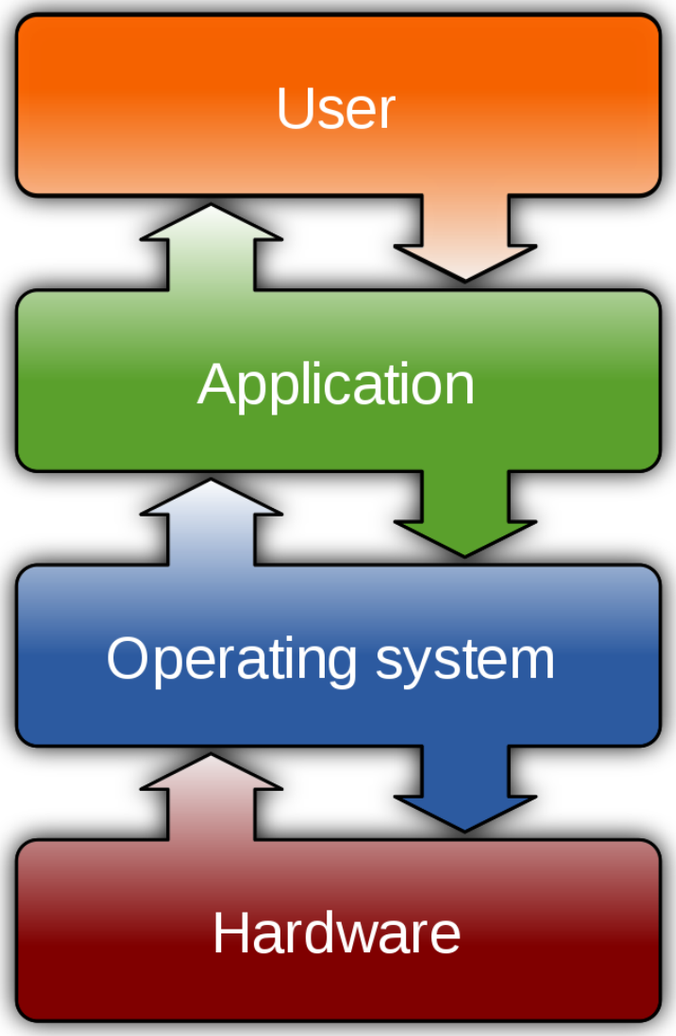
File system

- File system is a scheme for controlling how data is stored and retrieved from a partition.
    - File system organizes the sectors into blocks.
    - Organization:
        - Data is grouped into files.
            - Files have meta-data: e.g. when created, access permissions, …
        - Files are organized into directories.
    - FS holds a map Files names → Blocks.
        - Different file systems have different restrictions file names (e.g. allowed characters or case sensitivity).
        - General convention, “FOO.BAR”, FOO is filename, BAR is the extension which helps indicate the format of the file contents.
- FS enforce Access Control or Permissions… who can read what file.
- Different File system types:
    - Windows: FAT (FAT16, FAT32), exFAT NTFS, …
    - Mac: HFS, HFS+, APFS
    - Unix: ext2, ext3, ext4, ZFS, …
- SWAP: In some Operating Systems, the system can use storage as RAM when out of memory.



Software- Firmware

- There are several layers of software that interact with the computer at different levels.
    - During system boot, system starts with the Firmware, and hands control to next level, which then hands control on to then next level, and so on …
- ***Firmware***: Associated with the motherboard. Read Only Memory (ROM).
    - When you start a computer, it needs some instructions on what to do.
    - Test that everything is OK… e.g. memory.
    - Load user configuration.
    - Allow users to change configuration via menu system.
    - Start and configure peripherals, storage, …
    - Load the boot loader from storage into memory and hand-off
    - In the x86 world:
        - Basic Input/Output System (BIOS): Old
        - Unified Extensible Firmware Interface (UEFI): New



Boot Loader …

- Small program that loads the operating system kernel.
- Loaded by the firmware.
- Usually sits on the first sectors of the storage boot drive.
- May give the user the option of loading different operating systems sitting on different partitions of the storage.
- Must mount the partition, find the kernel (operating system), load it into memory, pass control to the kernel with potentially some configuration options (e.g. the root partition).

Operating System

- Software system that manages the computer hardware and software and provides common services for programs.
    - Sharing of resource between programs: processor, memory, storage, …
    - Intermediary between programs and hardware.
        - Provides Application Program Interface (API) / Software Development Kit (SDK) for building programs and interfacing them with OS.
- Examples: Windows, MacOS, Linux, iOS, Android, …
- Modern OSs are Multi-tasking: allow multiple programs to simultaneously run.
    - Each program ~ a process.
    - Pre-emptive multitasking: the OS gives slice of CPU time to each process.

Unix

- Multitasking, Multi-user, OS originally developed in 1970 by AT&T Bell Labs to run on mainframes with many connected terminals.
    - Written in C programming language.
- Many modern operating systems, including MacOS and Linux, implement Unix standards.
- “Unix Philosophy”
    - Plain text data storage.
    - Hierarchical file system.
    - Devices and inter-process communication via files
    - Main program that runs is the *kernel*.
    - Primary user interface is a *command-line* interpreter, called a shell.
    - Modular:
        - lots of small programs serve as tools
        - strung together via the *command-line* interpreter
        - passing information between each other via pipes

Graphical User Interface

- Though most of you associate an OS with its GUI, it is generally a layer on top of the core OS.
    - Establishes graphical metaphors aimed at simplifying user interaction with the OS.
        - Mouse cursor, windows, menus, buttons, …
        - Provides API for applications to build GUIs for themselves.
- For Unix, the X Windows system implements a client-server model:
    - The server runs the application that makes the API GUI calls.
    - These calls are transmitted to the network to a client.
    - The client runs the graphical environment, collects input, and draws the graphical elements.

Programming

- Each CPUs understands its own instruction sets.
- Low level operations:
  - Copy in/out data from memory into registers.
  - Perform operations on registers.
  - Conditional (if/then) and Control flow (jump)
  - Manage a stack (where small sets of data can be shared between different blocks of code)
- Each instruction are each assigned a specific binary value and are not human readable…
- Assembly is a human-read language that most closely mirrors Machine Language.

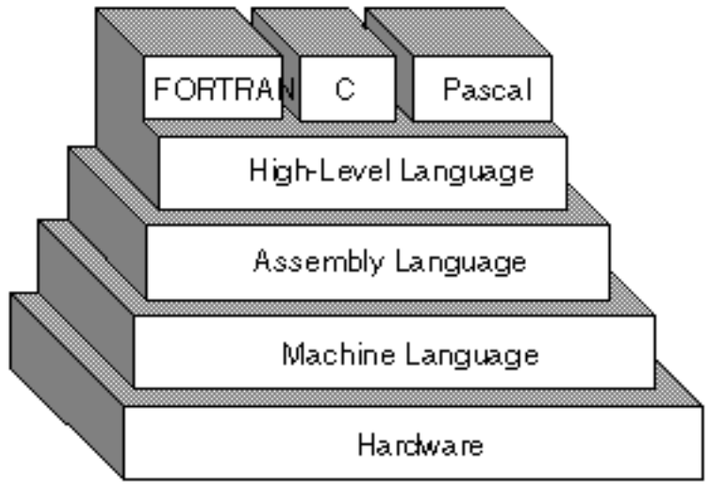
```
x3000 LD R1, x006          ; load data pointer
x3001 LDR R2, R1, #0       ; load data value
x3002 BRz x005             ; branch to end if zero
      ;
      ; repeating statements go here
      ;
x3003 ADD R1, R1, #1       ; increment data pointer
x3004 BRnzp x001           ; branch back to top
x3005 HALT
      ;
      ; data section
      ;
x3006 x4000 ; address of data

; The translation into LC-2 Machine Code
      0011 0000 0000 0000      ; load at x3000
X3000 0010 001 0 0000 0110     ; LD R1, x006
x3001 0110 010 001 000000      ; LDR R2, R1, #0
x3002 0000 010 0 0000 0101     ; BRz x005
      ;
      ; repeating statements go here
      ;
x3003 0001 001 001 1 00001     ; ADD R1, R1, #1
x3004 0000 111 0 0000 0001     ; BRnzp x001
x3005 1111 0000 0010 0101      ; HALT
      ;
      ; data section
      ;
x3006 0100 0000 0000 0000      ; x4000
```

from: http://www.eecs.umich.edu/courses/eecs284/example1.htm

Programming Languages

- Writing Assembly (Machine) code requires
    - knowledge of the specific CPU,
    - working at level of very small operations and
    - awareness of registers, memory, and hardware.



- High-level languages provide high level abstractions and human friendlier syntax for programming.
- Two types:
    - ***Compiled***: The text of the high level language is converted by a *compiler* into machine code. The machine code is run subsequently. Usually 2 steps:
        1. Compile to code into machine language → library
        2. Link the code against libraries → executable
    - ***Interpreted***: A program runs that reads the text of the high level language and performs the operations.

High-level Language

- 3 Fundamental Elements of any programing language:

    1. Primitives:

        - Numbers, Characters, …
        - Mathematical Operations: +, -, *, /, …
        - Logical Operations: and, or, …
        - Conditionals: if-then-else, …

    2. ***Means of Combination***: e.g. list

    3. ***Means of Abstraction***:

        - Assignment: x = 1
        - Definition: def x: 1
        - Function: def f(x): x

- Beyond these, there are universal programming concepts and patterns (e.g. object oriented programming) that enable or facilitate building sophisticated software.

- While we will learn these in python, look beyond the syntax and specifics of the programming language.

Why Python?

- Interpreted: no compilation time. Multi-platform.
    - Expense of speed, but the time consuming code are often in complied libraries.
- Large library: almost any package out there has a python API.
- Easy to read. Convenient syntax.
- Convenient data structures that are simple to build.
    - Advanced data structures: list, dictionaries, sets…
    - Dynamic typing: no need to declare the type of a variable.
    - Built-in memory management (reference counting + garbage collection): No need to worry about memory addresses or allocating/freeing memory.
    - Dynamic name resolution (late binding): same code can be reused for different data.
- Multi-paradigm:
    - structured programming: functions, sub-routines, etc…
    - functional programming: filter, map, reduce, lambda, generators, …
    - object-oriented programming: class, inheritance, …
    - …
- Language of choice for Data Science.

Operating Systems

- ***Linux***- implementation of a Unix Kernel
    - Distributions- Packaging Linux Kernel with rest of OS software (mostly from GNU project)
        - Examples: RedHat, Debian, Ubuntu, CentOS, …
- ***BSD***- implementation of Unix Kernel + full OS
- ***MacOS***- Officially certified as Unix
    - Built on BSD + Apple Kernel + Custom Apple Modules + Mac Interface
- ***iOS***- Shares a lot with MacOS
- ***Windows***
    - Windows Subsystem for Linux

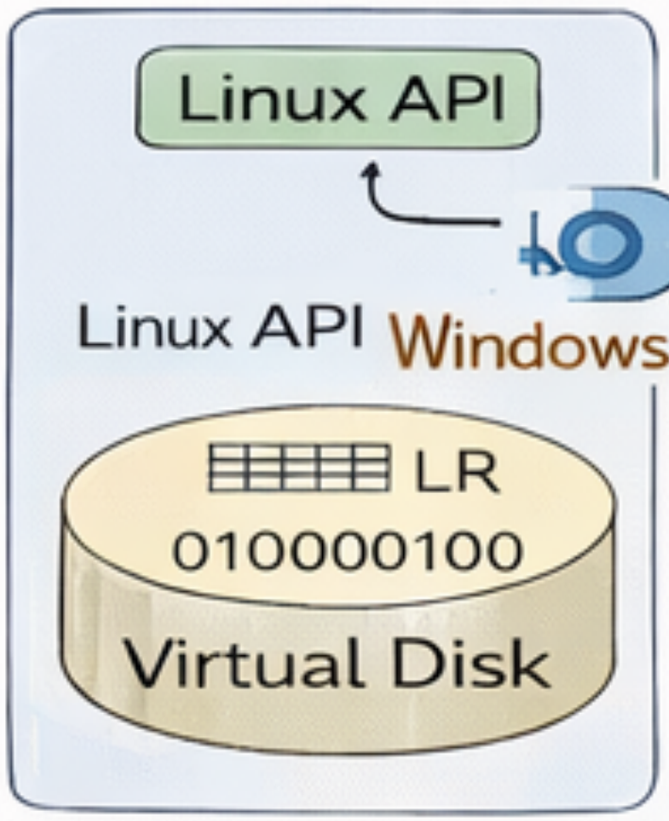
Mixing OSs

- Each operation system has different API/SDK that allow programs to interact with the computer and its components.
- All the Unix-base OS can compile and execute any basic Unix program that do not rely on any additional APIs provided by the OS.
    - One of the reasons that Unix is widely used is that it makes it easy to make your software work on different architectures, since almost all architectures have some Unix-like OS.

- How can you run a program from OS A (e.g. Linux) to run on OS B (Windows)?
    - Two things to think about:
        - Difference in OS API/SDK
        - Difference in Architecture (e.g. x86 vs ARM)

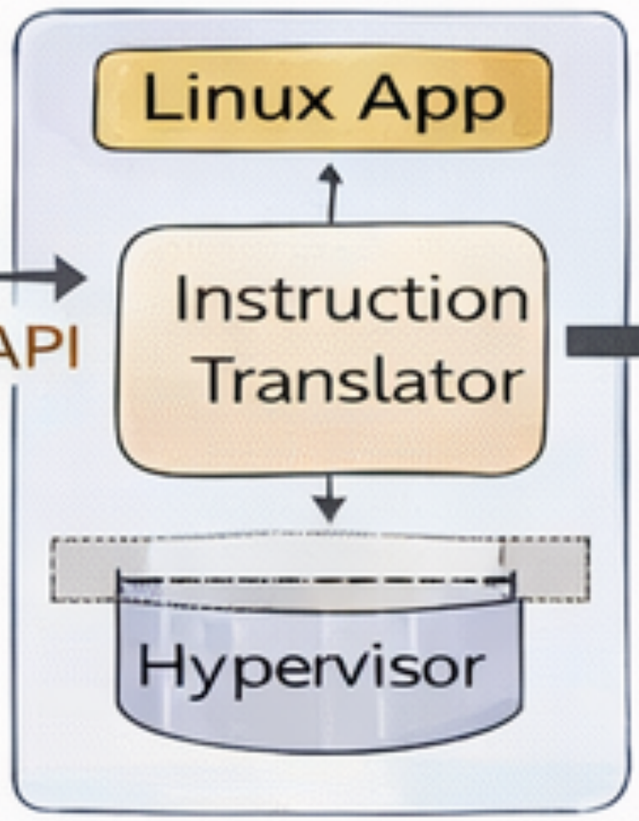
Approaches to Mix OSs

Wrapper



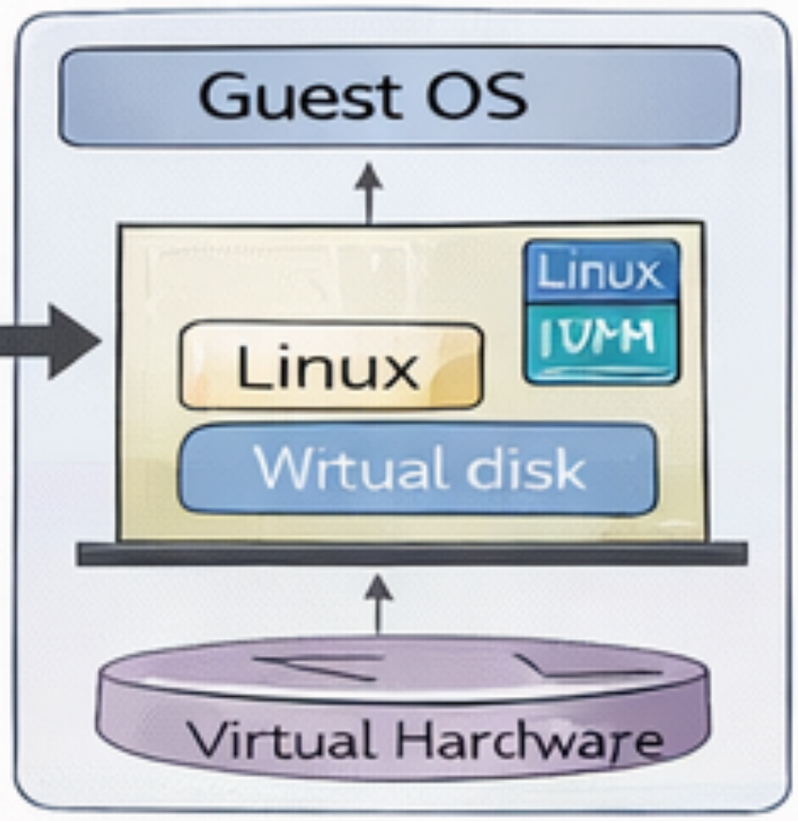
Create an image file, boot the VM,

Translation



I Create image file, install geest OS Into AFM.

Virtual Machine



Create an image file, boot VM, install OS into Guest OS inside a ⊟

Approaches to Mix OSs

- ***Wrapper***- Thin layer that presents OS A's API to programs, but wraps OS B's APIs.
    - ***WSL 1***: used this approach to enable running Linux programs in Windows
    - ***Wine/SteamOS***: similar approach enabling running Windows in Unix
- ***Translation-*** In case of same OS but different Architecture, convert machine code from A to B.
    - Rosetta (Mac OS transition from PowerPC → Intel) did on the fly translation.
    - Rosetta 2 (Mac OS transition from Intel → Apple M1) translates Intel code to ARM first time you run.
    - Windows on ARM does something similar to run Intel x86 Apps.
- ***Virtual Machine***- An application that simulates (usually with help from underlying OS and hardware) a computer and it's components
    - "Image" files on host machine appear as storage devices
    - For example: Create image file, boot VM, install OS into image file.
    - ***WSL 2:*** Uses VMs to run an actual Linux kernel.
    - You can use VM software to run one OS inside of another (e.g. Parallels or VMWare for running Windows programs on the Mac)
- ***Hypervisor***- Separates computer resources and simultaneously runs several VMs
    - For example in the cloud
- ***Container-*** Encapsulates everything needed to run specific software (including OS), but shares same Kernel and Resources between containers (and possibly host OS).

Why not Anaconda?

- While it appears to make things easy to install on on Windows…
    - It isn’t necessary. All the packages we need can be installed in one command using python’s pip installer.
        - It “fakes” a unix like environment, with many limitations.
            - I want you to be exposed to Unix ASAP.
            - Its difficult for me to support.
- If you are using a windows computer, I require you to use WSL.

Windows Subsystem for Linux
+ Data Science Stack

1. Enable Windows Linux Subsystem
    - Settings → Apps → Programs and Features → Turn Features on or off → Windows Subsystem for Linux
2. Install Ubuntu 24 LTS
    - Windows App Store
3. Startup Ubuntu (starts a shell)
    - First time will ask for username/password to make an account
    - Remember this username / password
4. Update package list (the rest of these commands you type into the terminal/shell window)
    - `sudo apt-get update`
5. `sudo apt-get install` <package_name>
    - python3, python3-pip
6. Close and re-open the Ubuntu Shell (sets up path correctly)

WSL Disk vs Windows Disk

- The WSL virtual machine emulates a storage volume where your install of linux and everything else you do will sit.
- This is distinct from the usual storage volumes in Windows.
- The WSL virtual volume is just an “image” file on your windows volume which WSL uses to emulate a storage disk.
- But Windows “mounts” the WSL volume and WSL mounts the Windows volume, meaning
    - You can get to the Windows volume from WSL (for example the Ubuntu terminal or jupyter notebook)
    - You can get to the WSL volume from Windows (for example your browser, Explorer, or any Windows App)
- Your browsers are running in Windows, so your downloads are on the Windows Volumes. You’ll have to copy them into the WSL volumes.
- How? From Google:
    - To access WSL volumes from Windows, navigate to the path "\\wsl\\" in your File Explorer, where you will see a list of your installed Linux distributions, allowing you to browse their file systems as if they were network shares; for example, to access the Ubuntu distribution, go to “\\wsl\\Ubuntu"
    - To access Windows volumes from within WSL (Windows Subsystem for Linux), navigate to the /mnt directory within your Linux distribution, where each Windows drive letter will be represented as a folder, like /mnt/c for your C: drive; essentially, you can access your Windows files by using the path /mnt/<drive letter> within your WSL terminal.

For Wednesday's Lecture

- Bring your laptops.
- I'll give out Lab 2.
- Make an attempt to get WSL setup (if on Windows), install python, Jupyter.
- Have jupyter running with Lab 2 loaded.
- The TAs will go around and check your laptops.
- Don't panic if you have trouble.
    - Most of you won't have any issues.
    - If you are having issues, TA and I will help resolve them.