Computer Architecture

COMP SCI 2GA3

Chapter 1 - Computer Abstractions and Technology

Based on:
RISC-V Chapter 1 textbook slides
COMPSCI 2GA3 2016 fall - Chapter 1
SOFTENG 2GA3 2020 winter - Chapter 1

Dr. Bojan Nokovic, P.Eng.
McMaster University, Fall Term 2021/22

Industrial Revolutions

"Civilization advances by extending the number of important operations which we can perform without thinking about them" - Alfred North Whitehead, An Intro. to Mathematics, 1911

Skills must be mastered to the point of automaticity in order to free the mind for the task at hand.

Modern era technological revolutions:
1 - Financial-agricultural revolution (1600-1740)
2 - First industrial revolution (1780-1840)
3 - Technological revolution - Second industrial revolution (1870-1920)
4 - Scientific-technical revolution (1940-1970)
5 - Information and telecommunications revolution - Digital or Third industrial revolution (1975-2020)
6 - Fourth industrial revolution - A fusion of technologies that is blurring the lines between the **physical**, **digital**, and **biological** spheres…

The Computer Revolution

- **Progress in computer technology**
  - Rapidly become cheaper and more powerful
  - Underpinned by Moore's Law
- **Makes novel applications feasible**
  - Computers in automobiles
  - Smart phones
  - Human genome project
  - World Wide Web
  - Search Engines
  - Laniakea
- **Computers are pervasive !**

Classes of Computers

- **Personal computers**
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
  - Single user, used with mouse, keyboard, and monitor

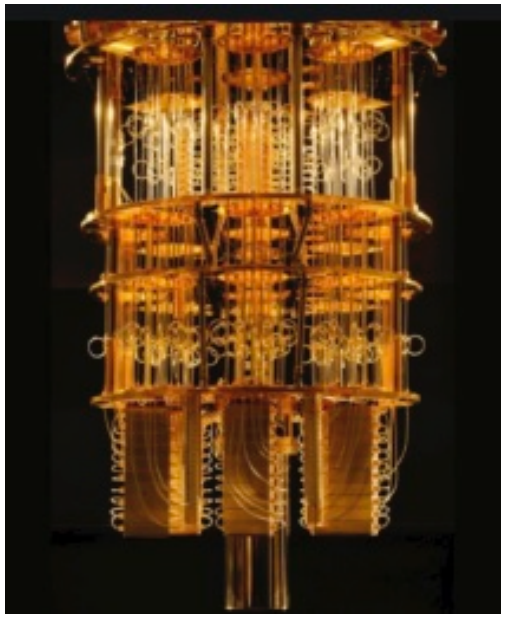
- **Server computers**
  - Usually accessed over network
  - Typically no monitor or keyboard/mouse
  - High capacity, performance, reliability
  - May run a single, complex application, or handle many small jobs
  - Range from small servers to building sized

Classes of Computers

- **Supercomputers**
  - May consist of tens of thousands of processors and terabytes (10^{12} bytes) of memory
  - High-end scientific and engineering calculations
  - https://en.wikipedia.org/wiki/TOP500 (Fugaku)
- **Embedded computers**
  - Hidden as components of systems
  - Designed to run a single application and comes integrated with hardware
  - Stringent power/performance/cost constrains; have low tolerance for failure

Classes of Computers

- **Quantum computers**
  - Perform Quantum computations
  - Based on supercomputing **qubits**
  - Able to solve certain problems that no classical computer could solve in any feasible amount of time
  - Cryptography, drug design, quantum teleportation



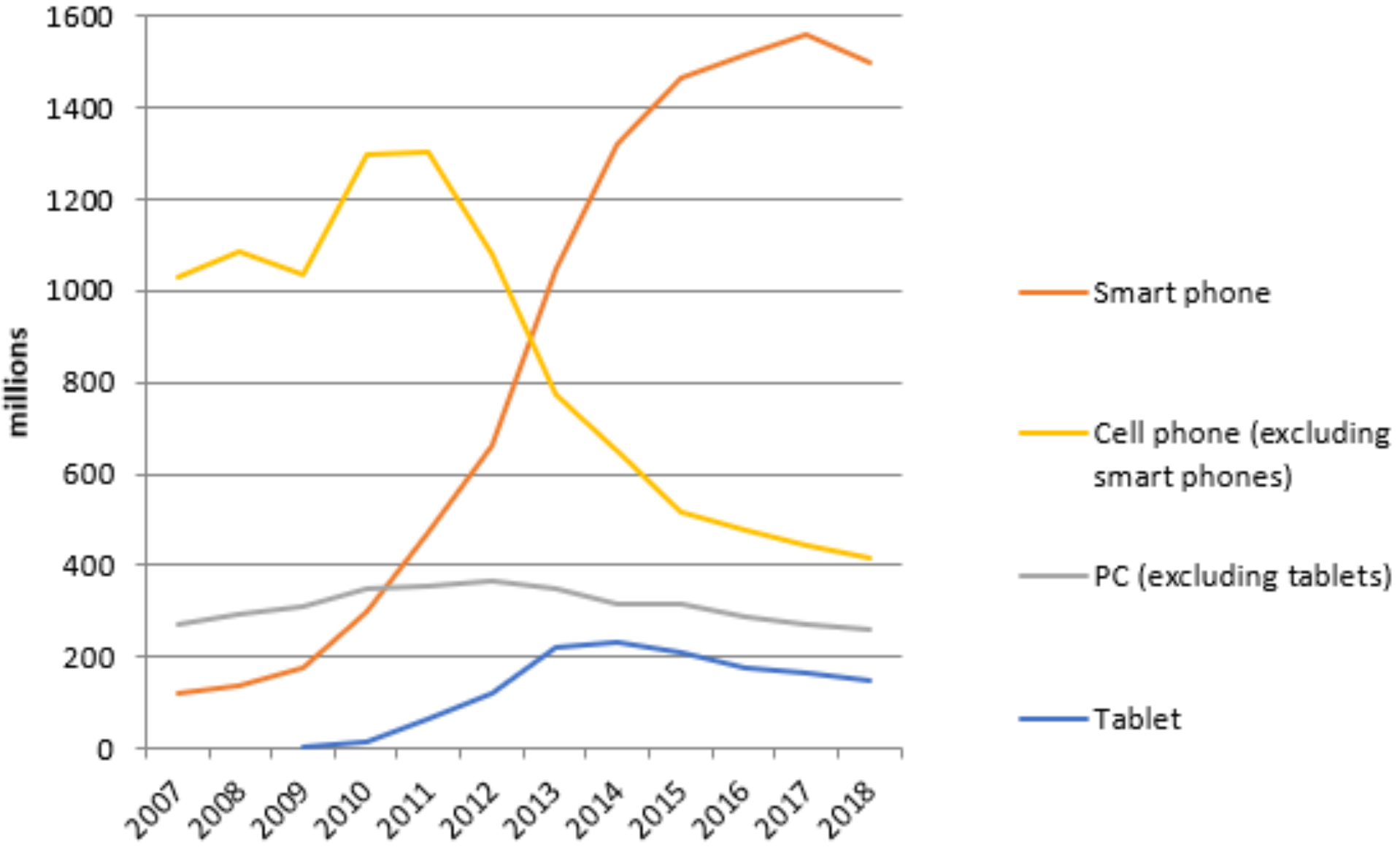
https://www.youtube.com/watch?v=QuR969uMICM

Common Memory Sizes

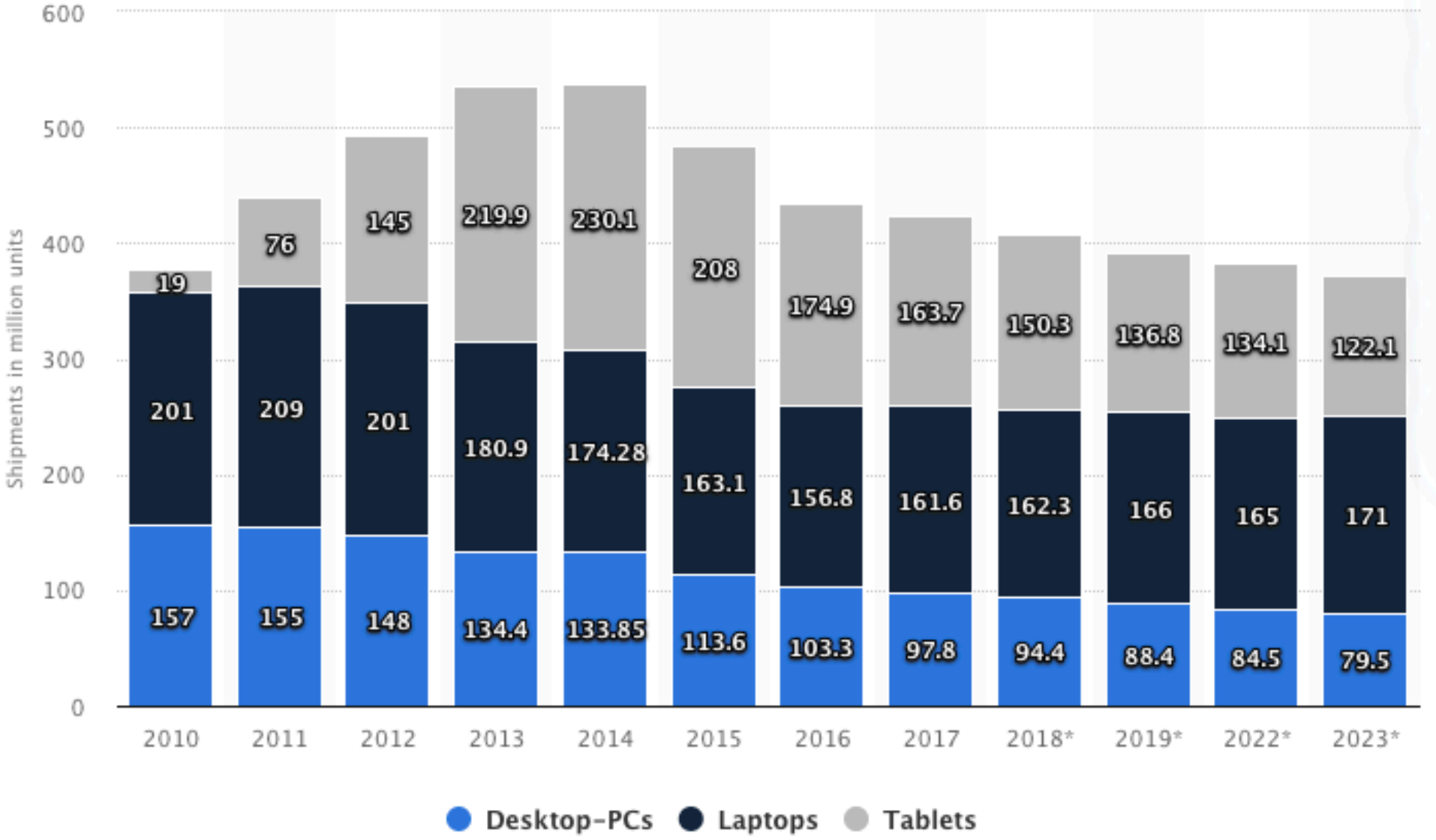
Decimal term	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

- Computer memory originally defined as powers of 2 (i.e. kilobyte of memory was 2^{10} = 1024bits)
- 10^3 = 1000
- 2.4% lagrer

The Post PC Era



The Post PC Era



https://www.statista.com/statistics/272595/global-shipments-forecast-for-tablets-laptops-and-desktop-pcs/

Post-PC Devices

- **Personal Mobile Device (PMD)**
  - Battery Operated
  - Connections to the Internet wirelessly
  - Costs hundreds of dollars
  - Smart phones, tablets. Then electronic glasses?
- **Cloud computing**
  - Warehouse Scale Computers (WSC) i.e. giant data centres. Companies rent portions so they don't need their own - 100000 servers
  - Software as a Service (**SaaS**). Portion of software runs on a PMD and a portion runs in the Cloud
  - Amazon and Google are examples

https://blog.hubspot.com/service/iaas-paas-saas

Why to Study Architecture?

You will use computers extensively. Good to know how things work!

- **Performance**
  - User want their software to run as fast as possible
  - Understanding hardware can result in improvements by a factor of 2 to 200
  - Used to be about minimizing memory usage
  - Now, need to understand hierarchy of memory and parallel nature of processors

For cloud and PMD, need to minimize energy usage!

What You Will Learn in Course?

• How programs are translated from high-level languages into machine code and how the hardware executes them

- The hardware/software interface
- What determines program performance and how it can be improved
- How hardware designers improve performance and energy efficiency (and how software can help or hinder)
- What is parallel processing

Understanding Performance

- Algorithm design - determines number of operations executed
- Programming language, compiler, instruction set architecture - determines number of machine instructions executed per operation
- Processor and memory system - determine how fast instructions are executed
- I/O system - hardware and operating system (OS) - determines how fast I/O operations are executed

Seven Great Ideas in Computer Architecture

Use abstraction to simplify design

- Lower-level details hidden, so higher-levels are simpler

Make the common case fast

- Optimize the most often used parts of code, rather than the rare parts

Seven Great Ideas in Computer Architecture - II

Performance via parallelism

- Hardware designers improve performance by adding means to do operations in parallel
- Could mean multiple computation/execution units or even out of order or speculative computation

Performance via pipelining

- Very common form of parallelism
- Complex operations broken down into multiple (*n*) steps and then each step performed in a parallel sequence
- Allows first step of next operation to start, as soon as first operation step completes
- Once pipeline is full, completes *n* step operation once per clock cycle instead of once per *n* clock cycles

Seven Great Ideas in Computer Architecture - III

Performance via prediction

- If future instructions not known because of branch in code, make best guess and start in advance

Hierarchy of memories

- Want memory to be fast, large, and cheap as memory speed often shapes performance
- Fastest memory can be expensive and power and space hungry
- Conflict addressed by hierarchy where fastest, smallest, and most expensive at the top, and largest, slowest and cheapest at bottom

Dependability via redundancy

- Use redundant components that can help detect errors, and take over when failure occurs

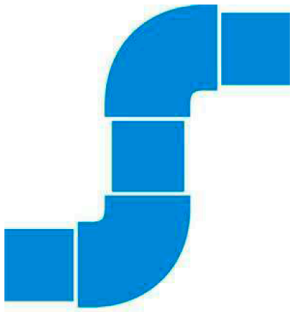
Textbook Icons

MOORE'S LAW

ABSTRACTION

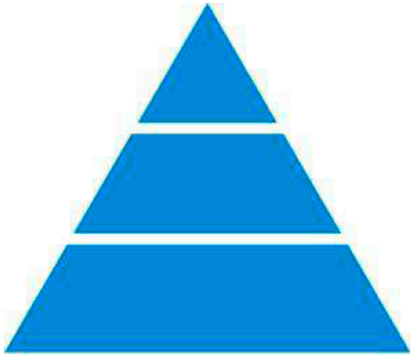
COMMON CASE FAST

PARALLELISM



PIPELINING

PREDICTION



HIERARCHY

DEPENDABILITY

Below Your Program

- Application software is written in a high-level language (HLL)
- Typically relies on software libraries that implement complex, often used operations
- Hardware can only execute simple low-level instructions
- To go from a complex application to primitive instructions requires several layers of software to translate high-level operation into simple computer instructions

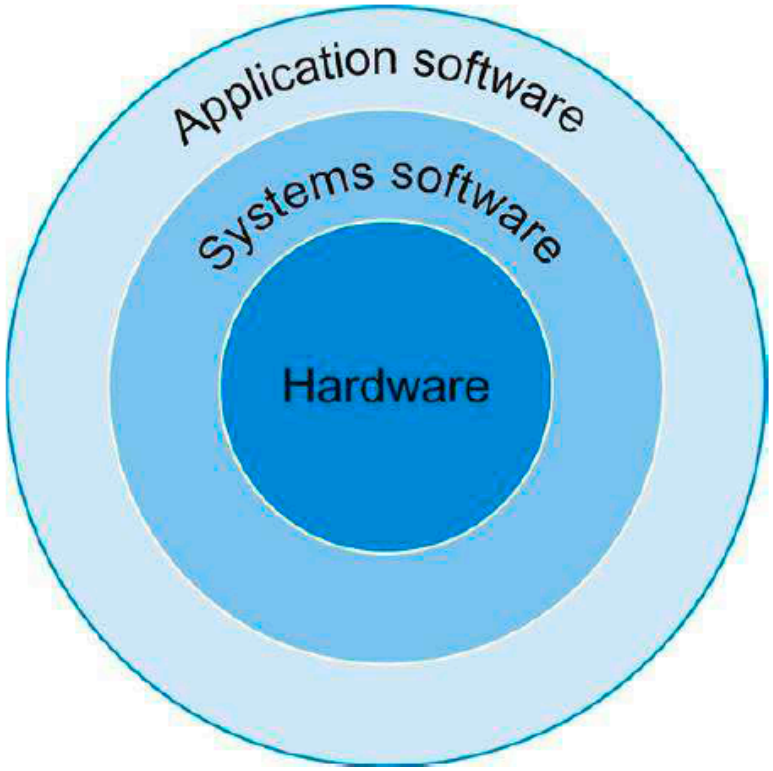
Below Your Program - II

Layers of software organized in hierarchical fashion

Application software - written in high-level language

System software

- Compiler: translates HLL code to machine code
- **Operating System:** service code
  - Provides high-level libraries to application
  - Handles input/output operations
  - Manages memory and storage
  - Schedules tasks & shares resources



Hardware

- Processor, memory, I/O controllers

Hardware Language

- To speak directly to hardware, you need to send the appropriate electronic signal
- Computer alphabet is just two letters, 0 (off) and 1 (on)
- We think of machine code as numbers in base 2, thus **binary** numbers
- You can encode anything as binary digits (called bits; 8 bits is called a byte), you just have to have enough of them.

Hardware Language II

- If you have n digits, you have 2^n unique combinations
  - For n=2, we have: 00, 01, 10, 11

- Computers execute our commands, called **instructions**, exactly as we tell them to

- An instruction is just a sequence of bits that the computer can understand. These sequences are referred to as **machine code**.
  - i.e. “1000110010100000” tells the computer to add two numbers

Assembly Language

- First programmers had to program computers by directly entering in the desired binary numbers for desired operations. Tedious!?

- They invented new notation that was closer to how humans think.

- They gave meaningful names to individual instructions (such as “add” for the add machine code) and a syntax to specify the needed parameters (such as the two numbers to add together).

Assembly Language II

- They then created a program called an **assembler** that would then translate these symbolic commands into actual machine code
  - i.e. programmer would write:
    ADD A,B
    and the assembler program would convert this to:
    "1000110010100000"

- They called this new symbolic language **assembly language**.

- Assembly language is still used to write low-level code that interacts directly with hardware such as in embedded applications and some operating system functions.

- It is also used when speed or control is paramount

High-Level Language (HLL)

- Assembly language is better, but still far from the notation that we would like to use to express a complex application
- Assembly requires too much detail – one line of assembly for each machine instruction
- High-level languages (such as C or Java) allow us to express complex operations in a more natural, compact way
- We use a program called a **compiler** to translate the HLL into either assembly or, more typically, directly into machine code
- HLL are more portable - machine code and assembly language are processor architect specific.

Levels of Program Code

High-level language

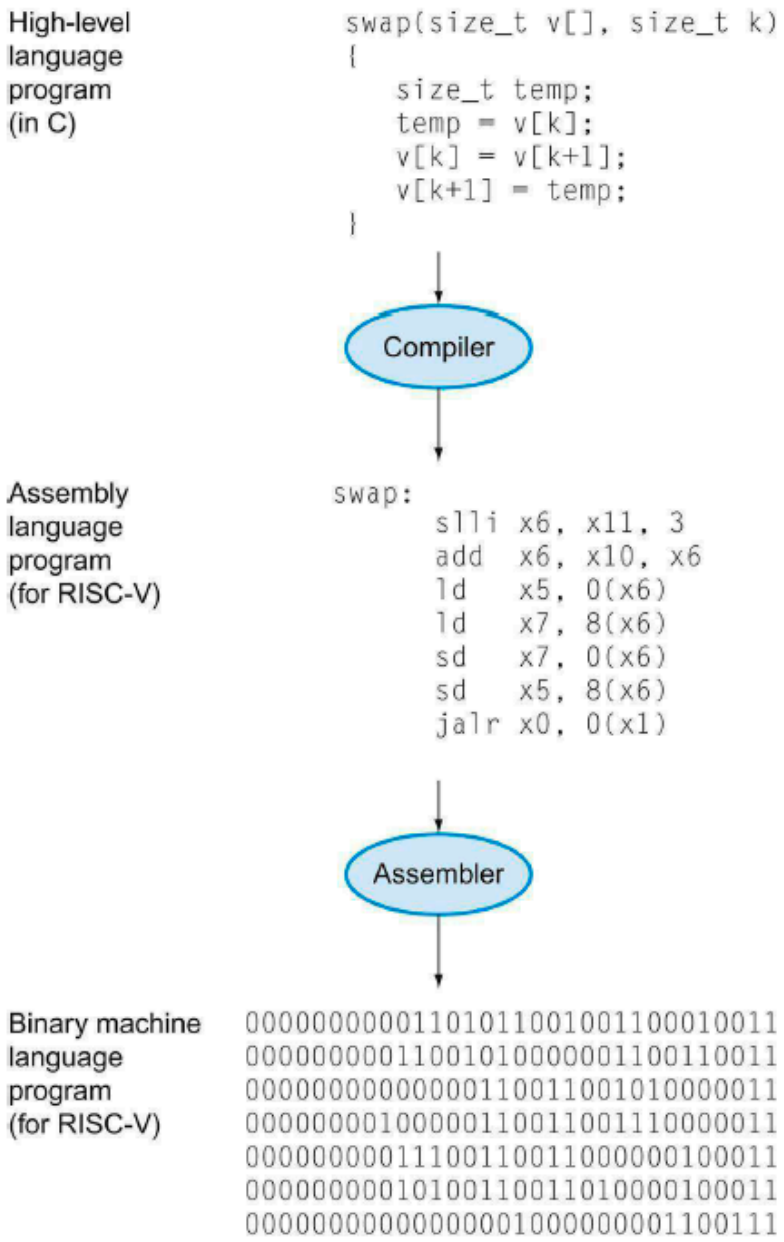
- Level of abstraction closer to problem domain
- Provides for productivity and portability

Assembly language

- Textual representation of instructions

Hardware representation

- Binary digits (bits) represented as zeros (off) and ones (on)
- Encoded instructions and data



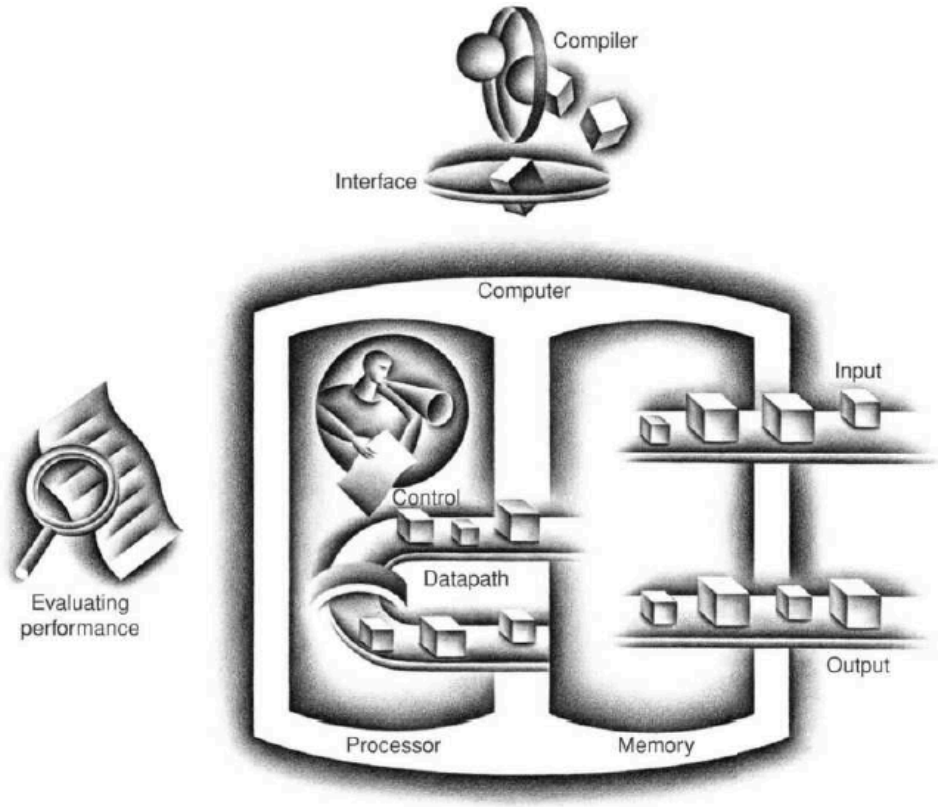
Components of a Computer

Same components for all kinds of computer

- Desktop, server, tablets

When we think of a computer, we think of a device that contains:

- Input (1) and output (2) devices
- Memory (3) for storing programs and data
- Processor that consists of a datapath (4) and a control unit (5)

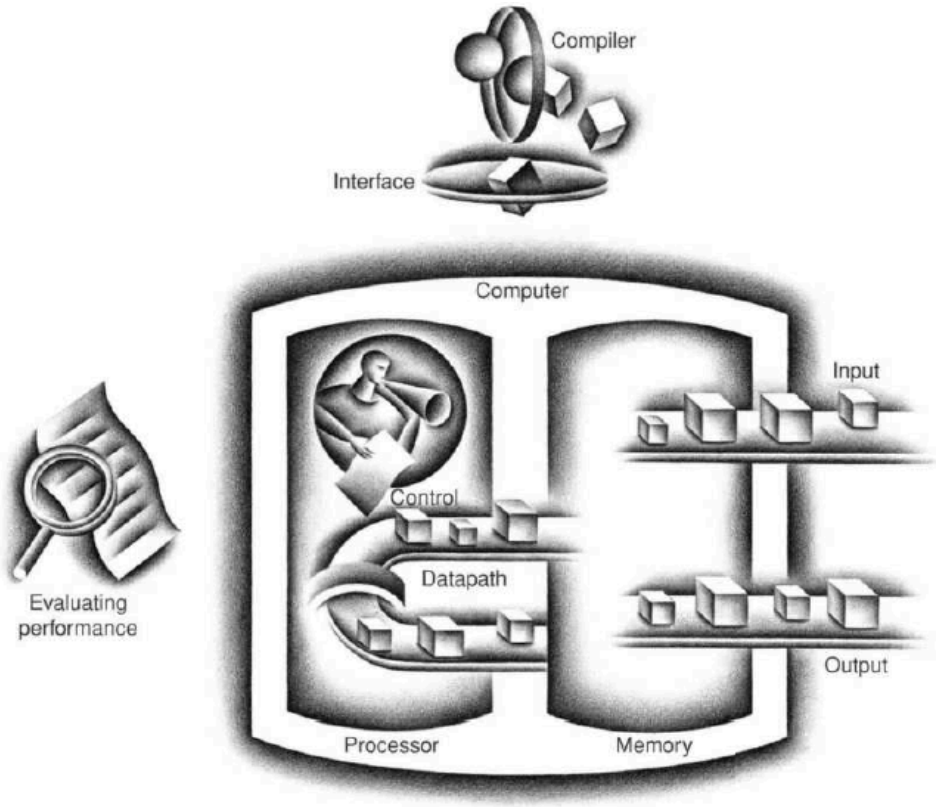


Components of a Computer II

Input/output includes

- User-interface devices: Display, keyboard, mouse
- Storage devices: hard disk, CD/DVD, flash
- Network adapters for communicating with other computers

Memory where programs and their data are kept when they are running

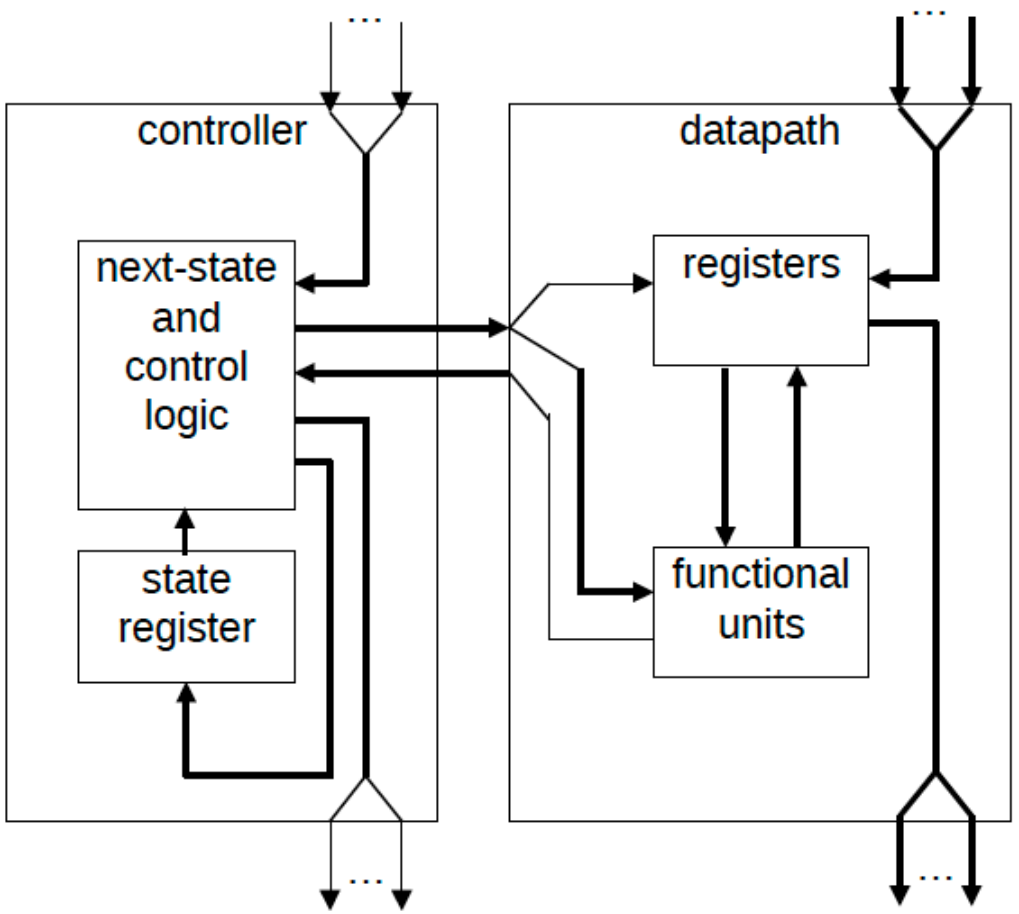


Components of a Computer III

The processor includes

- Datapath: The component of the processor that performs arithmetic operations.

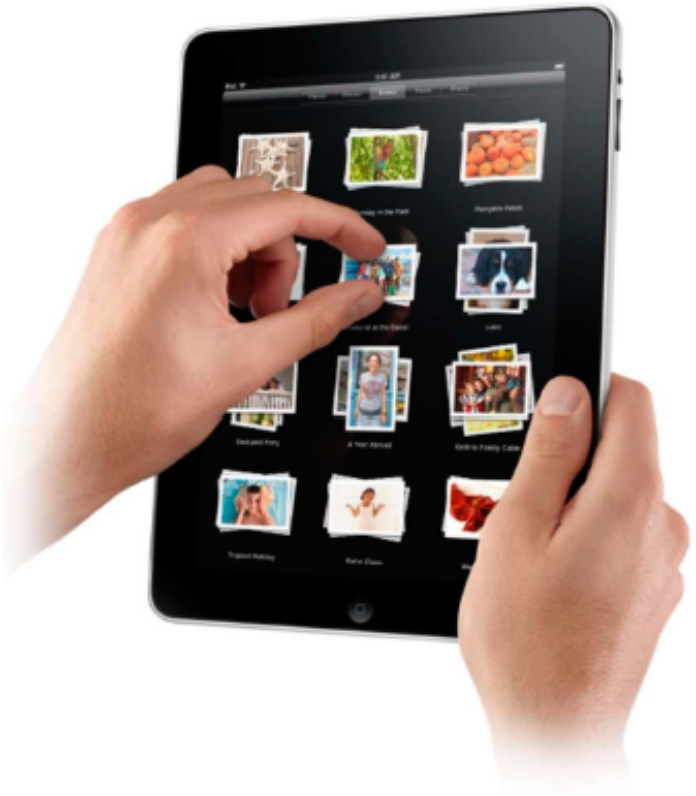
- Control unit (controller): this is the part that keeps track of what needs to be done, and configures the datapath to perform the desired actions to implement the current machine code instruction



Touchscreen

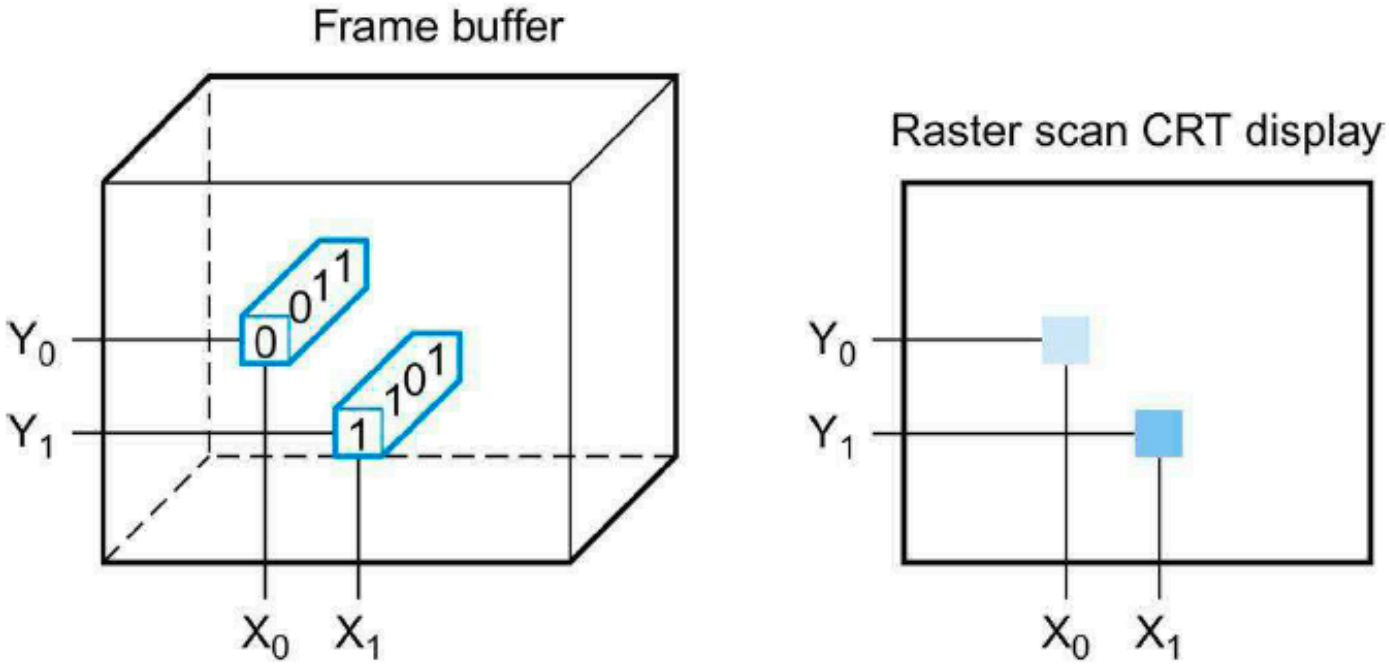
- For PostPC devices, a touchscreen supersedes keyboard and mouse

- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously

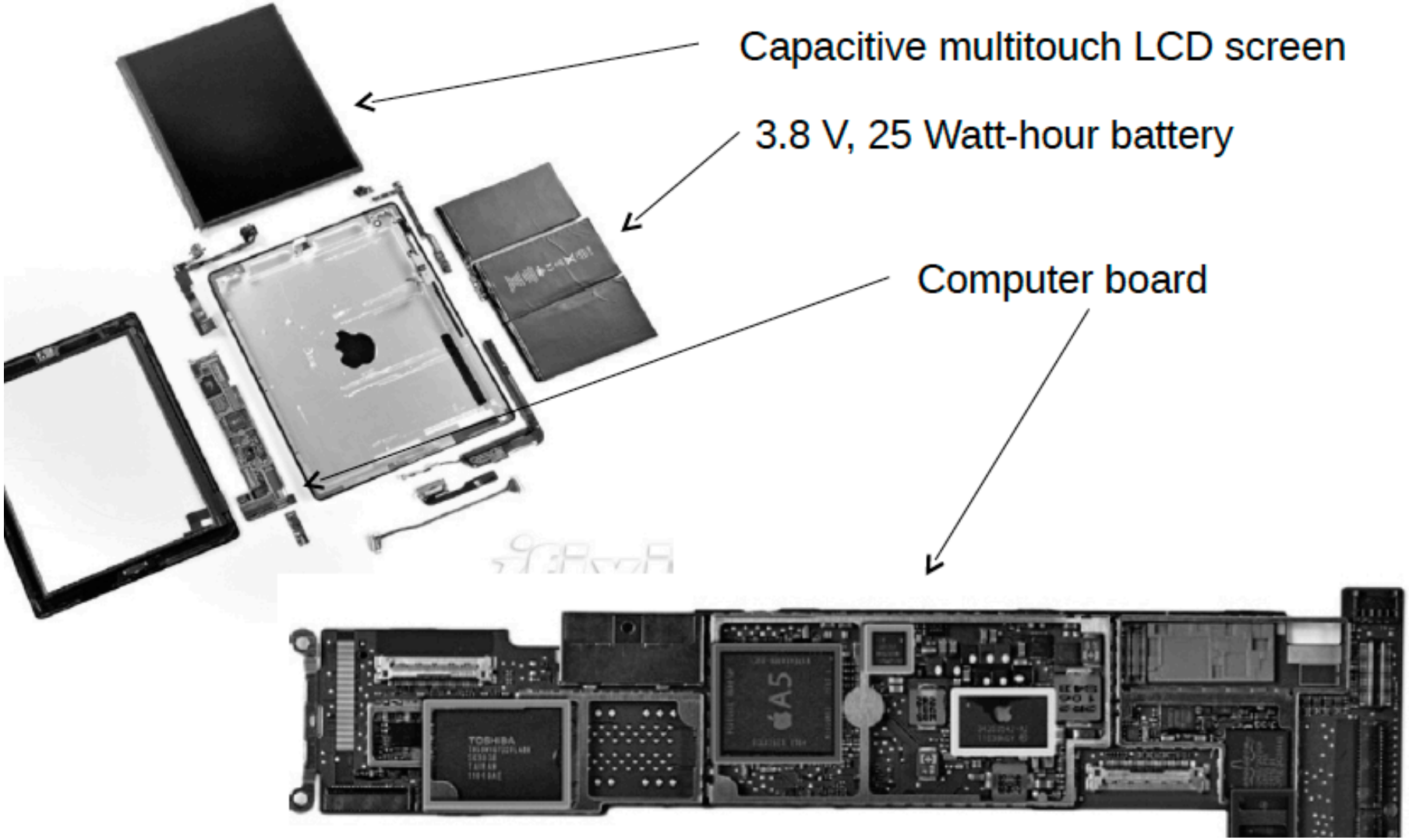


Through the Looking Glass

- A graphics display is today typically an LCD screen
- Image composed of a matrix of picture elements called **pixels**
- A color display might use 8 bits for each of the three colors (red, blue, green), for 24 bits per pixel



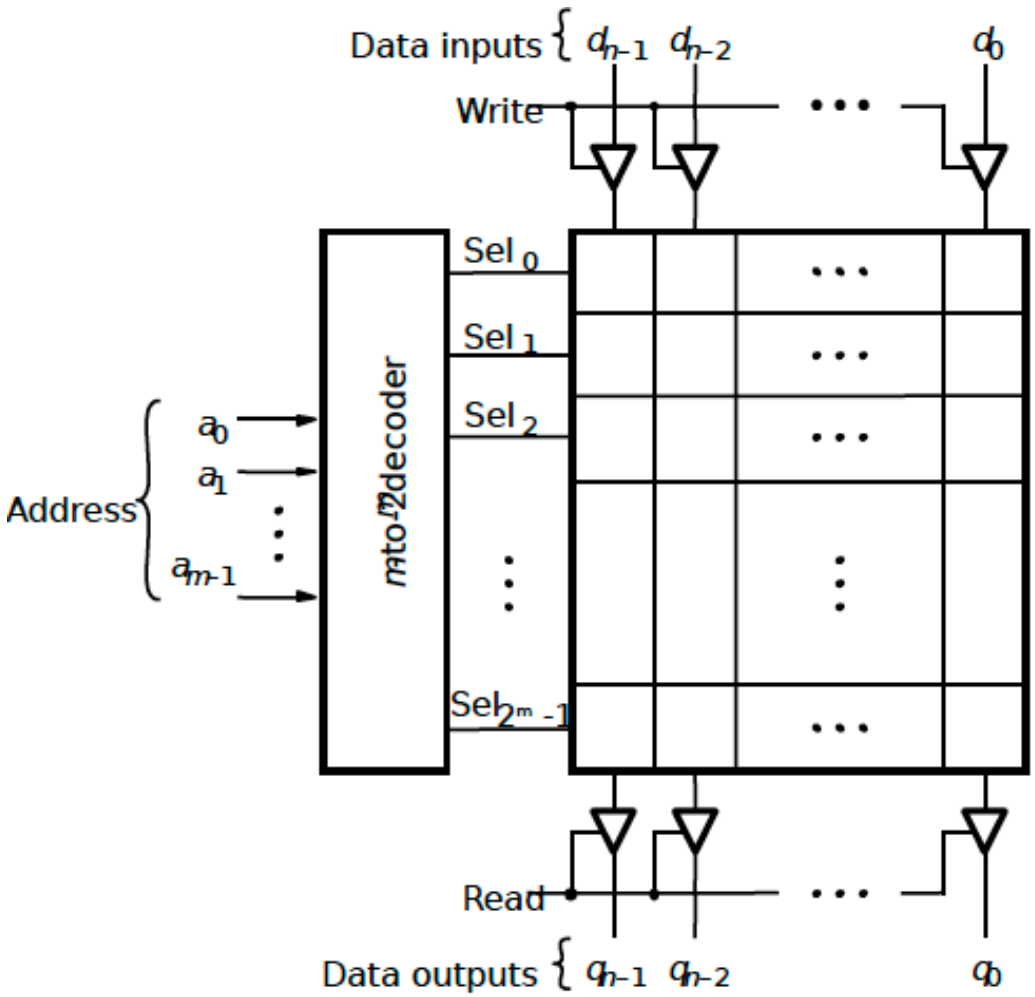
Opening the Box



Apple iPad 2 tablet computer

Main Memory

- Main Memory is composed of r**andom access memory** (RAM)
- RAM can be read from and written to
- It is volatile. The data is lost when power is turned off
- Any memory location can be directly accessed by applying the correct binary address to the *m* address lines
- Each memory location contains *n* bits of data



Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: tells datapath, memory, I/O devices what to do
- Two main types of RAM
  - DRAM: stands for dynamic RAM. Used for main memory as high density, thus lower cost. Data needs to be periodically refreshed.
  - SRAM: stands for static RAM. Faster than DRAM, but less dense, thus more expensive.
- Cache memory
  - Small fast SRAM memory for immediate access to data

Inside the Processor (CPU)

- Apple A5 Chip
- Processor is also called the **central processor unit (CPU)**
- Contains two ARM processors, or “cores”
- Contains a PowerVR **graphical processor unit (GPU)**

Compare to A12Z Chip:

https://en.wikipedia.org/wiki/Apple_A12Z

Abstractions

Abstraction helps us deal with complexity

- Hides lower-level detail

Instruction set architecture (ISA) is an important one

- ISA provides the hardware/software interface
- It includes everything a programmer needs to know to make a binary machine language program work properly

Abstractions II

Application binary interface

- Operating systems will encapsulate details of low-level system functions such as doing I/O, allocating memory etc.
- Hides details from the programmer
- The ISA plus the Operating system's interface is called the Application Binary Interface **(ABI)**

An implementation of an ISA is hardware that obeys the architecture abstraction. This allows many implementations of different cost and performance to run the same software.

RISC-V

A Safe Place for Data

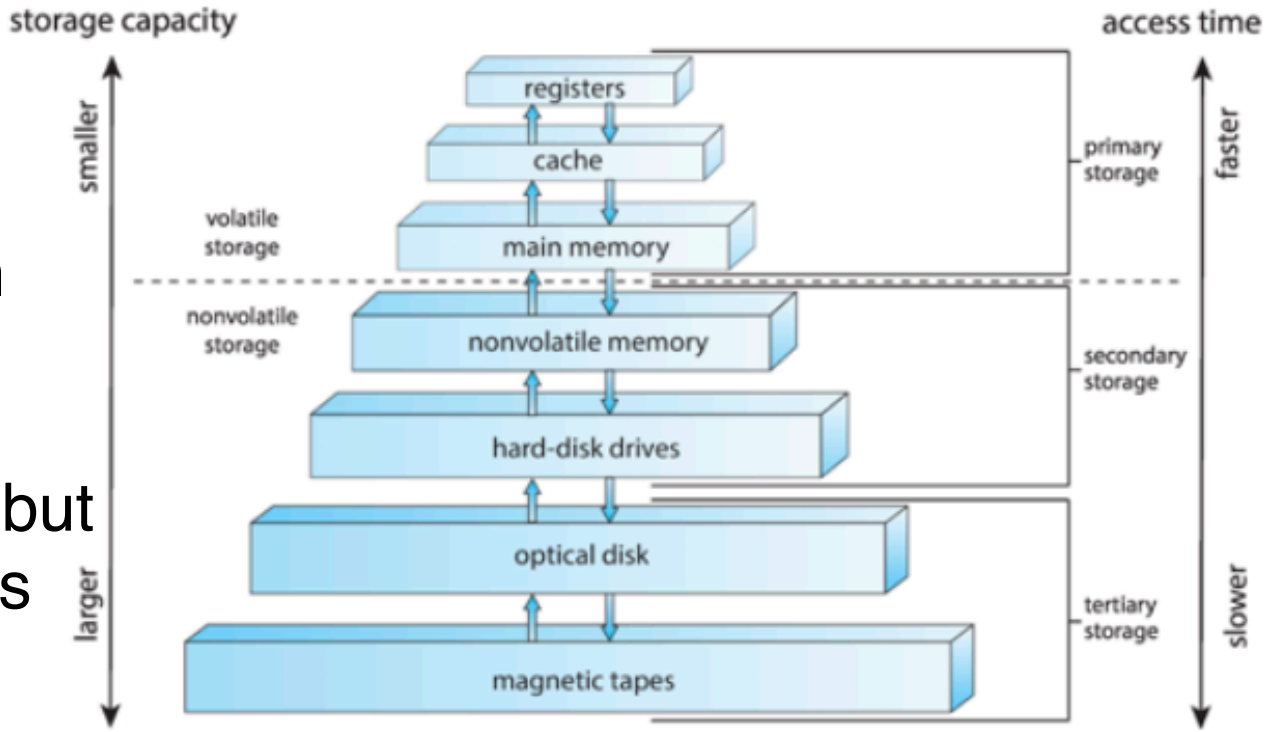
Volatile main memory (RAM)

- Loses instructions and data when power off

Non-volatile secondary memory used for long term storage

Slower than main memory but cheaper on a per byte basis

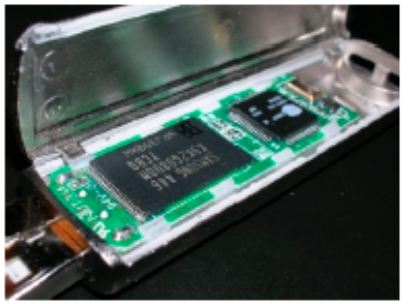
Forms the next layer of memory hierarchy



A Safe Place for Data

Magnetic Disk
- Primary form of non-volatile memory for computers
- Fast, cheap, and reliable

Flash Memory
- Used by PMD as smaller, and more rugged and power efficient
- Wears out after 100,000 to 1,000,000 writes



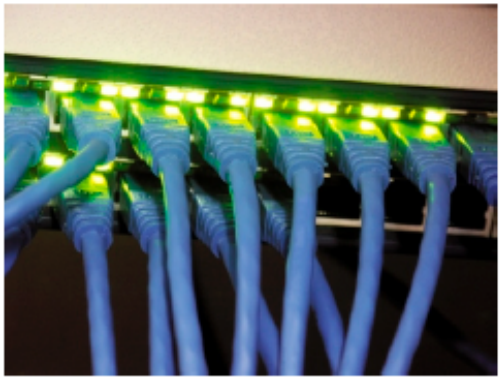
Optical disk (CDROM, DVD)
- slowest, but cheapest option

Computer Networks

Allow computers to exchange data with computers nearby and around the world

Key advantages:

- Communication: computers exchange data at high speeds
- Resource sharing: Computers on network can share I/O devices
- Non-local access: users can access computers remotely



Types of Computer Networks

Networks vary based on cost and performance, as well as if they are a “wired” solution or not

Local area network (LAN): e.g. Ethernet

- Interconnected with switches that provide routing and security

Wide area network (WAN): e.g. the Internet

- Span continents and usually based on optical fibers and leased from telecommunication companies

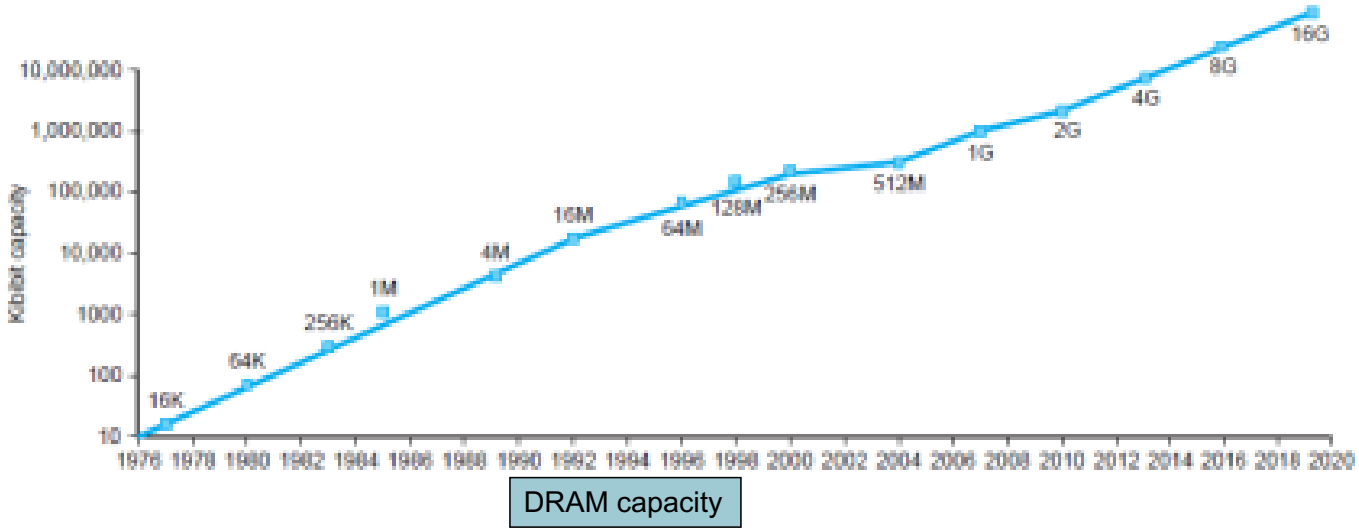
Wireless network: e.g. WiFi (IEEE 802.11), Bluetooth

- Can be a LAN, or device-to-device technology

Technology Trends

Electronics technology continues to evolve

- Increased capacity and performance
- Reduced cost

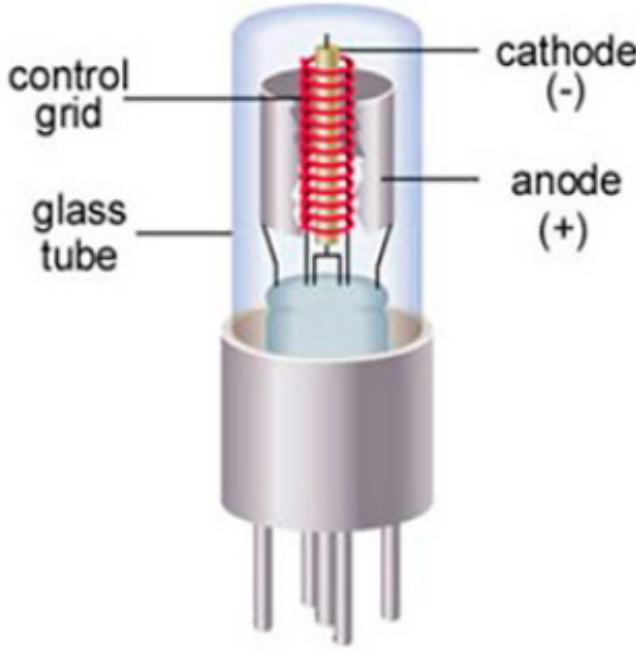


Year	Technology	Relative
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2400000
2013	Ultra large scale IC	250000000000

Vacum Tubes

The original building block of computers
Consists of three elements in a glass tube

- Cathode that emits electrons
- Anode that receives them
- Control grid that only allows electrons to flow when a voltage applied.



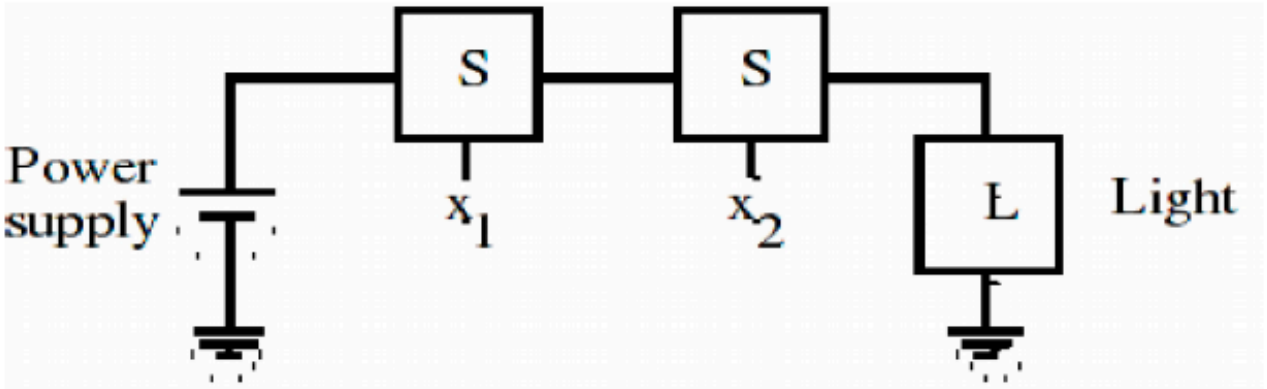
Acts like a switch that turns current on or off based on the voltage applied. Using a **switch**, one can create a logic **AND**, **OR**, **NOT** functions

Disadvantage:

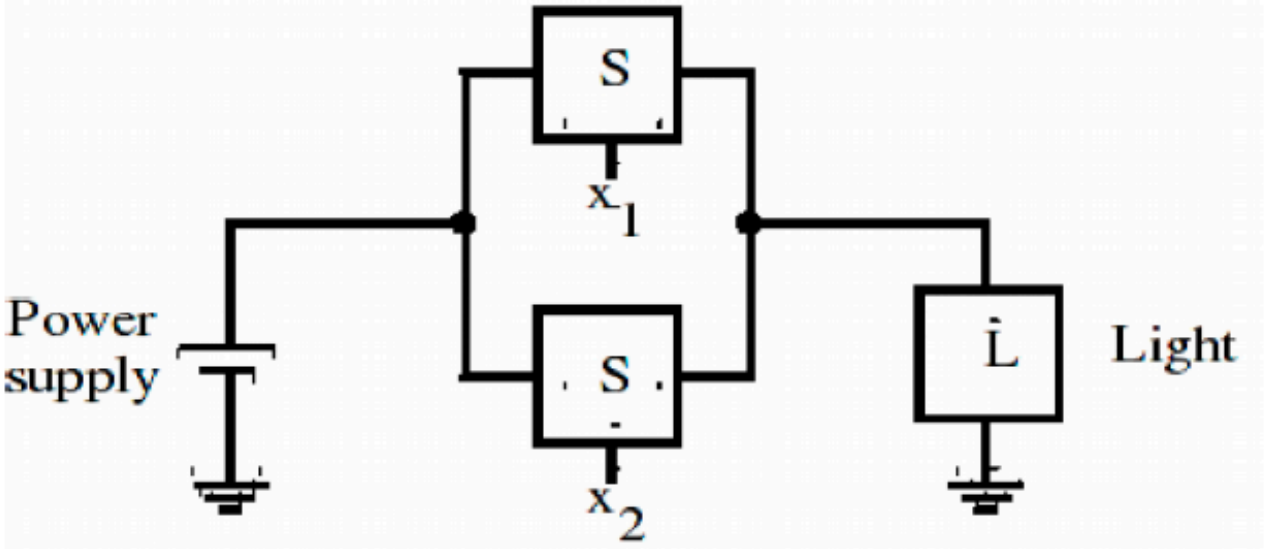
- Cathode must be heated produce electrons
- Heat means power consumption and wear and tear

https://www.youtube.com/watch?v=PbJ1GZMi_ho

Switches as Logic Function



(a) The logical AND function (series connection)

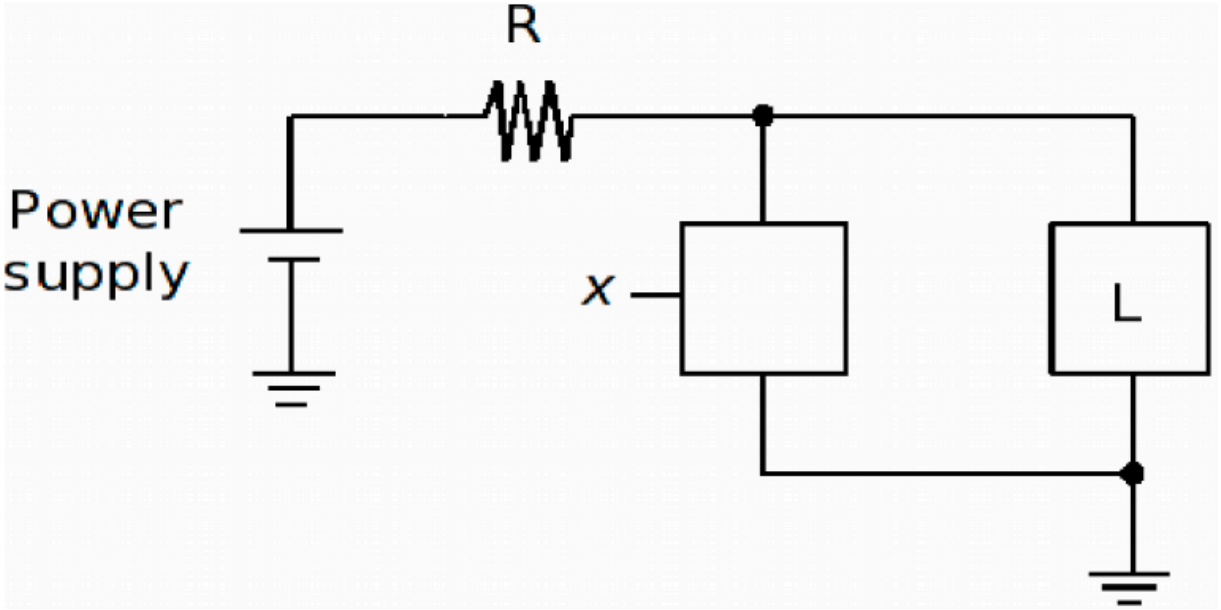


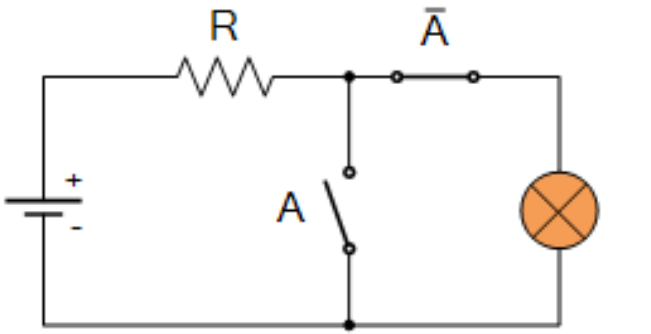
(b) The logical OR function (parallel connection)

Switches as Logic Function

- For **logical NOT**, the output function is the logical negation or the complement of the input variable

- The output is true (1) if the input variable equals false (0), else the output is false



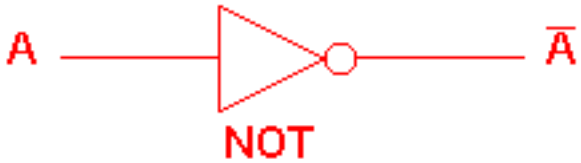


Switch A – Open = "0", Lamp – ON = "1"
Switch A – Closed = "1", Lamp – OFF = "0"

Basic Logic Gates

2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1



NOT gate	
A	$\overline{A}$
0	1
1	0

Semiconductor Technology

Transistors replaced vacuum tubes

They are lower power, more reliable, and can be produced far smaller (thus more dense)

Created out of a semiconductor called silicon

Called a semiconductor as it is normally a poor conductor of electricity

However, if an electric field is applied correctly, it can become a very good conductor

Types of Transistors

Most common technology used is called **Metal Oxide Semiconductor Field-Effect Transistors (MOSFET)**

Two distinct types, **NMOS (negative channel)** and **PMOS (positive channel)**

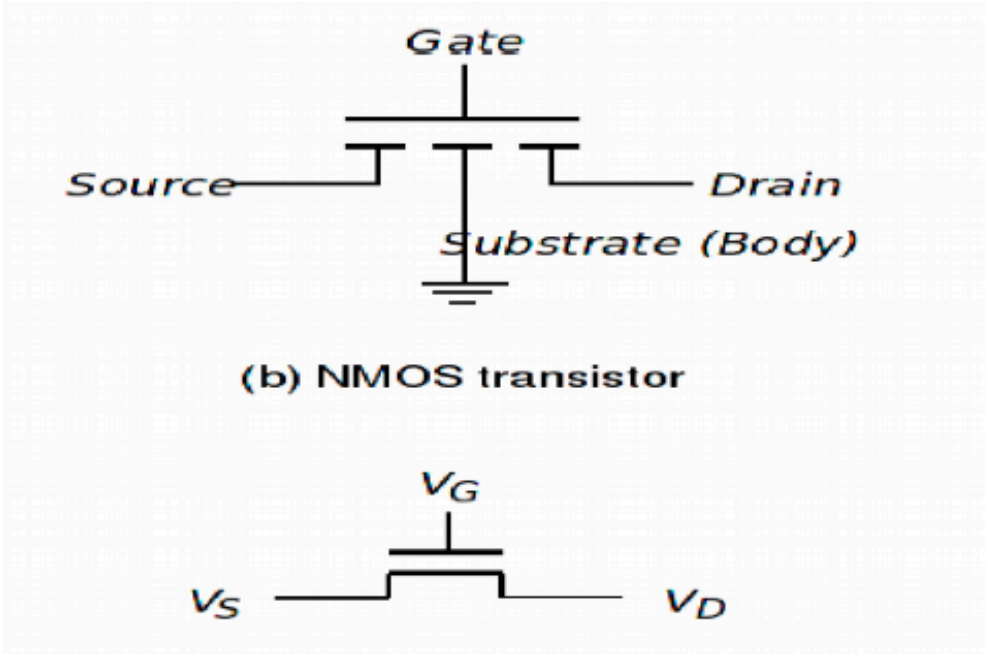
Both types contain *n-type* (silicon doped so that the charge carriers are negatively charged) and *p-type* (silicon doped so that the charge carriers are positively charged)

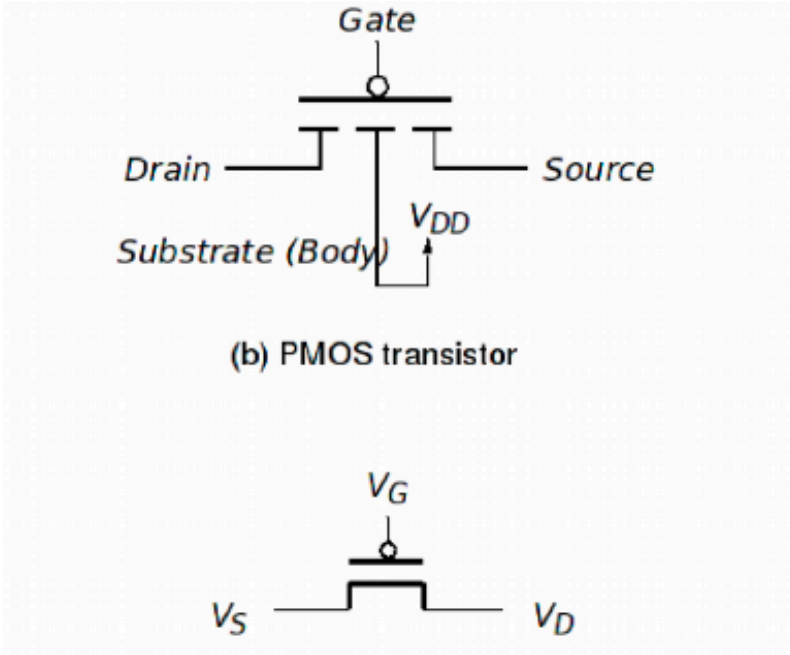
Electronic gates can be created out of either NMOS or PMOS transistors

Types of Transistors II

We can view an NMOS/PMOS transistor as a switch that conducts or not depending on the value (VG) applied to the gate input.

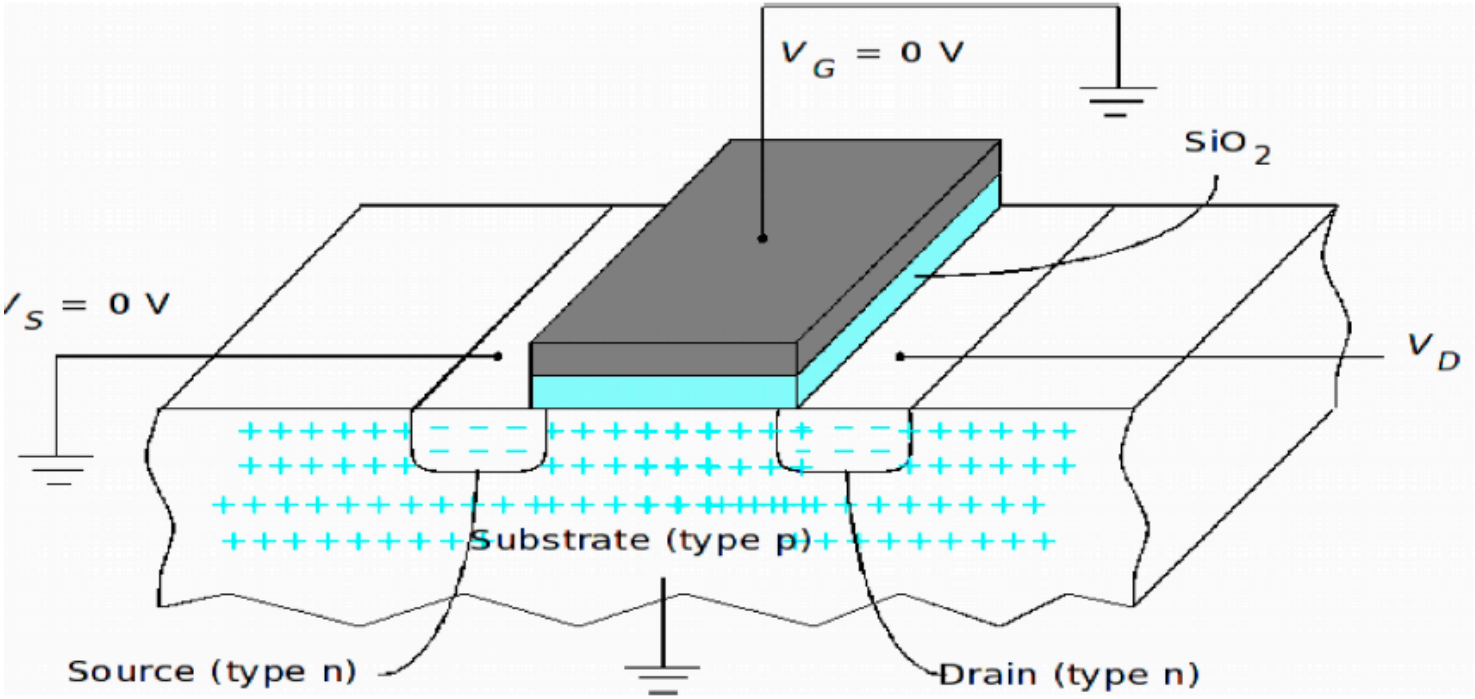
An NMOS (PMOS) “switch” is open (closed) when VG = 0V, and closed (open) when VG = 5V.





Example of an NMOS Transistor

A transistor is built upon a silicon wafer by adding different types of silicon, conductors, and insulators by means of chemical processes

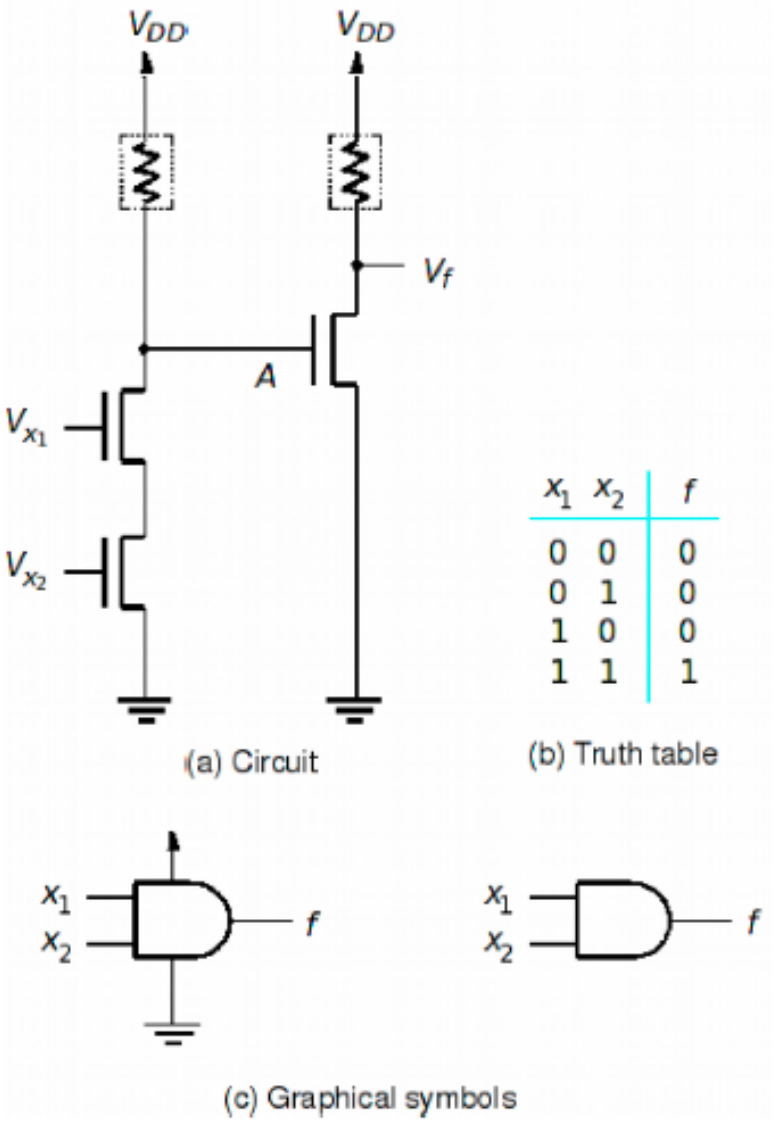


CMOS Gates

When a logic gate (see right) made of only NMOS or only PMOS transistors is conducting, current is flowing and consuming power

For CMOS (Complementary MOS) technology, we build gates using both NMOS and PMOS transistors

Advantage of CMOS: Under steady state conditions (every input voltage stable at either 0V or 5V) there are virtually no current flows



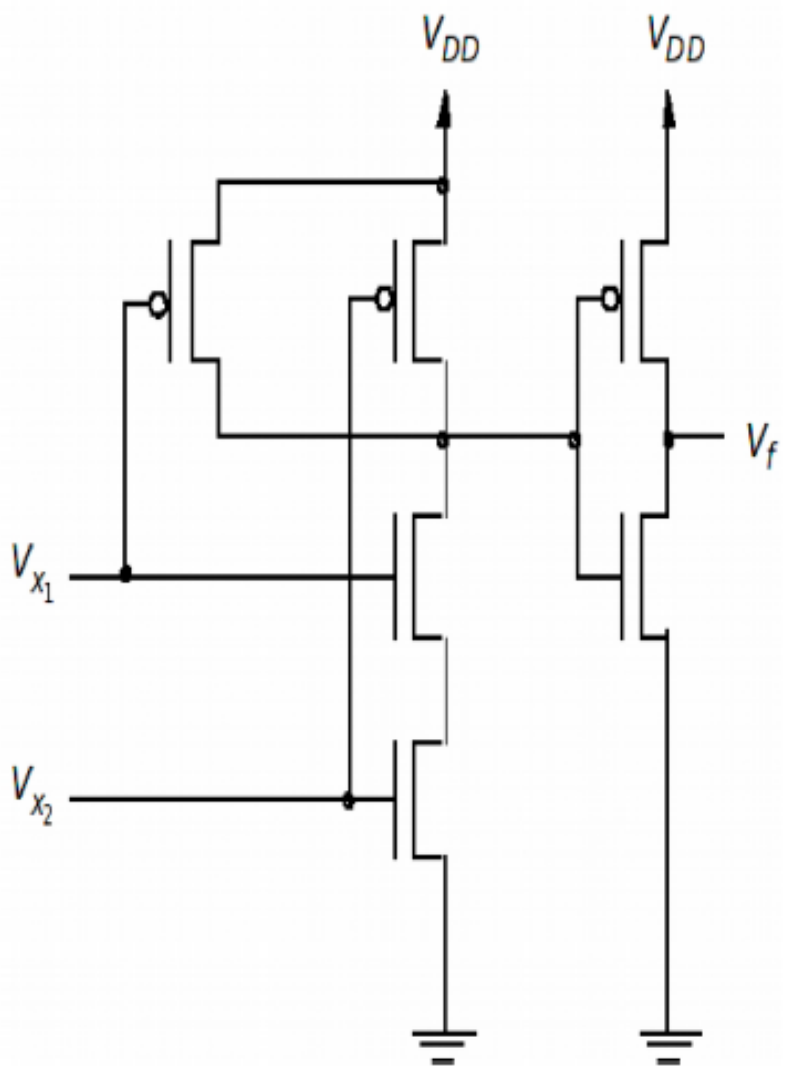
CMOS AND Logic Gate

The CMOS AND gate contains NMOS transistors at the bottom and PMOS transistors at the top

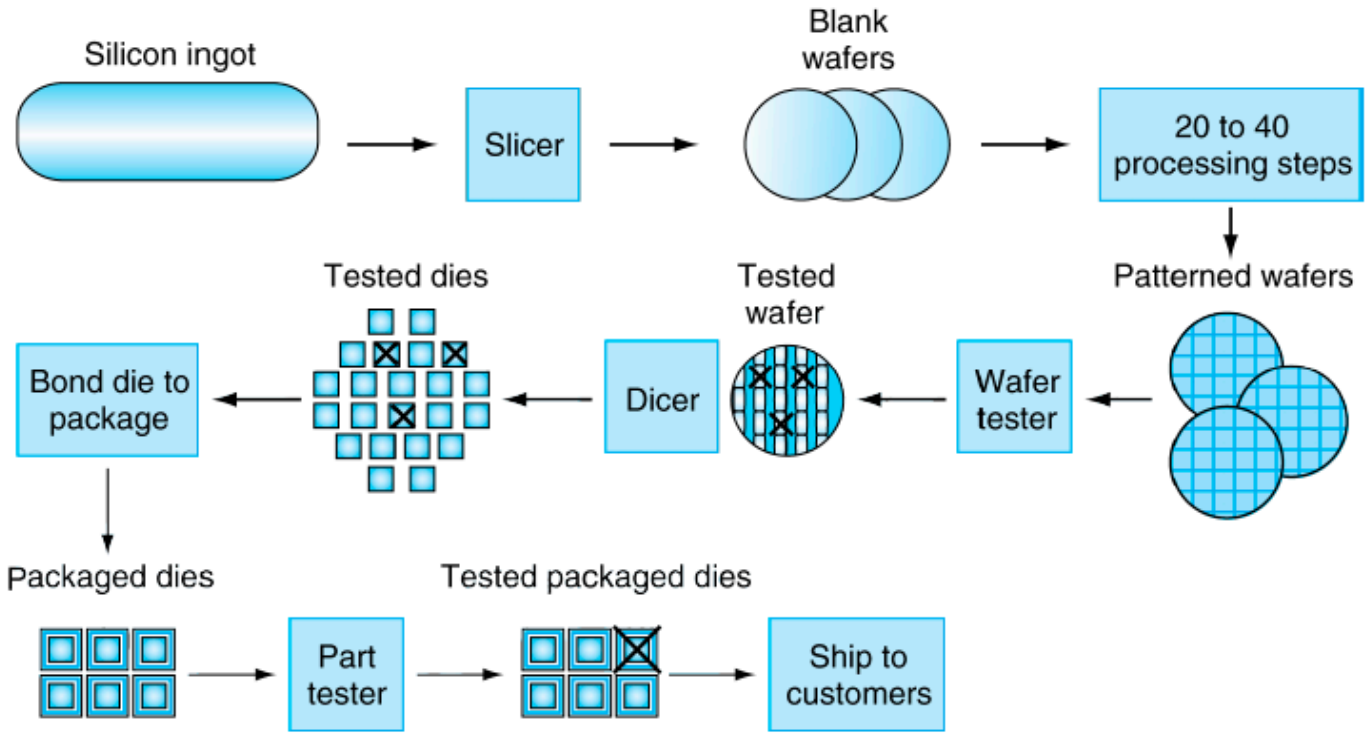
The NMOS part implements a complementary logic function to the PMOS part, thus only one part ever conducts at any given time

When NMOS conducts, output pulled to 0V, but as the PMOS part is an open circuit, no current flows.

When PMOS part conducts, output is pulled to 5V, but no current flows.



Manufacturing ICs

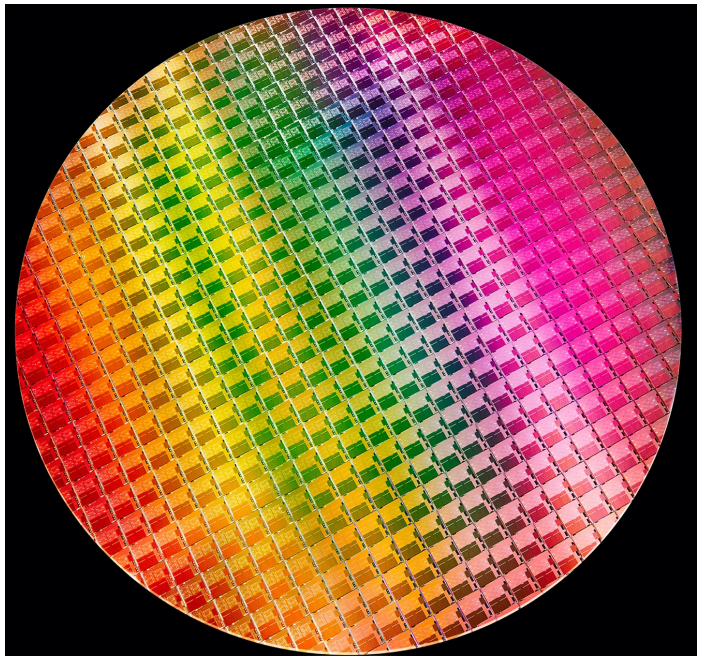


- Many independent components are created on a single wafer so that defects in one area will not cause others to fail

- Yield: proportion of working dies (chips) per wafer

Intel® Core 10th Gen

300mm wafer, 506 chips,
10nm technology



Each chip is 11.4 x 10.7 mm

Cost of integrated circuit rises quickly as die size increases due to lower yield and fewer dies fitting on wafer

Integrated Circuit Cost

$$Cost\ per\ die = \frac{Cost\ per\ wafer}{Dies\ per\ wafer\ \times\ Yelid}$$

Yield - The percentage of good dies from the total number of dies on the wafer.

$$Dies\ per\ wafer \approx \frac{Wafer\ area}{Die\ area}$$

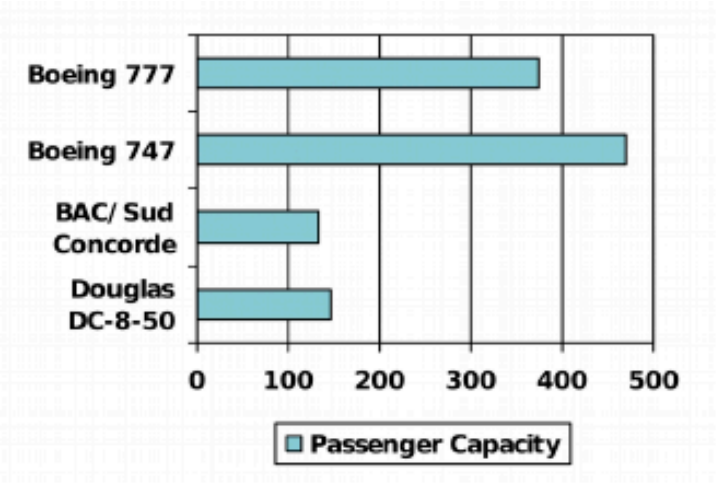
$$Yield = \frac{1}{(1 + (Defects\ per\ area\ \times\ \frac{Die\ area}{2}))^2}$$

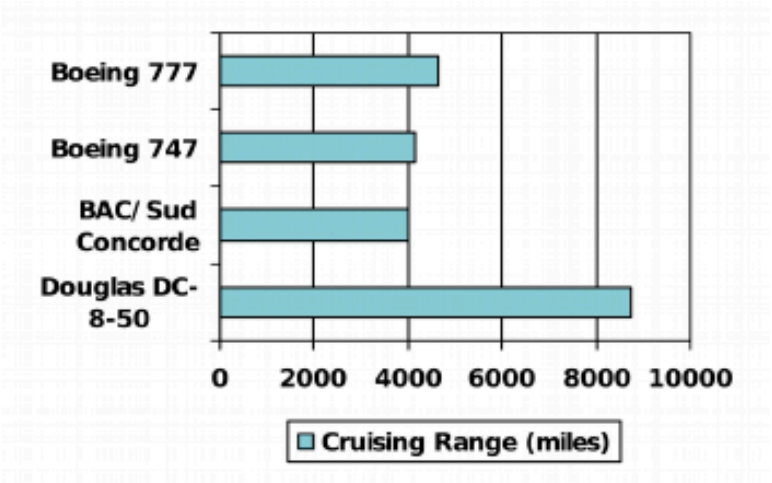
Nonlinear relation to area and defect rate

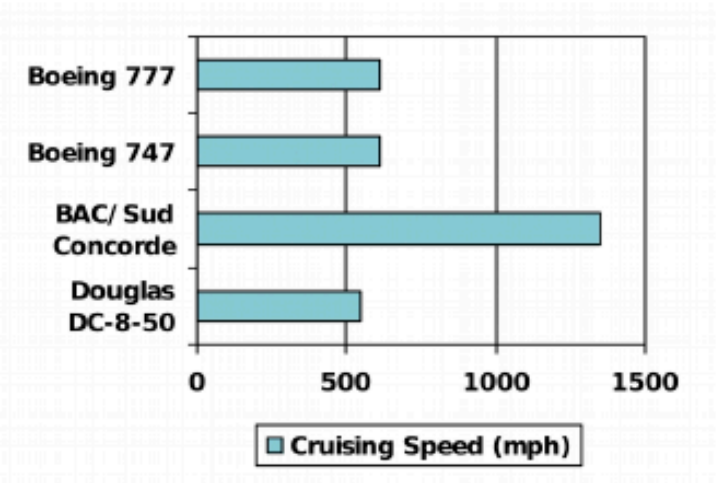
- Wafer cost and area are fixed
- Defect rate determined by manufacturing process
- Die area determined by architecture and circuit design

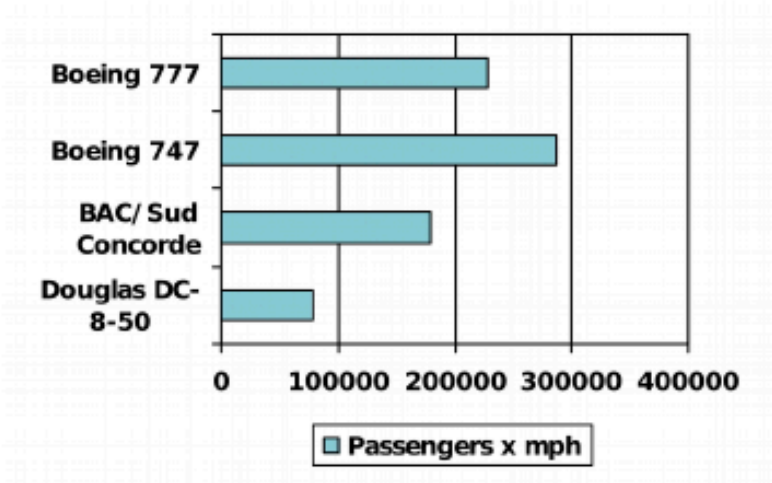
Defining Performance

Which airplane has the best performance?









Response Time and Throughput

Response time

- How long it takes to do a task

Throughput

- Total work done per unit time
  - - e.g., tasks/transactions/… per second

How do they differ?

- Response or "execution time," focuses on the time of a single task in isolation
- Throughput or "bandwidth," focuses on the average time to perform multiple tasks over a given amount of time
- This allows throughput to take advantage of parallelism in the operating system and hardware.

Response Time and Throughput II

How are response time and throughput affected by:

- Replacing the processor with a faster version?
- Adding more processors?

We'll focus on response time for now…

Relative Performance

$$Performance = \frac{1}{Execution\ Time}$$

"Computer *X* is *n* time faster than Computer *Y* if"

$$\frac{Performance_X}{Performance_Y} = \frac{Execution\ time_Y}{Execution\ time_X} = n$$

Example: time taken to run a program

- 10*s* on A, 15*s* on B
- Execution Time$_B$ / Execution Time$_A$ = 15s / 10s = 1.5
- So A is 1.5 times faster than B

Measuring Execution Time

Elapsed time: the time found by measuring the start and end time of the task

- Total response time of task, including all aspects:
  - Processing, I/O, memory access, OS overhead, idle time
- Determines system performance as it includes factors other than just time to execute instructions
- A system may be doing several tasks at once, and may optimize for throughput, as opposed to minimizing our programs execution time

Measuring Execution Time II

We often want to distinguish between execution time and the time over which the CPU has been working on our task

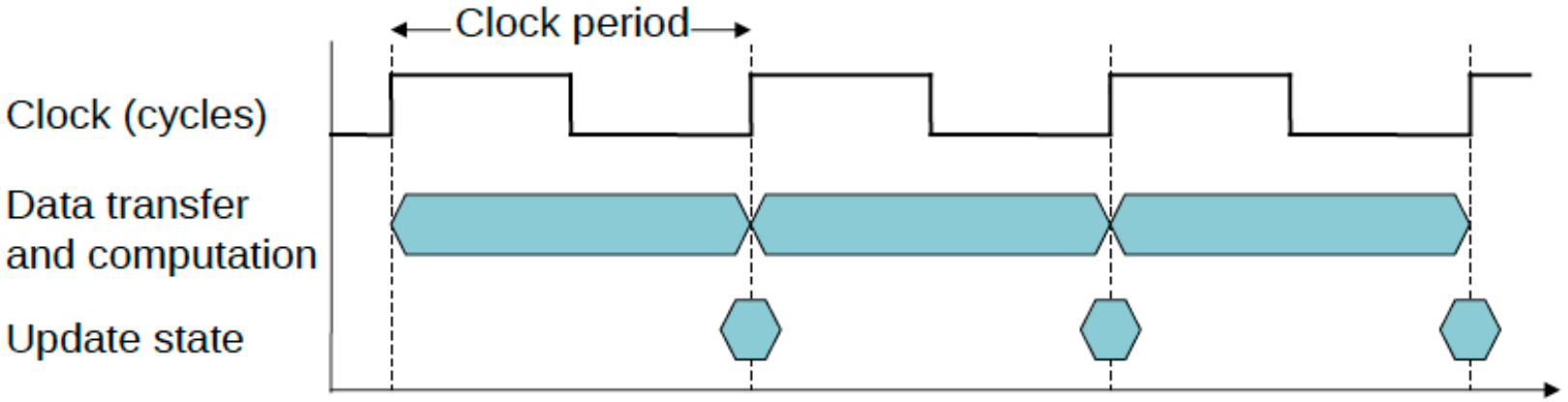
- CPU time is defined to be:
  - The time spent processing a given job
    - Discounts I/O time, other jobs' shares

- CPU time comprises user CPU time (time spent on the task itself) and system CPU time (time spent by the OS performing actions on behalf of the task)

- We use term CPU performance to refer to user CPU time and use system performance to refer to elapsed time on an unloaded system

CPU Clocking

Operation of digital hardware is governed by a constant-rate clock



Clock period: duration of a clock cycle

- e.g., 250ps = 0.25ns = 250×10^{-12}s

Clock frequency (rate): cycles per second

- Frequency is the inverse of clock period.
- e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

CPU Time

$$CPU\ Time = CPU\ Clock\ Cycles \times Clock\ Cycle\ Time = \frac{CPU\ Clock\ Cycles}{Clock\ Rate}$$

Here we are calculating how much time the CPU spends executing instructions from our program

"Clock cycle time" means clock period and "clock rate" means clock frequency

Performance can be improved by:

- Reducing number of clock cycles required by program
- Increasing clock rate (which reduces clock period)

Hardware designer must often trade off clock rate against cycle count

Instruction Count and CPI

$$CPU\ Clock\ Cycles = Instruction\ Count \times Cycles\ per\ Instruction$$

$$CPU\ Time = Instruction\ Count \times CPI \times ClockTime = \frac{Instruction\ Count \times CPI}{ClockRate}$$

Cycles per Instruction (CPI) is an average cycles per instruction for program

- Determined by CPU hardware
- If different instructions have different CPI
  - Average CPI affected by instruction mix

Instruction Count (IC) for a program is determined by:

- The program itself, the Instruction Set Architecture ISA and the compiler

CPI in More Detail

If different instruction classes take different numbers of cycles

$$Clock\ Cycles = \sum_{i=1}^{n} (CPI_i \times Instruction\ Count_i)$$

Weighted average CPI

$$CPI = \frac{Clocl\ Cycles}{Instruction\ Count} = \sum_{i=1}^{n} (CPI_i \times \underbrace{\frac{Instruction\ Count_i}{Instruction\ Count}}_{\text{Relative frequency}})$$

CPI Example

Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

Which code sequence executes the most instructions?
Which will be faster?
What is the CPI for each sequence?

Performance Summary

$$CPU\ time = Instruction\ Count \times CPI \times Clock\ Cycle\ Time$$

Changing the items below can affect performance as follows:

- Algorithm: affects IC, possibly CPI
- Programming language: affects IC, CPI
- Compiler: affects IC, CPI
- Instruction set architecture (ISA): affects IC, CPI, Tc (clock period)

Power Usage

Increased power usage means:

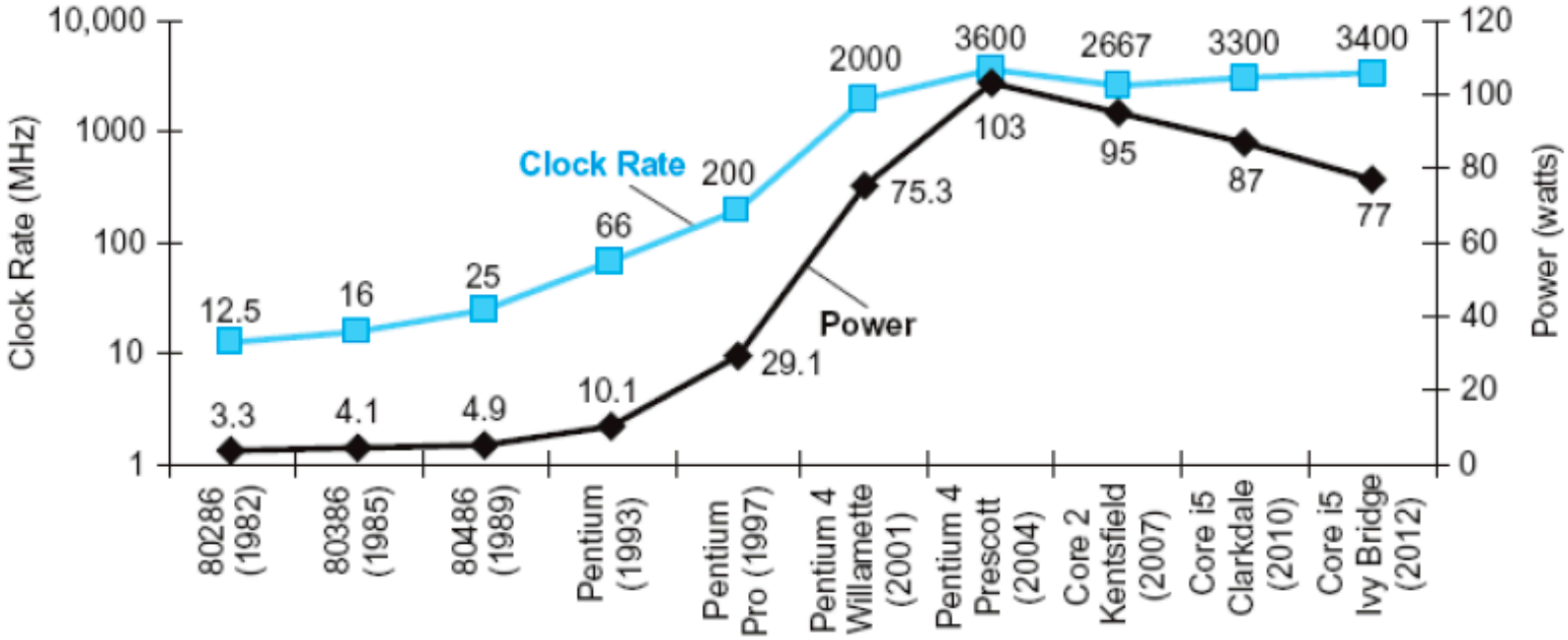
Increased heat production

- Limit to what we can cool in a commercial PC
- Cooling costs; for a data centre can be very expensive
- If a CPU overheats, it will cause errors, and will decrease time before it permanently fails

Increased electricity bills to run and cool a CPU, particularly for data centres with 100,000 servers

Increased energy usage, which means lower battery life (important for PMD)

Power Trends



- Shows increase in clock rate and power for Intel processors over 30 years
- Clock frequency increased 1000 times
- Power usage by processors ONLY increased by 30 times
- Why?

Power Equation for CMOS

- In CMOS IC technology, the primary energy consumption occurs when transistors switch state, so-called dynamic energy
- Dynamic energy depends on the capacitive loading of each transistor
- Power also depends on voltage of the circuit and the CPU clock rate
- The formula for **dynamic power** is:

$$Power = 1/2 \times Capacitive\ load \times Voltage^2 \times Frequency$$

x30 | 5V -> 1V | x1000

Reducing Power

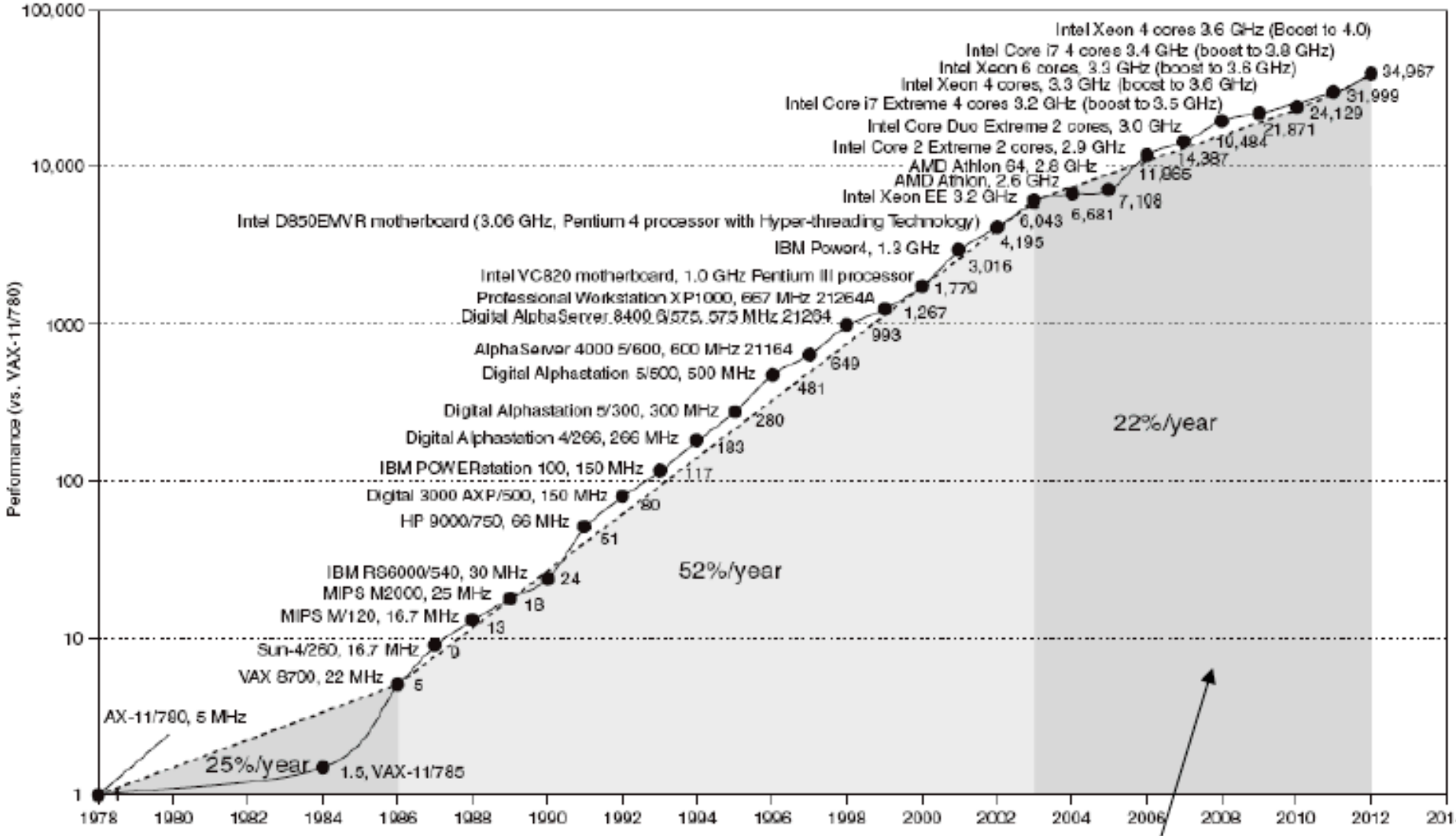
Hardware designers have hit the **“power wall”**

- We **can’t reduce voltage further**
  - Reducing voltage further means there is too much leakage current (unwanted current flow when the transistor should be off)
  - Leakage currently accounts for 40% of power consumption in server chips. This is referred to as **static power** consumption.

- We can’t remove more heat
  - Already attaching large cooling devices and turning off parts of chips but are running out of tricks

How else can we improve performance?

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

https://en.wikipedia.org/wiki/Normal_distribution

Multiprocessors

Industry changed focus:

- From decreasing response time of one program on a single processor..
- To shipping computers with multiple cores on a single chip

Multicore microprocessors

- More than one processor per chip
- Each core (processor) is simpler than the previous single core chips
- Focus is more on throughput that individual response time

Multiprocessors II

Before, could rely on improvements in hardware, architecture, and compilers to double program performance every 18 months

Now, multiple cores require explicitly parallel programming

- Instruction level parallelism
  - This is when hardware executes multiple instructions at once
  - Hidden from the programmer

- Adding parallelism to code is hard to do as it requires:
  - Programming for performance, as opposed to just correct behavior
  - Program behavior needs to spread across processors such that they are all equally busy (load balancing)
  - Optimizing communication and synchronization

Additional Reading

Benchmarking

Read Section 1.9 for your own interest

Fallacies and Pitfalls

Read Section 1.10 for your own interest

Concluding Remarks

Read Section 1.11 on your own

Thank You

https://en.wikipedia.org/wiki/Fallingwater