COMP2300-COMP6300-ENGN2219
Computer Organization & Program Execution

Convener: Shoaib Akram
shoaib.akram@anu.edu.au

Australian National University

Shoaib Akram
School of Computing, (Jan 2020 –)
Ph.D., 2019

Teaching: COMP2300, COMP2310, Computer Microarchitecture (3710)

Interests: Hardware/software interaction, storage systems

- Interesting time to learn and research about computer systems
- Technology (physical) limits vs. societal demand for more compute power; cost and energy efficiency; and sustainability
- Big Data, AI/ML, social media platforms, search engines, communications, mobile apps, drones, among others.

My current research focus is on efficiently processing big datasets

Logistics

- Course webpage: https://comp.anu.edu.au/courses/comp2300/

- **Lectures (**on the website**)**
  - Lecture slides
  - Lecture videos are available via Echo360

- **Policies**
  - General conduct, assignment submissions, support, management, grading, late submissions

- **Resources**
  - Frequently asked questions
  - Writing design documents
  - Stuff needed to finish the assignments

Communication

- We will use **Ed** for
  - Announcements
  - Addressing your concerns
  - Answering your questions
- Students are added automatically
  - If not send an email to **comp2300@anu.edu.au**

Communication

- The course email is an alternative form of communication

  - **comp2300@anu.edu.au**

Tutorials

- Labs are a critical component of this course (one every week)

- Handouts should be posted on the website

- First six labs
  - Assignment 1

- Next six labs
  - Assignment 2

- Excellent tutors with deep technical knowledge of the subject
  - Almost all of them have taken multiple courses in the systems & architecture specialization

Lectures

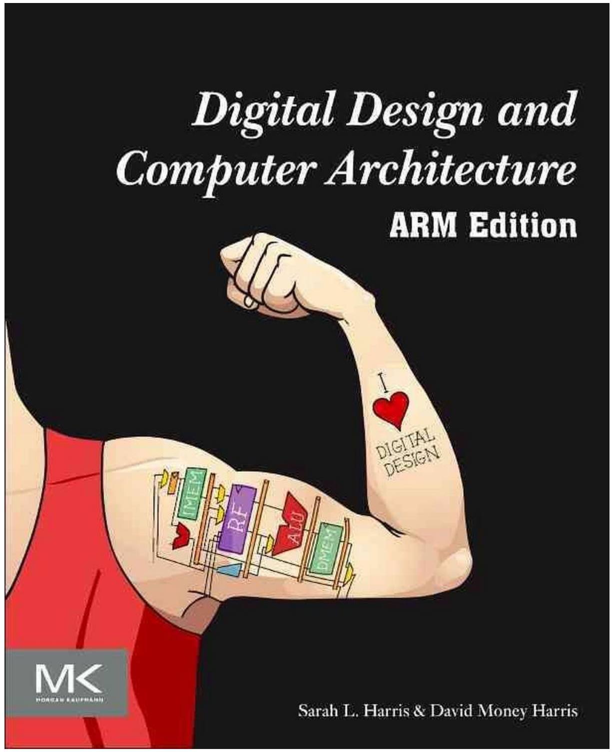
- Go well beyond what is covered in the tutorials
- Exam and quiz questions based on lectures
- Tutorials teach you how to build and program a simple computer to manage complexity
- Lectures will systematically build up from simple to a very complex, state of the art, computer

Assignment Submission

- Assignment submissions are handled via Gitlab
  - You will learn about it in the labs
  - Make a habit of using Gitlab properly!
  - Push often, always pull the latest

- Extensions
  - https://comp.anu.edu.au/courses/comp2300/policies/#extensions

Textbook



- Freely available online (check Ed or course webpage)
- I will post the chapters/sections on the Lectures page after the lecture

COMP2300

- Asks a question
    - Let’s give it a shot!

COMP2300

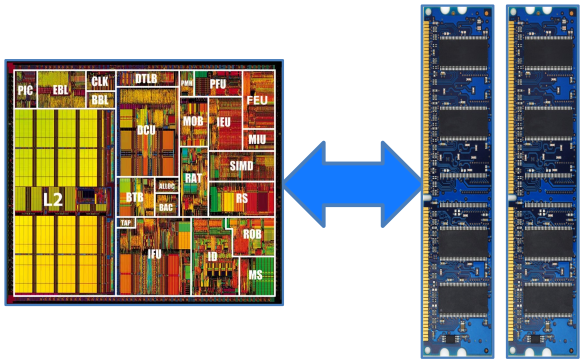
How does a computer **actually** work?

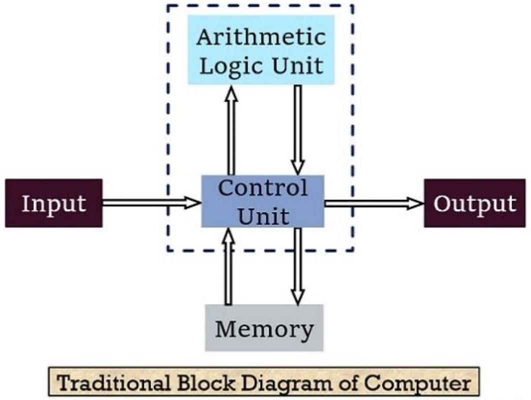
COMP2300

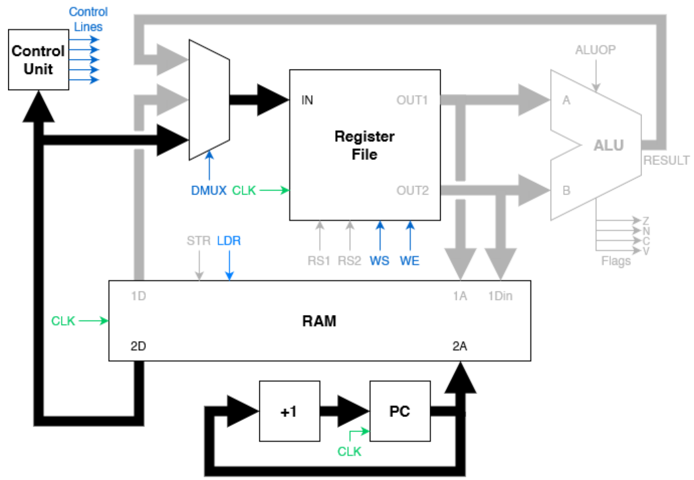
How does a computer work **under the hood**?

COMP2300

How does a computer work **under the hood**?





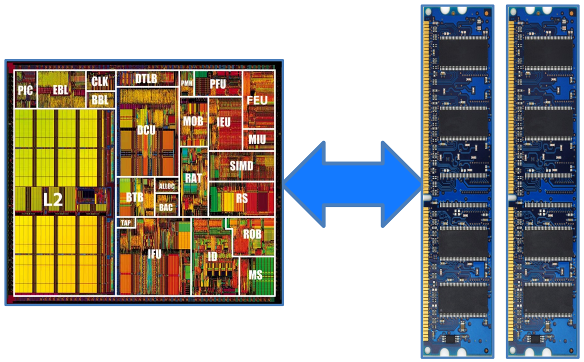


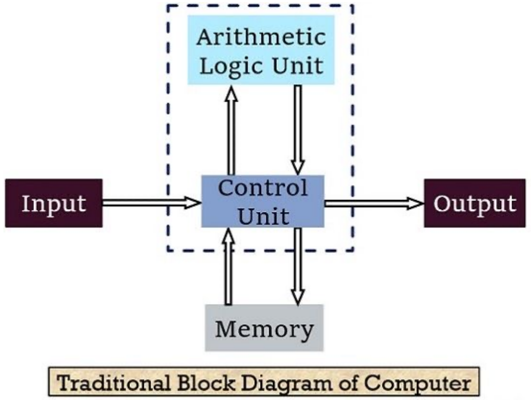
High abstraction level

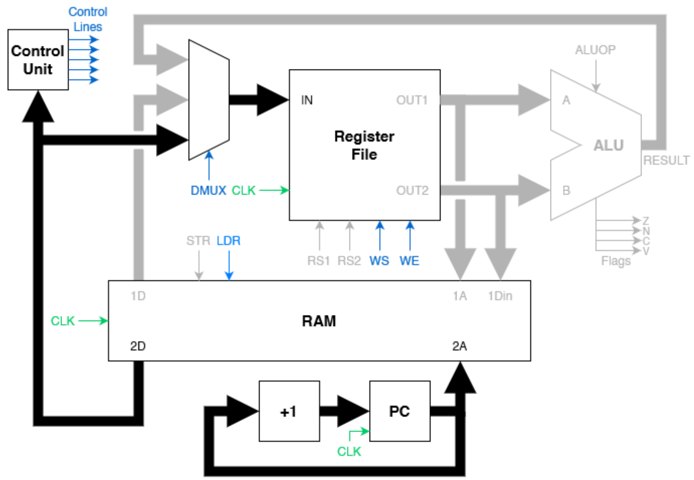
Low abstraction level

COMP2300

How does a computer **work** **under the hood**?







High abstraction level → Low abstraction level

COMP2300

Let’s be more specific

....

COMP2300

How does a computer perform a useful task?

COMP2300

How does a computer **perform** a **wide variety** of useful tasks?

COMP2300

How do we make a computer perform a wide variety of useful tasks?

COMP2300

How do we make electrons perform a wide variety of useful tasks?

COMP2300

How do we make electrons ~~perform~~ a wide variety of useful ~~tasks~~?

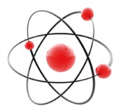
COMP2300

How do we make electrons execute a wide variety of useful programs?

- Computer Organization & Program Execution

How do we make electrons do the work?

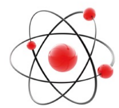
Problem Statement: “Save the planet”



How do we make electrons do the work?

Problem Statement: “Save the planet”

The Algorithm

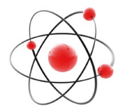


How do we make electrons do the work?

Problem Statement: “Save the planet”

The Algorithm

Program in a High-Level Language



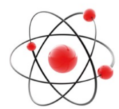
How do we make electrons do the work?

Problem Statement: “Save the planet”

The Algorithm

Program in a High-Level Language

Instruction Set Architecture (ISA)



How do we make electrons do the work?

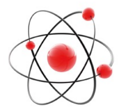
Problem Statement: "Save the planet"

The Algorithm

Program in a High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture



How do we make electrons do the work?

Problem Statement: "Save the planet"

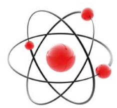
The Algorithm

Program in a High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture

Circuits



How do we make electrons do the work?

Problem Statement: "Save the planet"

The Algorithm

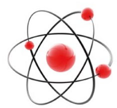
Program in a High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture

Circuits

Devices



How do we make electrons do the work?

- Using a *sequence of systematic transformations* developed over six decades

- Each step must be studied and improved for the whole **"compute stack"** to work/operate efficiently

Transformation Hierarchy

- We call the steps of the process: **Levels of transformation** OR **transformation hierarchy**

- At each level of the stack, we have choices
  - Language→ Java, Python, Ruby, Scala, C++, C#
  - ISA→ ARM, x86, SPARC, PowerPC, RISC-V
  - Microarchitecture→ Intel, AMD, IBM, Apple

- If we ignore any of the steps, then we cannot
  - Make the best use of computer systems
  - Build the **best** system for a set of programs

Problem

Algorithm

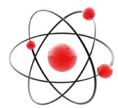
Program

Architecture

micro-arch

circuits

devices



Transformation Hierarchy

Problem Statement: “Save the planet”

The Algorithm

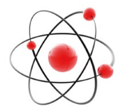
Program in a High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture

Circuits

Devices



Program Execution { Program in a High-Level Language / Instruction Set Architecture (ISA) / Microarchitecture }

Transformation Hierarchy

Problem Statement: “Save the planet”

The Algorithm

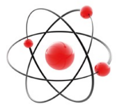
Program in a High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture

Circuits

Devices



Program Execution: Program in a High-Level Language, Instruction Set Architecture (ISA), Microarchitecture

Computer Organization: Microarchitecture, Circuits

Transformation Hierarchy

Problem Statement: “Save the planet”

The Algorithm

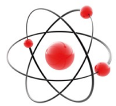
Program in a High-Level Language

Instruction Set Architecture (ISA)

Microarchitecture

Circuits

Devices



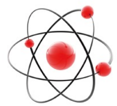
Program Execution

Computer Organization

Computer Architecture

Transformation Hierarchy & Us

	Problem Statement: "Save the planet"
	The Algorithm
	Program in a High-Level Language
2300	Instruction Set Architecture (ISA)
2300	Microarchitecture
2300	Circuits
	Devices



Program Execution

Computer Organization

Computer Architecture

Transformation Hierarchy & Us

Problem Statement: "Save the planet"

The Algorithm

Program in a High-Level Language

2310 Compiler and Third-Party Libraires/Binaries

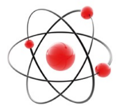
2310 Operating System

2300 Instruction Set Architecture (ISA)

2300 Microarchitecture

2300 Circuits

✗ Devices



Program Execution: Program in a High-Level Language through Microarchitecture

Computer Architecture: Instruction Set Architecture (ISA) through Devices

Computer Organization: Microarchitecture and Circuits

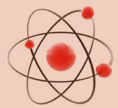
Hardware and Software

Problem Statement: "Save the planet"

Software	The Algorithm
	Program in a High-Level Language
	Compiler and Third-Party Libraires/Binaries
	Operating System

ISA = Hw/Sw boundary/interface

Instruction Set Architecture (ISA)

	Microarchitecture
	Circuits
Hardware	Devices 

ISA vs. Microarchitecture

- **ISA:** **Specification** of a set of definite instructions that the computer can carry out

  - All computers (CPUs/microprocessors) perform the same set of basic instructions

  - ADD, MULTIPLY, DIVIDE, MOVE, BRANCH

  - Two manufacturers might differ in *which set* of basic instructions

- **Microarchitecture:** **Implementation** of the ISA using circuits

ISA vs. Microarchitecture

- **ISA:** What the driver needs to know as she sits inside the automobile to make the automobile carry out the driver's wishes
- If the middle pedal (brake) is pressed, the car stops
- Steering wheel, ignition key, the gears, windshield wipers
- ISA specifies two things (?)
- All cars have the same ISA (hopefully). There could be differences!

- **Microarchitecture:** What goes underneath the hood
- Different cost/performance tradeoffs
- Some are turbocharged. Some have disc brakes. Some cost a million $. Some are more fuel efficient than others
- But you don't need a separate license to drive a Honda and a BMW
- Must not take a Honda to a BMW factory for repair!

Two Recurring Themes

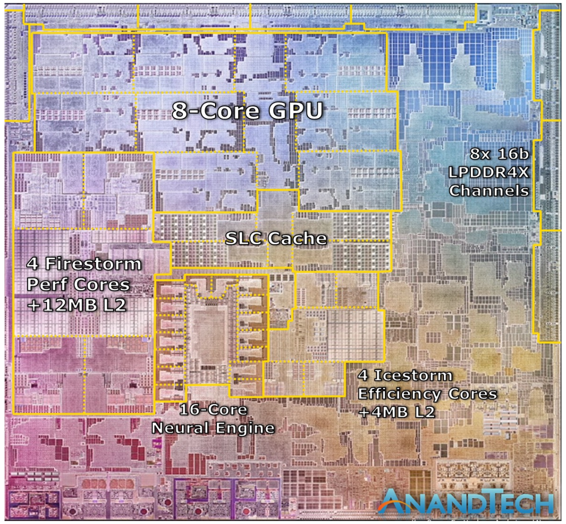
- The notion of abstraction

- Hardware versus software

The Notion of Abstraction

- **Abstraction:** Know components from a high level of detail

No human (programmer) can track 10 billion elements. **Computer systems work because of abstraction!**



Apple M1 Chip
Billions of transistors
All working in parallel

The Notion of Abstraction

- Abstraction: View the world from a higher level
- Focus on the important aspects
- It is a way to enhance productivity and efficiency

The Notion of Abstraction

Put pressure on the pedal known as the accelerator

Drive 0.7 KM at an appropriate speed so we don't get fined

Turn the steering wheel to the left

Press the middle pedal so the automobile comes to a halt

....

The Notion of Abstraction

First go straight

Then take a left

Then go straight again

...

The Notion of Abstraction

Take me to 108 North Road, please!

The Notion of Abstraction

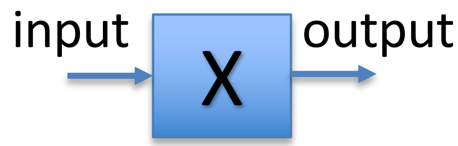
- Important lesson in the previous example

- It is efficient to abstract and there is really nothing to be gained from the ridiculous elaboration to the driver

- **Except when**
  - Driver does not know how to drive
  - Driver does not know where is 108 North Road
  - This is where COMP2300 is *unique*. It teaches you to un-abstract when the world you are trying to abstract does not work as expected.

The Notion of Abstraction

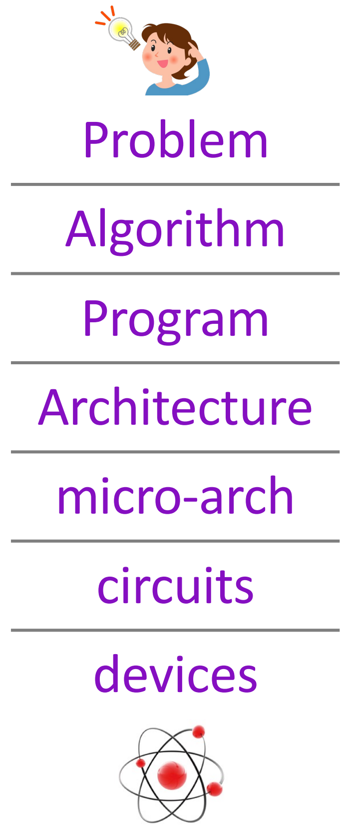
- Focus on the important aspects
  - What is input? What is output?
  - What is the function: Is X an ADD or MULTIPLY unit



- If the world below does not work as expected?
  - If the transistors X is built from do not behave as expected (unlikely in this course)
  - To deal with it, we need to go below the abstraction layer
- Deconstruction: To un-abstract when needed
  - Important skill
  - If the CPU does not work as expected, first test each of the big sub-components
  - Then, check to see if the next component in the hierarchy works as expected

The Notion of Abstraction

- We will raise the level of abstraction in this course every couple weeks
- We start from the ground (up) because this is how computers have evolved
- Each layer in the *transformation hierarchy* is an abstraction layer

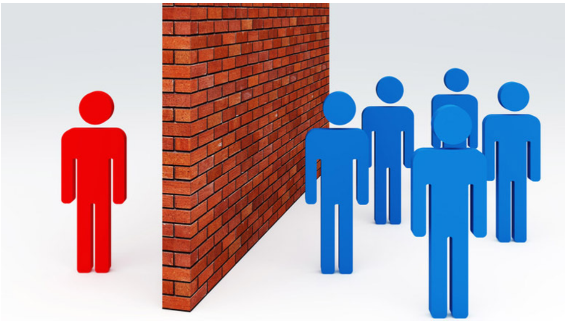


Hardware versus Software

- Hardware
  - CPU, memory, storage device, disk, SSD, network card, USB device, AI accelerator, FPGA, PLA, motherboard, PCI express bus, SATA drive

- Software
  - Programs, operating systems, compilers, virtual machines, device drivers,

- **One view:** Ok to be an expert at one of these

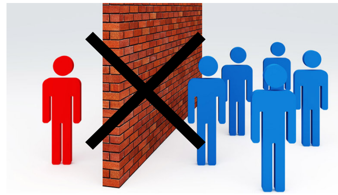


Hardware versus Software

- Hardware and Software
  - Two interacting parts of the computer system

- **COMP2300 view:** Knowing the capabilities and limitations of each leads to better overall systems

Hardware-Software Interplay

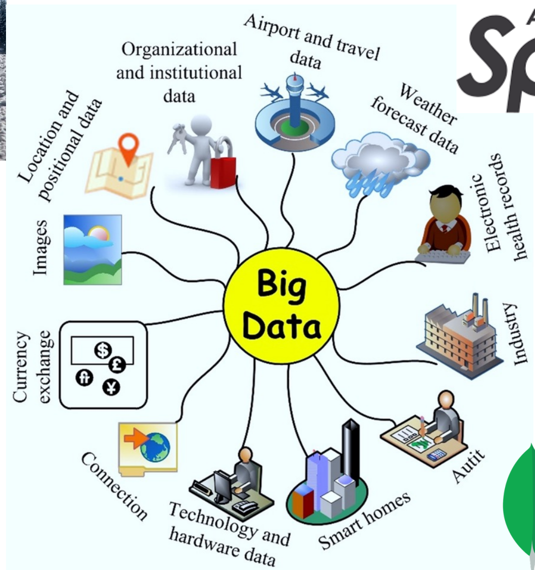


Software Applications

Requirements →

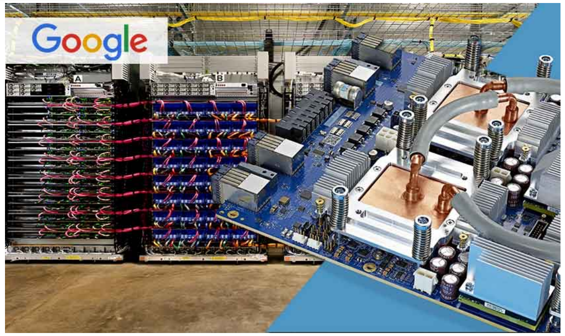
← Opportunities

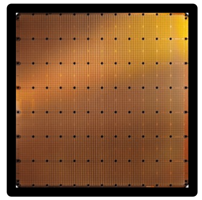
Hardware Systems



APACHE Spark

mongoDB





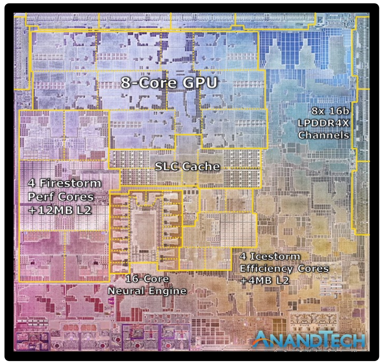
Why so many hardware choices?

Hardware: General Purpose vs. Special Purpose

Common building block of computers

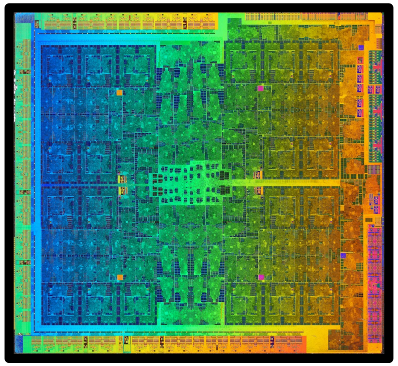
General Purpose

CPUs



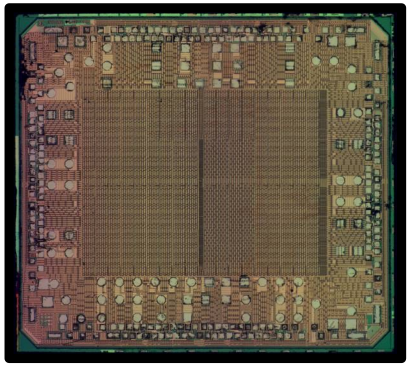
Apple M1

GPUs



Nvidia GTX 1070

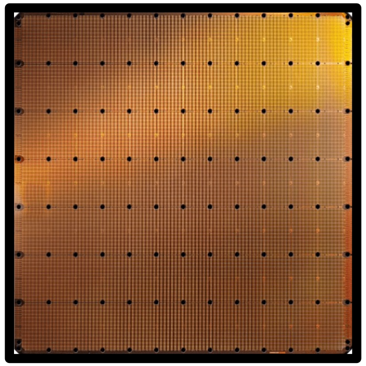
FPGAs



Xilinx Spartan

Special Purpose

ASICs



Cerebras WSE-2

Flexible: Can execute any program
Easy to program & use
Not the best performance & efficiency
C/C++/Java/...

Efficient & High performance
(Usually) Difficult to program & use
Inflexible: Limited set of programs
Domain-specific, special purpose languages

General Purpose vs. Special Purpose

General Purpose

Flexible: Can work with any bolt

Easy to use

Not the best fit, results or efficiency

Special Purpose

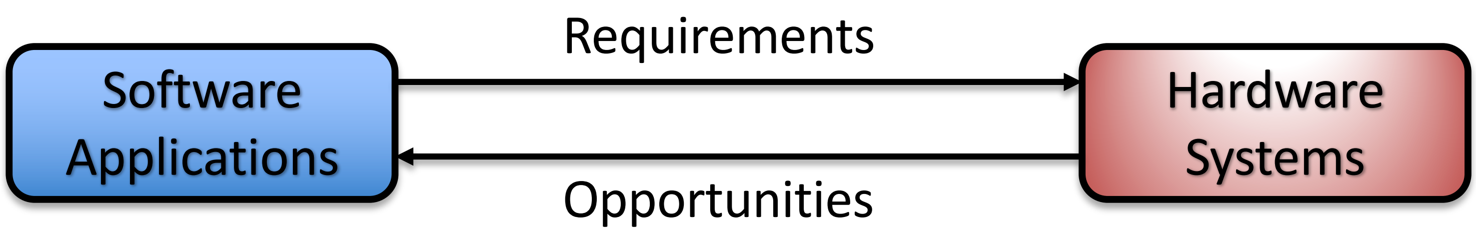
Efficient & High performance

(Usually) Difficult to use

Inflexible: Only for fitting bolts

Modern Software Trends

- Many important and emerging applications are data-intensive
- It is easier and convenient to produce data than to process, analyze, and store it
- This trend of producing data at high velocity is driving new applications and correspondingly novel hardware

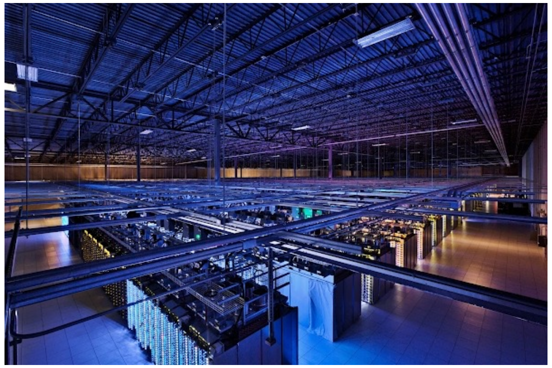


Two Key Ideas

- Key components of a computer are the same

- All computers can compute the same things

Idea 1: Key components are same



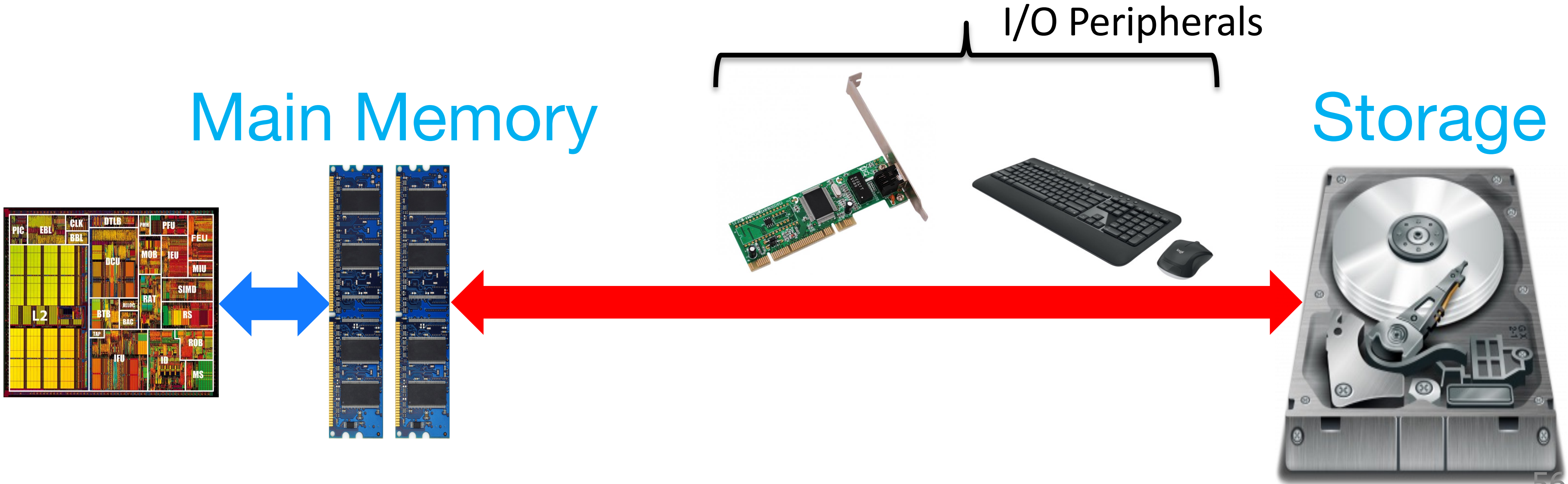
Council Bluffs, Iowa
data center, Google
(115, 000 sq. feet)

Self-flying nano drone
94 milli-watts

Research server for my
students with special memory
& storage devices

A Canonical Computer System

- **Most computer systems can be viewed as below**
  - Key resources: CPU, memory, I/O devices, storage
  - CPU (processor/microprocessor) does the actual computation
  - Processor can access memory much faster than storage



Idea 1: Key components are same

- Key Idea 1: All computer systems, big or small, have a few fundamental components
  - **Microprocessor** (processor or central processing unit or CPU) for doing computation
  - **Main memory** for storing temporary information and program data close to the processor
  - **Storage** devices (disks or SSDs) for storing long-term or persistent information
  - **I/O devices** to communicate with the external environment
    - Sensors
    - Peripherals

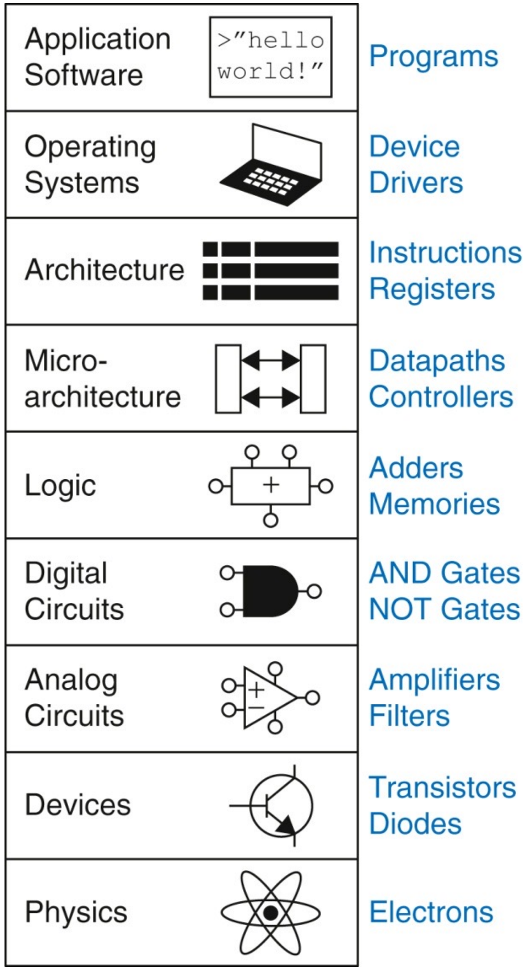
Idea 2: They all can solve the same problems

- Key Idea 2: All computers regardless of size, cost, and speed can compute the same things if they are given enough time and memory

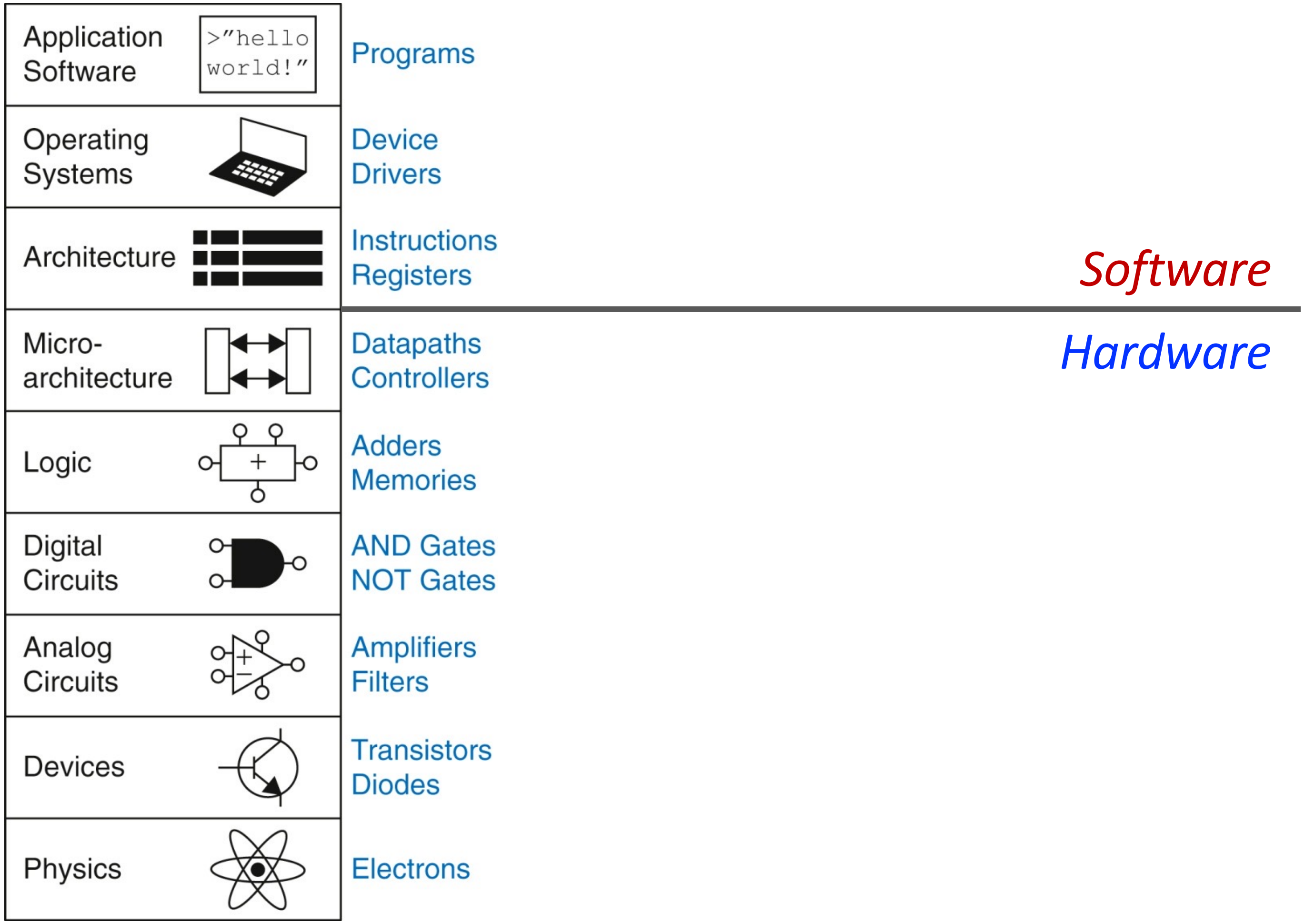
  - Anything a fast computer can do, a slow computer can do

  - Let’s explore this idea further ....

Program Ex.

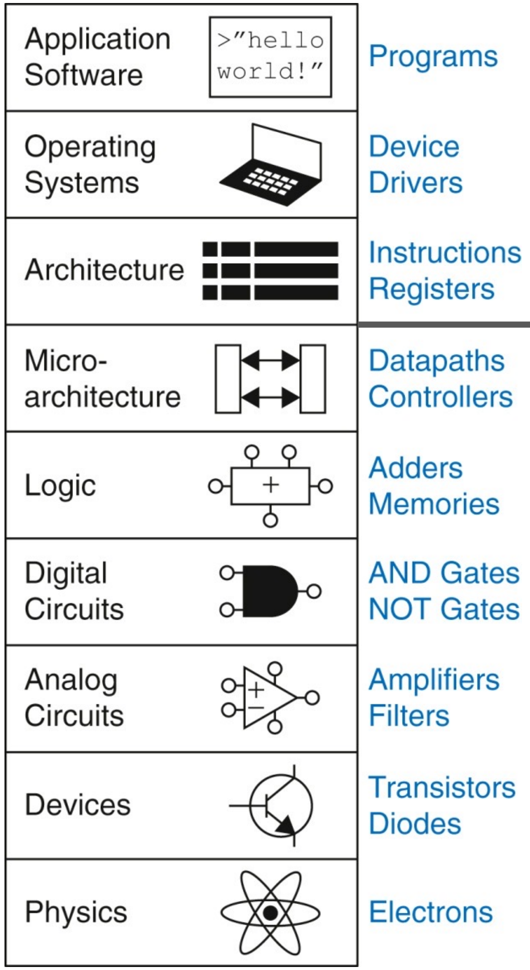

Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions Registers
Micro-architecture
Datapaths Controllers
Logic
+
Adders Memories
Digital Circuits
AND Gates NOT Gates
Analog Circuits
+
−
Amplifiers Filters
Devices
Transistors Diodes
Physics
Electrons

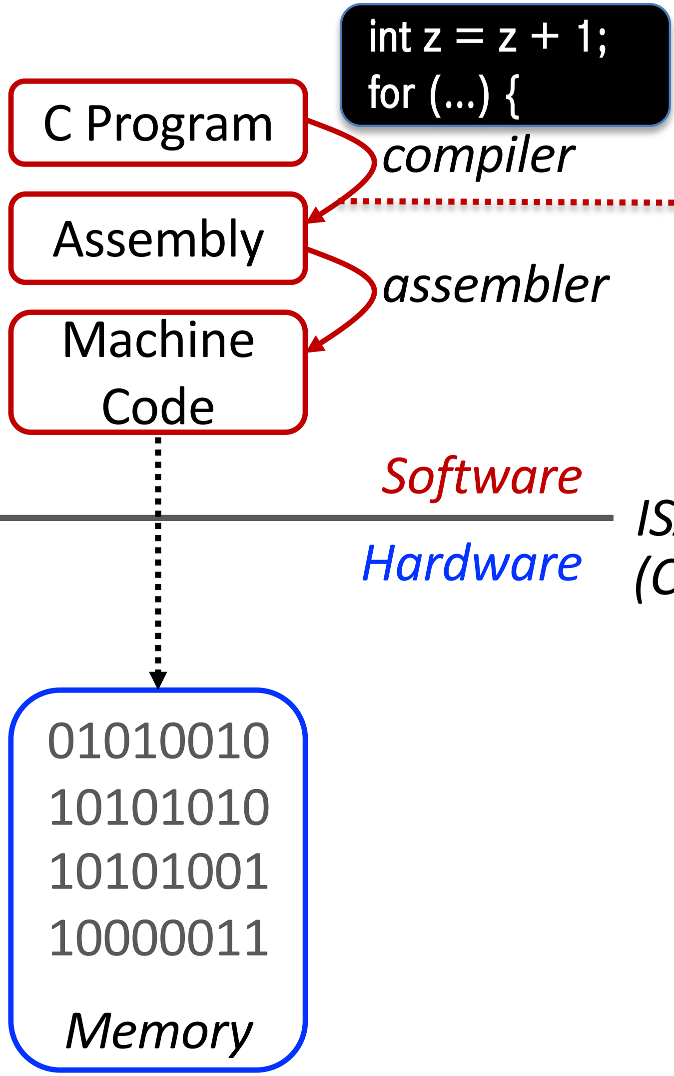
Program Ex.



ISA is the boundary (Contract)

Program Ex.





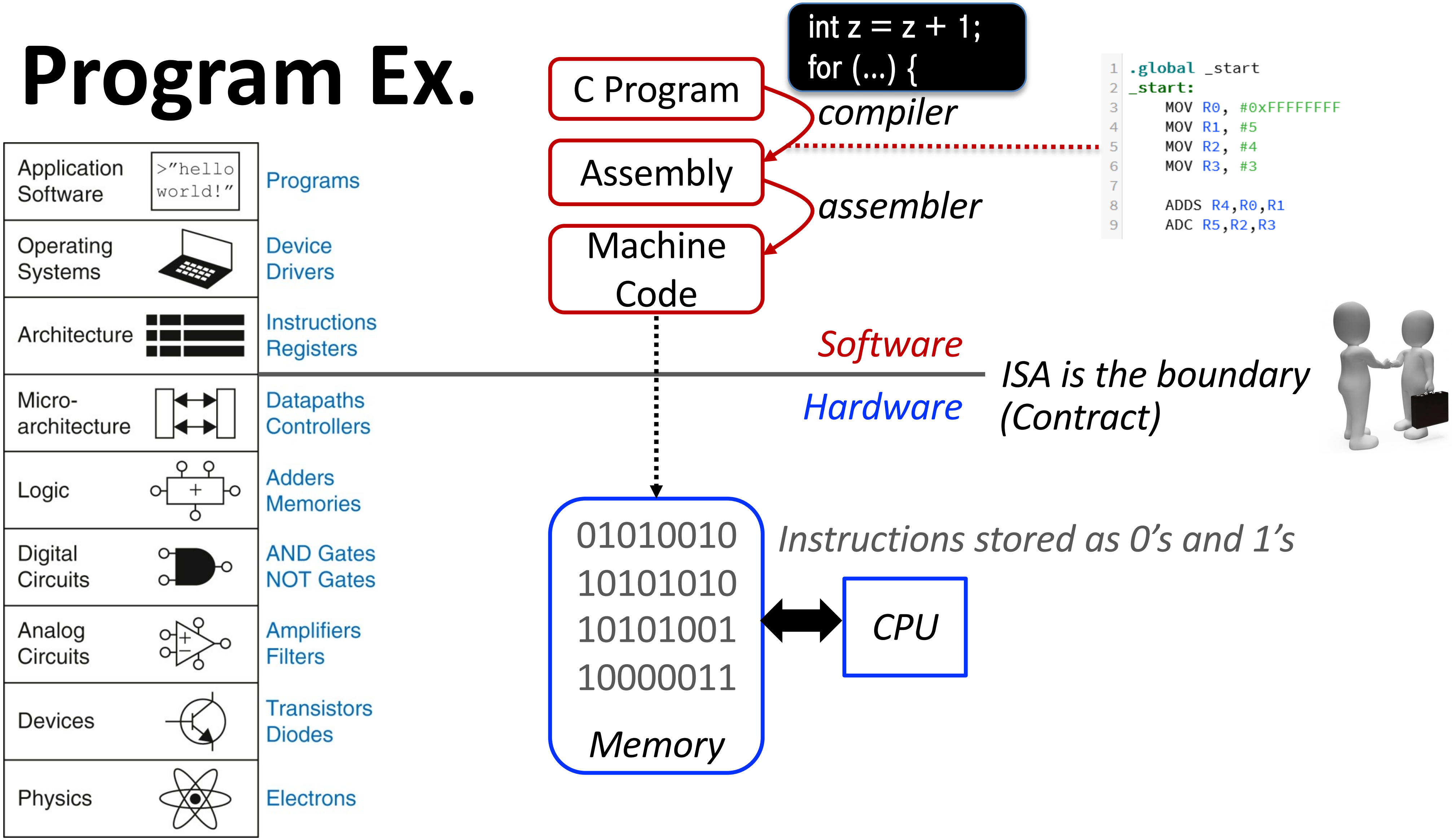

```
.global _start
_start:
    MOV R0, #0xFFFFFFFF
    MOV R1, #5
    MOV R2, #4
    MOV R3, #3

    ADDS R4,R0,R1
    ADC R5,R2,R3
```

Software

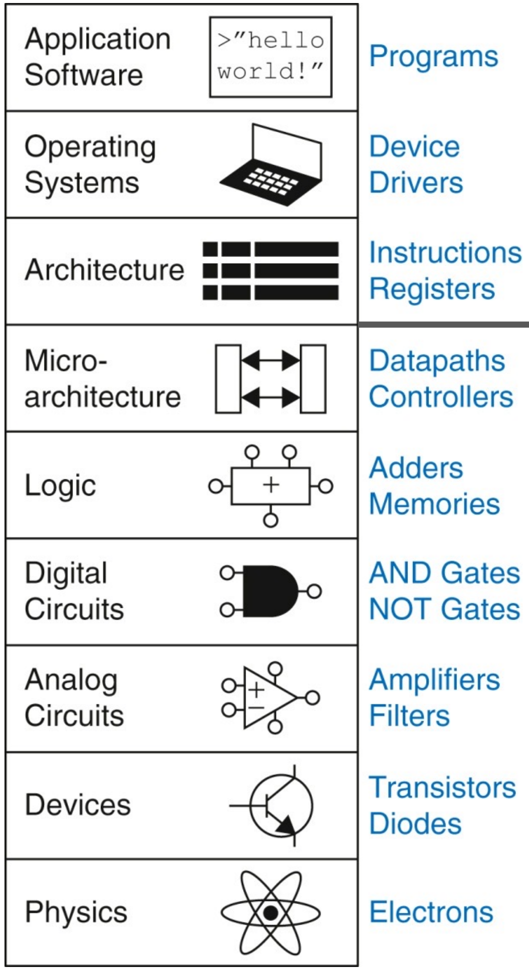
Hardware

ISA is the boundary (Contract)

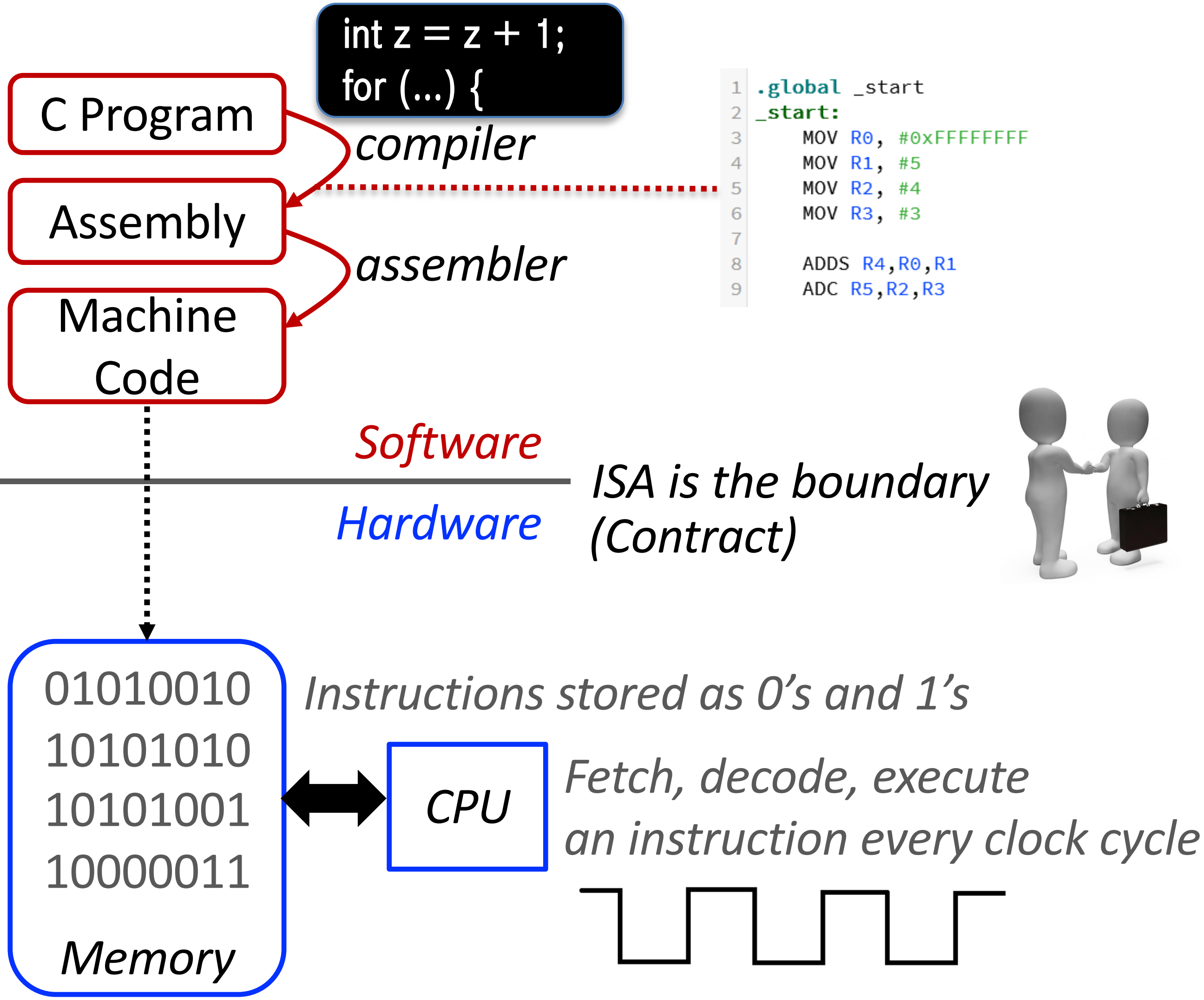
Program Ex.


Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions Registers
Micro-architecture
Datapaths Controllers
Logic
Adders Memories
Digital Circuits
AND Gates NOT Gates
Analog Circuits
Amplifiers Filters
Devices
Transistors Diodes
Physics
Electrons
int z = z + 1;
for (...) {
C Program
compiler
Assembly
assembler
Machine Code
.global _start
_start:
MOV R0, #0xFFFFFFFF
MOV R1, #5
MOV R2, #4
MOV R3, #3
ADDS R4,R0,R1
ADC R5,R2,R3
Software
Hardware
ISA is the boundary (Contract)
01010010
10101010
10101001
10000011
Memory
Instructions stored as 0's and 1's
CPU

Program Ex.

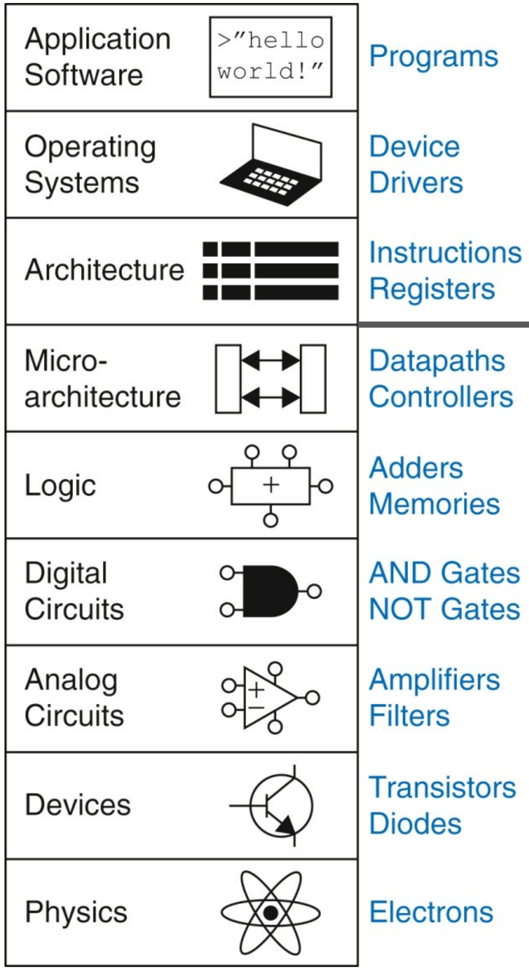


Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions Registers
Micro-architecture
Datapaths Controllers
Logic
Adders Memories
Digital Circuits
AND Gates NOT Gates
Analog Circuits
Amplifiers Filters
Devices
Transistors Diodes
Physics
Electrons



int z = z + 1;
for (...) {
C Program
compiler
Assembly
assembler
Machine Code
.global _start
_start:
MOV R0, #0xFFFFFFFF
MOV R1, #5
MOV R2, #4
MOV R3, #3
ADDS R4,R0,R1
ADC R5,R2,R3
Software
Hardware
ISA is the boundary (Contract)
01010010
10101010
10101001
10000011
Memory
Instructions stored as 0's and 1's
CPU
Fetch, decode, execute
an instruction every clock cycle

Program Ex.




```
int z = z + 1;
for (...) {
```

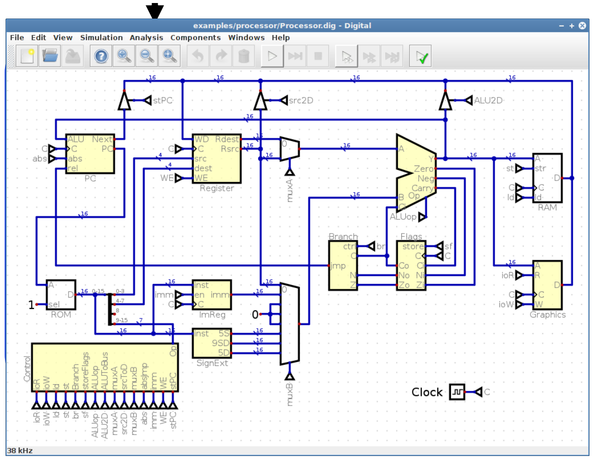
C Program → *compiler* → Assembly → *assembler* → Machine Code

```
.global _start
_start:
    MOV R0, #0xFFFFFFFF
    MOV R1, #5
    MOV R2, #4
    MOV R3, #3

    ADDS R4,R0,R1
    ADC R5,R2,R3
```

Software

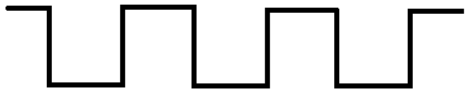
Hardware

ISA is the boundary (Contract)



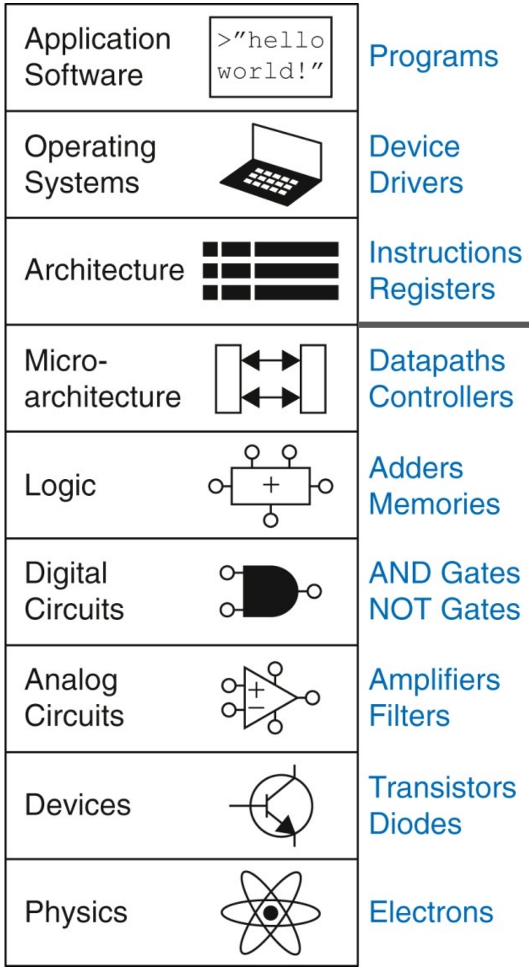
stored as 0's and 1's

Fetch, decode, execute an instruction every clock cycle



Assignment 1: Build CPU

Program Ex.




```
int z = z + 1;
for (...) {
```

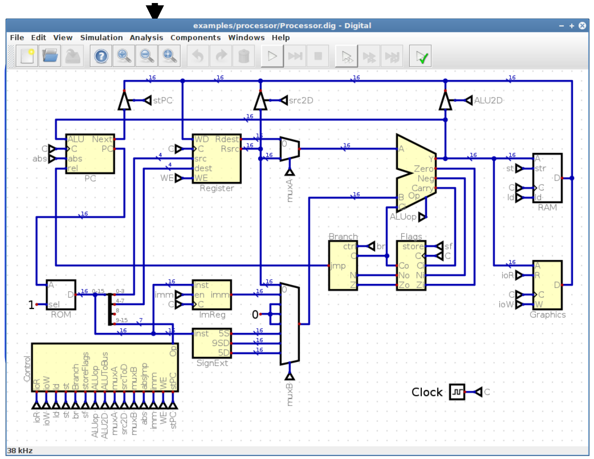
Assignment 2: Program CPU

C Program → *compiler* → Assembly → *assembler* → Machine Code

Software

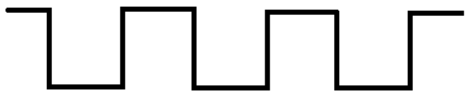
Hardware

ISA is the boundary (Contract)



stored as 0's and 1's

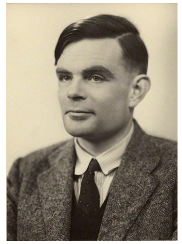
Fetch, decode, execute
an instruction every clock cycle



Assignment 1: Build CPU

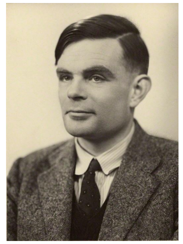
Computer = Universal Computational Device

- Anything that can be computed, can be computed by a computer
- Studying computers is studying the fundamentals of all computing
- The idea of universal computational device is due to Alan Turing (1937)

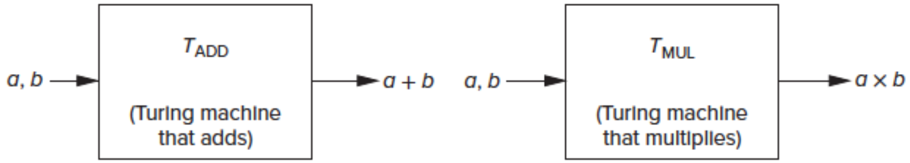


- He gave the mathematical description of Turing machine
  - A particular kind of machine called the Turing machine can carry out all computations
  - Let’s build this ultra powerful Turing machine

Computer = Universal Computational Device



- First, we have two simple example Turing machines
  - One for addition, and one for multiplication



- What we now call a computer is a Turing machine that can simulate all Turing machines
- What does it need as inputs?
  - Inputs and the description of the Turing machine to simulate
  - Can you draw the black box model?
  - We say that Turing machines are programmable

Universal Computational Device

- Anything that can be computed, can be computed by a computer provided it has enough time and enough memory

- We instruct the computer how to do X, and it obliges by interpreting our instructions

  - Instructions are stored in memory like regular data

- Computer is **programmable** because we can rewrite instructions to make it do something else

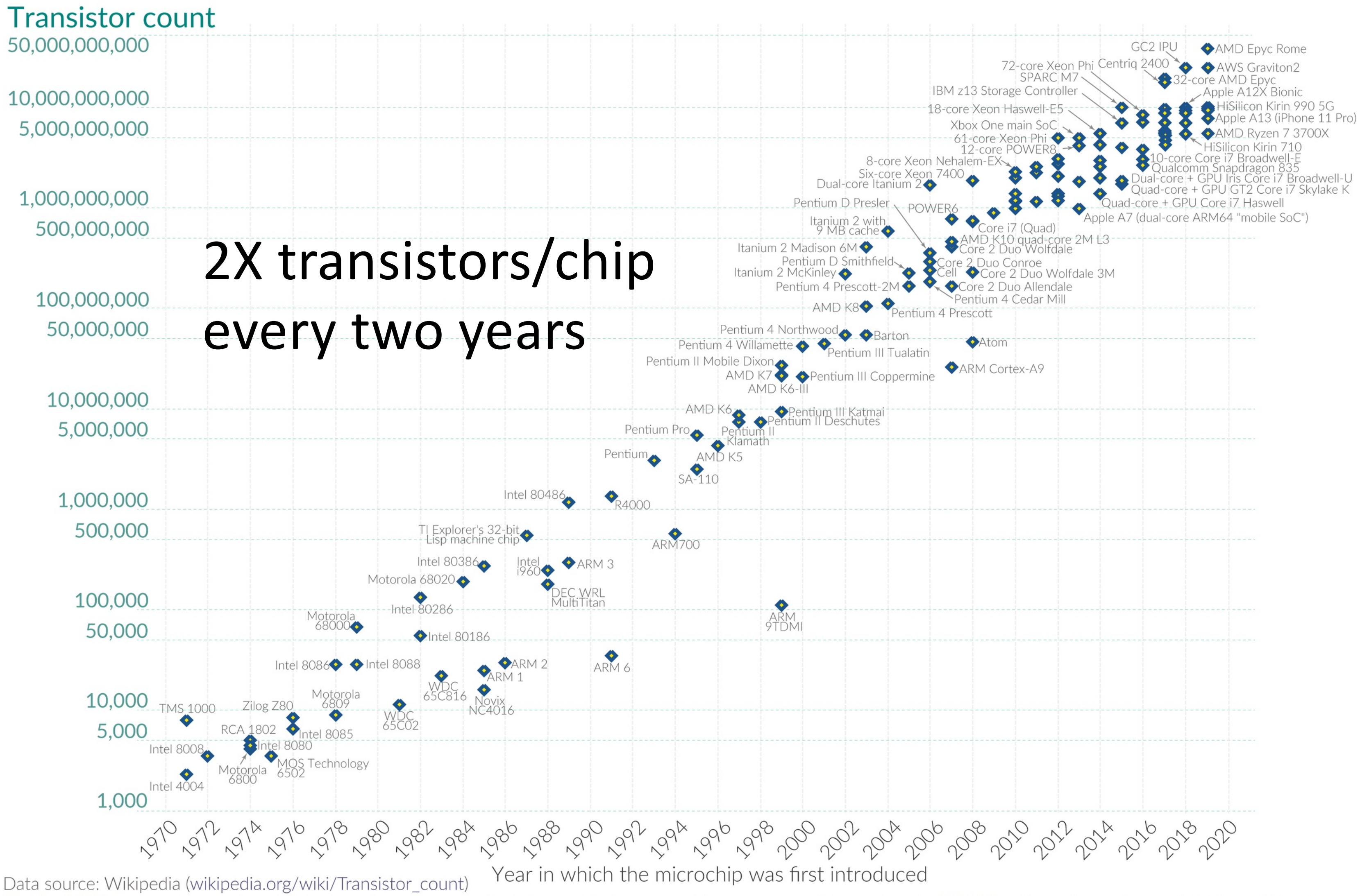
Some Technology Trends

- Moore's Law
- Uniprocessor performance
- Memory wall
- Memory hierarchy

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.

Today, technologists have internalized it and grown accustomed to believing that computer speed doubles every 18 months. However, over the last few years, the semiconductor industry has reached a point where Moore's Law is becoming obsolete. In fact, Nvidia's founder and CEO Jensen Huang has declared Moore's Law to be done.

The most recent statement made by Huang was to *The Protocol* in a recent interview where he said "the semiconductor industry is near the limit." He added, "It's near the limit in the sense that we can keep shrinking transistors but we can't shrink atoms — until we discover the same particle that Ant Man discovered. Our transistors are going to find limits, and we're at atomic scales. And so [this problem] is a place where material science is really going to come in handy."

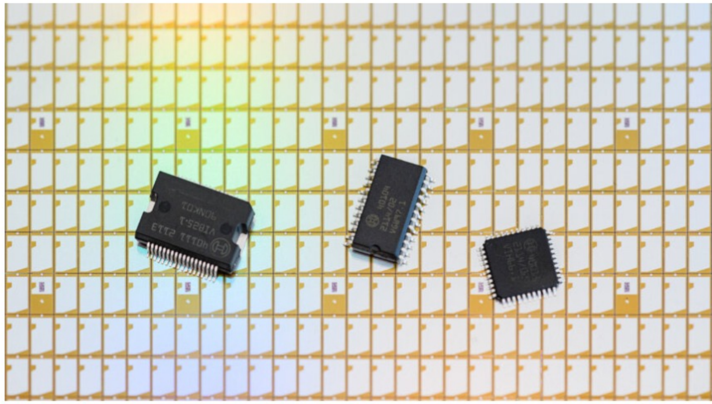
T_HQ

ARTIFICIAL INTELLIGENCE

The semiconductor industry is 'near its limit,' says Nvidia CEO

CEO Jensen Huang believes that the long-held notion that the processing power of computers increases exponentially every couple of years has hit its natural limit.

17 October 2022



Dashveenjit Kaur

@DashveenjitK

dashveen@hybrid.co

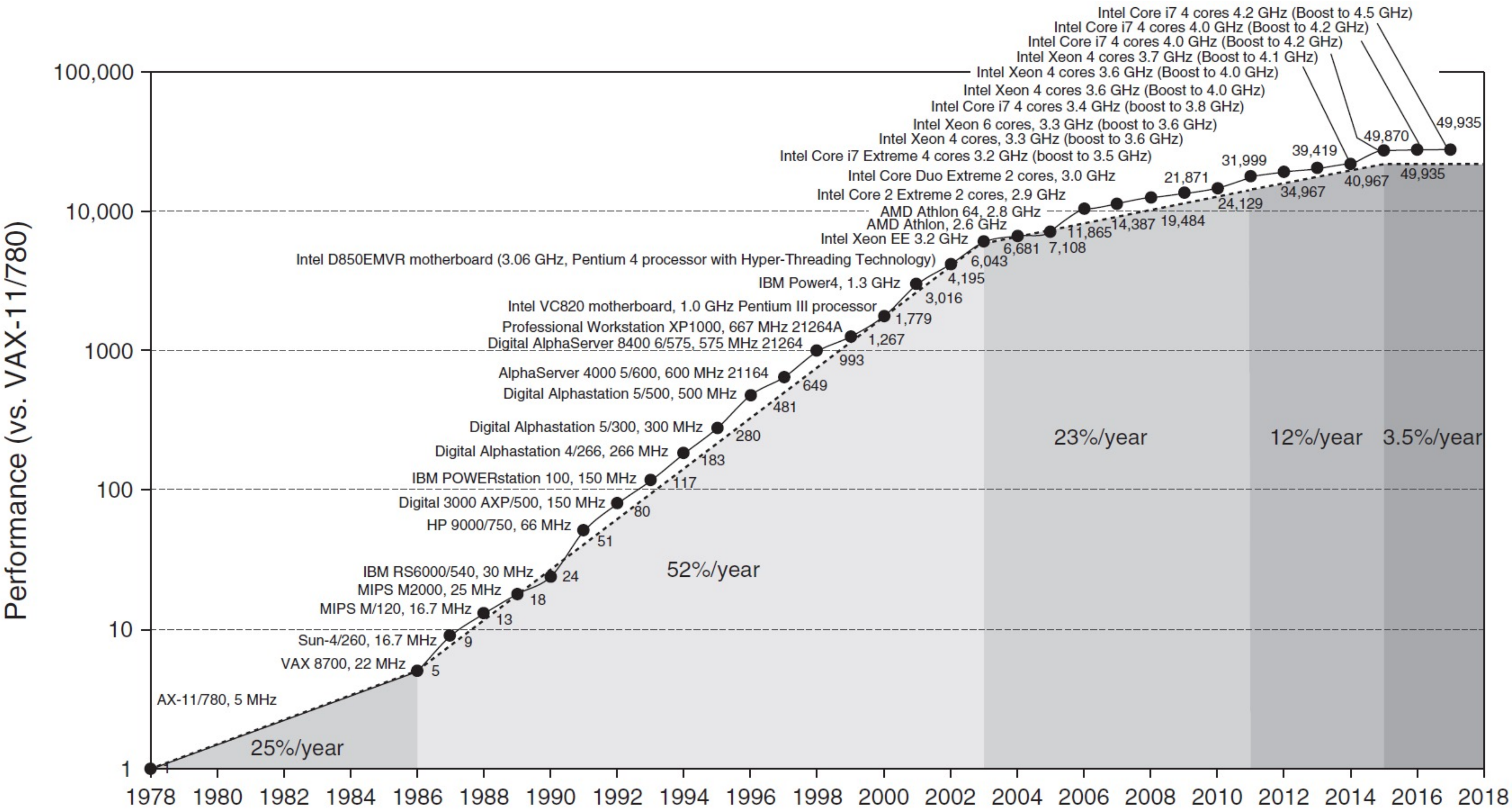
All stories

The semiconductor industry is near its limit, Nvidia CEO said. What does it mean? (Photo by JENS SCHLUETER / AFP)

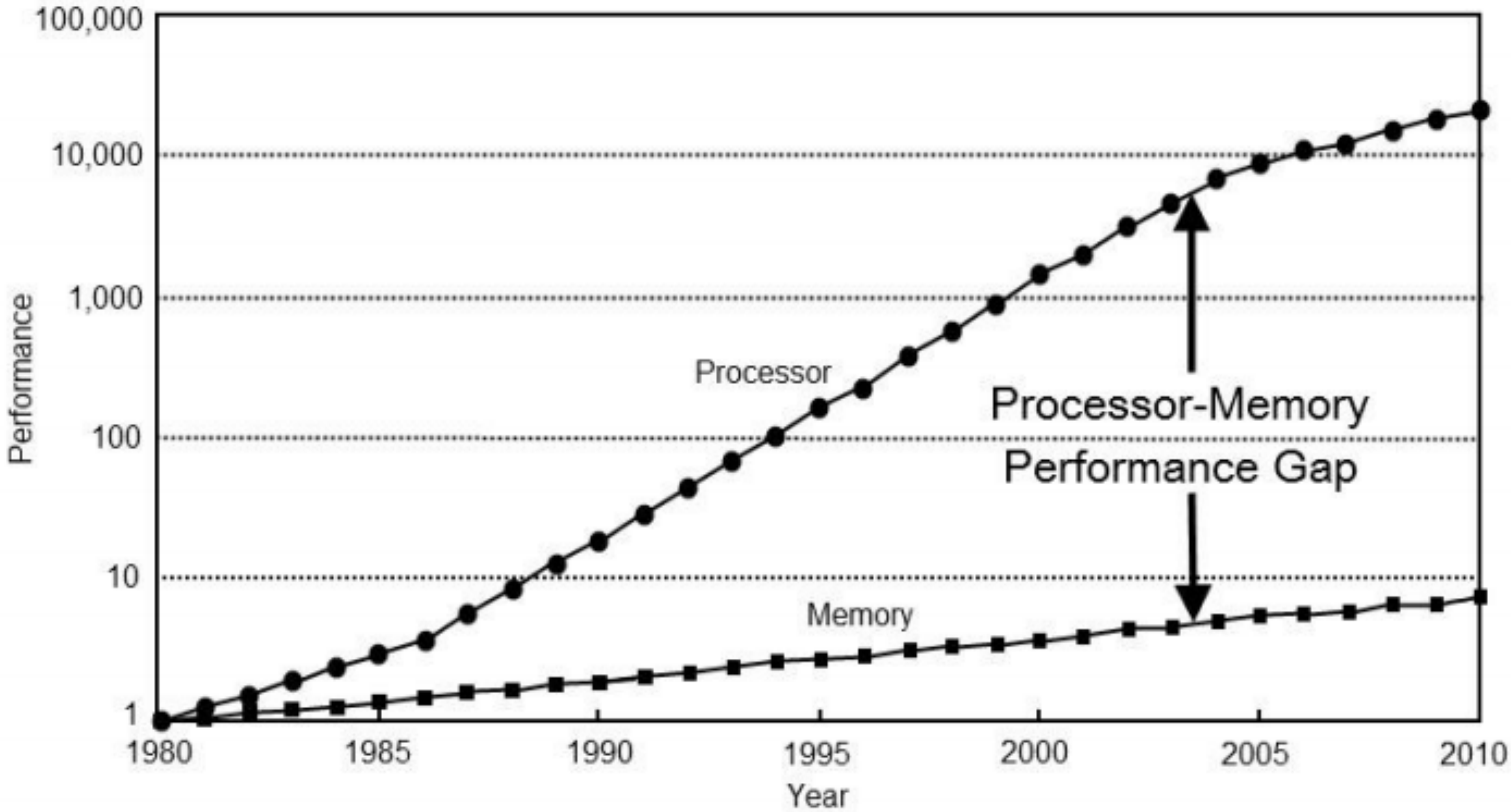
- **The semiconductor industry is at a point where the scale of chip components gets closer and closer to that of individual atoms, and that makes it harder to keep up the pace of Moore's Law, Huang said.**
- **Huang says "our transistors are going to find limits and we're at atomic scales."**

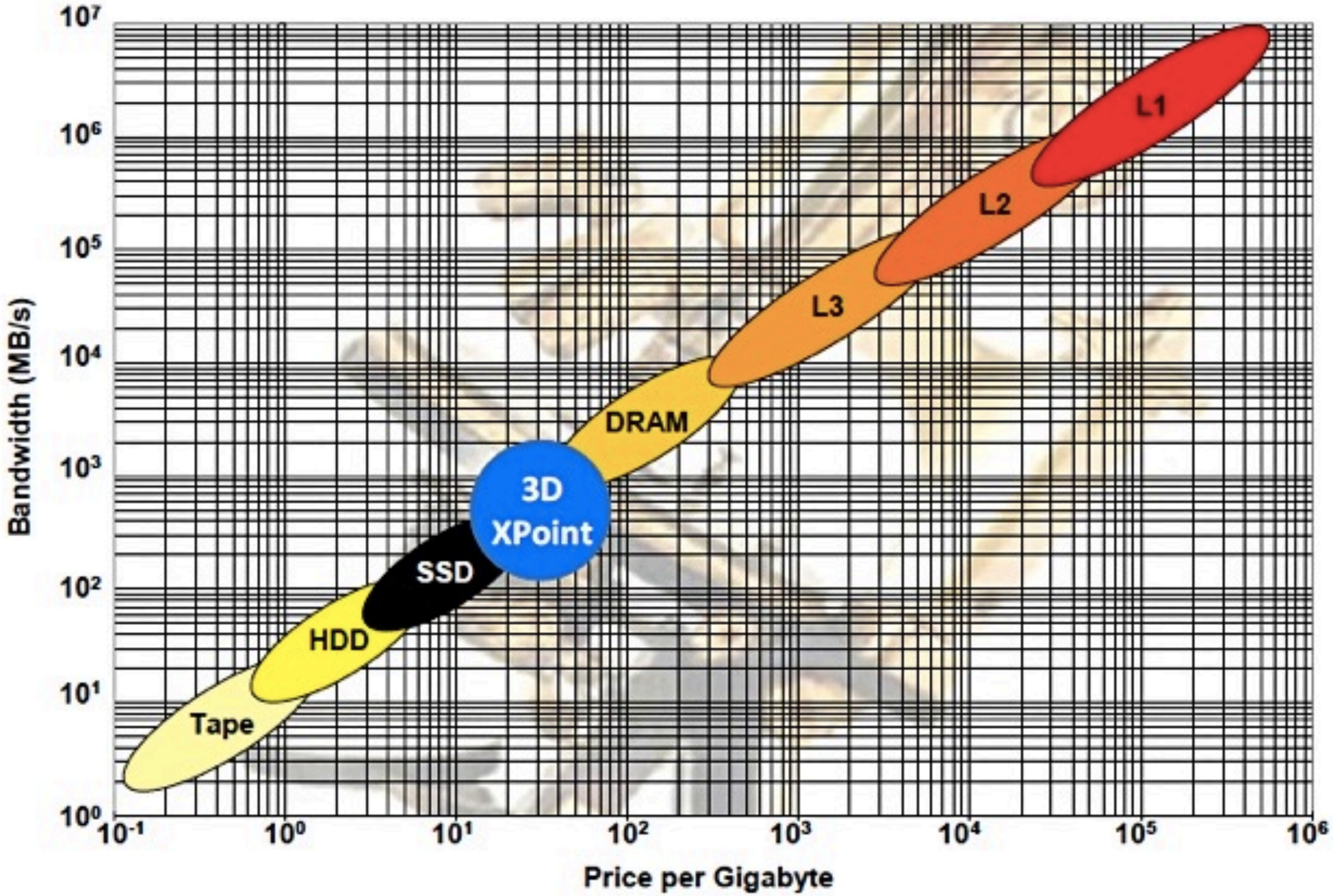
READ NEXT

Within the technology industry, overall electronics innovation is highly dependable on semiconductor advancements. After all, it is the shrinking of processors that improves battery life, lowers costs and boosts performance of devices. As Moore's Law suggests, the number of transistors packed onto the silicon chips that power the modern world has been steadily growing in


Performance (vs. VAX-11/780)
100,000
10,000
1000
100
10
1
1978 1980 1982 1984 1986 1988 1990 1992 1994 1996 1998 2000 2002 2004 2006 2008 2010 2012 2014 2016 2018
25%/year
52%/year
23%/year
12%/year
3.5%/year
AX-11/780, 5 MHz
VAX 8700, 22 MHz
Sun-4/260, 16.7 MHz
MIPS M/120, 16.7 MHz
MIPS M2000, 25 MHz
IBM RS6000/540, 30 MHz
HP 9000/750, 66 MHz
Digital 3000 AXP/500, 150 MHz
IBM POWERstation 100, 150 MHz
Digital Alphastation 4/266, 266 MHz
Digital Alphastation 5/300, 300 MHz
Digital Alphastation 5/500, 500 MHz
AlphaServer 4000 5/600, 600 MHz 21164
Digital AlphaServer 8400 6/575, 575 MHz 21264
Professional Workstation XP1000, 667 MHz 21264A
Intel VC820 motherboard, 1.0 GHz Pentium III processor
IBM Power4, 1.3 GHz
Intel D850EMVR motherboard (3.06 GHz, Pentium 4 processor with Hyper-Threading Technology)
Intel Xeon EE 3.2 GHz
AMD Athlon, 2.6 GHz
AMD Athlon 64, 2.8 GHz
Intel Core 2 Extreme 2 cores, 2.9 GHz
Intel Core Duo Extreme 2 cores, 3.0 GHz
Intel Core i7 Extreme 4 cores 3.2 GHz (boost to 3.5 GHz)
Intel Xeon 4 cores, 3.3 GHz (boost to 3.6 GHz)
Intel Xeon 6 cores, 3.3 GHz (boost to 3.6 GHz)
Intel Core i7 4 cores 3.4 GHz (boost to 3.8 GHz)
Intel Xeon 4 cores 3.6 GHz (Boost to 4.0 GHz)
Intel Xeon 4 cores 3.6 GHz (Boost to 4.0 GHz)
Intel Xeon 4 cores 3.7 GHz (Boost to 4.1 GHz)
Intel Core i7 4 cores 4.0 GHz (Boost to 4.2 GHz)
Intel Core i7 4 cores 4.0 GHz (Boost to 4.2 GHz)
Intel Core i7 4 cores 4.2 GHz (Boost to 4.5 GHz)
1
5
9
13
18
24
51
80
117
183
280
481
649
993
1,267
1,779
3,016
4,195
6,043
6,681
7,108
11,865
14,387
19,484
21,871
24,129
31,999
34,967
39,419
40,967
49,870
49,935
49,935

CPU/Memory performance

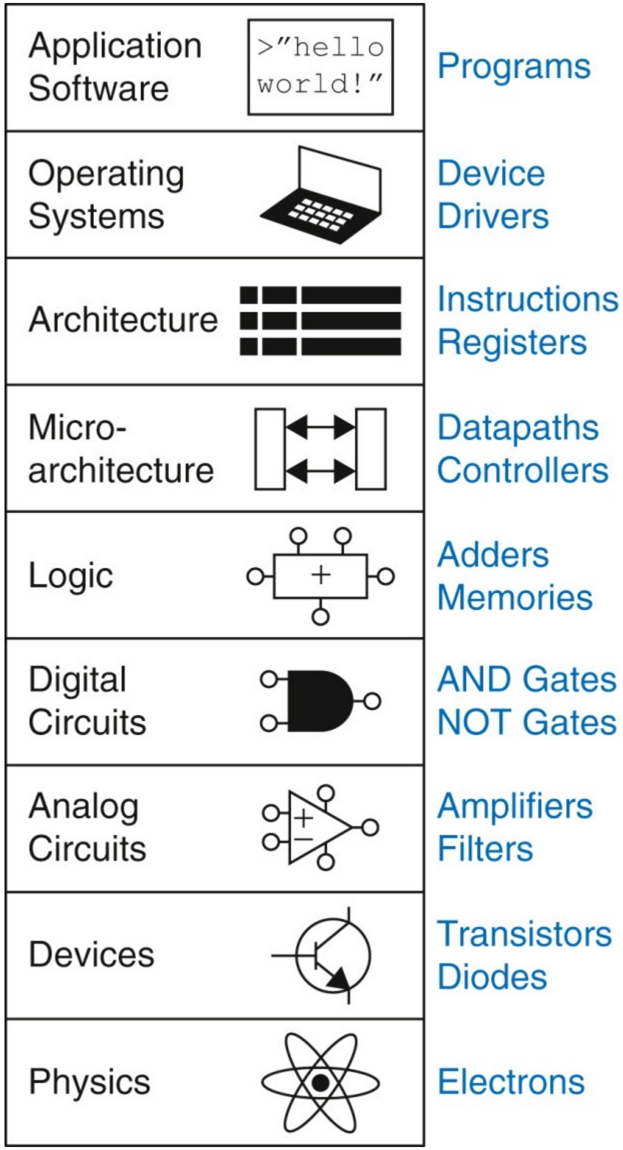




Source: Objective Analysis, 2019

Course Plan

- Navigate the transformation hierarchy
  - Bottom-up ✓
  - Top-down



Week 4

Week 7, 8, 9

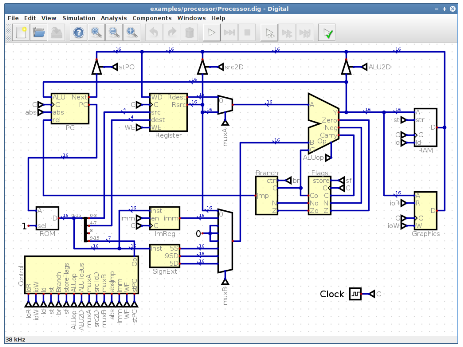
Week 5, 6

Week 10, 11, 12 (I/O and Advanced microarchitecture optimizations)

Week 2, 3

Week 1

Assignment 2

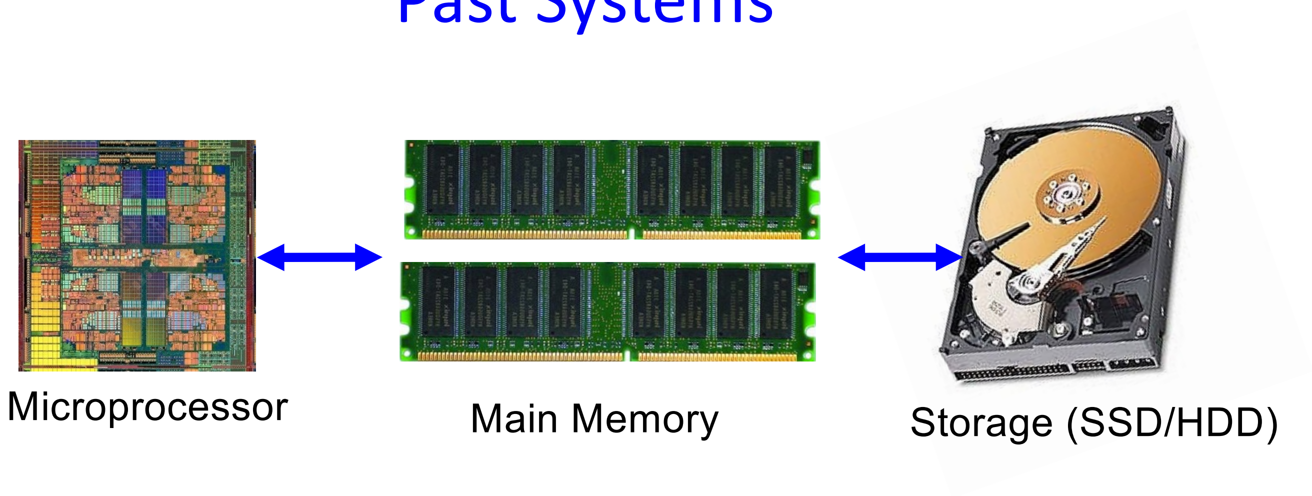


Assignment 1: Logic simulator

Hw/Sw Interaction an Important Skill

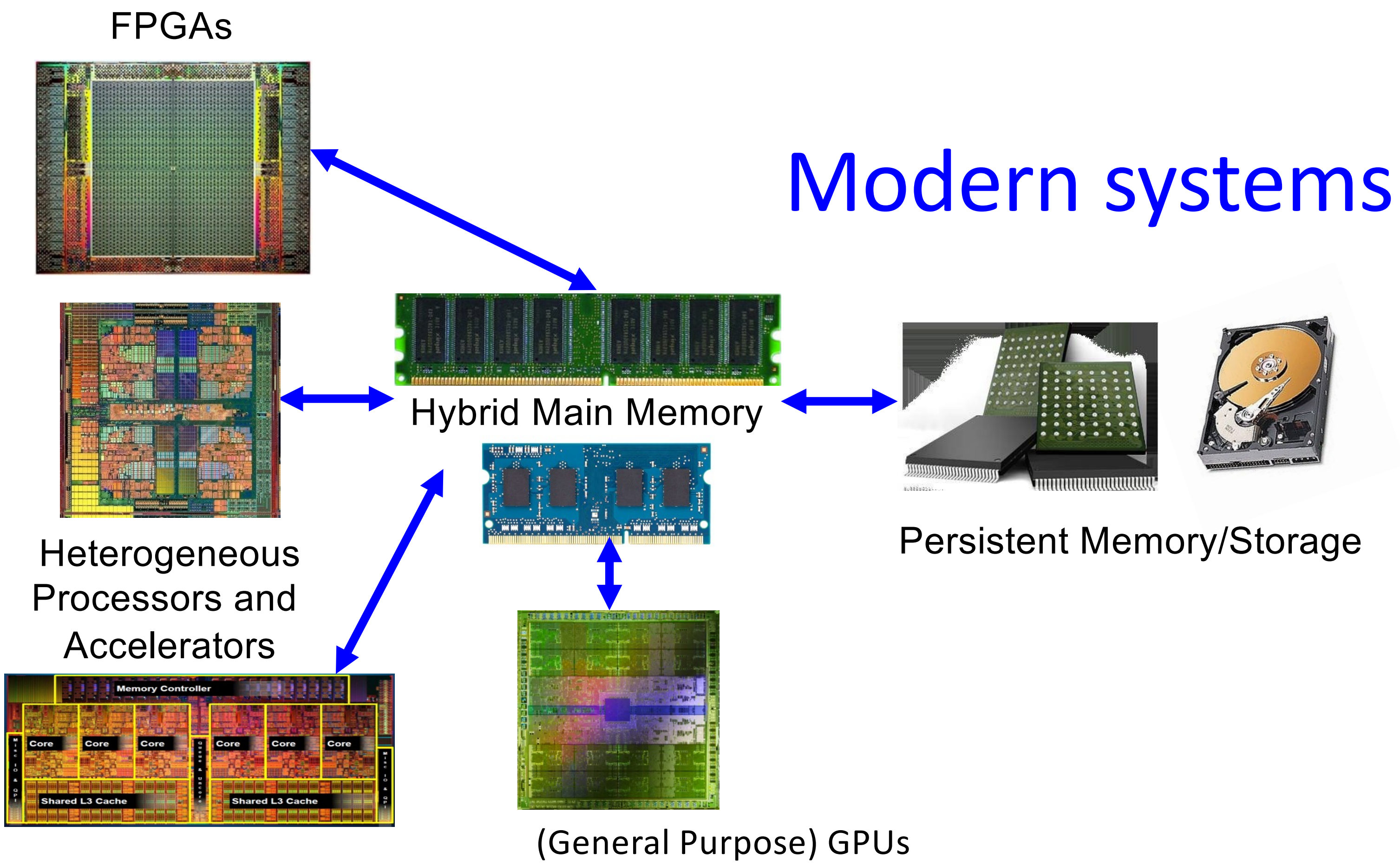
- Hardware is increasingly heterogeneous
- Programmers today need a good understanding of what the hardware offers

Past Systems



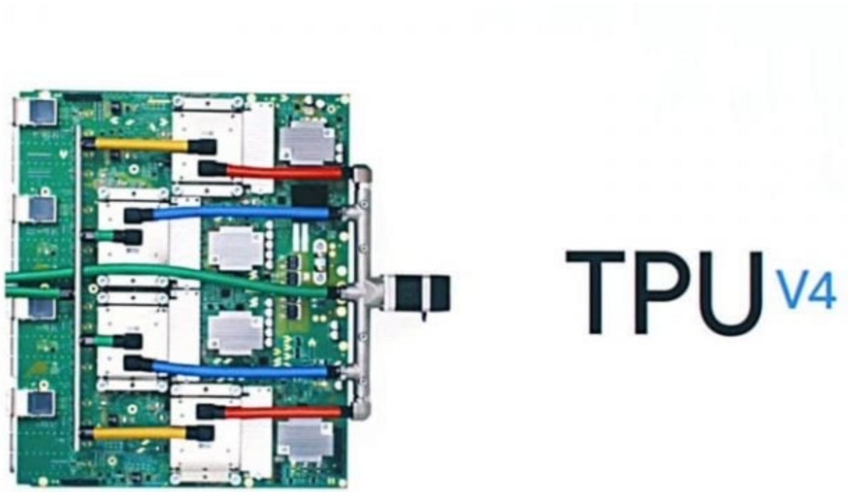
Hw/Sw Interaction an Important Skill

- Programmers today need a good understanding of what the hardware offers



Hw/Sw Interaction an Important Skill

- Programming models are domain-specific
  - New ML/AI accelerators
  - Programming models intertwined with hardware details
  - No luxury of a commodity language



New Important Metrics

- Traditional metrics to evaluate a computer
  - Performance
  - Energy efficiency
  - Battery life
  - Cost

New Important Metrics

- Today, we care a lot about
  - Reliability
  - Sustainability
  - Security

Sustainability

- Sustainability is an important concern
  - ICT contribution to global GHG emissions around 4%, and increasing
  - Need to understand emissions during manufacturing of computers and in deployment
  - Sustainability is the latest sub-area in the field of computer architecture

Security Demands Robust Hardware

- Security trumps performance in many environments
  - Recent cyber attacks target vulnerabilities at the bottom
  - Early mitigations handled in software
  - Today's hardware built for performance
- Need to build fundamentally secure Hw/Sw systems

Hw/Sw Interaction an Important Skill

- Writing compilers, operating systems, virtual machines

Hw/Sw Interaction an Important Skill

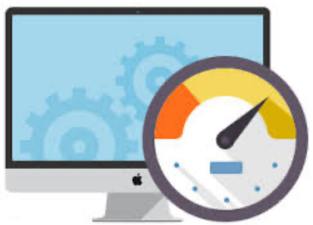
- Writing compilers, operating systems, virtual machines

- Programming embedded computers
  - Nano drones, IoT devices, wearable computing

Hw/Sw Interaction an Important Skill

- Writing compilers, operating systems, virtual machines
- Programming embedded computers
  - Nano drones, IoT devices, wearable computing

- Debugging performance issues better than the average programmer



Programming Perspective

- Characteristics of great code
  - Is easy to read and maintain
  - Well-commented
  - Follows good style guidelines
  - Easy to modify
  - Well-documented
  - Well-tested and correct

COMP2300 Emphasizes

- Characteristics of great code
  - Is easy to read and maintain
  - Well-commented
  - Follows good style guidelines
  - Easy to modify
  - Well-documented
  - Well-tested and correct
  - Uses the CPU efficiently
  - Uses the memory efficiently
  - Uses system resources efficiently

Our Ultimate Goals

- Make you think broadly *about computing*
- Make you think critically
- Give INSIGHT (into the NATURE of things)

Some Comments from Last Year

- COMP2300 is "empowering"
- **"After building the CPU, I know exactly what my program is doing"**
- "When I engage with a computer expert, I know what they are talking about"
- "I use the knowledge of modern processors from this course every single day" *Now a Security professional*

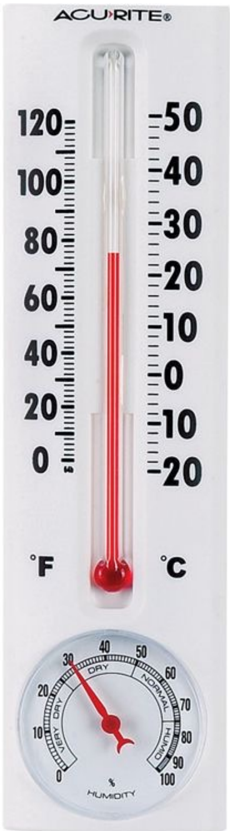
Representing Information

- Continuous or analog variables can take an infinite number of values
  - Frequency of oscillation
  - Voltage
  - Position
  - Volume

Representing Information

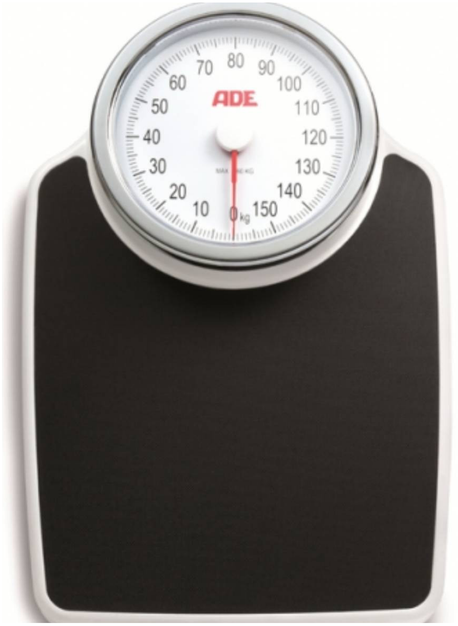
- **Using continuous variables to represent information is hard for the computer to deal with**
  - Sometimes difficult to measure precisely
  - Difficult to store
  - Difficult to copy

Representing Information



- Mercury's volume represents temperature

Representing Information



- Position of needle represents weight

Representing Information

- Modulated grooves on vinyl record represent sound

Representing Information

- Chemical properties of film represent captured image

Representing Information

- Measuring things and storing data by **analogy** has been the **predominant** approach in history

- Engineers call signals that can take an **infinite number of values** analog even when they are not an analogy for something

Representing Information

- If we don't want to represent data as something with **potentially infinitely varying analog values**, what can we do?

- Use the **Digital** approach

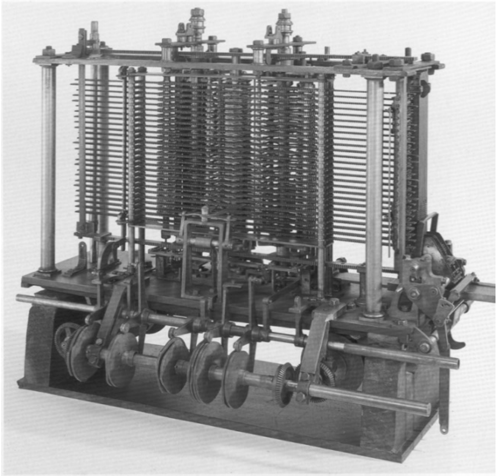
Representing Information

Digital Systems represent information with discrete-valued variables

Variables with a finite number of distinct values

Representing Information

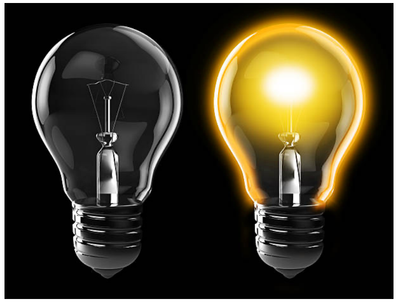
- Charles Babbage's analytical engine used ten discrete values



- Used mechanical parts such as gears with ten positions 0 – 9

Representing Information

Modern digital systems use a binary (*two-valued*) representation



0 1

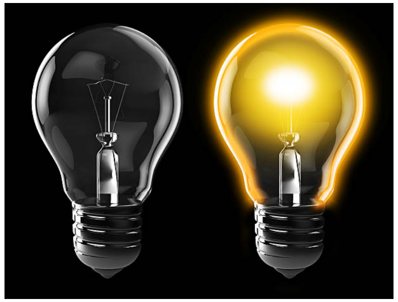
0 1

0 1

Representing Information

High voltage: Presence of something meaning **1**

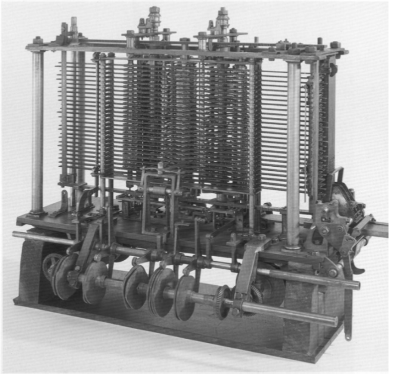
Low voltage: Absence of something meaning **0**



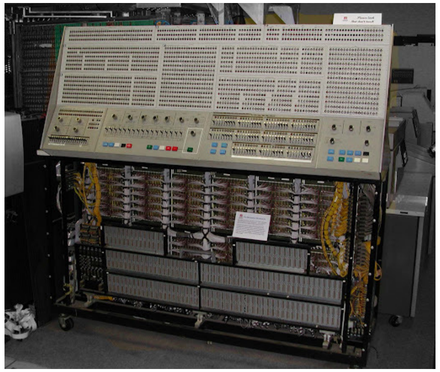
0 1

Why Voltage?

- Mechanical parts are not easy to scale to do large computations – circa 1850 Babbage engine



- Some 1964 computers (CDC 6600 & IBM 360)

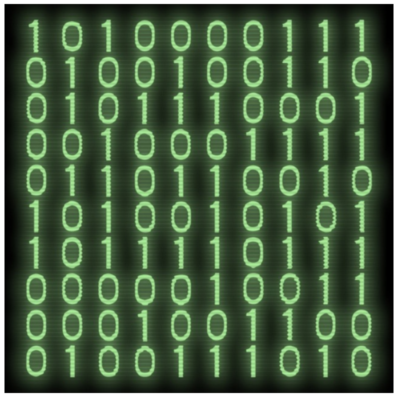


- 2020 Apple M1 400 mm^2 and 16 billion transistors

Representing Information

We need more than 0 and 1 to represent large quantities and sets

Data is represented using a sequence of symbols where each symbol is 0 or 1



Binary Representation

Digital systems internally use “voltages” for representing binary variables

→ Low voltage means 0

→ High voltage means 1

B I N A Y   D I G I T

- A bit is a unit of information
- A binary variable represents one bit of information
- To represent discrete sets with more than two elements, we combine multiple bits into a binary code

Binary Codes

Suppose we want to represent four colors: {red, blue, green, black}

- How many bits of information do I need?
- (00, 01, 10, 11)
- The assignment of the **2-bit binary code** to colors is *ad-hoc*
- Also legitimate is: (10, 11, 00, 01)

How many bits of information do we need to represent the alphabet set in English?

- **For 26 alphabets, we need 5 bits**

Information Content in a Binary Code

$$D = Log_2 N \; bits$$

- The color set has four states: N = 4, # bits = 2
- The alphabet set has 26 states: N = 26, # bits = 5
- Conversely,
  - If D is 2, N = 4
  - If D is 5, N = 32

Why do computers use binary?

- Two symbols enable simplified hardware and improved reliability

- Keep **complexity** and **cost** under control

- It is easy to use the amazing transistor as a **switch**!
  - We will see later

TRUE and FALSE

0 1

F T

False Off

True On

- True and False are called logical values
  - Logical variable is one that can be 1 or 0 (True or False)

Decimal Number System

- Base 10 means 10 digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Multiple digits form longer decimal numbers
- Each column of a decimal number has 10 times the weight of the previous column

1's column
10's column
100's column
1000's column

coefficient

power of 10

$$9742 = 9 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 2 \times 10^0$$

nine thousands, seven hundreds, four tens, two ones

Range of Decimal Numbers

- An N-digit decimal number represents one of 10^N possibilities
  - 0, 1, 2, 3, ..., $10^N - 1$

- **3 digits:** 1000 possibilities in the range 0 – 999

Binary Numbers

- Base 2 means 2 digits (0, 1)
- Multiple bits form longer binary numbers
- Each column of a binary number has **2** times the weight of the previous column

8's column
4's column
2's column
1's column

$1001 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

coefficient

power of 2

one eight	one four	zero two	one one

Range of Binary Numbers

An N-bit binary number represents one of 2^N possibilities

- 0, 1, 2, 3, ..., $2^N - 1$
- 3 bits: 8 (= 2 X 2 X 2) possibilities in the range 0 – 7
- 4 bits: ?
- 5 bits: ?
- 10 bits: ?

Powers of 2

Columns #	Power of 2	Weight
0	2^0	1
1	2^1	2
2	2^2	4
3	2^3	8
4	2^4	16
5	2^5	32
6	2^6	64
7	2^7	128
8	2^8	256
9	2^9	512

Columns #	Power of 2	Weight	
10	2^{10}	1024	Kilo
11	2^{11}	2048	
12	2^{12}	4096	
13	2^{13}	8192	
14	2^{14}	16384	
15	2^{15}	32768	
16	2^{16}	65536	

Powers of 2

Power of 2	Decimal Value	Abbreviation	
2^{10}	1024	Kilo (K)	~ 1000
2^{20}	1048576	Mega (M)	~ 1000, 000
2^{30}	1073741824	Giga (G)	~ 1000, 000, 000

What is 2^{24} in decimal?

- 2^{20} X 2^{4} = 1 M X 16 = 16 M

What is 2^{17} in decimal?

- 2^{10} X 2^{7} = 1 K X 128 = 128 K

Terminology

Byte

- 8 bits

Nibble

- 4 bits

Word

- Machine-dependent
- 8 – 16 bits (gadgets)
- 32 – 64 bits (high-end)

Most Significant Bit

1	0	0	0	0	0	0	1

The bit in the highest position

Least Significant Bit

1	0	0	0	0	0	0	1

The bit in the lowest position

Terminology

Most Significant Byte

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

The byte in the ***highest*** *position*

Least Significant Byte

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

The byte in the ***lowest*** *position*

Rev: Binary Codes

Column weight	2^2	2^1	2^0
Column #	2	1	0
	0	0	0
	0	0	1
	0	1	0
	0	1	1
	1	0	0
	1	0	1
	1	1	0
	1	1	1

3-bit
8 elements

- Write combinations in a systematic way
- Note how often the bit flips in each column
- Can represent any arbitrary set with a code

Rev: Binary Codes

0
1

1-bit
2 elements

0	0
0	1
1	0
1	1

2-bit
4 elements

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

3-bit
8 elements

0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

4-bit
16 elements

- Write combinations in a systematic way
- Note how often the bit flips in each column
- Can represent any arbitrary set with a code

Decimal to Binary Conversion

Method # 1: Find the largest power of 2, subtract, and repeat

Example: Convert 53_{10} to binary

53	32 X 1
53 – 32 = 21	16 X 1
21 – 16 = 5	4 X 1
5 – 4 = 1	1 X 1

2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	1

Decimal to Binary Conversion

Method # 2: Repeatedly divide by 2, remainder goes in each column

Example: Convert 53_{10} to binary

53/2	=	26	R: 1
26/2	=	13	R: 0
13/2	=	6	R: 1
6/2	=	3	R: 0
3/2	=	1	R: 1
1/2	=	0	R: 1

2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	1

Hexadecimal Numbers

Motivation: Tedious and error-prone to write long binary numbers

Hexadecimal or base 16: A group of four bits represent 2^4 or 16 possibilities

16 digits: 0 – 9, A, B, C, D, E, F

Column weights: 1, 16, 16^2 (or 256), 16^3 (or 4096)

Hexadecimal Numbers

Hex Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Binary to Hexadecimal

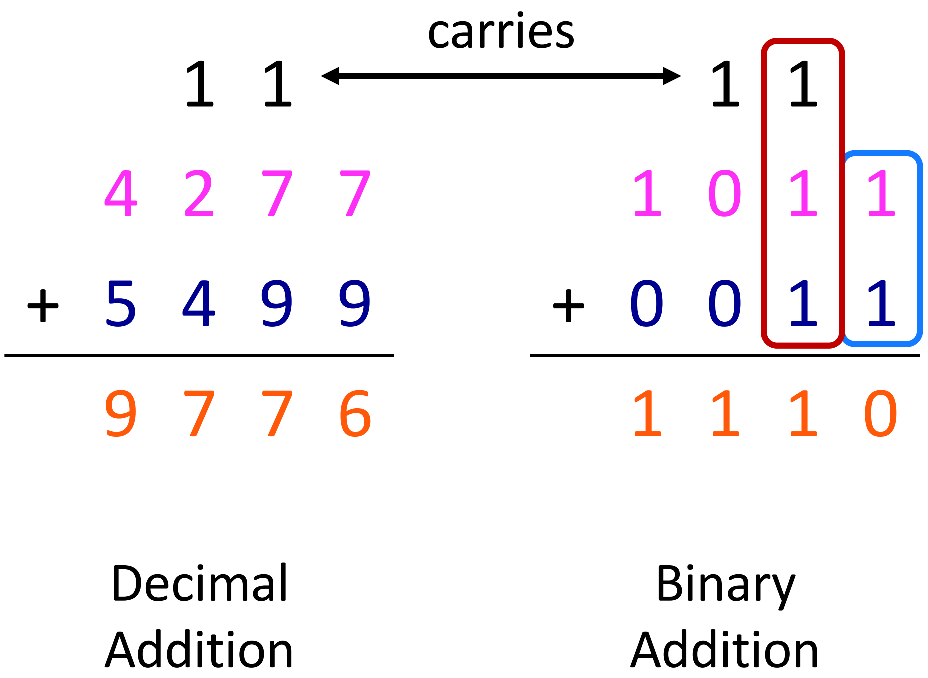
Binary	1	1	1	1	0	1	1	1
Hexa	F	7						

Binary	1	1	1	1	1	1	1	0
Hexa	F	E						

Hexadecimal to Binary

Hexa	D	7	4	1												
Binary	1	1	0	1	0	1	1	1	0	1	0	0	0	0	0	1

Binary Addition



1 + 1 = 2 (10 in binary), but a binary variable can either be 0 or 1

- We record the 1's digit (0), and carry the 2's digit (1) over to the next column

1 + 1 + 1 = 3 (11 in binary), but a binary variable can either be 0 or 1

- We record the 1's digit (1), and carry the 2's digit (1) over to the next column

Overflow

- A = 1111 and B = 1111
  - A + B does not fit in the largest value four bits can represent
- **Overflow:** When the result is too big to fit inside the available bits
- **Detection:** If there is a carry bit out of the most significant column

	1	1	1		
	1	1	1	1	15
+	1	1	1	1	15
1	1	1	1	0	30

Signed Binary Numbers

Signed Binary Numbers

- We need both positive and negative numbers to solve real-world problems

- *How do we make a string of 1 and 0 represent both positive and negative numbers?*

- If we write all possible combinations of 0 and 1 in a disciplined fashion, maybe we can find a way

Most significant bit		least significant bit
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Signed Binary Numbers

- Use the *most significant bit* to represent the sign: 0 means positive and 1 means negative

- **N bit sign/magnitude system:** 1 bit for sign and N – 1 bits for magnitude (**absolute**)

			Decimal
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	-0
1	0	1	-1
1	1	0	-2
1	1	1	-3

Drawbacks of Sign/Mag Rep.

- Ordinary binary addition does not work for sign/magnitude numbers
  - What is the sum of +3 and -3 and does the result make sense?

- Zero has two representations (awkward)

			Decimal
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	-0
1	0	1	-1
1	1	0	-2
1	1	1	-3

One's Complement

- 1's complement was tried in early computers, such as, Control Data Corporation (CDC) 6600
- **Negative number:** Flip all bits of the binary representation of a positive integer
- Suffers from the same problems as the sign/magnitude representation

			Decimal
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	-3
1	0	1	-2
1	1	0	-1
1	1	1	-0

Two’s Complement

- A third system of representation for signed integers
- Ordinary addition works
- There is a single representation for *zero*
- Used in almost all computers today

Problem: If A + B = C, and A is known, then find B, such that C = 0

			Binary				Decimal
	A	=	0	1	0	1	?
+	B	=	?	?	?	?	?
	C	=	0	0	0	0	0

Problem: If A + B = C, and A is known, then find B, such that C = 0

			Binary				Decimal
	A	=	0	1	0	1	+5
+	B	=	?	?	?	?	?
	C	=	0	0	0	0	0

Problem: If A + B = C, and A is known, then find B, such that C = 0

			Binary				Decimal
	A	=	0	1	0	1	+5
+	B	=	?	?	?	?	-5
	C	=	0	0	0	0	0

Problem: If A + B = C, and A is known, then find B, such that C = 0

			Binary				Decimal
	A	=	0	1	0	1	+5
+	B	=	1	0	1	1	-5
	C	=	0	0	0	0	0

Problem: If A + B = C, and A is known, then find B, such that C = 0

What is the relationship between A and B?

			Binary				Decimal
	A	=	0	1	0	1	+5
+	B	=	1	0	1	1	-5
	C	=	0	0	0	0	0

Some Observations

Observation # 1: If A + B = C, and A is +5, and C is 0, then B must be -5. (We have found a new representation for negative numbers.)

Observation # 2: To transform A to B (i.e., +5 to -5), we need to take 1's complement of A and then add +1. We say that B is **2's complement** of A

Observation # 3: Like sign/magnitude numbers, positive numbers have the MSB set to 0, and negative numbers have the MSB set to 1

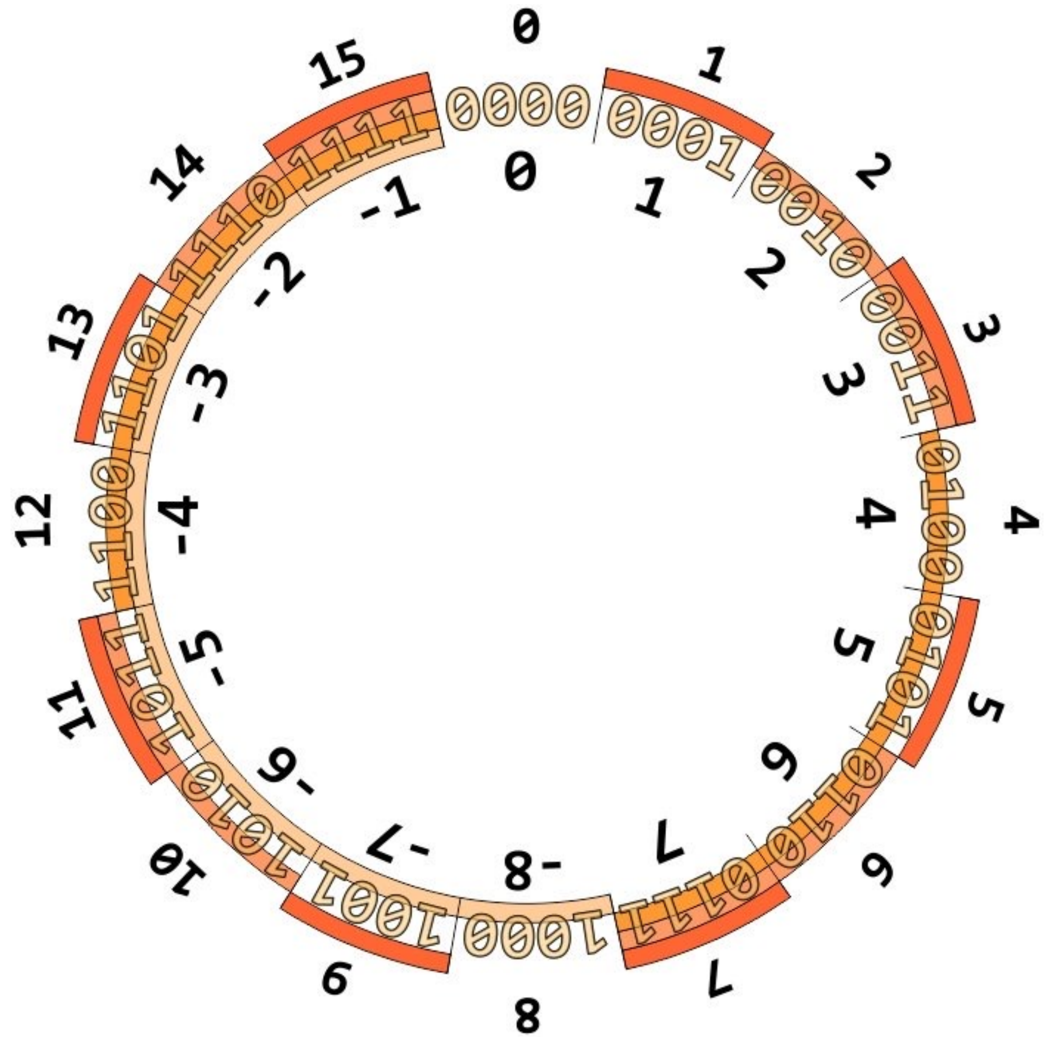
Some Observations

Observation # 4: Ordinary addition works

- What is the sum of +3 and -3 in two's complement system, and *does the result make sense*?
- Since ordinary addition works, a circuit to add numbers can handle both addition and subtraction
  - Recall that, X – A is equivalent to X + (–A)

		1		
	0	1	1	+3
+	1	0	1	-3
	0	0	0	

2's Complement Circle



More Observations

Observation # 5: There is only one representation for *zero*

Observation # 6: There is one more negative number than positive number

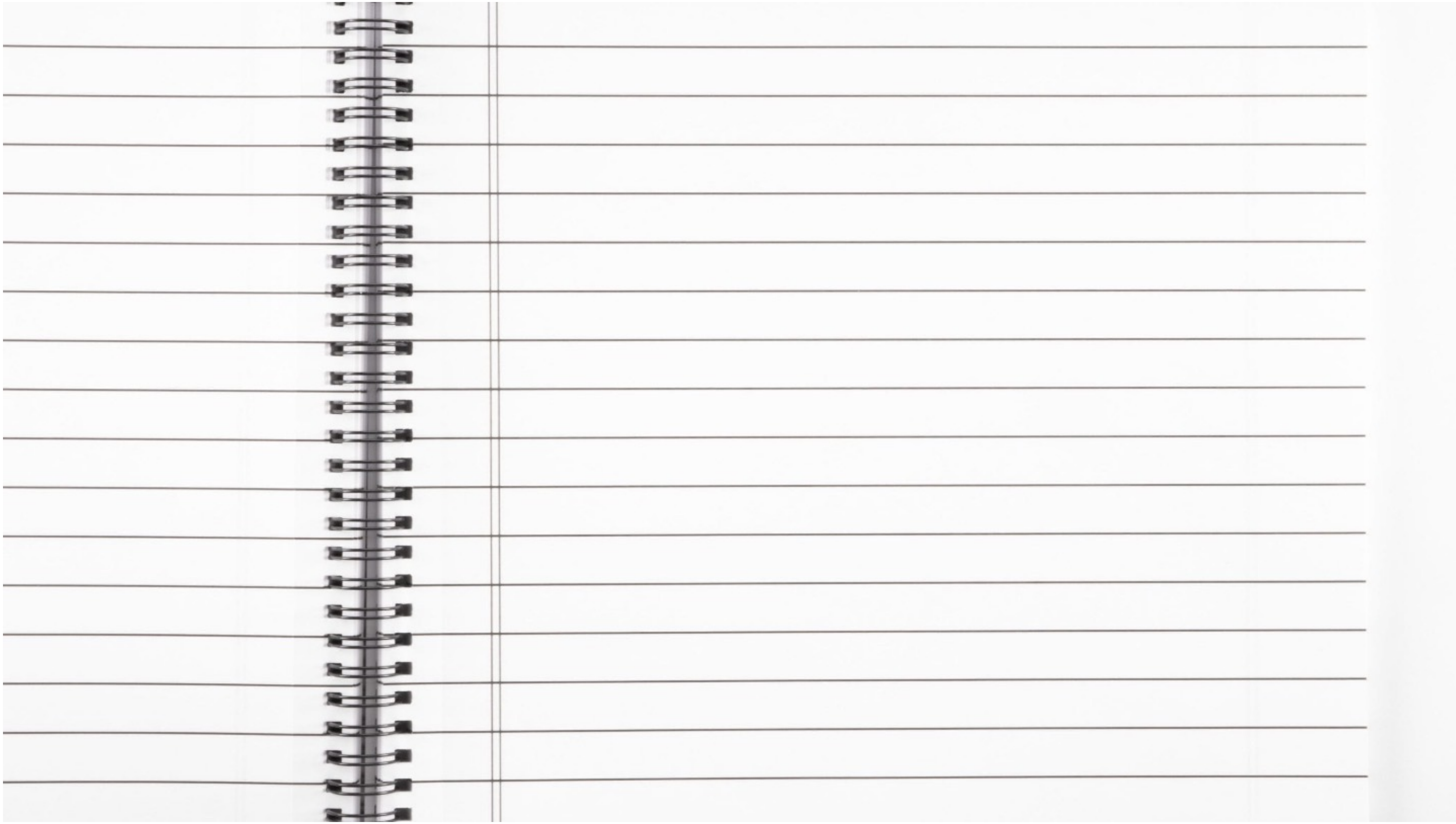
- With 3 bits, this number is 100
- With 4 bits, this number is 1000
- This negative number has no positive counterpart
- It is called the *weird number*
- The 2’s complement of the weird number is the weird number (verify!)

2's Complement to Decimal

- If **MSB is 0**
  - It is a positive number. The magnitude is represented by the remaining N – 1 bits

- If **MSB is 1**
  - It is a negative number. Take the two's complement of the (binary) number. The magnitude of the negative number) is represented by the N – 1 bits

Practice and test your understanding using the two's complement circle



Overflow in 2's Complement

- Suppose we have two 5-bit numbers
  - A = 01001 and B = 01011
  - What is A + B?
  - What is the largest value 5 bits can represent in 2's complement?
- Overflow
  - The result is too big to fit inside the available bits
  - Sum of two positive integers cannot produce a negative integer!

			1	1		
	0	1	0	0	1	+9
+	0	1	0	1	1	+11
	1	0	1	0	0	-12

Overflow in 2's Complement

- Suppose we have two 5-bit numbers
  - A = 10100 and B = 11010
  - What is A + B?
  - What is the *smallest* value 5 bits can represent in 2's complement?
- Overflow
  - The result is too big to fit inside the available bits
  - Sum of two negative integers cannot produce a positive integer!

	1	0	1	0	0	-12
+	1	1	0	1	0	-6
	0	1	1	1	0	14

Overflow in 2's Complement

Observation # 1: If two number being added have the same sign bit and the result has the opposite sign bit (easy!)

Observation # 2: Unlike unsigned numbers, a carry out of the most significant bit does not indicate overflow

Observation # 3: The sum of a negative number and a positive number never produces an overflow (**prove yourself!**)

Range of Number Systems

Number System	Minimum	Maximum
Unsigned	0	$2^N - 1$
Sign/Magnitude	$-2^{N-1} + 1$	$2^{N-1} - 1$
Two's Complement	-2^{N-1}	$2^{N-1} - 1$

N = 3

Unsigned: 0 to 7

Sign/Magnitude: -3 to 3

2's Complement: -4 to 3

N = 4

Unsigned: 0 to ?

Sign/Magnitude: -? to ?

2's Complement: -? to ?

Comparing Number Systems

Binary Representation	Decimal Value Represented			
	Unsigned	Signed Magnitude	1's Complement	2's Complement
000	0	0	0	0
001	1	1	1	1
010	2	2	2	2
011	3	3	3	3
100	4	-0	-3	-4
101	5	-1	-2	-3
110	6	-2	-1	-2
111	7	-3	-0	-1

Quiz: See any errors?

				Unsigned	Signed	1's Comp.	2's Comp.
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	2	2	2	2
0	0	1	1	3	3	3	3
0	1	0	0	4	4	4	4
0	1	0	1	5	5	5	5
0	1	1	0	6	6	6	6
0	1	1	1	7	7	7	7
1	0	0	0	8	-0	-7	-1
1	0	0	1	9	-1	-6	-2
1	0	1	0	10	-2	-5	-3
1	0	1	1	11	-3	-4	-4
1	1	0	0	12	-4	-3	-5
1	1	0	1	13	-5	-2	-6
1	1	1	0	14	-6	-1	-7
1	1	1	1	15	-7	-0	-8

Sign Extension

Question: What is the difference between the 16-bit and 4-bit numbers below?

16-bit number	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
4-bit number	**0**	**1**	**0**	**1**												

Answer: None. They both represent the positive number 5

Leading zeros do not impact the magnitude of a binary number

There are times when it is useful to allocate a small number of bits to represent a value

Sign Extension

- What value does the two numbers below represent?

16-bit number (**A**)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
4-bit number (**B**)	**1**	**1**	**0**	**1**												

- What is the sum of **A** and **B**?
  - **Scenario # 1:** Assume the absence of bits in **B** to be 0
  - **Scenario # 2:** Assume the absence of bits in **B** to be 1

Scenario # 1

	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	+13
+	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	-3

X

The assumption that appending 0's will lead to correct addition was wrong

Scenario # 2

	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	+13
+	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	-3

✓

The assumption that appending 1's will lead to correct addition was right

Sign Extension

- Leading 0's do not change the magnitude of the positive number

- Leading 1's do not change the magnitude of the negative number

When a 2's complement number is extended to more bits, the sign bit must be copied into the most significant bit positions. We refer to the operation as Sign-EXTension or SEXT

Building Block of Computers

Transistors

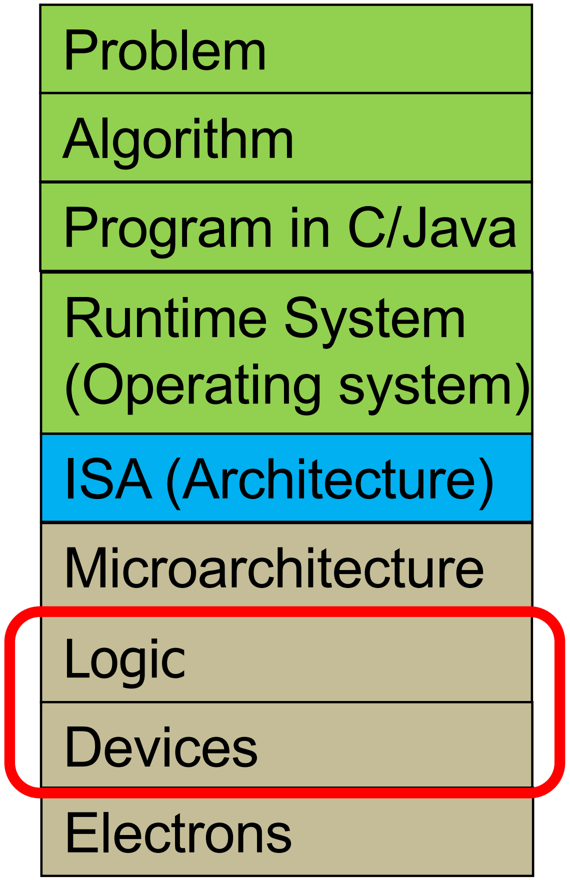
157

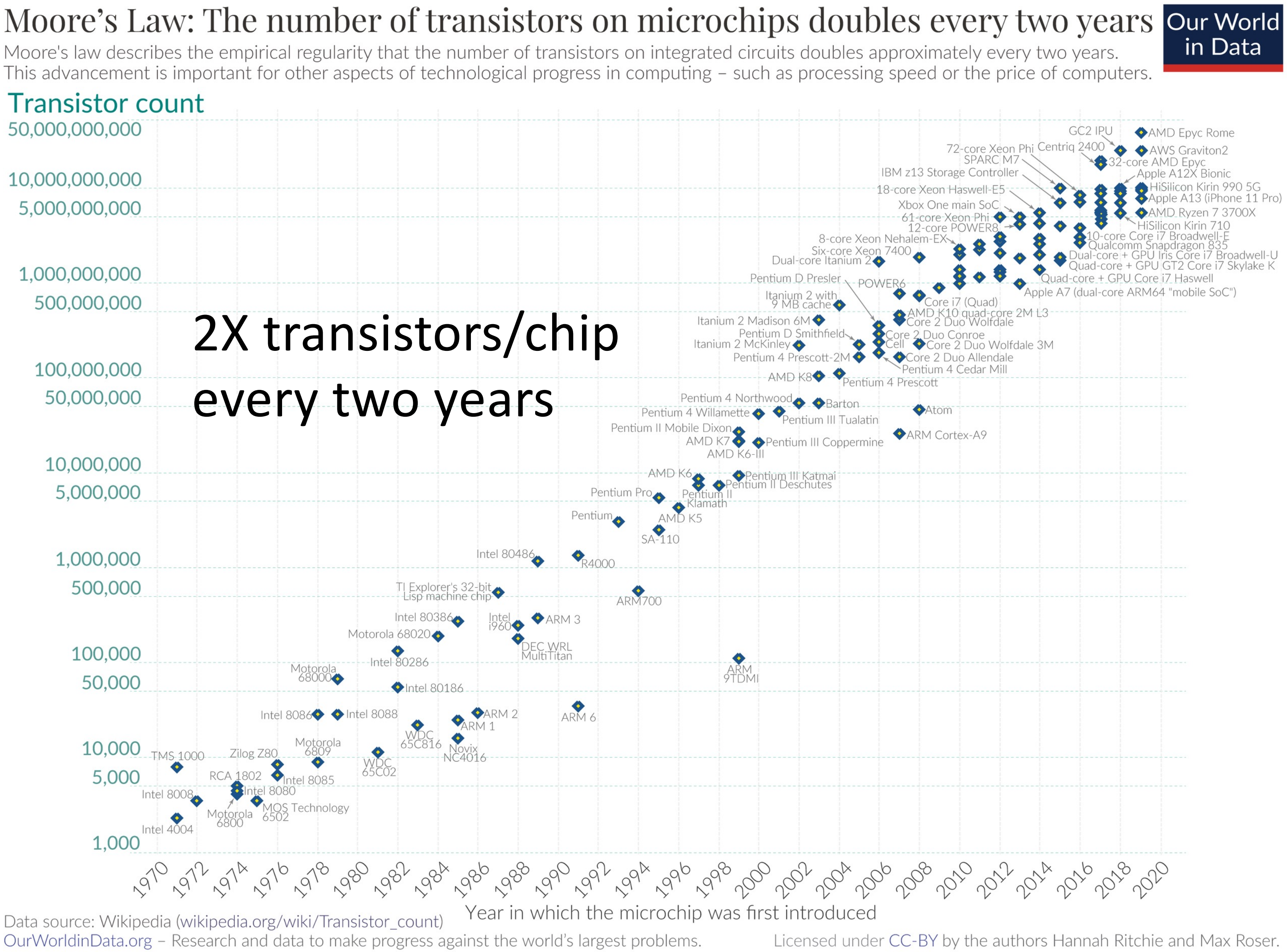
We are Here.

Transistors

- **Computers are built from billions of small and simple structures called transistors**
  - 1970: Few 1000s of transistors
  - Apple's M2 Max: **50+ Billion** transistors
  - **Moore's Law:**Transistor count double in 18 months
  - Computers with improved capability over time due to a large # transistors at the device level

- **We will cover**
  - How MOS transistor works (as a logic element)?
  - How transistors are connected to form logic gates?
  - How logic gates are interconnected to form larger units that are needed to construct a computer





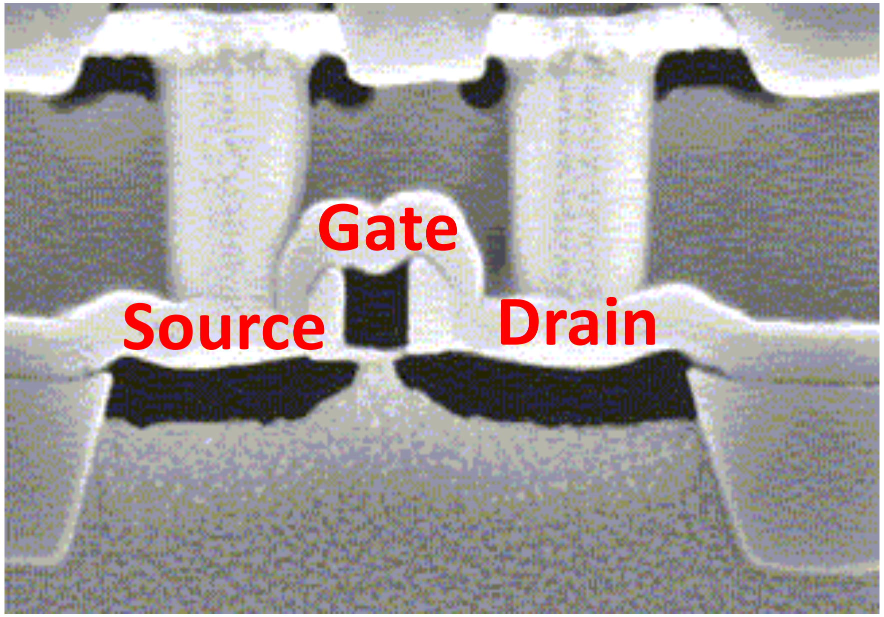
2X transistors/chip
every two years

Transistors

- Sections 1.6 and 1.7 in Harris & Harris provide more technical explanations than what we will cover

MOS Transistor

- **MOS** stands for
  - Conductors (**M**etal)
  - Insulators (**O**xide)
  - **S**emiconductors

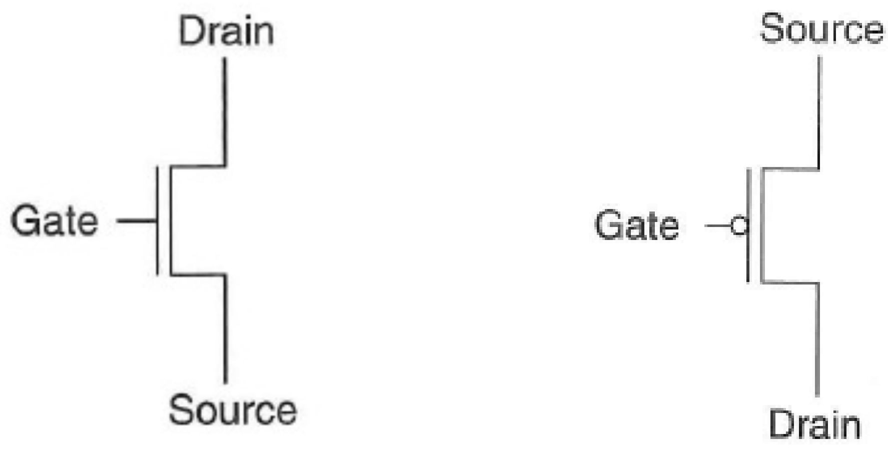


- MOS transistor has three terminals

- We can combine many of these to form logic gates
  - *The electrical properties of metal-oxide semiconductors are well beyond the scope of what we want to understand in this course*
  - They are below our **lowest level of abstraction**
  - If transistors **misbehave**, an architect is at their mercy (unlikely to happen)

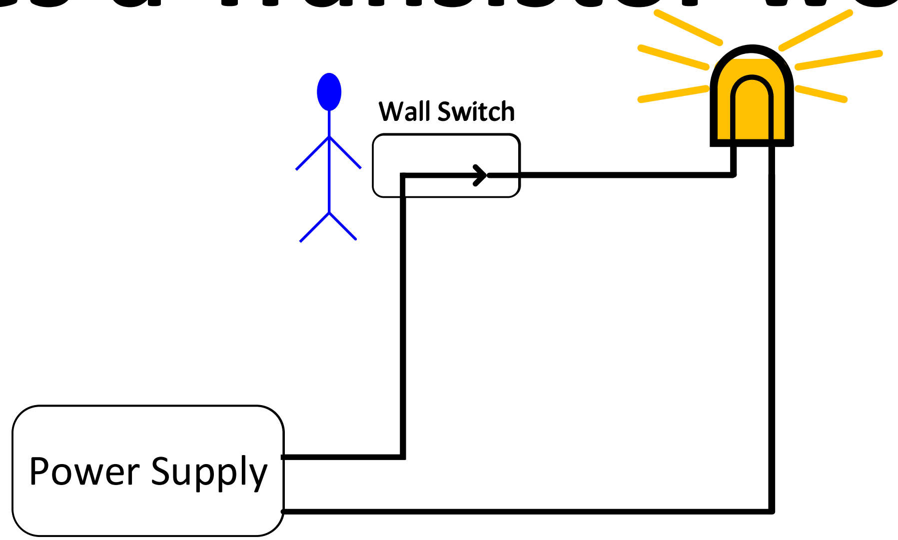
Two Types of MOS Transistors

- Two types: n-type and p-type
- They both operate “logically,” very similar to the way wall switches work



n-type p-type

How Does a Transistor work?



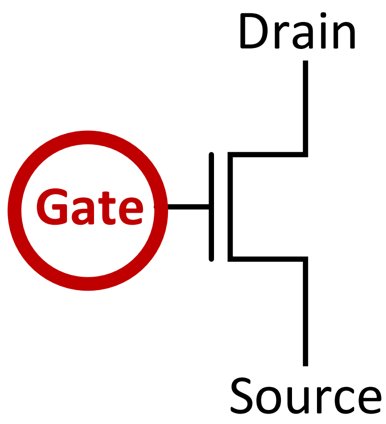
- For the lamp to glow, electrons must flow
- For electrons to flow, there must be a closed circuit from the power supply to the lamp and back to the power supply
- The lamp can be turned on and off by simply manipulating the wall switch to make or break the closed circuit

How Does a Transistor work?

- Instead of the wall switch, we could use an n-type of a p-type MOS transistor to make or break the closed circuit

If the gate of the n-type transistor is supplied with a **high** voltage, the connection from source to drain acts like a piece of wire (**we have a closed circuit**)

If the gate of the n-type transistor is supplied with **zero** voltage, the connection between source and drain is broken (**we have an open circuit**)

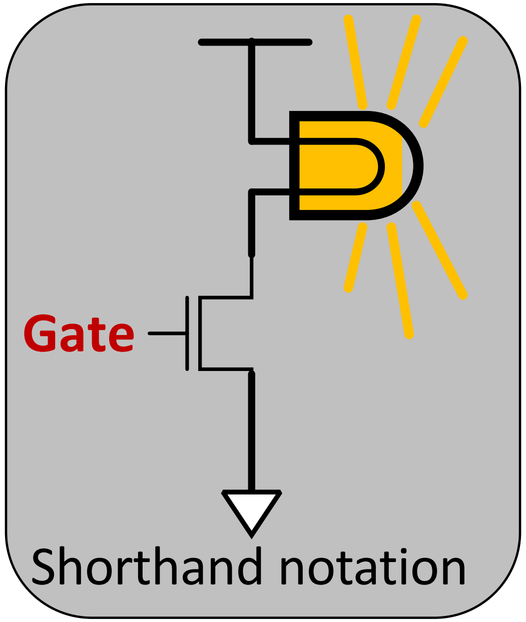


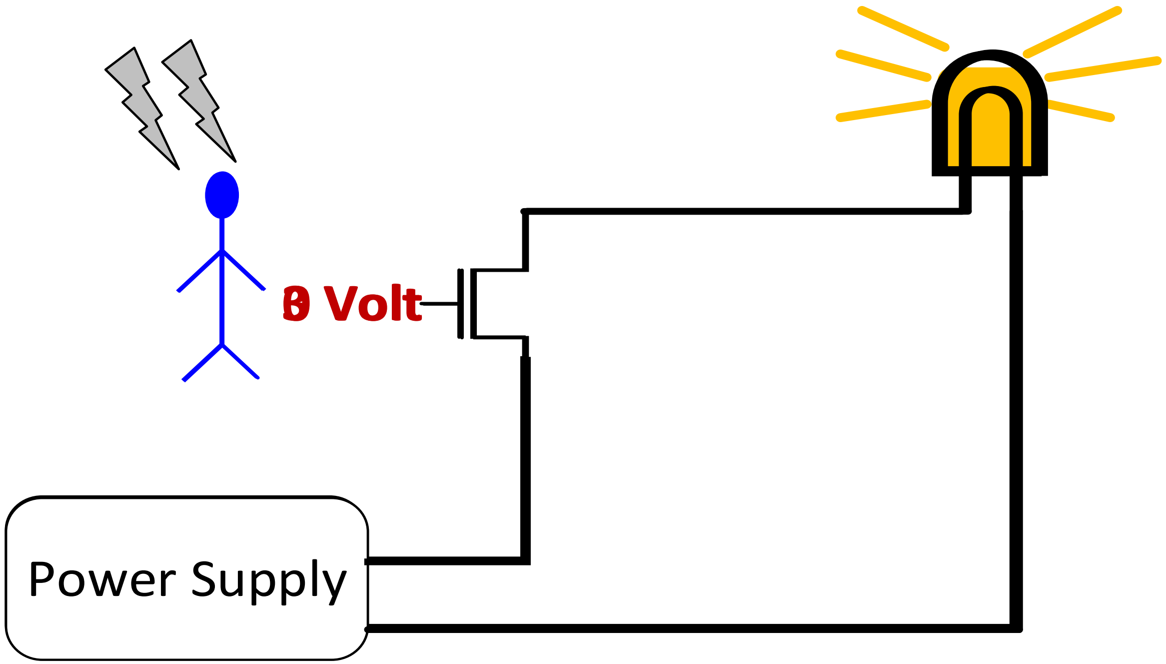
Schematic of an n-type MOS transistor

- Depending on the technology, high voltage can range from **0.3V** to **3V**

How Does a Transistor work?

- The n-type transistor in a circuit with a battery and a bulb



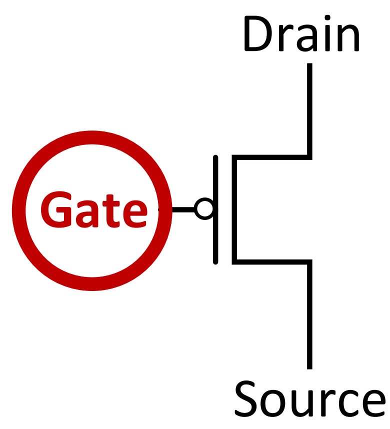


How Does a Transistor work?

- The p-type MOS transistor works in exactly the opposite fashion from the n-type transistor

The circuit is **open** when the gate is supplied with 3V

The circuit is **closed** when the gate is supplied with 0V

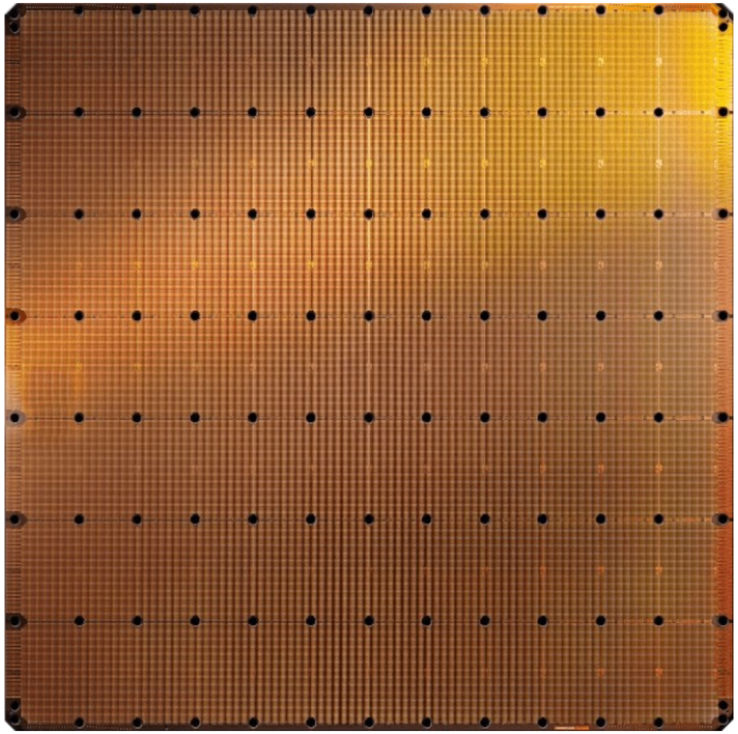


- Depending on the technology, high voltage can range from **0.3V** to **3V**

Some Examples of
Transistors as Building Blocks

168

Modern Special-Purpose ASIC



- The largest ML accelerator chip (2021)
- 850,000 cores

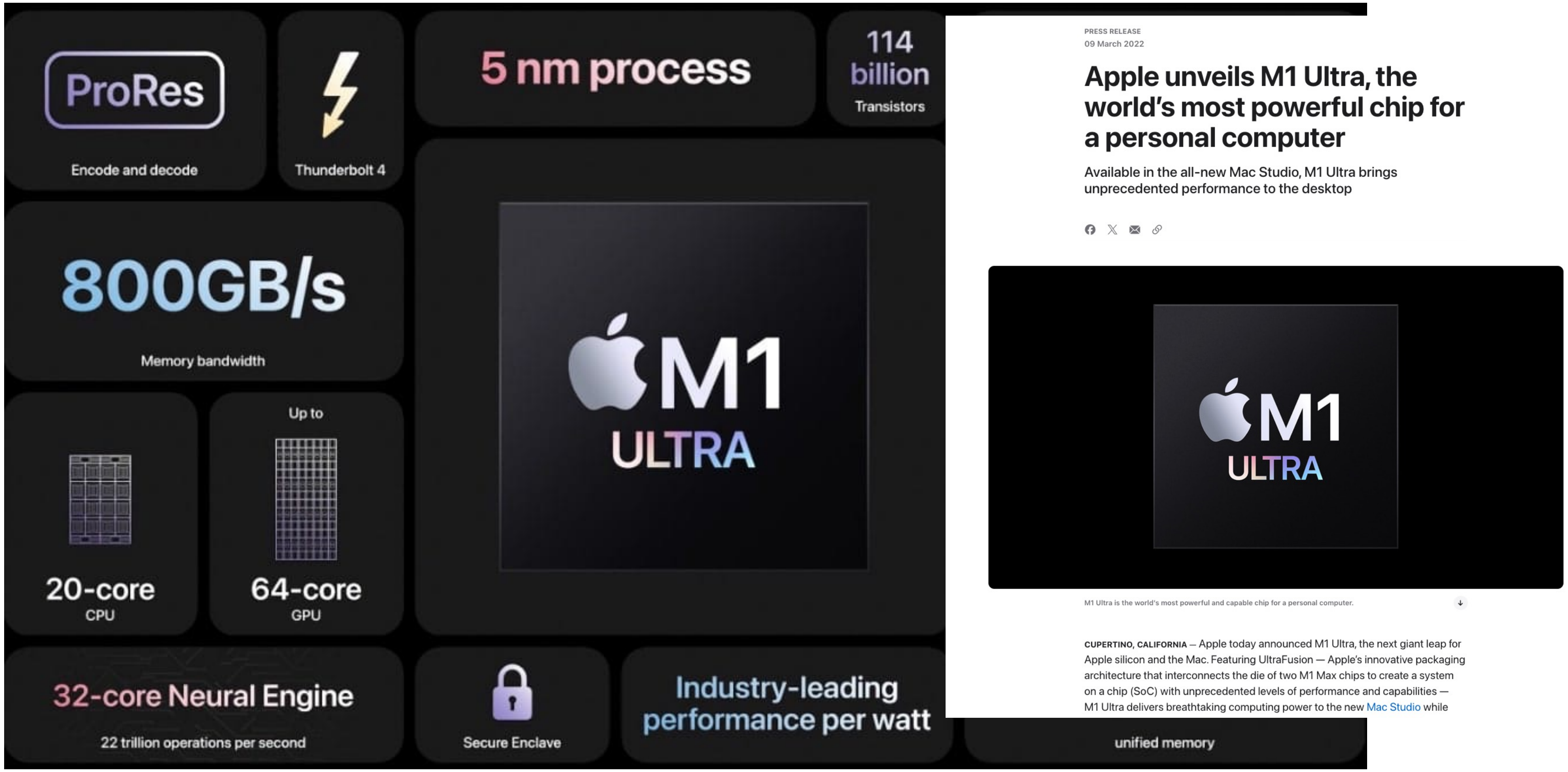
Cerebras WSE-2
2.6 Trillion transistors
46,225 mm^2

Largest GPU
54.2 Billion transistors
826 mm^2
NVIDIA Ampere GA100

https://www.anandtech.com/show/14758/hot-chips-31-live-blogs-cerebras-wafer-scale-deep-learning

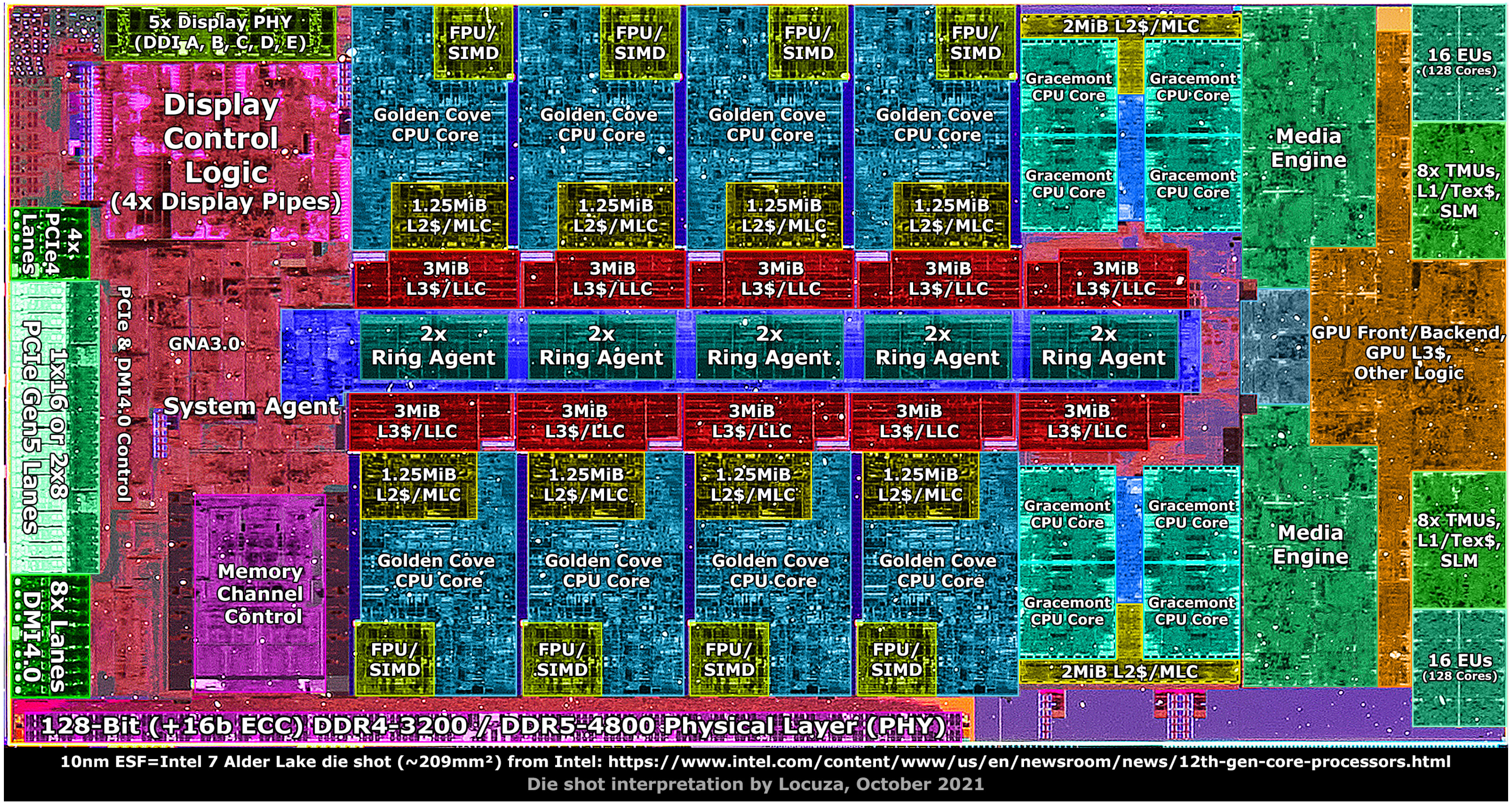
https://www.cerebras.net/cerebras-wafer-scale-engine-why-we-need-big-chips-for-deep-learning/

Apple M1 Ultra (2022)



https://stadt-bremerhaven.de/apple-neuer-m1-ultra-chip-ist-offiziell/

Modern General-Purpose Microprocessor

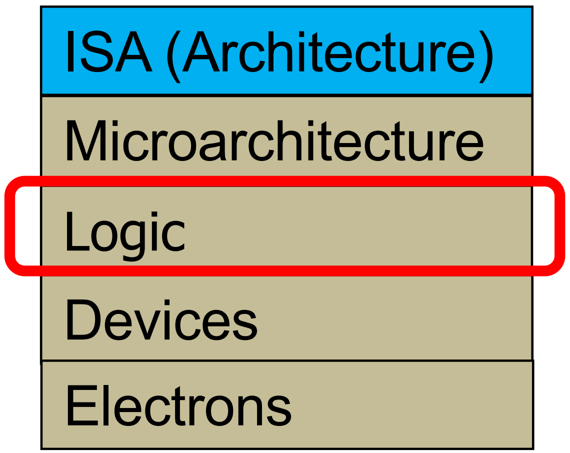


Intel Alder Lake, 2021

Logic Gates

One Level Higher in Abstraction

- Now, we know how a **MOS transistor** works
- How do we build logic structures out of individual MOS transistors
- We called these logic structures logic gates and they implement simple **Boolean** functions



Boolean Logic

- A system for describing logical statements where variables are TRUE or FALSE

- Defines *important but simple* logical operations on binary logical variables

Logic

- **Logic comes from reasoning or thinking**
- When presented with some facts, how to derive a valid conclusion
- A statement is either TRUE or FALSE
- **When many statements are combined, what is the conclusion?**

Origin of Logic Functions

- Canberra is the **CAPITAL** of Australia
- **AND** today I am in Canberra
- **Therefore, today, I am in the CAPITAL city of Australia**

- When it rains, I am **NOT** in office
- **AND** today it is raining
- **Therefore, today, I am NOT in office**

Boolean Logic Functions

- **Logical operations are the steppingstone for composing sophisticated digital circuits for performing arithmetic**

- **Boolean logic is a system of logic for describing statements consisting of binary variables**
  - Operations, rules, axioms, etc

Logic Functions vs Gates

- Logic gates are digital circuits that take one or more inputs and produce a binary output
- Logic gate is the physical realization of a logical function built with transistors
- The inputs are to the left, and the output is to the right
- The relationship between inputs and the output is described by a **truth table** or a **Boolean equation**

Truth Table

- A convenient way to describe the behavior of logical functions

- Suppose **A** and **B** are input operands and **Y** is the output
  - **A** can be 0 or 1
  - **B** can be 0 or 1
  - Four combinations (rows)
  - Three columns (2 inputs and an output)

- The Boolean equation for **Y**: **Y** = 0
  - The values of **A** and **B** does not matter

A	**B**	**Y**
0	0	0
0	1	0
1	0	0
1	1	0

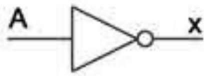
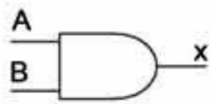
Truth Table with More Inputs

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

- Boolean Equation for output **Y**: **Y** = 1

Note: *Soon we will see more interesting logic functions than Y = 0 and Y = 1*

Logic Gates

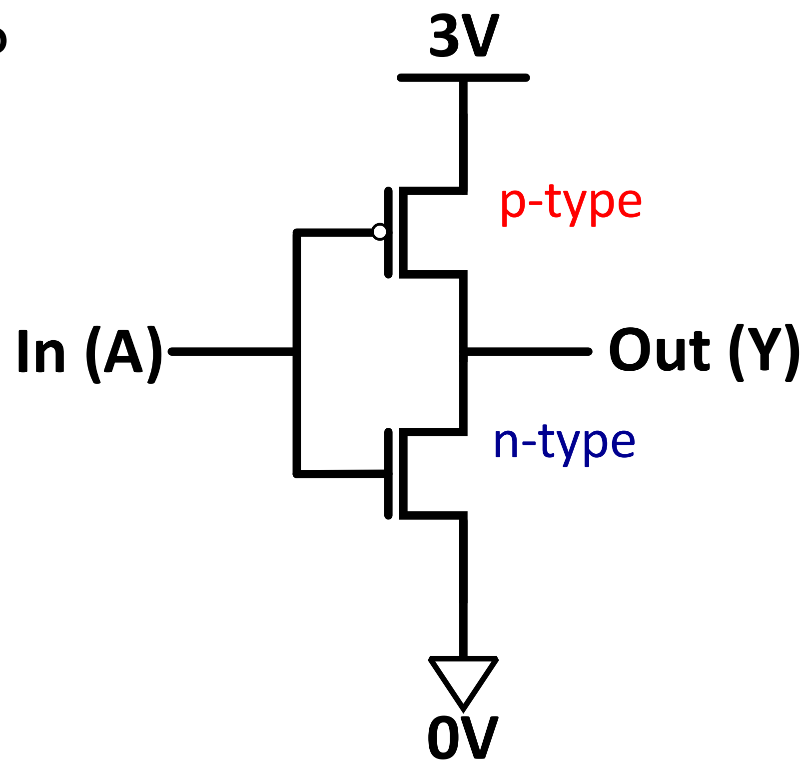
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol							
Truth Table	**A** | **X** 0 | **1** 1 | **0**	**B** **A** | **X** 0 0 | **0** 0 1 | **0** 1 0 | **0** 1 1 | **1**	**B** **A** | **X** 0 0 | **1** 0 1 | **1** 1 0 | **1** 1 1 | **0**	**B** **A** | **X** 0 0 | **0** 0 1 | **1** 1 0 | **1** 1 1 | **1**	**B** **A** | **X** 0 0 | **1** 0 1 | **0** 1 0 | **0** 1 1 | **0**	**B** **A** | **X** 0 0 | **0** 0 1 | **1** 1 0 | **1** 1 1 | **0**	**B** **A** | **X** 0 0 | **1** 0 1 | **0** 1 0 | **0** 1 1 | **1**

Making Logic Gates Using CMOS Technology

- Modern computers use both n-type and p-type transistors, called Complementary MOS (CMOS) technology
  - **nMOS + pMOS = CMOS**

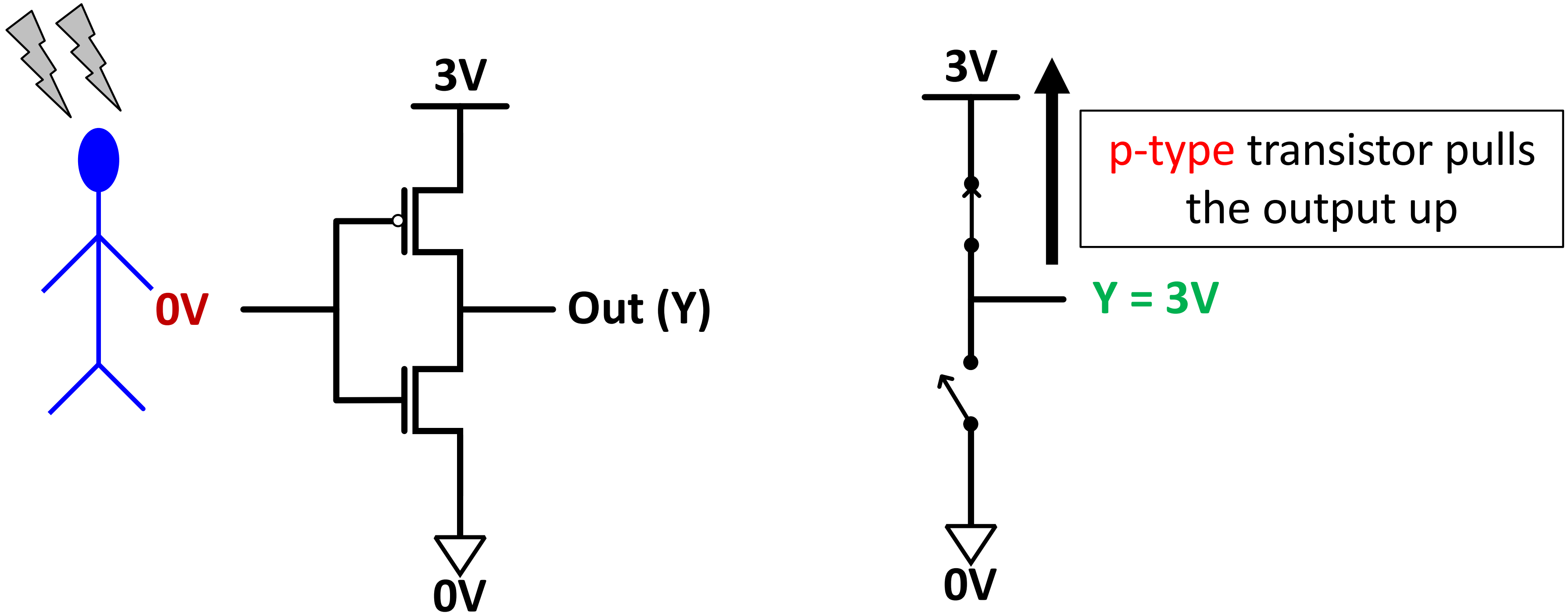
CMOS Technology Fundamentals

- Let’s look at the simplest logic structure that exists in a modern computer
  - What does this circuit do?



CMOS Technology Fundamentals

- What happens when the input is connected to 0V?

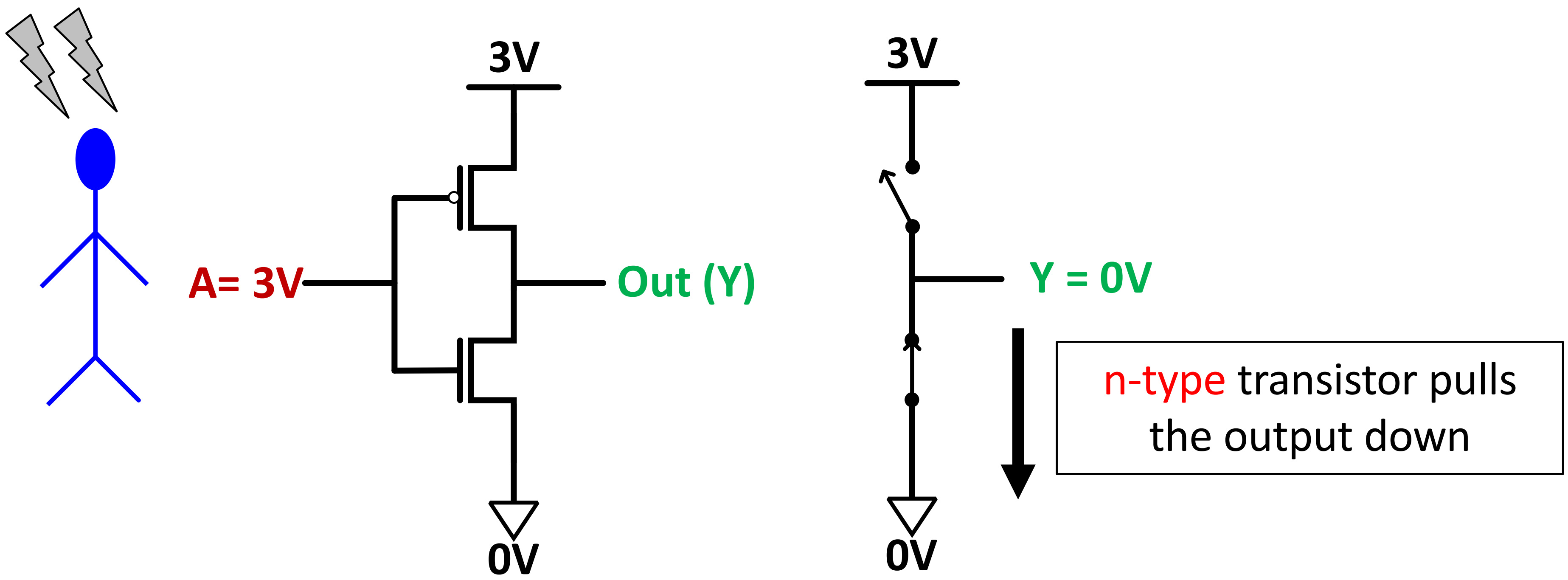


CMOS Technology Fundamentals

- **p**-type transistors are good at pulling u**p** the voltage

CMOS Technology Fundamentals

- What happens when the input is connected to 3V?

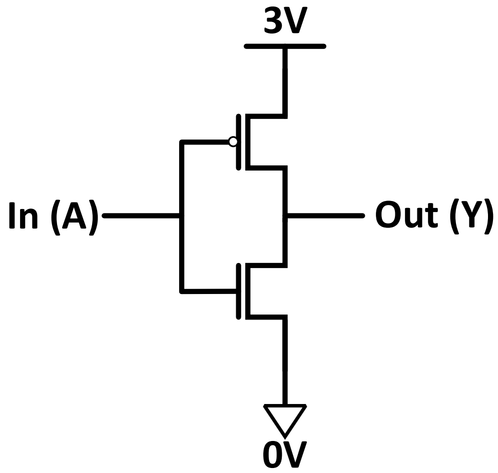


CMOS Technology Fundamentals

- **n**-type transistors are good at pulling dow**n** the voltage

CMOS NOT Gate (Inverter)

- We have seen a NOT gate at the transistor level
  - If A = **0V** then Y = **3V**
  - If A = **3V** then Y = **0V**



- Interpretation of voltage levels
  - Interpret **0V** as logical (binary) 0 value
  - Interpret **3V** as logical (binary) 1 value

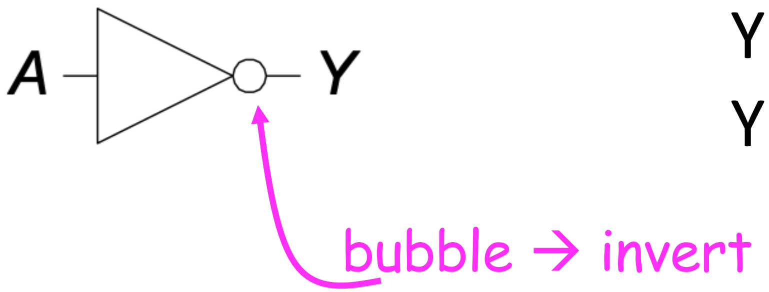
A	P	N	Y
0	**ON**	OFF	1
1	OFF	**ON**	0

$$Y = \bar{A}$$

CMOS NOT Gate (Inverter)

A	Y
0	1
1	0

The NOT gate has only one input (unary)



$Y = A'$

$Y = \overline{A}$

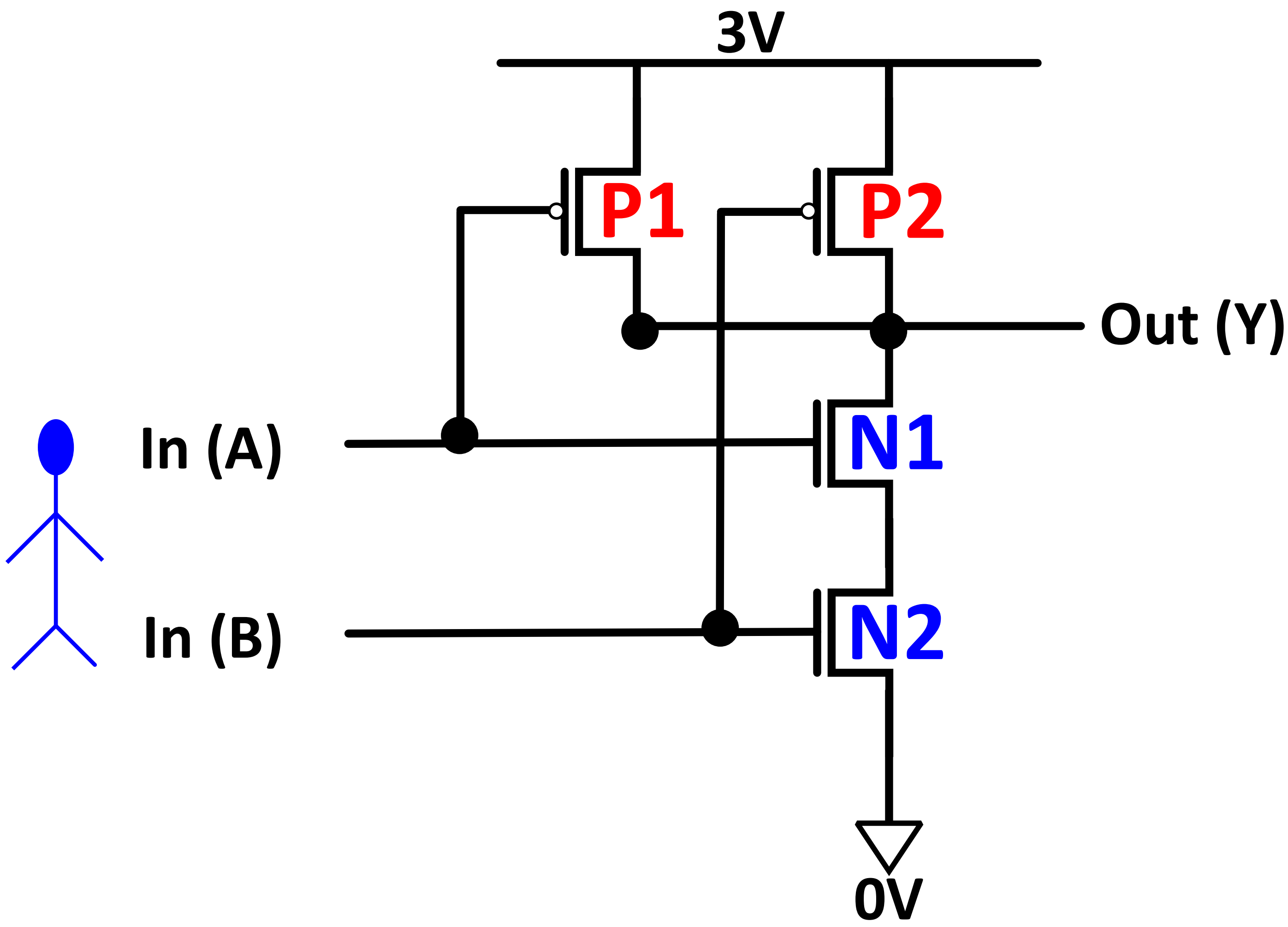
Truth Table

NOT Logic Gate

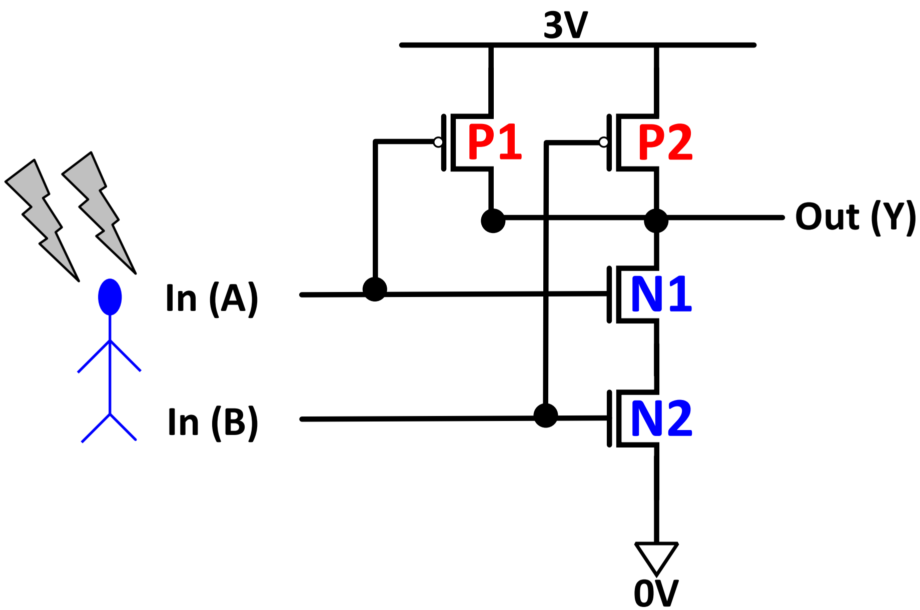
Boolean Equation

NOT Function: *The output Y is the inverse of the input A*

Another CMOS Gate: What is this?



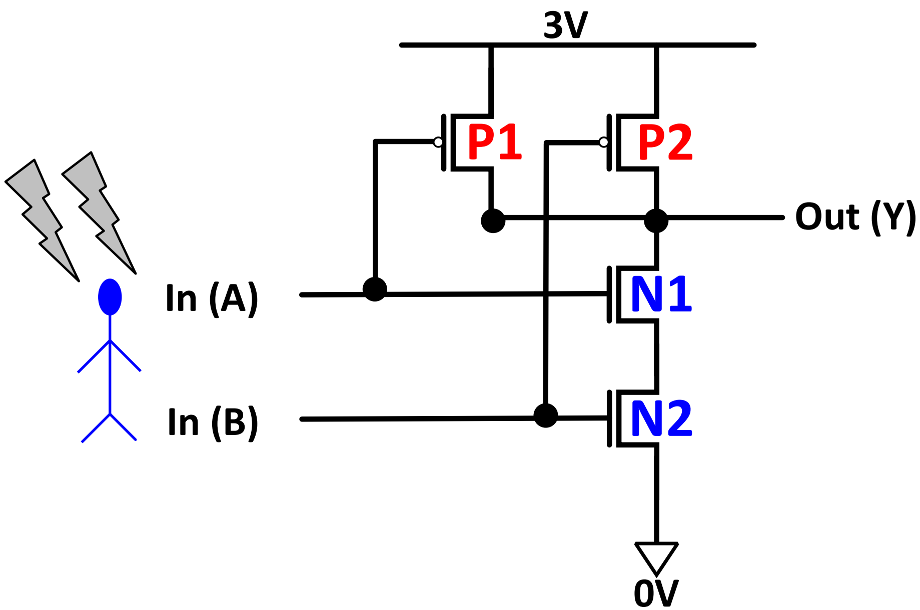
CMOS NAND Gate



A	B	P1	P2	N1	N2	Y

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

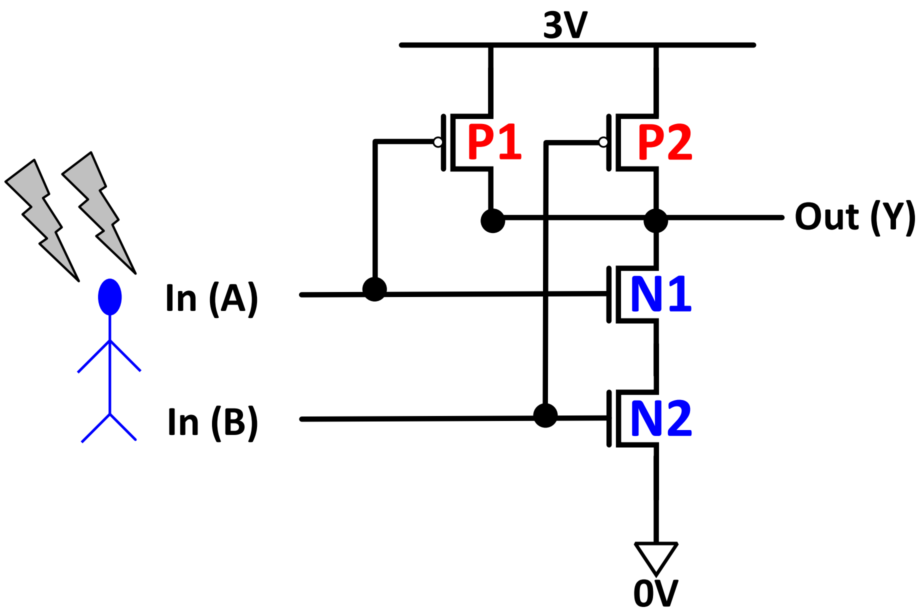
CMOS NAND Gate



A	B	P1	P2	N1	N2	Y
0	0	ON	ON	OFF	OFF	1

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

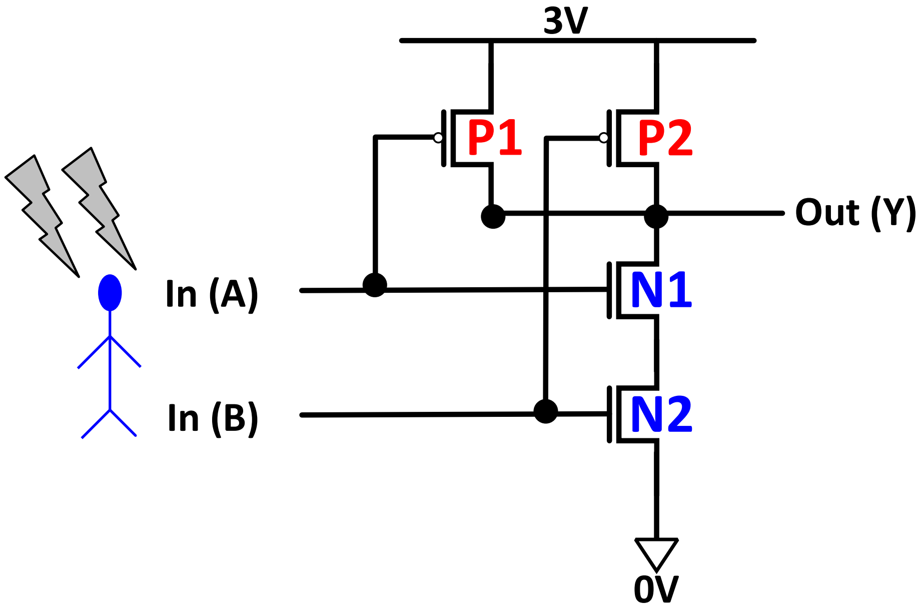
CMOS NAND Gate



A	B	P1	P2	N1	N2	Y
0	0	**ON**	**ON**	OFF	OFF	1
0	1	**ON**	OFF	OFF	**ON**	1

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

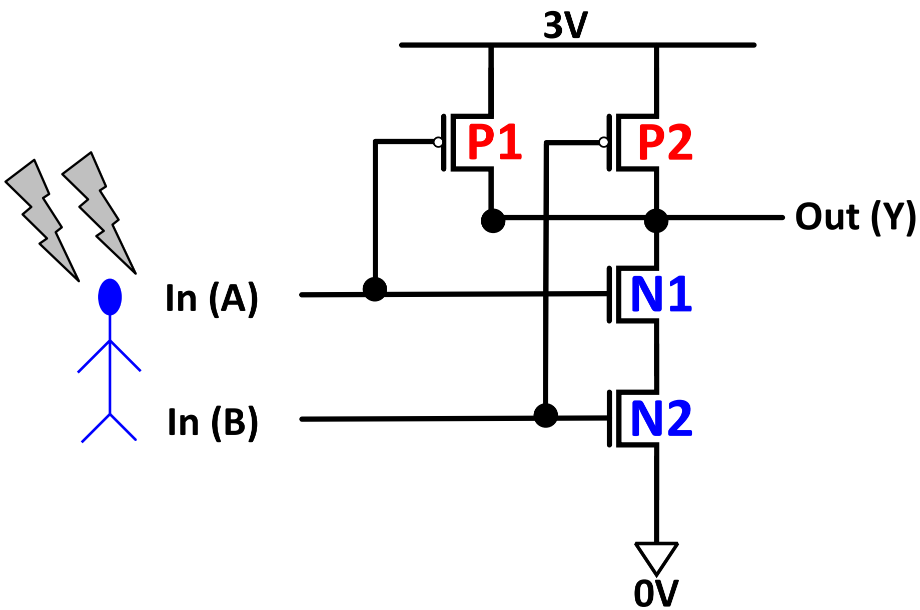
CMOS NAND Gate



A	B	P1	P2	N1	N2	Y
0	0	**ON**	**ON**	OFF	OFF	1
0	1	**ON**	OFF	OFF	**ON**	1
1	0	OFF	**ON**	**ON**	OFF	1

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

CMOS NAND Gate



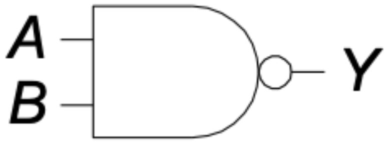
A	B	P1	P2	N1	N2	Y
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

- P1 and P2 are in parallel; only one must be **ON** to pull up the voltage to 3V
- N1 and N2 are connected in series; both must be **ON** to pull down the voltage to 0V

CMOS NAND Gate

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table



NAND Logic Gate

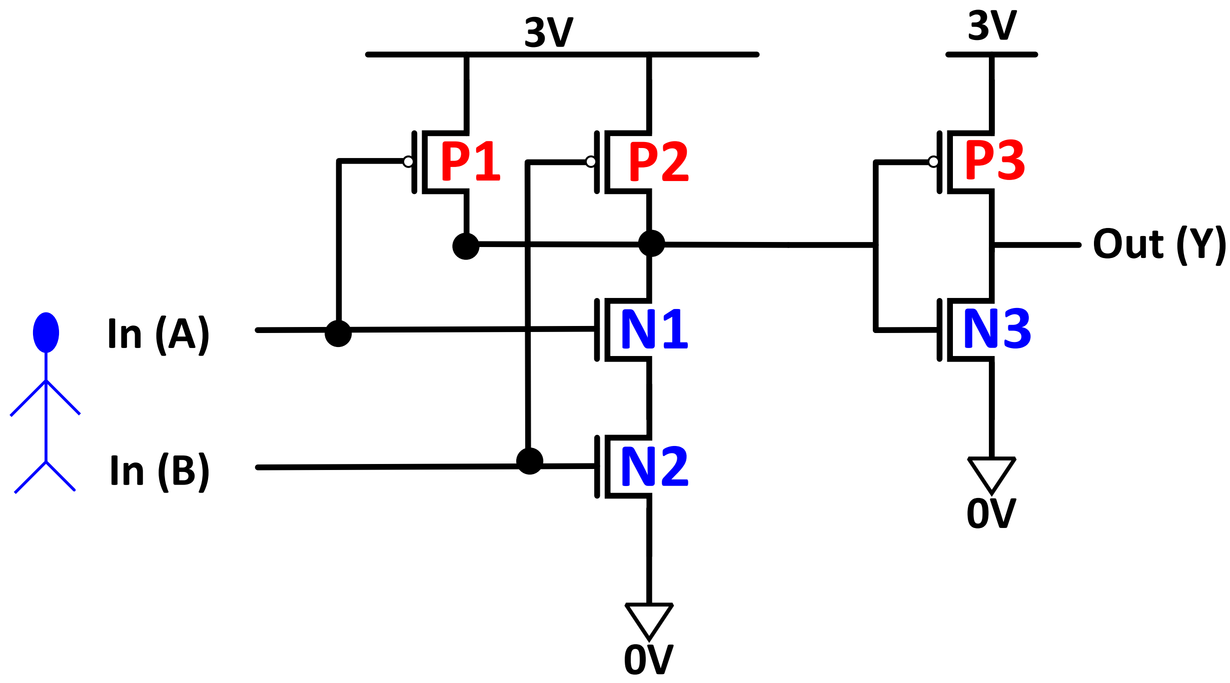
$Y = (AB)'$

$Y = \overline{AB}$

Boolean Equation

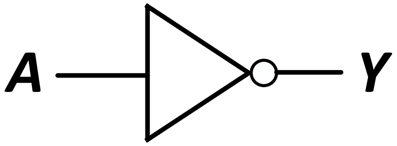
NAND Function: *The output Y is 1 unless both inputs are 1*

CMOS AND Gate

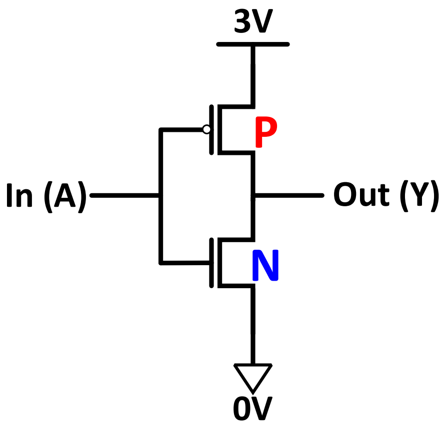


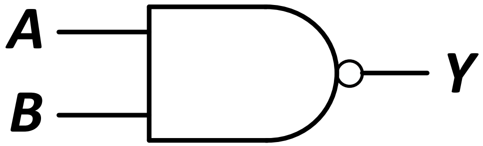
- We make an AND gate using one NAND gate and one NOT gate
- **Homework:** Can we not use fewer transistors for the AND gate?

CMOS NOT, NAND, and AND Gates

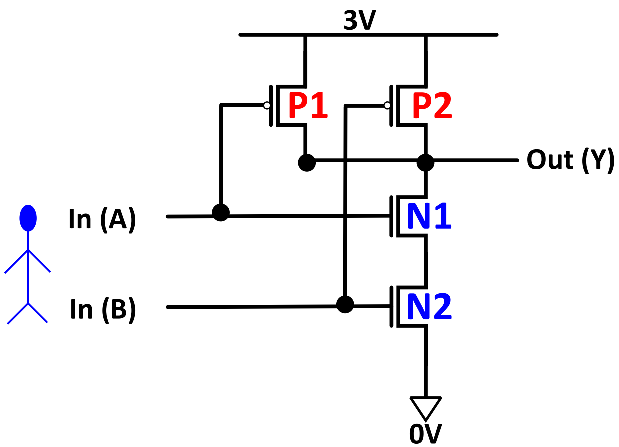


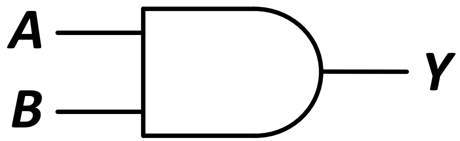
A	Y
0	1
1	0



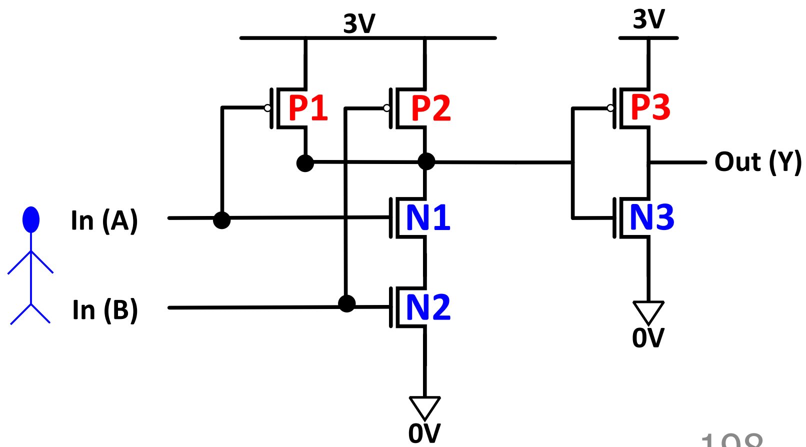


A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



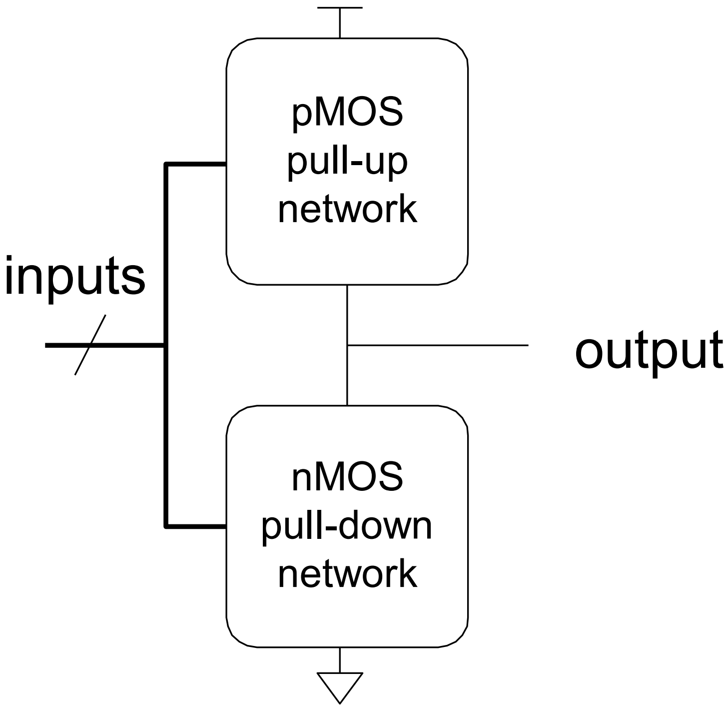


A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



General CMOS Gate Structure

- We have a general form to construct any inverting logic gate, such as, NOT, NAND, NOR
  - The networks may consist of transistors in series of in parallel
  - When transistors are in **parallel**, the network is **ON** if one of the transistors is **ON**
  - When transistors are in **series**, then network is **ON** only if all transistors are **ON**



pMOS transistors are used for pull-u**p**

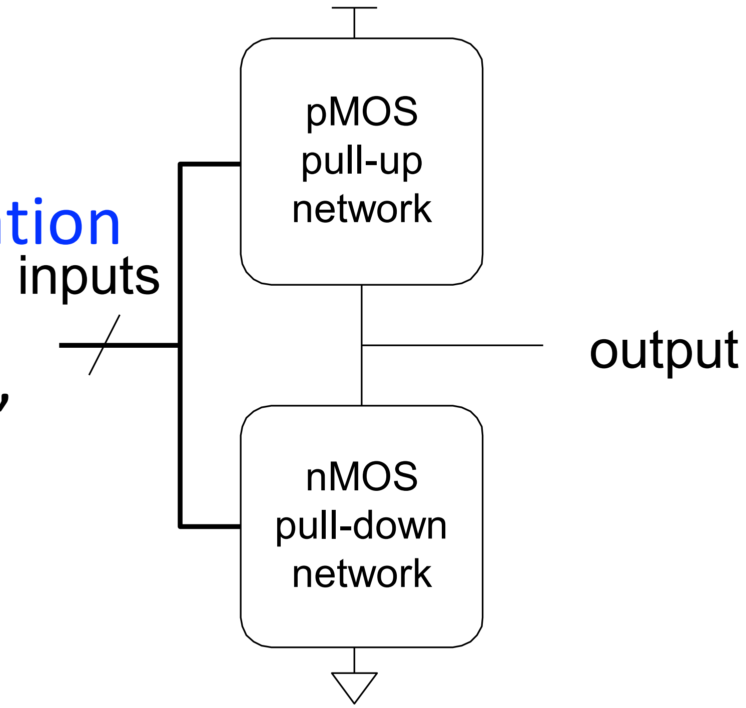
nMOS transistors are used for pull-dow**n**

General CMOS Gate Structure

- Exactly one network should be ON, and the other network should be OFF at any given time

- If both networks are ON simultaneously, there is a short circuit → incorrect operation

- If both networks are OFF simultaneously, the output is floating → undefined



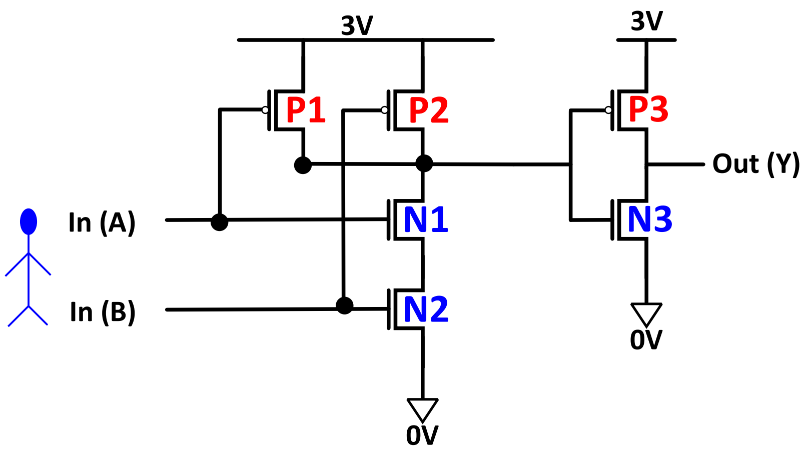
pMOS transistors are used for pull-u**p**

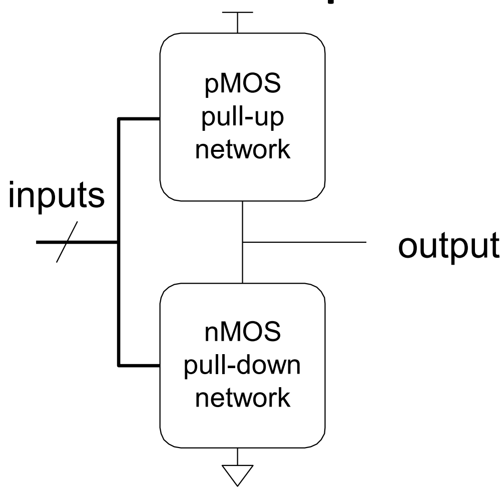
nMOS transistors are used for pull-dow**n**

Why This Structure?

- MOS transistors are imperfect switches

- pMOS transistors pass 1's well but 0's poorly
  - **p**MOS transistors are good at "pulling u**p**" the output
- nMOS transistors pass 0's well but 1's poorly
  - **n**MOS transistors are good at "pulling dow**n**" the output





Latency

- Which one is faster?
  - Transistor in series
  - Transistors in parallel

- Series connections are slower than parallel connections
  - More **resistance** on the wire

- **Remember: Latency of series vs. parallel circuits extend from transistors to gates and larger circuits**

- See Section 1.7.8 for more details

Gates with More Than Two Inputs

- We can create larger gates with more than 2 inputs
  - 3-input NOR gate or 11-input NAND gate

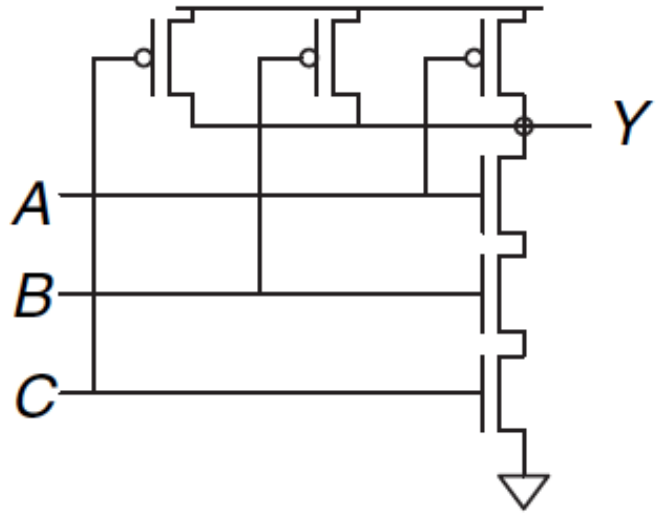
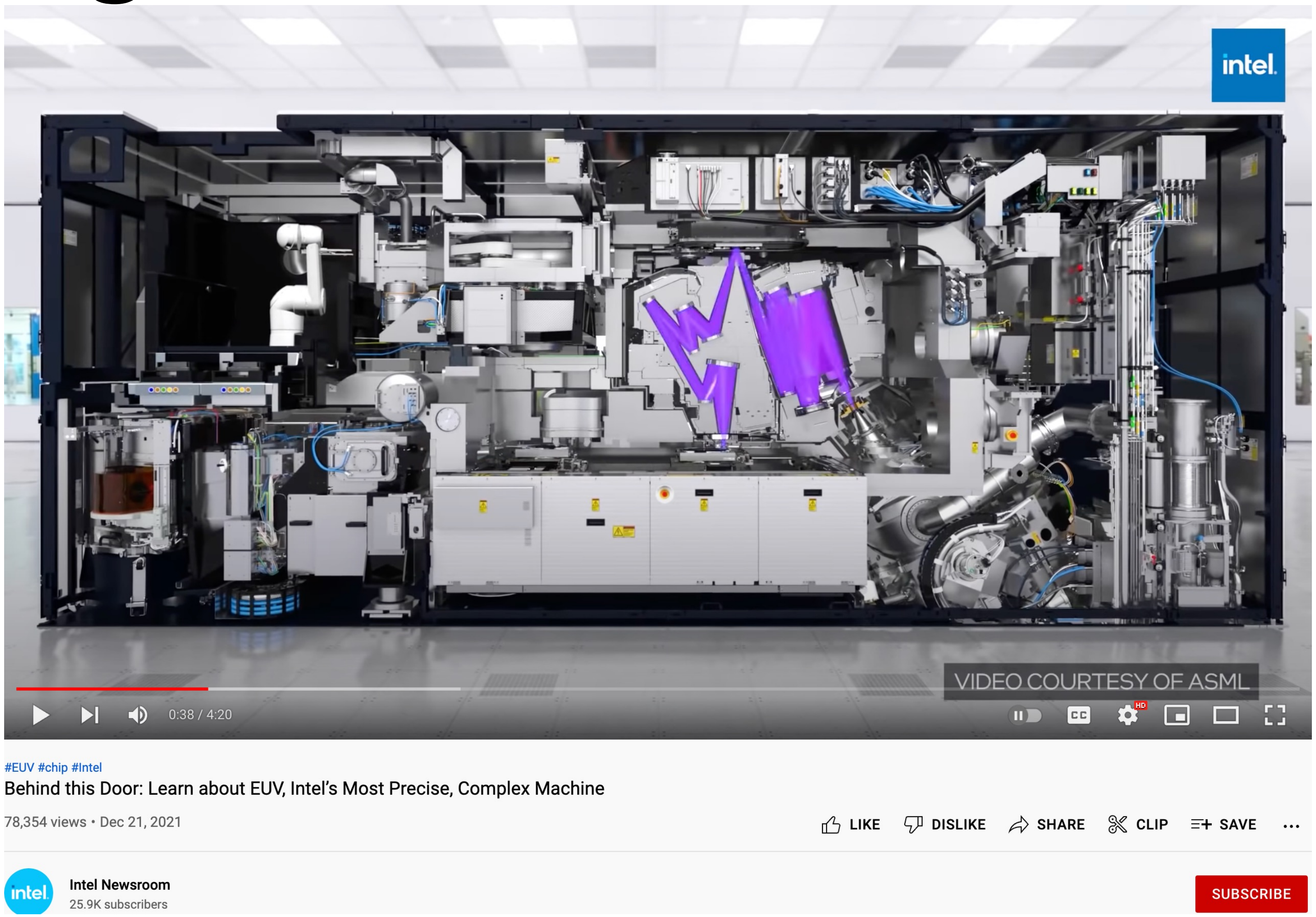


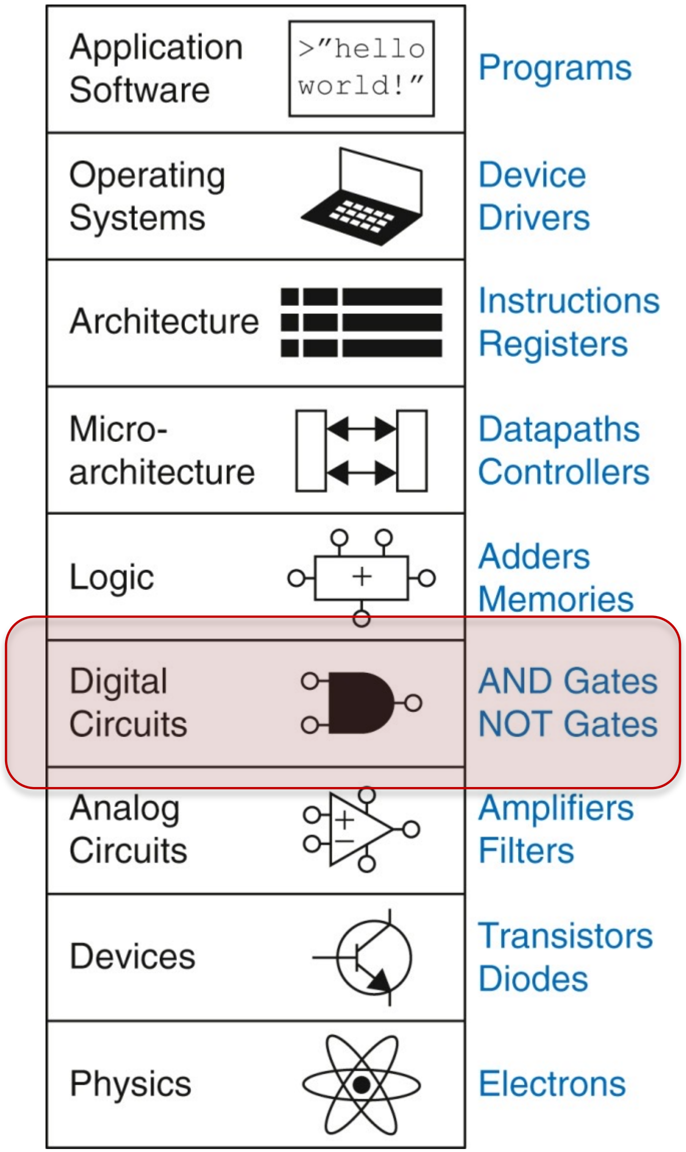
Figure 1.35 **Three-input NAND gate schematic**

Manufacturing Tech is the Enabler

- **Precision Manufacturing**
  - Extreme Ultraviolet (EUV) light to pattern <10nm structures



https://www.youtube.com/watch?v=Jv40Viz-KTc

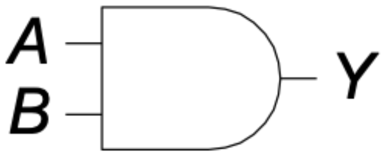

Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions Registers
Micro-architecture
Datapaths Controllers
Logic
Adders Memories
Digital Circuits
AND Gates NOT Gates
Analog Circuits
Amplifiers Filters
Devices
Transistors Diodes
Physics
Electrons

We are here

The AND Function

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table



AND Logic Gate

Y = AB

Y = A.B (product)

Y= A∩B (intersection)

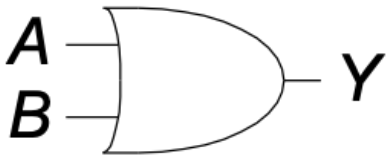
Boolean Equation

AND Function: *The output Y is 1 if and only if both A and B are 1*

The OR Function

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table



OR Logic Gate

Y = A + B (sum)

Y = A ∪ B (union)

Boolean Equation

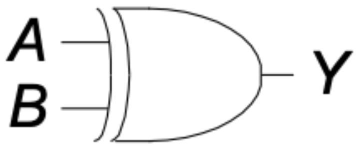
OR Function: *The output Y is 1 if either A or B are 1*

The XOR Function

eXclusive-OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table



XOR Logic Gate

$Y = A \oplus B$

Boolean Equation

XOR Function: *The output Y is 1 if A or B, but not both, are 1*

OR and XOR

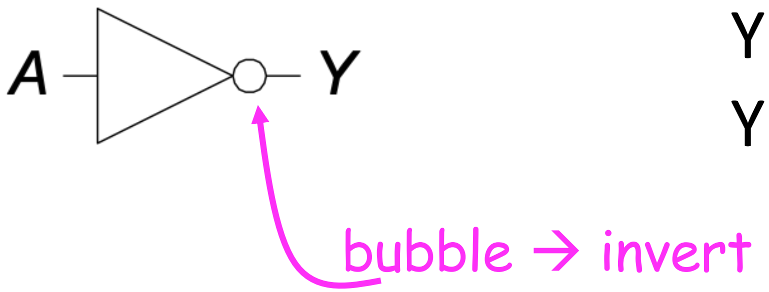
- The term exclusive is used because the output is 1 if only one of the inputs is 1 (mutually exclusive)

- OR produces an output 1, if only one of the two sources is a 1, or both sources are one (inclusive OR)

The NOT Unary Function

A	Y
0	1
1	0

The NOT gate has only one input (unary)



$Y = A'$

$Y = \bar{A}$

Truth Table

NOT Logic Gate

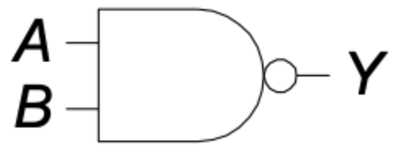
Boolean Equation

NOT Function: *The output Y is the inverse of the input A*

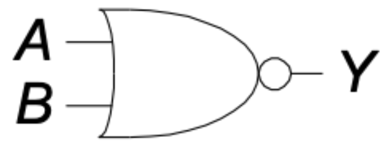
Inverting a Gate's Operation

Any gate can be followed by a bubble to invert its operation

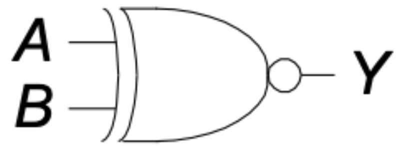
NOT AND → NAND



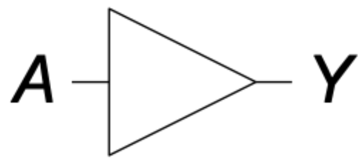
NOT OR → NOR



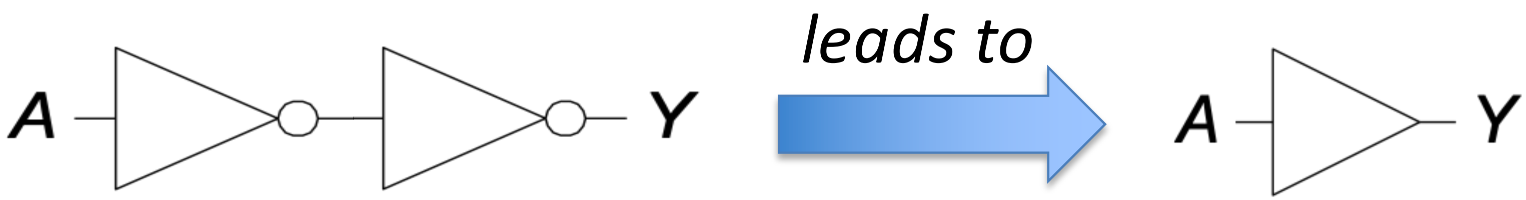
NOT XOR → XNOR



NOT NOT → BUF



In Boolean logic, two wrongs make a right!

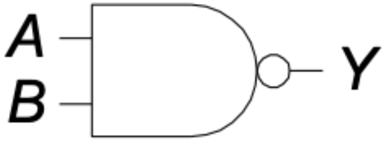


We say that two bubbles cancel each other's effect

The NAND Function

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table



NAND Logic Gate

Y = (AB)’

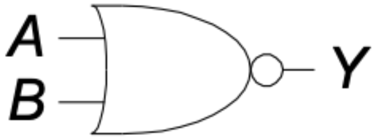
Boolean Equation

NAND Function: *The output Y is 1 unless both inputs are 1*

The NOR Function

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table



NOR Logic Gate

$$Y = (A + B)'$$

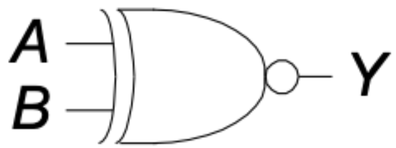
Boolean Equation

NOR Function: *The output Y is 1 if neither A nor B is 1*

The XNOR Function

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Truth Table



XNOR Logic Gate

$$Y = (A \oplus B)'$$

Boolean Equation

XNOR Function: *The output Y is 1 if both A and B are 1*

XOR and XNOR are special

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

XOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

XNOR

XOR: Output is 1 when inputs are not equal (odd number of 1's)

Parity Gate

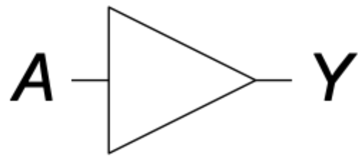
XNOR: Output is 1 when inputs are equal (even number of 1's)

Equality Gate

Buffer (BUF)

A	Y
0	0
0	0
1	1
1	1

Truth Table



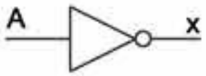
BUF Logic Gate

Y = A

Boolean Equation

Buffer: *The output Y is equal to the input A*

Logic Gates

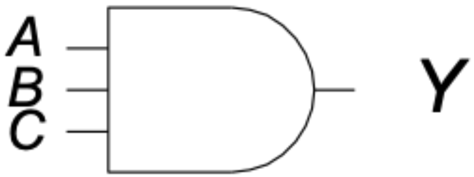
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol							
Truth Table	A X 0 **1** 1 **0**	B A X 0 0 **0** 0 1 **0** 1 0 **0** 1 1 **1**	B A X 0 0 **1** 0 1 **1** 1 0 **1** 1 1 **0**	B A X 0 0 **0** 0 1 **1** 1 0 **1** 1 1 **1**	B A X 0 0 **1** 0 1 **0** 1 0 **0** 1 1 **0**	B A X 0 0 **0** 0 1 **1** 1 0 **1** 1 1 **0**	B A X 0 0 **1** 0 1 **0** 1 0 **0** 1 1 **1**

Multiple-Input Gates

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Gates with multiple inputs are possible

Looking at the truth table, can you guess the 3-input gate?



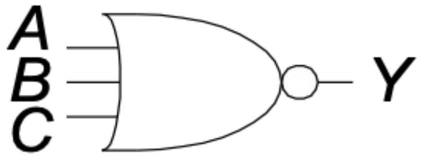
Y = ABC

Multiple-Input Gates

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Gates with multiple inputs are possible

Looking at the truth table, can you guess the 3-input gate?



Y = (A + B + C)’

Bitwise Operations

- All logical operators are applicable to two bit-patterns (group of bits or **bit vectors**) of m bits each, m is any # bits (8, 16, ...)
  - We apply the operation individually to each pair of bits
  - If A and B are **8-bit input sources** (or source operands), then their **AND** or **product**, C, is also 8 bits

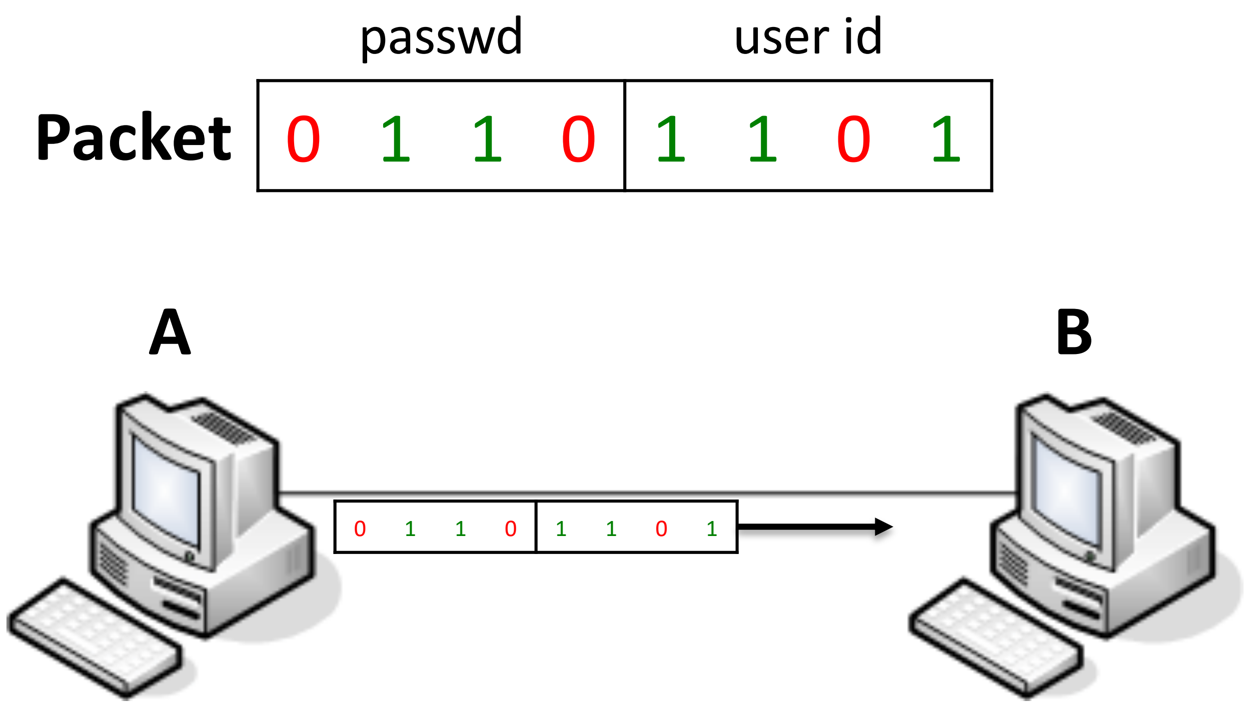
C = AB (bit-wise AND)

A	0	0	0	0	1	1	0	1
B	1	1	1	1	1	1	1	1
C	0	0	0	0	1	1	0	1

C = A + B (bit-wise OR)

A	0	0	0	0	1	1	0	1
B	0	0	0	0	0	0	0	0
C	0	0	0	0	1	1	0	1

Bit Masks



- **B** wants to create a new packet with user id set to **A**'s id and passwd bits set to 0 (e.g., to send a packet to another computer)

Bit Masks

- **Bit mask:** A binary pattern (B) that separates the bits of A into two halves

- Suppose we are interested in extracting the least significant four bits from A, while ignoring the right-most four bits

  - If we **AND** A with B, and choose B as 00001111, then we get the desired bit pattern in C

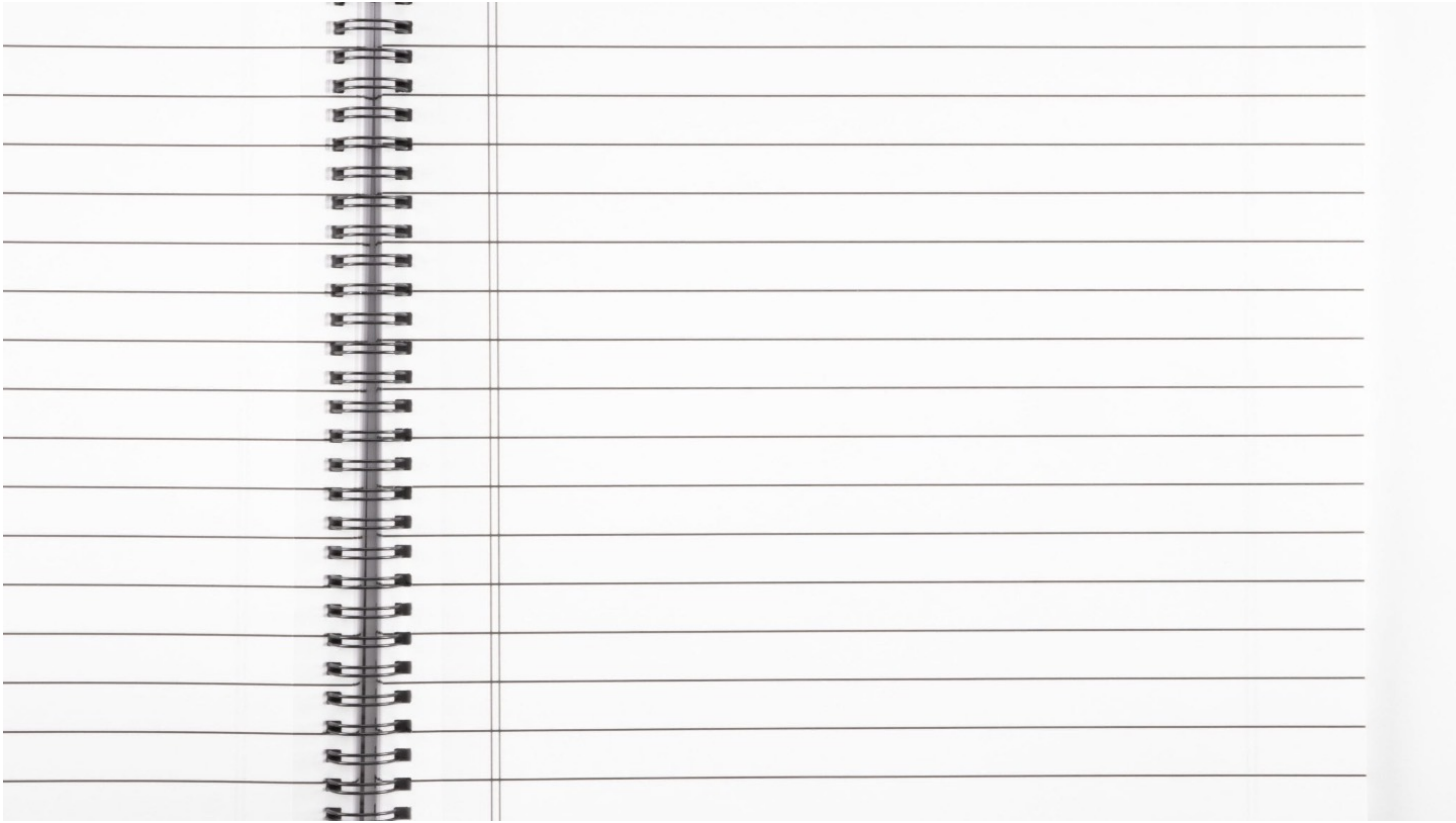
C = AB (bit-wise AND)

A	0	1	1	0	1	1	0	1
B	0	0	0	0	1	1	1	1
C	0	0	0	0	1	1	0	1

Exercises

- Suppose, A = 11000010, and the rightmost two bits are of particular significance. Find a bitmask and a logical operation to mask out the values in the rightmost positions in a new bit pattern B. (**All other bits in B are set to 0.**)

- Suppose, A = 10110010, and the leftmost two bits are of particular significance. Find a bitmask and a logical operation to mask out the values in the leftmost positions in a new bit pattern B. (**All other bits in B are set to 1.**)

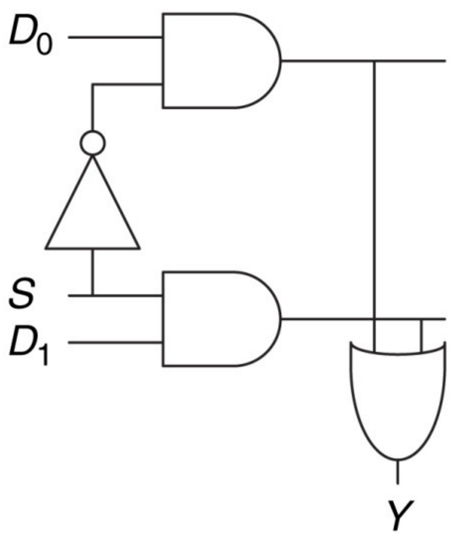


Exercise

- How can we find out if two bit-patterns A and B are identical?
- Verify that, 1 AND X = X, where X is a binary variable. Also, verify that, 0 OR X = X.
- Verify that, B AND B = B, where B is a binary variable. Also, verify that, B OR B = B.
- Verify that, B AND B’ = 0, where B is a binary variable. Also, verify that, B OR B’ = 1.

A Useful Circuit

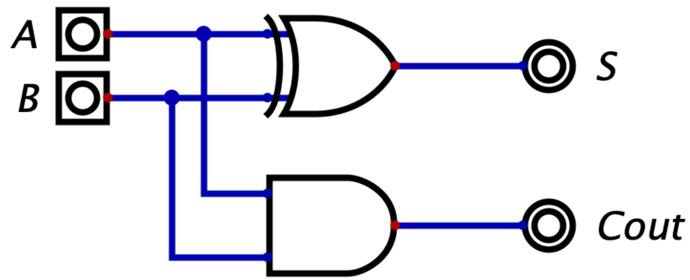
- **What does this circuit do?**



- Multiplexer
- **Used for decision making and often found inside control logic**

Another Useful Circuit

- **What does this circuit do?**



- Half adder (no carry input)
- **Used for making an ALU – Arithmetic & Logic Unit**

Coming Attractions

- We will learn to **systematically build circuits from a specification**

- We **will look at many useful circuits**

- We will study two types of logic circuits – Combinational and sequential