

ENGN2219/COMP6719

Computer Systems & Organization

Convener: Shoaib Akram

shoaib.akram@anu.edu.au



Australian
National
University

Shoaib Akram

Lecturer, School of Computing, (Jan 2020 –)

Ph.D., 2019

Teaching: Computer Microarchitecture (semester 2)

Interests: Hardware/software interaction and performance analysis

It is an interesting time to learn and research about computer systems. Traditional semiconductor laws are breaking down. But the society needs more compute power and storage capacity: Big Data, AI/ML, communication needs, among others.

My current focus is on enabling fast access to large amounts of information (data) using emerging memory and storage devices.

Logistics

Course webpage: <https://comp.anu.edu.au/courses/engn2219/>

Lectures (on the website)

- Lecture slides
- Lecture videos (I will do my own recording)
- Weekly problem sets (for your practice only, *Not Graded*)
- Key Ideas and summary

Policies (will be up shortly)

- General conduct, assignment groups/submissions, support, management, grading

Resources

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

Piazza

I will use Piazza for all communication

- If you ignore Piazza, *you will miss key announcements*
 - Drop-in sessions, make-up lectures, problems, exercises, corrections, lecture timing (ENGN2218 conflict)
- Ask questions on Piazza first (most likely you will receive a response quickly)
- Post solutions to weekly problems, ask your classmates if you are on the right track
- *Ask private questions on Piazza to instructors*
- Students are added/dropped automatically

Tutorials/Labs

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

Handout will be posted on the website “Labs” before each lab

First six labs

- In each lab, you will finish a sub-component of design assignment 1
- If you finish the first six labs, you will finish ~50% of the first assignment
- We only mark the week 2 – 3 labs to make sure you are making progress and to give you feedback (due Monday 6 pm week 4)
- Week 3 lab accounts towards 5 points (out of 100) for assignment 1

Week 7 lab (after the teaching break)

- Test your design project

Week 8 – 11 Labs

- C programming (*not really about C, but about learning key computer systems concepts, more on this later*)

Assessments

Two assignments (60%)

- CPU design assignment (30%)
 - 5% due on 6 pm, Monday of week 4
 - Full assignment due on 12 pm, Monday of week 8
- Programming assignment (30%)
 - Due date: Monday 6 pm, week 13

Final Exam (40%)

- I will release problem sets and exercises throughout the course for preparation
- Mock-up exams will be available during/after the break

Assignment Submission

Extensions will be granted on a per-request basis

- Via Email: Shoaib.Akram@anu.edu.au

Assignment submissions are handled via Gitlab

- You will learn about it in the labs
- Make a habit of using Git properly
- Push often, always pull the latest

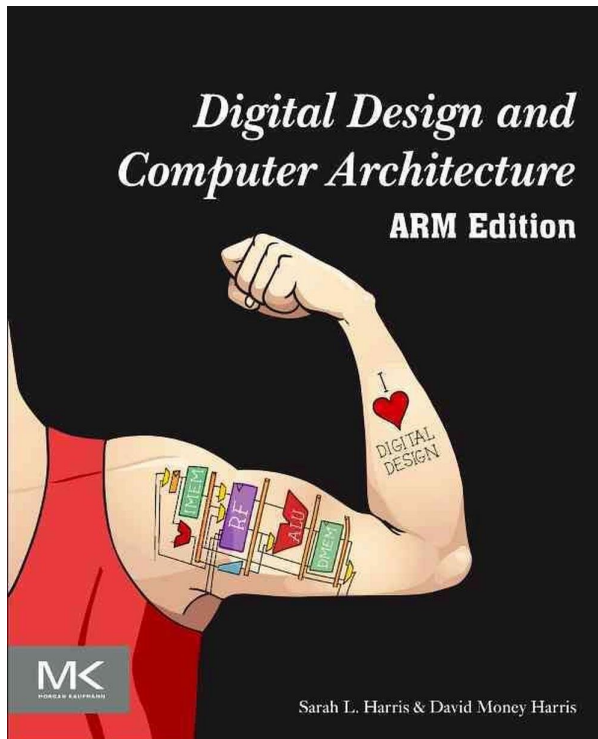
You can form group of up to two students to work on the assignment (one submission per group)

Ink/Whiteboard in Lectures

Try to take notes

- Help you think and most of the time I will do this is to solve a problem
- Ink + Wacom (not very stable)

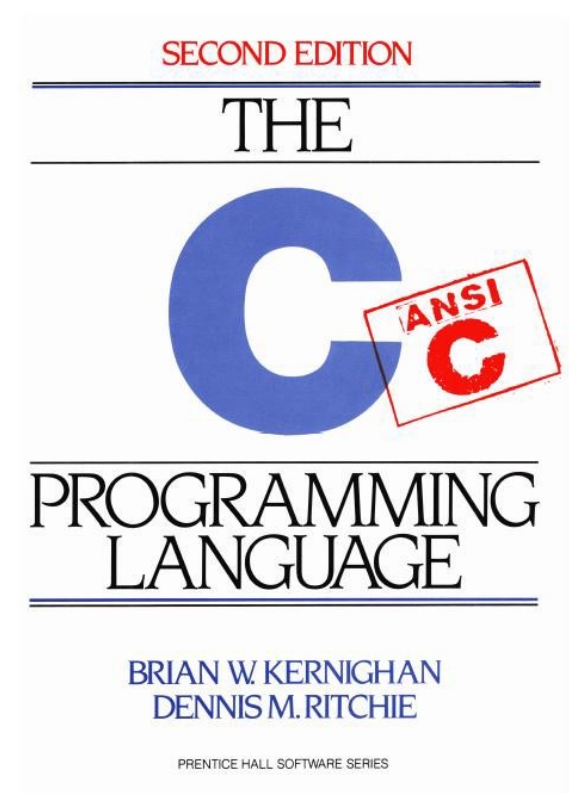
Textbook



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

Kernighan & Ritchie, The C Programming Language, 2nd Edition

- “ANSI” (old-school) C





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



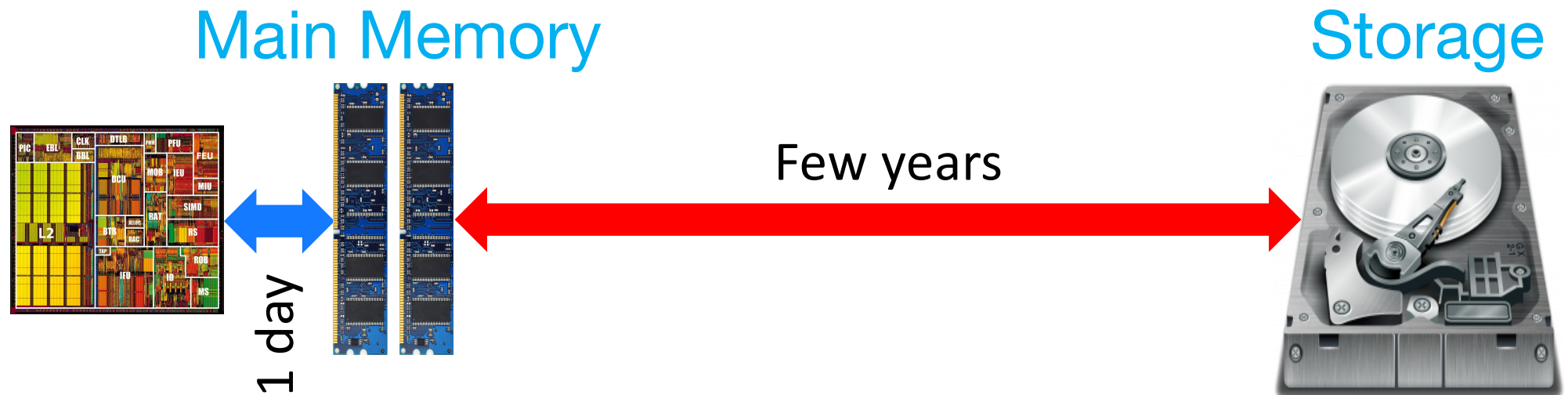
Fundamentals are important

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 (e.g., disks or SSDs) for storing long-term or persistent information
- **I/O devices** to communicate with the external environment
 - Sensors
 - Peripherals

Most computer systems can be viewed as below

- Three key resources: CPU, memory, storage
- CPU is the heart of a computer system
- Processor can access memory much faster than storage



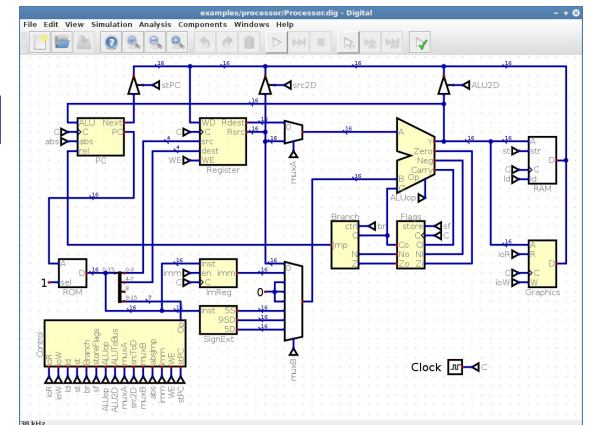
This Course is about ...

How does the general-purpose processor work? How do modern processors perform a wide variety of tasks?

How do processors interact with main memory and storage? How does the memory and storage system work?

The best way to learn how something works is to build one

- You will build a processor in Digital
- URL: <https://github.com/hneemann/Digital>



This Course is also about ...

A computer system is more than just hardware

- How does hardware and software interact?
- What should programmers know about hardware?
- C is a good vehicle for answering the above questions
- You can talk about hardware resources in high-level terms but still stay close to the hardware
- Key learning outcome of this course: *How can you shoot yourself in the foot when writing C programs?*
- **Remember:** It's not about C or Java or Python. It's about gaining a deep insight into computer systems!

A 5-Step Recipe for Failure

Stay out of the loop

- Do not check Piazza
- Do not ask questions
- Do not care what is going on in lectures

And do not learn to use Gitlab

Do not attempt end of week problem sets

- Okay, not every week

Do not come to labs OR do not read/attempt the lab handouts

- And start the assignment on your own the weekend before due date (no way!)

Do not seek help from tutors

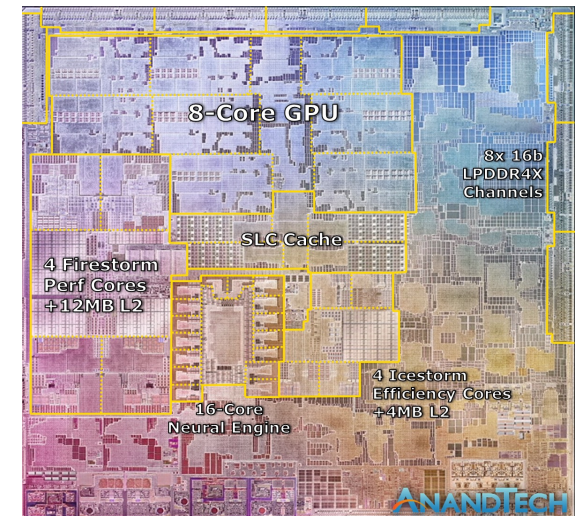
Do not communicate your problems to me!



How do engineers manage complexity?

- Look at components from a higher level
- Get into detail if necessary

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



Apple M1 Chip
Billions of transistors
All working in parallel

Transformation Hierarchy

- We think of problems in English
 - Sort students by their UIDs
- The actual work is done by electrons
 - Do electrons speak English?
- How do we make the electrons do the work?
 - *We use a systematic **transformation hierarchy** to transform the problem in English into electron movement*
 - *This transformation hierarchy is driven by our need to abstract away complexity*



Problem

Algorithm

Program




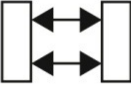
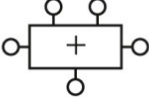

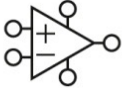

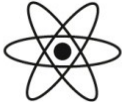
Architecture

micro-arch

circuits

devices



Application Software		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

Definitions

Abstraction: Hiding details to view the system from a high level




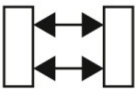
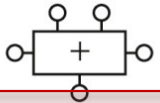

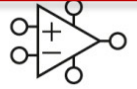


Deconstruction: Going from abstraction back to its component parts (breaking abstraction)

Low-level language: Languages that are tied to the machine architecture. Each architecture supports at least one low-level language called assembly.

High-level language: Programming languages that are at distance higher than the architecture. They are machine-independent. E.g., C, Java, Python, Rust, Ruby, Go

Instruction Set Architecture: A specification of all the instructions a processor can perform. Each instruction is an arithmetic (add, sub) or a data movement (fetch from memory) operation.

Microarchitecture: ISA has no physical significance. Microarchitecture is the physical implementation of an ISA.

Application Software		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

Week 4

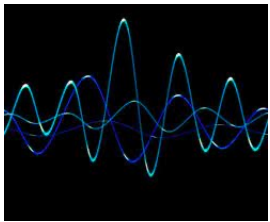
Week 5 – 6

Weeks 2 – 3

Week 1

Representing Information

Question: How many different values can each of these *physical* variables take?



Frequency of oscillation



Voltage on a wire



Temperature

Answer: Infinite

All these are continuous signals
They contain infinite amount of information

Representing Information

Digital Systems: Represent information with discrete-valued variables, i.e., variables with a finite # distinct values

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



0 1



0 1



0 1

Binary Representation

Digital systems internally use “voltages” for representing binary variables

→ Low voltage means 0

→ High voltage means 1

B I N A R 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 I need to represent the alphabet set in English?

- For 26 alphabets, we need 5 bits

Information Content in a Binary Code

$$D = \text{Log}_2 N \text{ bits}$$

The color set has four states: $N = 4$, # bits = 2

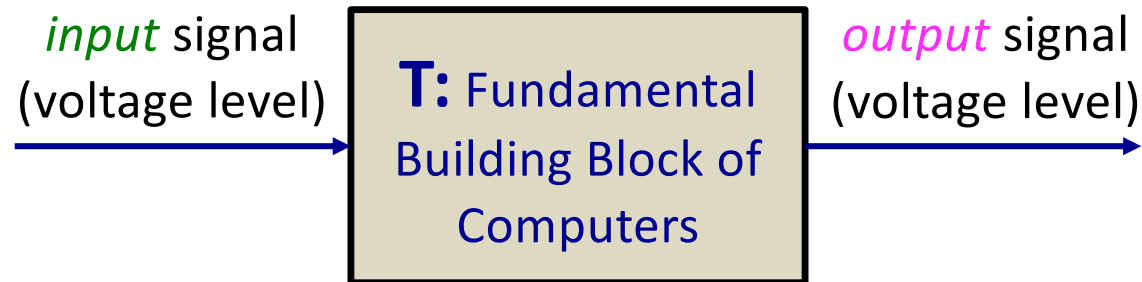
The alphabet set has 26 states: $N = 26$, # bits = 5

Conversely,

If D is 2, $N = 4$

If D is 5, $N = 32$

Why do computers use binary?

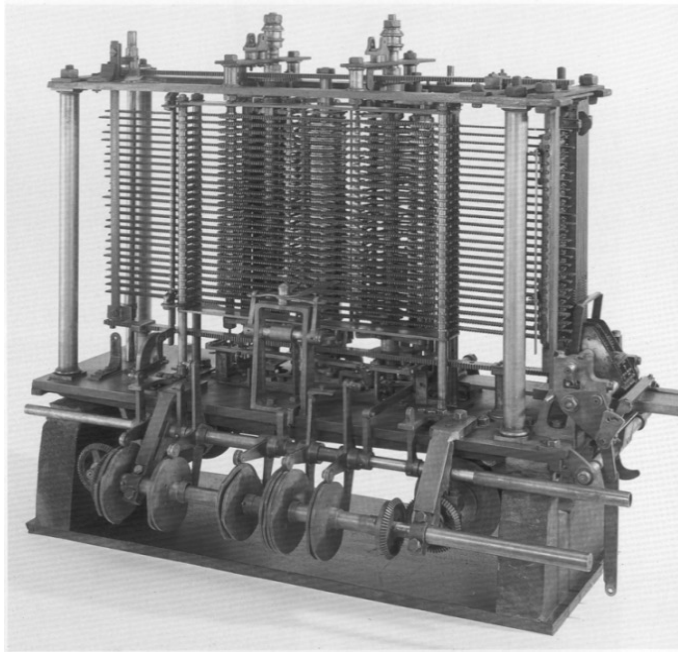


We can divide a continuous voltage range into ten levels to represent 0 – 9, but that would make **T** very complex

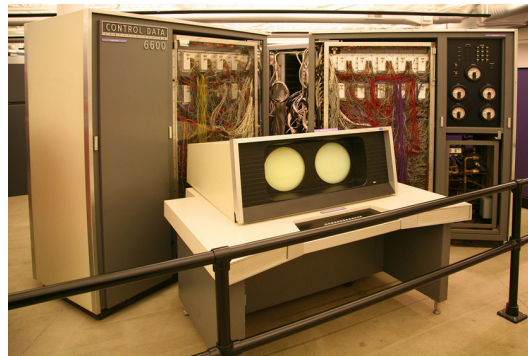
The fundamental building block of all computers is a transistor. A transistor can only distinguish two voltage levels. We call these voltage levels 1 and 0

Voltages and Transistors, Why?

Mechanical parts: Not easy to scale to do large computations



The Analytical Engine
Charles Babbage
1834 – 1871



*CDC 6600, 1964, \$ 2.5 M
Slower than my phone*



IBM 360, 1964



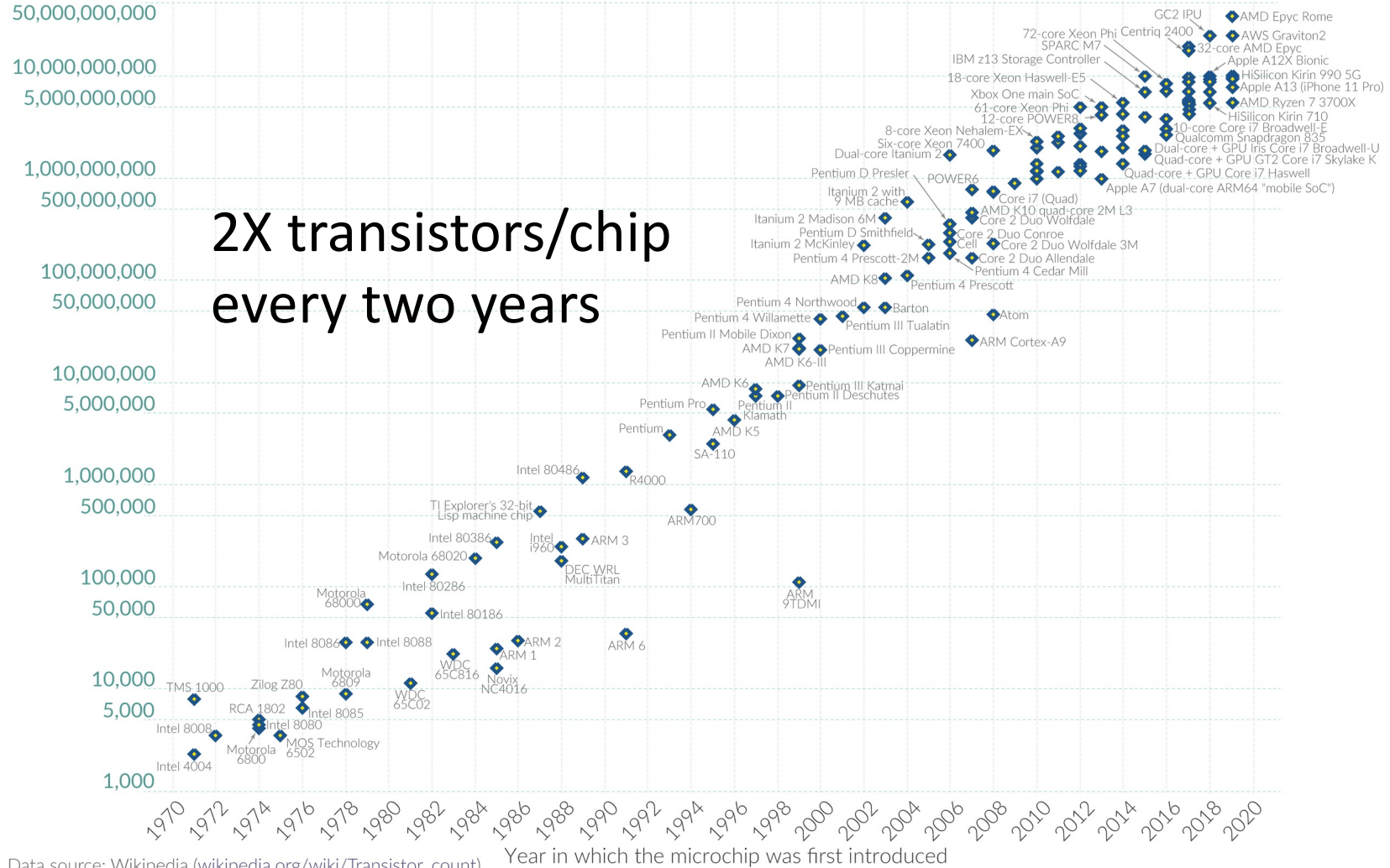
*Apple M1, 2020
400 mm²
16 billion transistors*

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.

Transistor count



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

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)
- Boolean logic defines operations on logic variables

Our Plan

Presenting information to digital circuits

- *Representing numbers as a string of 1's and 0's*
- *Number systems: to set a foundation for efficient manipulation (add and subtract)*

```
0101010100110
100110011010100
101001101011010
111011110101001
100010110010010
001001000010001
```

Operations on binary variables (1's and 0's)

- *Logic gates to perform operations on binary variables*

Breaking the digital abstraction (**self study**)

- *1's and 0's as continuous physical quantities (voltage)*

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

9742 = 9 X 10³ + 7 X 10² + 4 X 10¹ + 2 X 10⁰

nine thousands seven hundreds four tens two ones

coefficient

power of 10

Range of Decimal Numbers

An N-digit decimal number represents one of 10^N possibilities

- $0, 1, 2, 3, \dots, 10^N - 1$
- 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

coefficient

power of 2

$$\begin{array}{l} \text{1's column} \\ \text{2's column} \\ \text{4's column} \\ \text{8's column} \\ \color{red}{1} \color{blue}{0} \color{green}{0} \color{magenta}{1} \end{array} = \begin{array}{l} \color{red}{1} \times 2^3 \\ \text{one} \\ \text{eight} \end{array} + \begin{array}{l} \color{blue}{0} \times 2^2 \\ \text{one} \\ \text{four} \end{array} + \begin{array}{l} \color{green}{0} \times 2^1 \\ \text{zero} \\ \text{two} \end{array} + \begin{array}{l} \color{magenta}{1} \times 2^0 \\ \text{one} \\ \text{one} \end{array}$$

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
11	2^{11}	2048
12	2^{12}	4096
13	2^{13}	8192
14	2^{14}	16384
15	2^{15}	32768
16	2^{16}	65536

Kilo

Powers of 2

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

What is 2^{24} in decimal?

- $2^{20} \times 2^4 = 1 \text{ M} \times 16 = 16 \text{ M}$

What is 2^{17} in decimal?

- $2^{10} \times 2^7 = 1 \text{ K} \times 128 = 128 \text{ K}$

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 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 0 1

The byte in the lowest position

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 the most significant position

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)

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

Hexadecimal Numbers

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						← carries →	1	1			
	4	2	7	7					1	0	1	1	
+	5	4	9	9					+	0	0	1	1
<hr/>													
	9	7	7	6					1	1	1	0	

Decimal
Addition

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

- Suppose we have two 4-bit numbers
 - If $A = 1111$ and $B = 1111$
 - What is $A + B$?
 - What is the largest value 4 bits can represent?
- Overflow
 - The result is too big to fit inside the available bits
 - We check the carry bit out of the most significant column to detect overflow

$$\begin{array}{r} 1\ 1\ 1 \\ \hline 1\ 1\ 1\ 1\ 15 \\ + 1\ 1\ 1\ 1\ 15 \\ \hline 1\ 1\ 1\ 1\ 0\ 30 \end{array}$$

Signed Binary Numbers

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

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



If we write all possible combinations of 0's and 1's 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

Signed Binary Numbers

What are the drawbacks of sign/magnitude representation?

Ordinary binary addition does not work for sign/magnitude numbers

- What is the sum of +3 and -3 and *does the result make sense?*

Awkward to have two different representations of the same number (0)

			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

Tried in some early computers, Control Data Corporation (CDC) 6600

Negative number: Take the representation of a positive integer and flip all the bits

Known commonly as 1's complement

- 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 for which

- Ordinary addition works
- Single representation for *zero*

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

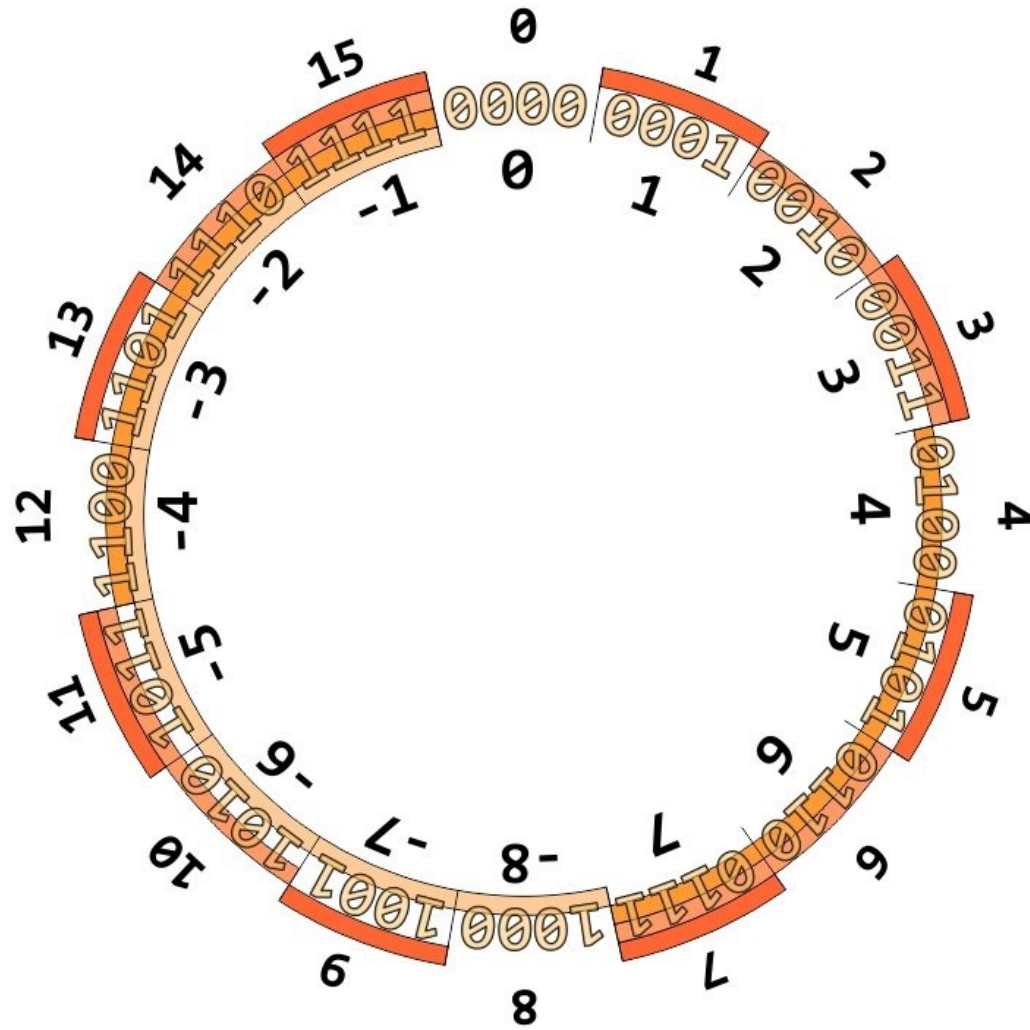
Some Observations

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 $X + (-A)$

$$\begin{array}{r} \\ \\ + \\ \hline \\ \\ \end{array}$$

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

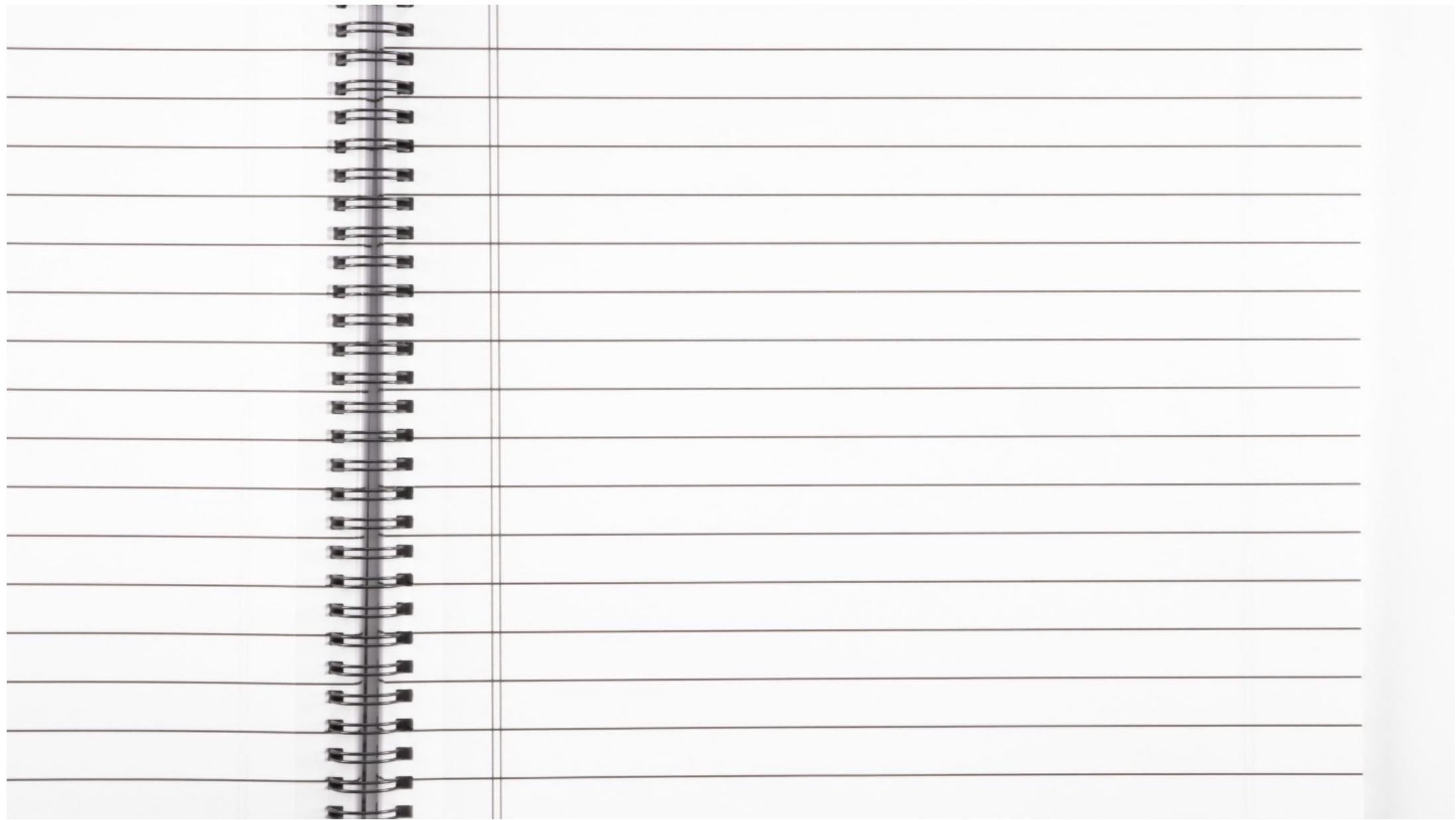
If MSB is 0

- *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
+		0	1	0	1	1
		1	0	1	0	0

+9
+11
-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!

$$\begin{array}{r} \overline{10100} \quad -12 \\ + \phantom{\overline{}} 11010 \quad -6 \\ \hline 01110 \quad 14 \end{array}$$

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 the difference between the 16-bit and 8-bit numbers below?

16-bit number **0 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 to the two numbers below represent?

16-bit number (A)

0	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

Boolean Logic

A system of logic for binary variables

- Helps us reason about the interactions between two binary variables
 - X is 1 and Y is 0. What is $X \cap Y$?
- Formalizes *key* (simple) logical operations on binary variables
- Logical operations are the steppingstone for composing sophisticated operations (including arithmetic)

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
 - The logic **AND** gate (*built with transistors*) implements the logical **AND** function
- 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
 - A total of four combinations (rows)
 - Three columns (2 inputs and an output)

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

For the contrived example on the right, the Boolean equation for **Y** is, $Y = 0$

- The values of **A** and **B** does not matter

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

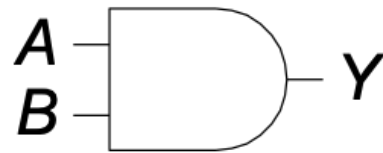
Boolean Equation for output Y: $Y = 1$

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

The AND Function

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

Truth Table



AND Logic Gate

$$Y = AB$$

$$Y = A.B \quad (\text{product})$$

$$Y = A \cap B \quad (\text{intersection})$$

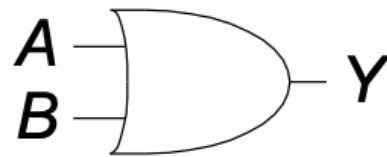
Boolean Equation

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

$$Y = A + B \text{ (sum)}$$
$$Y = A \cup B \text{ (union)}$$

Boolean Equation

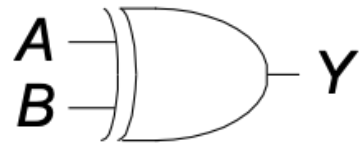
AND Function: *The output Y is 1 if either A or B are 1*

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

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