中国科学技术大学
University of Science and Technology of China

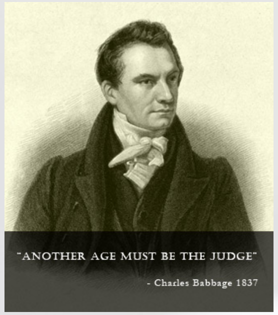
计算系统概论A
Introduction to Computing Systems
(CS1002A.03)

Chapter 2
Bits, Data Types, and Operations

陈俊仕
cjuns@ustc.edu.cn
2024 Fall

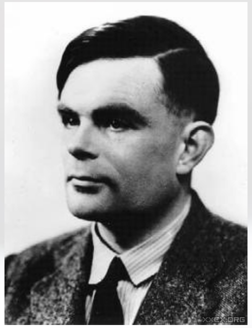
计算机科学与技术学院
School of Computer Science and Technology

Previously: from mechanical computer to electronic computer

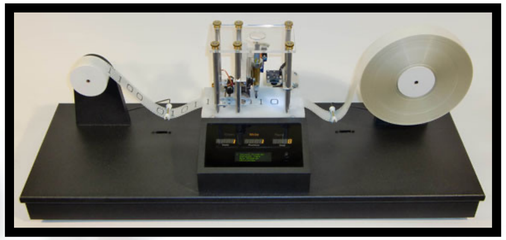


Charles Babbage,
1791 – 1871,England

1832,2002,2008
The Babbage Difference Engine, 17 years, 25,000 parts, 5ton, cost: £17,470

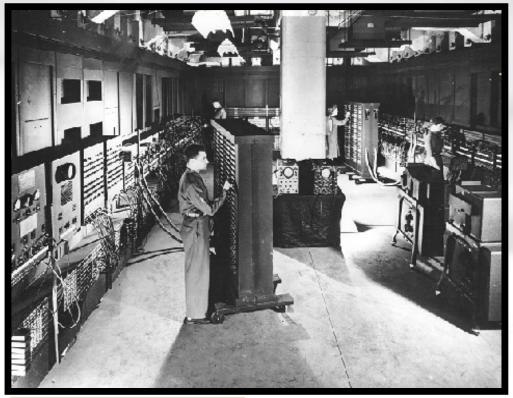


Alan Turing(24)



Turing Machine, 1936

Eckert(24) and Mauchly(36)



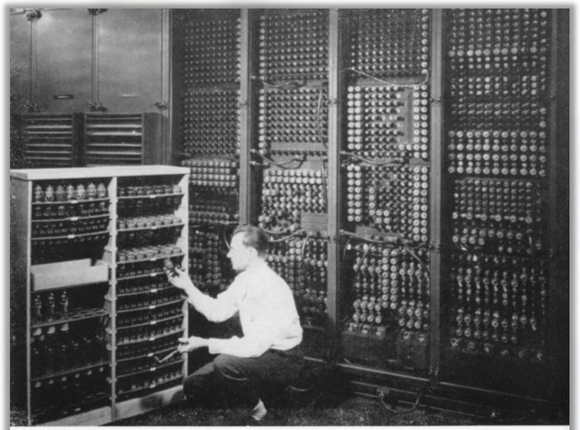
ENIAC

1946

Previously： First computer vs. First microprocessor chip

1946 , ENIAC(Electrical Numerical Integrator And Calculator)

- 18000 vacuum tubes
- 1500 relays
- 174 KW
- 30 tons
- 1800 sq. ft. footprint
- Clock: 100kHz
- RAM: ~230bytes
- IO: punched card

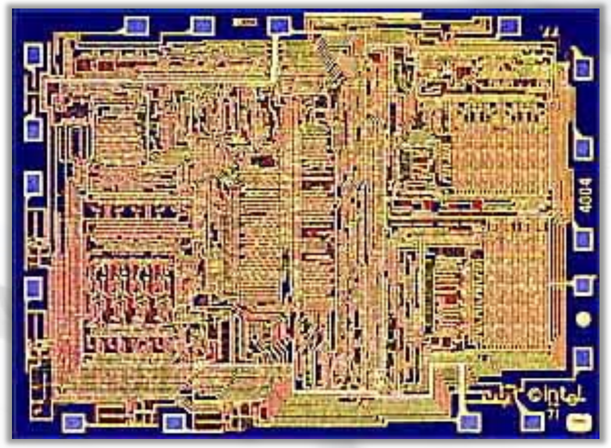


Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

After 25 years

1971, Intel 4004

- 10 micron process, NMOS-Only Logic
- 2,250 transistors
- 3cmx4cm die
- 4-bit bus
- Performance < 0.1 MIPS
- 640 bytes of addressable Memory
- 740 KHz



Previously： Thirty years after the first microprocessor chip was born

1971, Intel 4004

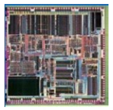
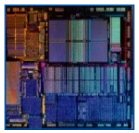
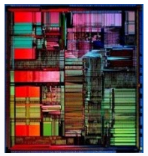
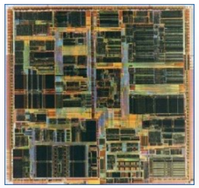
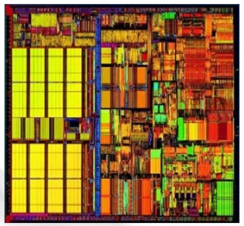
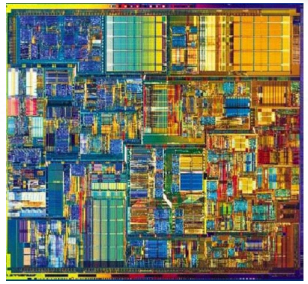
- 10 micron process
- 2,300 transistors
- 3x4 mm die
- 4-bit bus
- 640 bytes of addressable Memory
- 750 KHz

After 30 years →

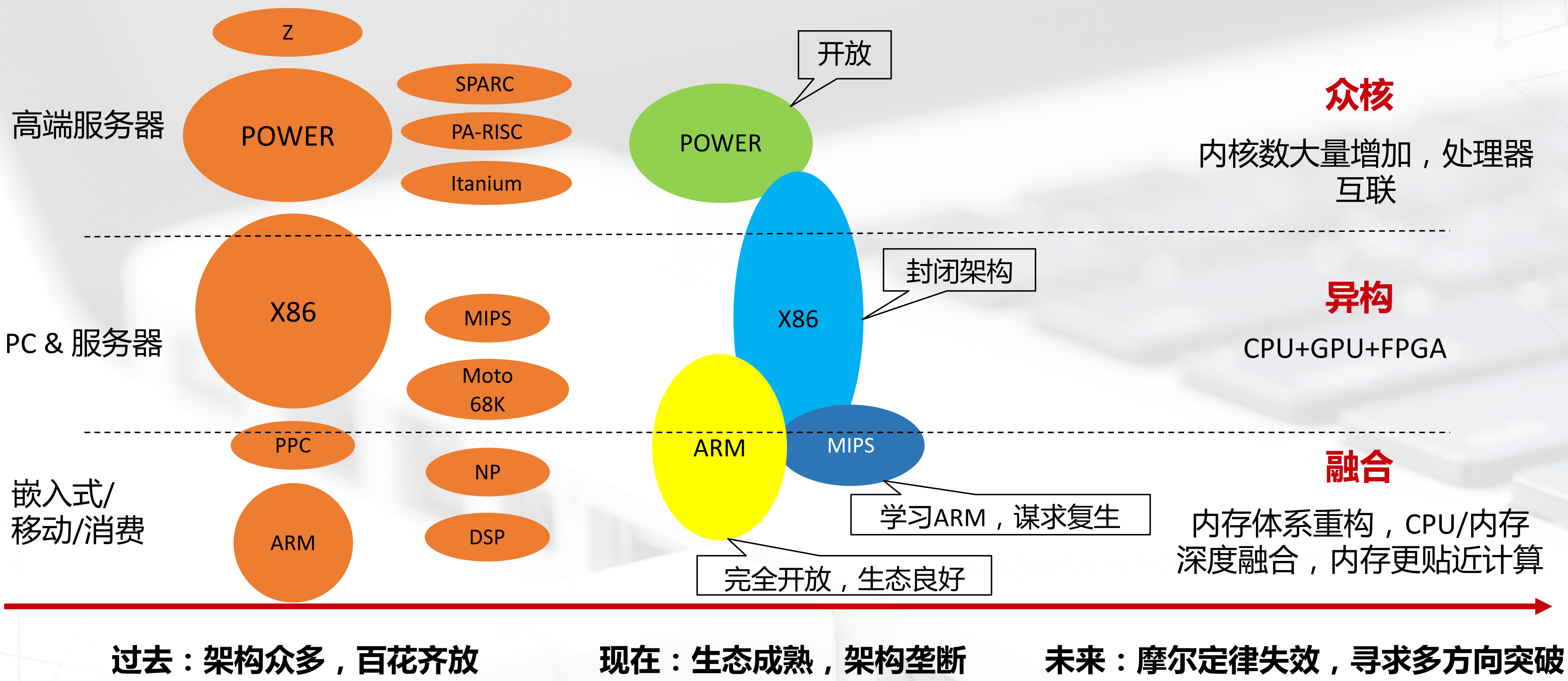
2000, Intel Pentium IV

- Issues up to 5 uOPs per cycle
- MMX, SSE, and SSE2
- 0.18 micron process
- 42 million transistors
- 217 mm die
- 64-bit bus
- 8KB D-cache, 12KB op trace cache (I-cache), 256KB L2 cache
- 1.4 GHz

Performance improved 5000x: smaller, faster, cheaper

Previously： State-Of-The-Art Microprocessor Chips



Previously： 人类如何实现从物理设备到问题求解的？

Application

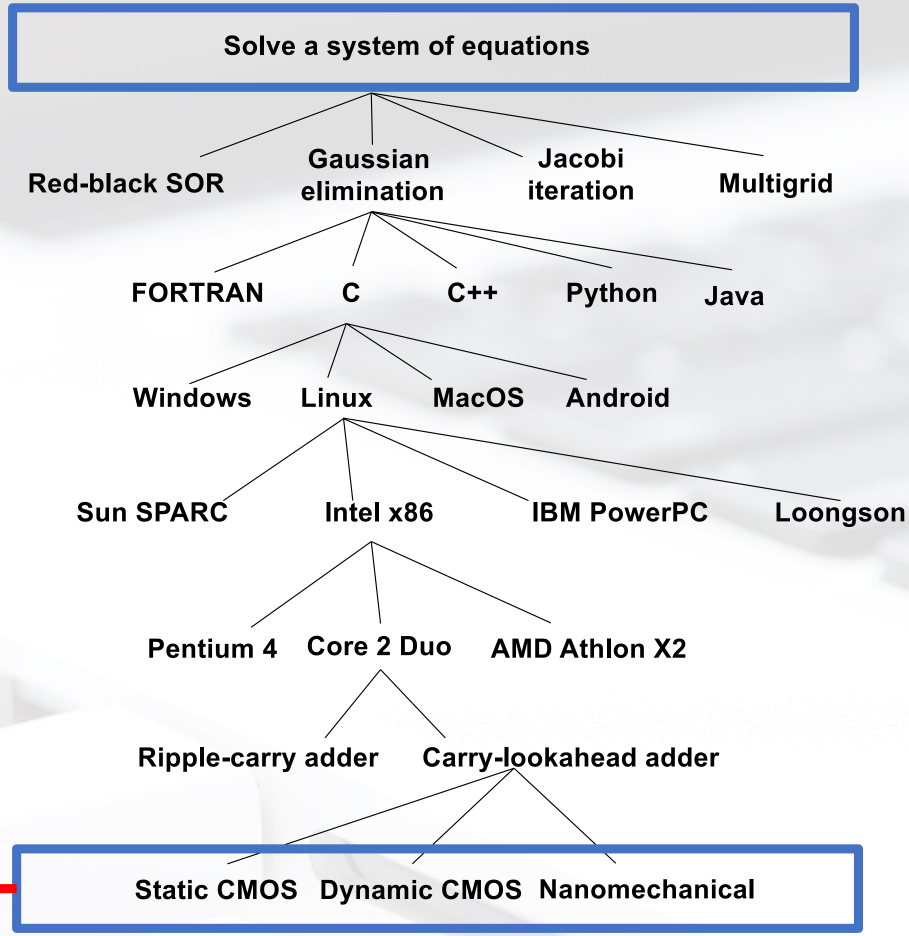
差距太大，
一步无法跨越！

也有例外，
例如罗盘

Physics

Previously : Many Choices at Each Level

- Application
- Algorithm and Data Structure
- Programming Language/Compiler
- Operating System/Virtual Machines
- Instruction Set Architecture (ISA)
- Microarchitecture
- Gates/Register-Transfer Level (RTL)
- Analog/Digital Circuits
- Electronic Devices
- Physics



Previously: Abstraction helps us Manage Complexity

USTC Courses

	USTC Courses
Application	
Algorithm and Data Structure	算法基础/数据结构
Programming Language/Compiler	程序设计/编译技术
Operating System/Virtual Machines	操作系统/虚拟机
Instruction Set Architecture (ISA)	计算机组成
Microarchitecture	计算机组成
Gates/Register-Transfer Level (RTL)	数字逻辑
Analog/Digital Circuits	集成电路
Electronic Devices	微电子
Physics	

需要一门**贯通课程**，帮助学生从底层物理到高层应用，整体上理解计算机系统。

从广义上讲，计算机系统结构是抽象层次的设计，它允许我们使用可用的制造技术有效地实现信息处理应用程序。

高性能计算学习路径

I. 基础知识学习

- 数学基础
  - 线性代数、微积分、概率论 …
- 计算机科学基础
  - C/C++/Fortran/Python/R等编程语言
  - 计算机体系结构、数据结构和算法、操作系统、机器学习等

II. 高性能计算

- 高性能计算架构
  - 多核处理器、GPU等高性能计算硬件架构
- 并行编程模型
  - OpenMP、MPI、CUDA
- 性能分析与优化技术
  - 并行算法优化、内存优化、向量化等

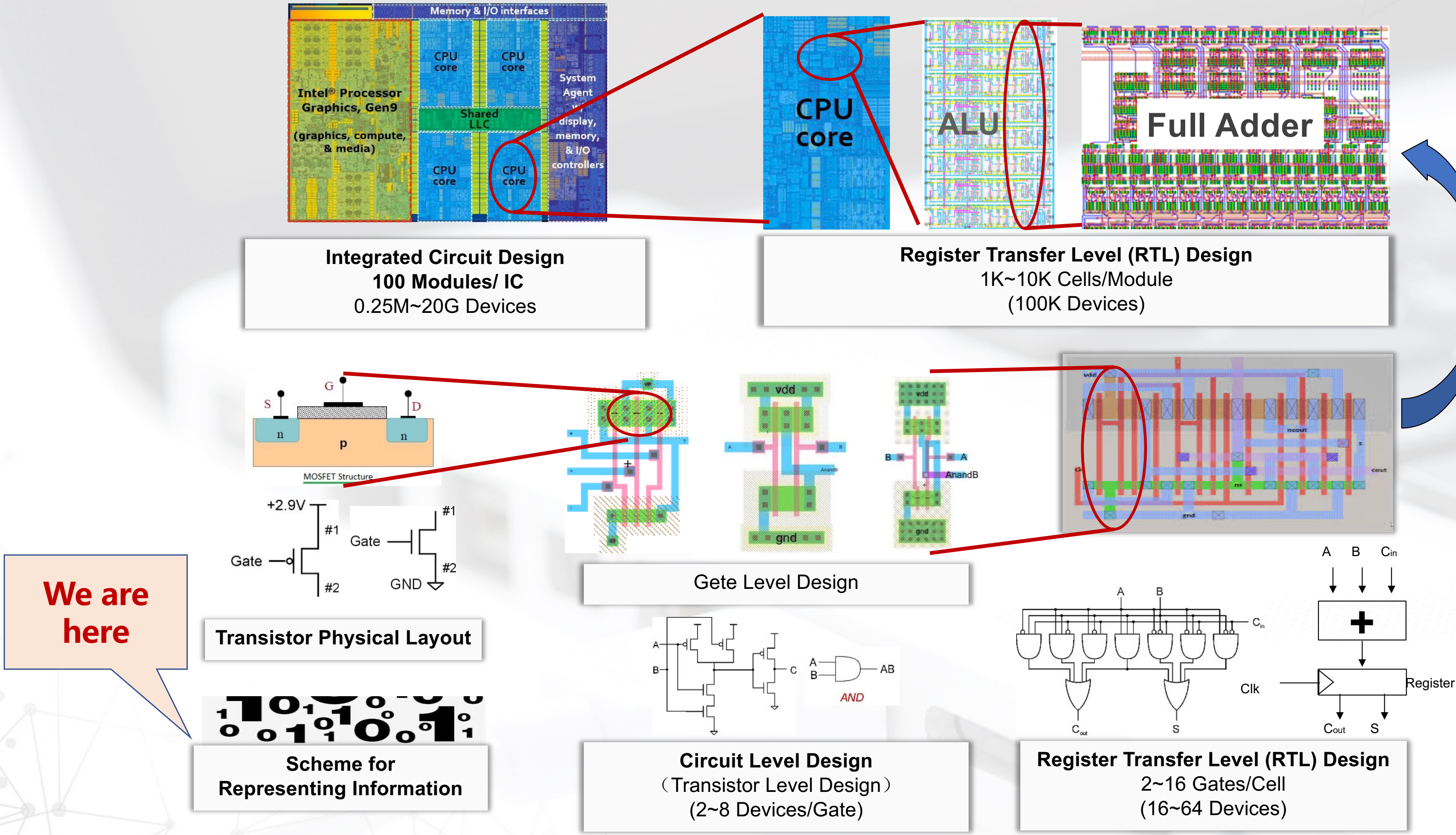
III. 领域应用计算方法

- 领域应用的数值计算方法
  - 计算流体力学、分子动力学、天体物理、分子生物学、量子计算、…
- 领域应用的常用软件和并行计算方法

参考资料：

- 请问高性能计算的学习路线应该是怎样的？ - https://www.zhihu.com/question/33576416
- Introduction to Parallel Computing Tutorial - https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial
- 高性能计算学习路线 - https://heptagonhust.github.io/HPC-roadmap/

Today



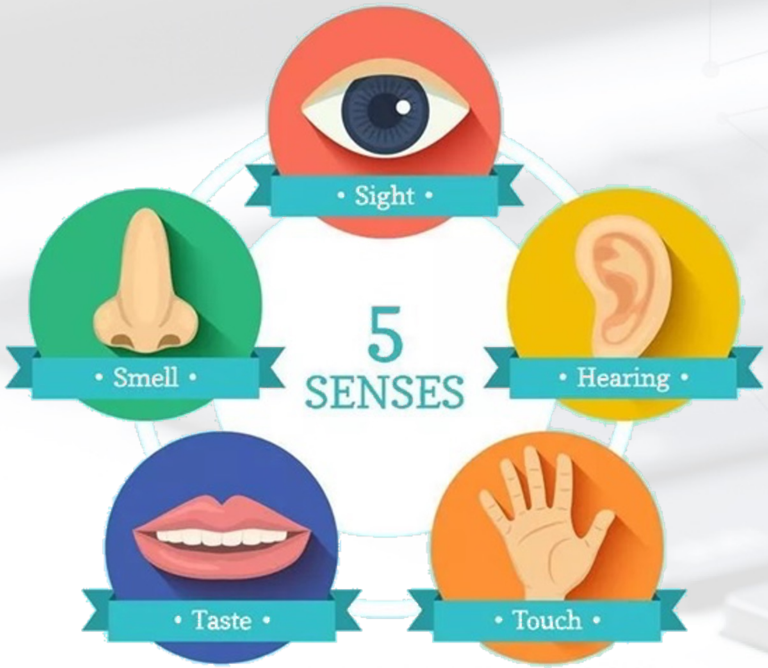
Memory & I/O interfaces
Intel® Processor Graphics, Gen9
(graphics, compute, & media)
CPU core
CPU core
Shared LLC
CPU core
CPU core
System Agent
display, memory, & I/O controllers
CPU core
ALU
Full Adder
Integrated Circuit Design
100 Modules/ IC
0.25M~20G Devices
Register Transfer Level (RTL) Design
1K~10K Cells/Module
(100K Devices)
MOSFET Structure
+2.9V
Gate
#1
#2
Gate
#1
#2
GND
vdd
gnd
AnandB
Gete Level Design
We are here
Transistor Physical Layout
Scheme for
Representing Information
AB
AND
Circuit Level Design
（Transistor Level Design）
(2~8 Devices/Gate)
A
B
Cin
Register
Clk
Cout
S
Register Transfer Level (RTL) Design
2~16 Gates/Cell
(16~64 Devices)

Outline

Outline

5 Senses of Human

- **Sight**
  - Image, picture, photo, vedio, …
- **Hearing**
  - Sound, voice, speech, music, …
- **Touch**
  - Shape, soft, hard, hurt, numb, …
- **Taste**
  - Sour, sweet, bitter, spicy, salty, …
- **Smell**
  - Sweet, smelly, …



to record by number, data, words, symbols, text, language, ……

What kinds of information do we need to represent?

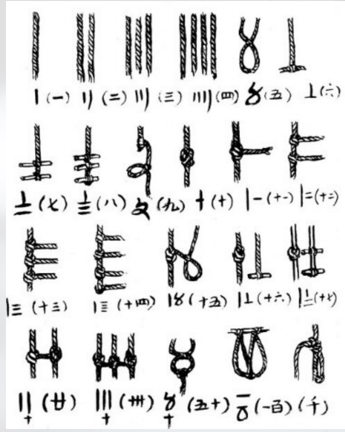
- Kinds of Information
  - **Numbers** – natural number, integers, positive/negative integers, integers/decimals, real, complex, rational, irrational, signed, unsigned, floating point, …
  - **Text** – characters, strings, …
  - **Logical** – true, false
  - **Images** – pixels, colors, shapes, …
  - **Sound** – sound of talk, sound of sing, …
  - **Video** – a series of images
  - **Instructions** – plus(+), minus(-), times (*) ,divided by(/) , …
  - …
- **Data type:** *representation* and *operations* within the computer

We'll start with **numbers…**

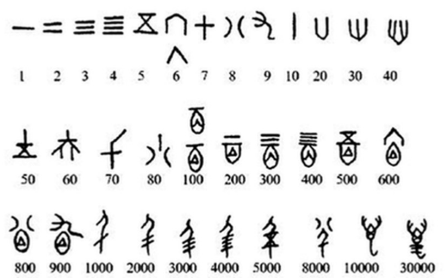
Number Notation

Counting stone(石头)

Counting rod(算筹)



Knotting(结绳)



Inscriptions on oracle bones
（甲骨文上刻字）

Number Notation

- Non-positional notation(like to **counting rod)**
  - Could represent a number ("5") with a string of ones ("11111")

  problems?

5

V

11111

Number Notation

Weighted positional notation

- decimal numbers(denary numbers): “329”
- “3” is worth 300, because of its position(with place value 100),
- while “9” is only worth 9, because of its position(with place value 1)

329

10^2 10^1 10^0

$3x100 + 2x10 + 9x1 = 329$

Denary numbers
base is 10,
place value according its position

Denary numbers - base ten

- $(5346)_{10}$

5346

Available digit	**0, 1, 2, 3, 4, 5, 6, 7, 8, 9**			
Place value	10^3=1000	10^2=100	10^1=10	10^0=1
Digit	5	3	4	6
Product of digit and place value	5x1000=5000	3x100=300	4x10=40	6x1=6

$(5346)_{10}$ = 5x1000+3x100+4x10+6x1

Great Idea from Ancient Chinese Philosophy

All things come into being, all things come into nothing

天下万物生于有，有生于无 ——《老子·四十章》

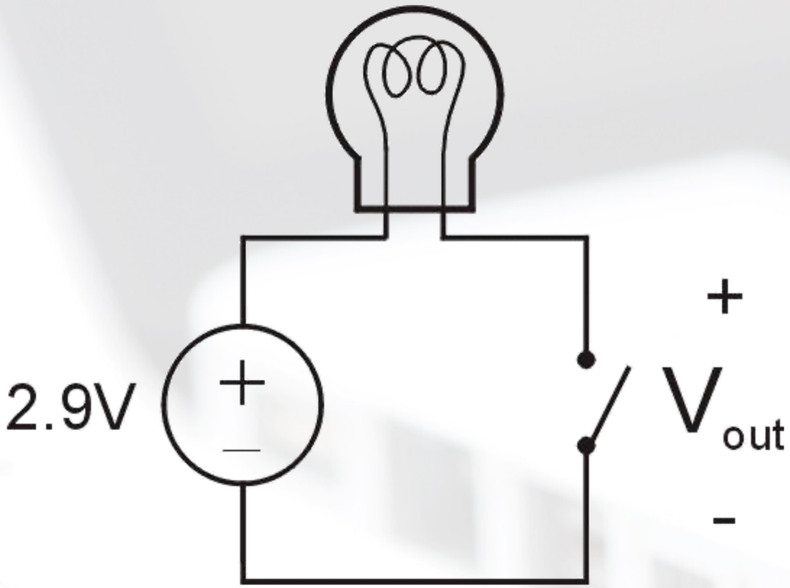
《易经》

太极生两仪，
两仪生四象，
四象生八卦，
八卦演万物。

How do we represent data in a computer?

- At the lowest level, a computer is an electronic machine.
  - works by controlling *the flow of electrons*
- Easy to recognize two conditions:
  - **presence** of a voltage – we'll call this state "1"
  - **absence** of a voltage – we'll call this state "0"
- Could base state on ***value*** of voltage, but control and detection circuits more complex.
  - compare turning on a light switch to measuring or regulating voltage
- We'll see examples of these circuits in the next chapter.

Simple Switch Circuit



Switch open:

- No current through circuit
- Light is off
- V_{out} is +2.9V

Switch closed:

- Short circuit across switch
- Current flows
- Light is on
- V_{out} is 0V

Switch-based circuits can easily represent two states:
on/off, open/closed, voltage/no voltage.

Computer is a binary digital system

Digital system:
- finite number of symbols

Binary (base two) system:
- has two states: 0 and 1

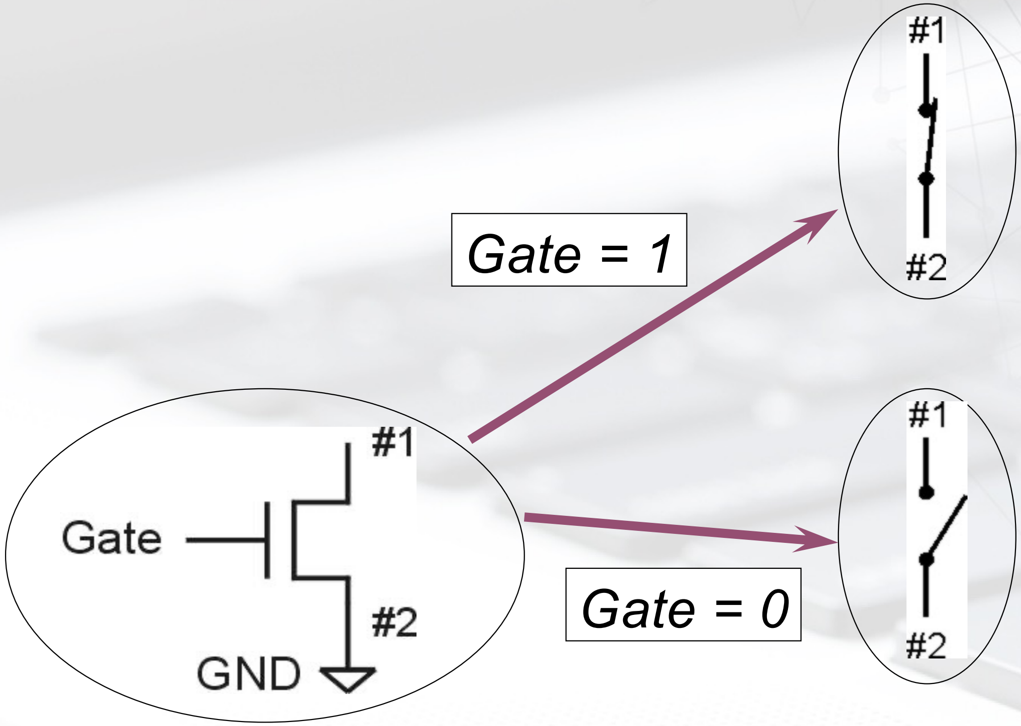
Basic unit of information is the *binary digit*, or *bit*.

Values with more than two states require multiple bits.

- A collection of two bits has four possible states:
  00, 01, 10, 11
- A collection of three bits has eight possible states:
  000, 001, 010, 011, 100, 101, 110, 111
- *A collection of n bits has 2^n possible states.*

N-type MOS Transistor

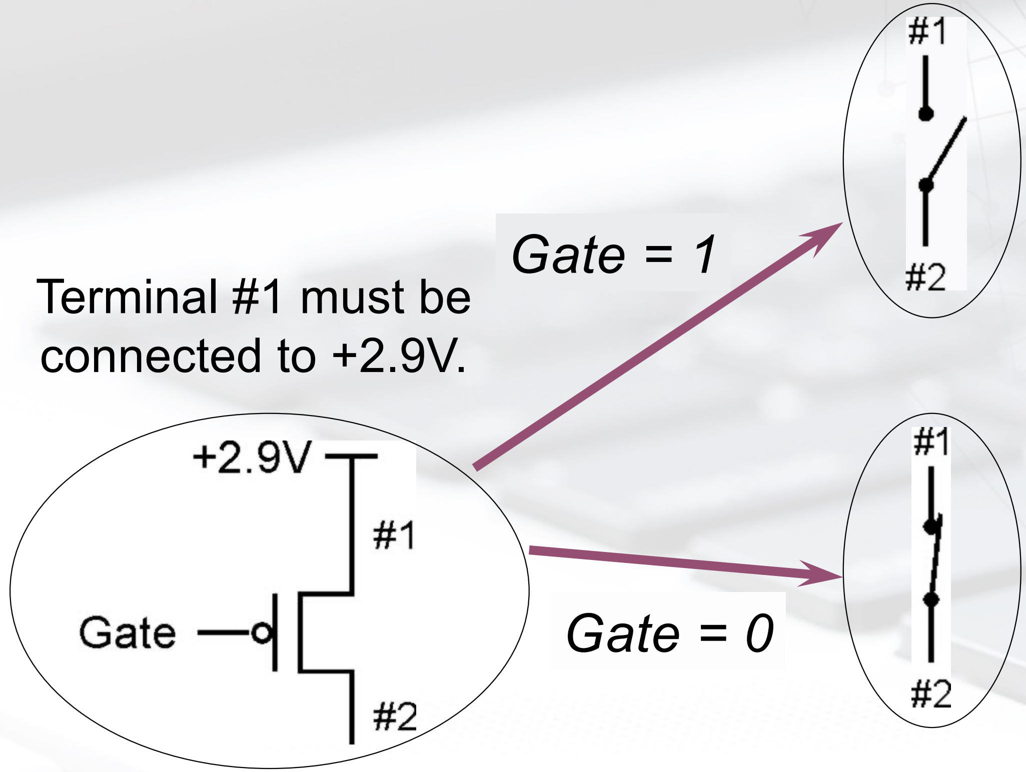
- **MOS = Metal Oxide Semiconductor**
  - two types: N-type and P-type
- **N-type**
  - when Gate has positive voltage, short circuit between #1 and #2 (switch closed)
  - when Gate has zero voltage, open circuit between #1 and #2 (switch open)



Terminal #2 must be connected to GND (0V).

P-type MOS Transistor

- **P-type** is *complementary* to N-type
  - when Gate has positive voltage, open circuit between #1 and #2 (switch open)
  - when Gate has zero voltage, short circuit between #1 and #2 (switch closed)



Logic Gates

- Use switch behavior of MOS transistors to implement logical functions: **AND, OR, NOT.**
- Digital symbols:
  - recall that we assign a range of analog voltages to each digital (logic) symbol

  - assignment of voltage ranges depends on electrical properties of transistors being used
  - typical values for "1": +5V, +3.3V, +2.9V, +1.1V for purposes of illustration, we'll use +2.9V

Binary numbers - base two

- $(101110)_2$

10 1110

Available digit	0, 1					
Place value	$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
Digit	1	0	1	1	1	0
Product of digit and place value	32	0	8	4	2	0

$(101110)_2 = 1x32+0x16+1x8+1x4+1x2+0x1 = (46)_{10}$

$(11110100)_2 = 1x128+1x64+1x32+1x16+0x8+1x4+0x2+0x1=(244)_{10}$

$(2790)_{10} = ($? $)_2$

$(5346)_{10} = ($? $)_2$

Within the Computer: Everything is a Number.

- **Numbers within the Computer**
  - Base 10 #s: Dec(imal)
    - Digits: 0,1,2,3,4,5,6,7,8,9
  - Base 2 #s: Bin(ary)
    - Digits: 0,1
  - Base 8 #s: Oct(al)
    - Digits: 0,1,2,3,4,5,6,7
  - Base 16 #s: Hex(adecimal)
    - Digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Dec(imal)	Hex(adecimal)	Oct(al)	Bin(ary)
00	0	00	0000
01	1	01	0001
02	2	02	0010
03	3	03	0011
04	4	04	0100
05	5	05	0101
06	6	06	0110
07	7	07	0111
08	8	10	1000
09	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Hexadecimal Notation

■It is often convenient to write binary (base-2) numbers as hexadecimal (base-16) numbers instead.

- `fewer digits -- four bits per hex digit`
- `less error prone -- easy  to corrupt long string of 1's and 0's`

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

Binary	Hex	Decimal
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

011101010001111010011010111

Converting from Binary to Hexadecimal

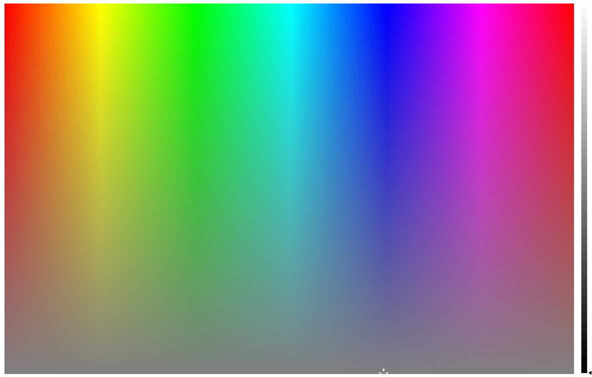
- **Every four bits is a hex digit.**
  - `start grouping from right-hand side`

```
 011101010001111010011010111
 3   A   8   F   4   D   7
```

This is not a new machine representation,
just a convenient way to write the number.

BIG IDEA：Bits can represent anything!!!

- Characters?
  - 26 letters ⇒ 5 bits ($2^5 = 32$)
  - upper/lower case + punctuation（符号） ⇒ 7 bits (in 8)(“ASCII”)
  - standard code to cover all the world’s languages ⇒ 8,16,32 bits (“Unicode”) www.unicode.com
- Logical values?
  - 0 → False, 1 → True
- colors ?
  - Ex: Red(00),Green(01),Blue(11)
- locations / addresses?
- commands?



MEMORIZE: N bits ⇔ at most 2^N things

Within the Computer: Everything is a Number.

- **Bit(BInary digiT)**
  - `1Bits=2things;`
  - `2Bits=4things;`
  - `4Bits=16things;`
  - `8Bits=256things`
  - ……
- **Byte**
  - `1Byte=8Bits`
  - `A byte is 8 bits`
- **But numbers usually stored with a fixed size**
  - `8-bit bytes;`
  - `16-bit half words;`
  - `32-bit words;`
  - `64-bit double words, ...`
  - `And there are really only two primitive "numbers": 0 and 1 is a "bit"`

Outline

1 How do we represent information in a computer?

2 **Integer Data Types**

3 2’ Complement Integers

4 Binary-Decimal Conversion

5 Operations on Bits: Arithmetic and Logical

6 Other Representation

Unsigned Integers

■ Weighted positional notation

- like decimal numbers: “329”
- “3” is worth 300, because of its position, while “9” is only worth 9

329: 10^2 10^1 10^0

3x100 + 2x10 + 9x1 = 329

101 — MSB (most significant) / LSB (least significant): 2^2 2^1 2^0

1x4 + 0x2 + 1x1 = 5

Unsigned Integers

- **An *n*-bit unsigned integer represents 2^n values: from 0 to 2^n-1.**

2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Unsigned Binary Arithmetic

- **Base-2 addition – just like base-10!**
  - add from right to left, propagating carry

carry

```
   10010        10010      1111
+   1001     +   1011   +     1
   11011        11101     10000

                10111
             +    111
```

- Subtraction, multiplication, division,…

Signed Integers

- With n bits, we have 2^n distinct values.
  - assign about half to positive integers (1 through 2^{n-1}) and about half to negative (-2^{n-1} through -1)
  - that leaves two values: one for 0, and one extra
- Positive integers
  - just like unsigned – zero in Most Significant (MS) bit
    00101 = 5
- Negative integers
  - sign-magnitude（原码） – set top bit to show negative, other bits are the same as unsigned
    10101 = -5
  - one's complement（反码） – flip every bit to represent negative
    11010 = -5
  - in either case, MS bit indicates sign: 0=positive, 1=negative

Three representations of signed integers

Representation	Value Represented: Signed Magnitude	1's Complement	2's Complement
0 0 0 0 0	0	0	0
0 0 0 0 1	1	1	1
0 0 0 1 0	2	2	2
0 0 0 1 1	3	3	3
0 0 1 0 0	4	4	4
0 0 1 0 1	5	5	5
0 0 1 1 0	6	6	6
0 0 1 1 1	7	7	7
0 1 0 0 0	8	8	8
0 1 0 0 1	9	9	9
0 1 0 1 0	10	10	10
0 1 0 1 1	11	11	11
0 1 1 0 0	12	12	12
0 1 1 0 1	13	13	13
0 1 1 1 0	14	14	14
0 1 1 1 1	15	15	15

Representation	Value Represented: Signed Magnitude	1's Complement	2's Complement
1 0 0 0 0	—0	—15	—16
1 0 0 0 1	—1	—14	—15
1 0 0 1 0	—2	—13	—14
1 0 0 1 1	—3	—12	—13
1 0 1 0 0	—4	—11	—12
1 0 1 0 1	—5	—10	—11
1 0 1 1 0	—6	—9	—10
1 0 1 1 1	—7	—8	—9
1 1 0 0 0	—8	—7	—8
1 1 0 0 1	—9	—6	—7
1 1 0 1 0	—10	—5	—6
1 1 0 1 1	—11	—4	—5
1 1 1 0 0	—12	—3	—4
1 1 1 0 1	—13	—2	—3
1 1 1 1 0	—14	—1	—2
1 1 1 1 1	—15	—0	—1

Signed Magnitude:

5 - 5 = 5 + (-5) =-10

```
    00101    (5)
+   10101     (-5)
    11010  (-10)
```

1's Complement:

5 - 5 = 5 + (-5) =- 0

```
    00101    (5)
+   11010    (-5)
    11111    (-0)
```

Outline

Two’s Complement Representation

- If number is positive or zero,
  - normal binary representation, zeroes in upper bit(s)
- If number is negative,
  - start with positive number
  - flip every bit (i.e., take the one’s complement)
  - then add one
- This representation makes the hardware simple!

	00101	(5)
	11010	(1's comp)
+	1	
	11011	(-5)

	01001	(9)
		(1's comp)
+	1	
		(-9)

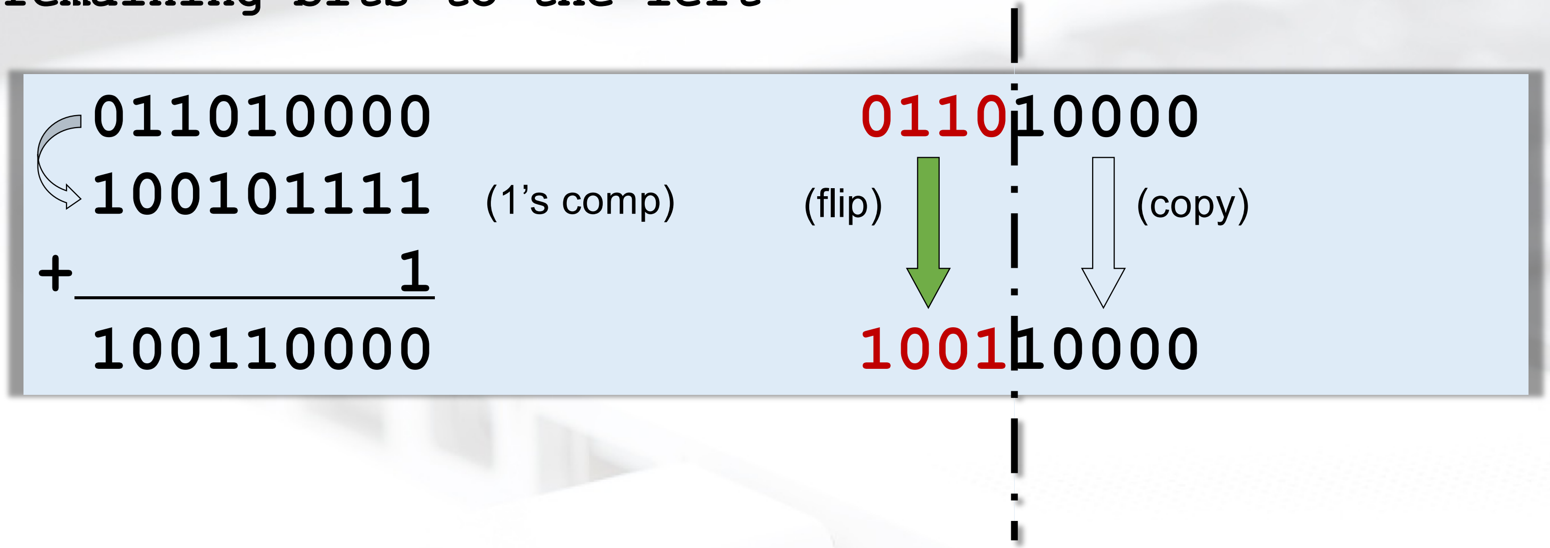
Two’s Complement

- **Problems with sign-magnitude and 1’s complement**
  - **two representations of zero (+0 and -0)**
  - **arithmetic circuits are complex**
    - **How to add two sign-magnitude numbers?**
      - e.g., try 2 + (-3)
    - **How to add two one’s complement numbers?**
      - e.g., try 4 + (-3)
- ***Two’s complement*** **representation developed to make circuits easy for arithmetic.**
  - **for each positive number (X), assign value to its negative (-X), such that X + (-X) = 0 with “normal” addition, ignoring carry out**

	00101	(5)		01001	(9)
+	11011	(-5)	+		(-9)
	00000	(0)		00000	(0)

Two’s Complement Shortcut

- To take the two’s complement of a number:
  - copy bits from right to left until (and including) the first “1”
  - flip remaining bits to the left



Two's Complement Signed Integers

- MS bit is sign bit – it has weight -2^{n-1}.
- Range of an n-bit number: -2^{n-1} through $2^{n-1} - 1$.
  - The most negative number (-2^{n-1}) has no positive counterpart.

-2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

-2^3	2^2	2^1	2^0	
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Three representations of signed integers

Representation					Value Represented: Signed Magnitude	Value Represented: 1's Complement	Value Represented: 2's Complement
0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1
0	0	0	1	0	2	2	2
0	0	0	1	1	3	3	3
0	0	1	0	0	4	4	4
0	0	1	0	1	5	5	5
0	0	1	1	0	6	6	6
0	0	1	1	1	7	7	7
0	1	0	0	0	8	8	8
0	1	0	0	1	9	9	9
0	1	0	1	0	10	10	10
0	1	0	1	1	11	11	11
0	1	1	0	0	12	12	12
0	1	1	0	1	13	13	13
0	1	1	1	0	14	14	14
0	1	1	1	1	15	15	15

Representation					Value Represented: Signed Magnitude	Value Represented: 1's Complement	Value Represented: 2's Complement
1	0	0	0	0	—0	—15	—16
1	0	0	0	1	—1	—14	—15
1	0	0	1	0	—2	—13	—14
1	0	0	1	1	—3	—12	—13
1	0	1	0	0	—4	—11	—12
1	0	1	0	1	—5	—10	—11
1	0	1	1	0	—6	—9	—10
1	0	1	1	1	—7	—8	—9
1	1	0	0	0	—8	—7	—8
1	1	0	0	1	—9	—6	—7
1	1	0	1	0	—10	—5	—6
1	1	0	1	1	—11	—4	—5
1	1	1	0	0	—12	—3	—4
1	1	1	0	1	—13	—2	—3
1	1	1	1	0	—14	—1	—2
1	1	1	1	1	—15	—0	—1

Signed Magnitude:

5 - 5 = 5 + (-5) =-10

```
    00101     (5)
+   10101      (-5)
    11010   (-10)
```

1's Complement:

5 - 5 = 5 + (-5) =- 0

```
    00101     (5)
+   11010    (-5)
    11111    (-0)
```

2's Complement:

5 - 5 = 5 + (-5) = 0

```
    00101      (5)
+   11011        (-5)
    00000      (0)
```

Q&A

- Suppose we had a 5-bit word. What integers can be represented in two’ s complement?
  - A. -32~+31
  - B. 0~+31
  - C. -16~+15
  - D. -15~+16
- Suppose we had a 8-bit word. What integers can be represented in two’ s complement?
- Suppose we had a 16-bit word. What integers can be represented in two’ s complement?
- Suppose we had a 32-bit word. What integers can be represented in two’ s complement?

Outline

Converting Binary (2’ s C) to Decimal

1. If leading bit is one, take two’s complement to get a positive number.
2. Add powers of 2 that have “1” in the corresponding bit positions.
3. If original number was negative, add a minus sign.

$$X = 01101000_{two}$$
$$= 2^6+2^5+2^3 = 64+32+8$$
$$= 104_{ten}$$

Assuming 8-bit 2’s complement numbers.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

More Examples

$X = 00100111_{two}$
$= 2^5+2^2+2^1+2^0 = 32+4+2+1$
$= 39_{ten}$

$X = 11100110_{two}$
$-X = 00011010$
$= 2^4+2^3+2^1 = 16+8+2$
$= 26_{ten}$
$X = -26_{ten}$

Assuming 8-bit 2's complement numbers.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Converting Decimal to Binary (2's C)

First Method: *Division*

1. Divide by two – remainder is least significant bit.
2. Keep dividing by two until answer is zero, writing remainders from right to left.
3. Append a zero as the MS bit; if original number negative, take two's complement.

$X = 104_{ten}$	104/2	=	52 r0	*bit 0*
	52/2	=	26 r0	*bit 1*
	26/2	=	13 r0	*bit 2*
	13/2	=	6 r1	*bit 3*
	6/2	=	3 r0	*bit 4*
	3/2	=	1 r1	*bit 5*
$X = 01101000_{two}$	1/2	=	0 r1	*bit 6*

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Converting Decimal to Binary (2's C)

Second Method: *Subtract Powers of Two*

1. Change to positive decimal number.
2. Subtract largest power of two less than or equal to number.
3. Put a one in the corresponding bit position.
4. Keep subtracting until result is zero.
5. Append a zero as MS bit; if original was negative, take two's complement.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

$X = 104_{ten}$

104 - 64 =	40	*bit 6*
40 - 32 =	8	*bit 5*
8 - 8 =	0	*bit 3*

$X = 01101000_{two}$

Outline

Operations: Arithmetic and Logical

- Recall: a data type includes ***representation*** and ***operations***.
- We now have a good representation for signed integers, so let's look at some arithmetic operations:
  - **Addition**
  - **Subtraction**
  - **Sign Extension ! ! !**
- We'll also look at **overflow** conditions for addition.
- Multiplication, division, etc., can be built from these basic operations.
- Logical operations are also useful:
  - **AND, OR, NOT**

Addition

- As we've discussed, 2's comp. addition is just binary addition.
  - assume all integers have the same number of bits
  - ignore carry out
  - for now, assume that sum fits in n-bit 2's comp. representation

```
   01101000 (104)             11110110 (-10)
 + 11110000 (-16)         +              (-9)
   01011000 (88)                         (-19)
```

Assuming 8-bit 2's complement numbers.

Subtraction

- **Negate subtrahend (2nd no.) and add.**
  - assume all integers have the same number of bits
  - ignore carry out
  - for now, assume that difference fits in n-bit 2's comp. representation

```
  01101000  (104)           11110110 (-10)
- 00010000   (16)         -           (-9)

  01101000  (104)           11110110 (-10)
+ 11110000  (-16)         +            (9)
  01011000   (88)                    (-1)
```

Assuming 8-bit 2's complement numbers.

Sign Extension

- To add two numbers, we must represent them with the **same number** of bits.
- If we just pad with zeros on the left:

4-bit	**8-bit**
0100 (4)	00000100 (still 4)
1100 (-4)	00001100 (12, not -4)

- Instead, replicate the most significant bit (MSB) -- the sign bit:

4-bit	**8-bit**
0100 (4)	00000100 (still 4)
1100 (-4)	11111100 (still -4)

Overflow

- Recall the represent range of n-bit 2' complement Signed Integers
- For an n-bit number:

$$-2^{n-1} \sim 2^{n-1} - 1$$

- Can we use n-bit 2' complement to represent a value larger than $2^{n-1}-1$? Or a value smaller than -2^{n-1} ?

Overflow

- If operands are too big, then sum cannot be represented as an *n*-bit 2' s comp number.

01000 (8)		11000 (-8)	
+ 01001 (9)		+ 10111 (-9)	
10001 (-15)		01111 (+15)	

- We have overflow if:
  - signs of both operands are the same, and
  - sign of sum is different.

- Another test -- easy for hardware:
  - carry into MS bit does not equal carry out

```
 01000(8)        11000(-8)
+01001(9)       +10111(-9)
 10001(-15)      01111(+15)
01              10
```

Logical Operations

- **Operations on logical TRUE or FALSE**
  - two states -- takes one bit to represent:
  - TRUE=1, FALSE=0
- **View *n*-bit number as a collection of *n* logical values**
  - operation applied to each bit independently

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT A
0	1
1	0

Examples of Logical Operations

■ **AND**

- `useful for clearing bits`
  - —AND with zero = 0
  - —AND with one = no change

```
     11000101
AND  00001111
     00000101
```

■ **OR**

- `useful for setting bits`
  - —OR with zero = no change
  - —OR with one = 1

```
     11000101
OR   00001111
     11001111
```

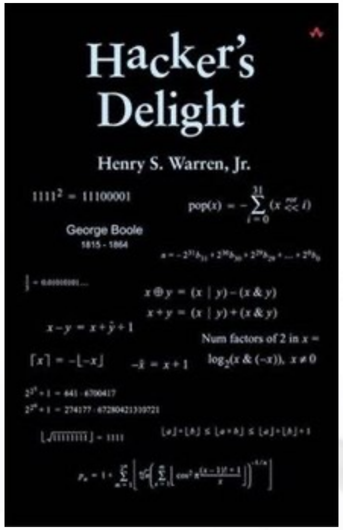
■ **NOT**

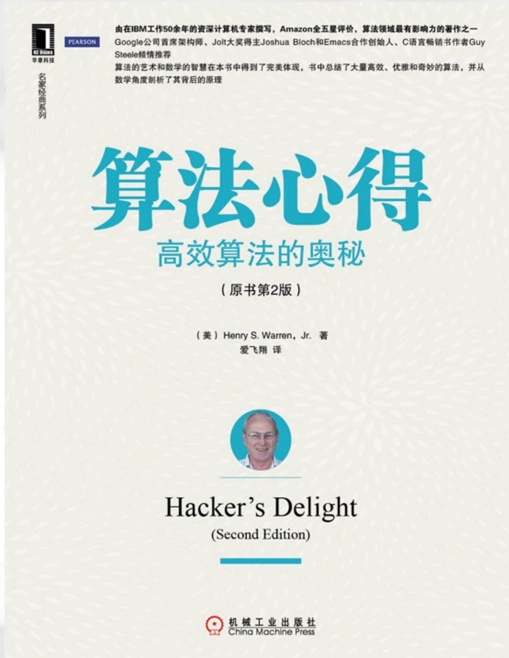
- `unary operation -- one argument`
- `flips every bit`

```
NOT  11000101
     00111010
```

Hacker’s Delight

- ***Hacker's Delight***
  - by Henry S. Warren, Jr.
  - first published in 2002
  - fast bit-level and low-level arithmetic algorithms
- **Bit Twiddling Hacks**
  - By Sean Eron Anderson
  - https://graphics.stanford.edu/~seander/bithacks.html





Outline

1. How do we represent information in a computer?
2. Integer Data Types
3. 2’ Complement Integers
4. Binary-Decimal Conversion
5. Operations on Bits: Arithmetic and Logical
6. **Other Representation**

Fractions: Fixed-Point

- **How can we represent fractions?**
  - Use a "binary point" to separate positive from negative powers of two -- just like "decimal point."
  - 2's comp addition and subtraction still work.
    - —if binary points are aligned

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

```
  00101000.101 (40.625)
+ 11111110.110 (-1.25)
  00100111.011 (39.375)
```

No new operations -- same as integer arithmetic.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Fractions: Fixed-Point

- **How can we represent fractions?**
  - Use a "binary point" to separate positive from negative powers of two -- just like "decimal point."
  - 2's complement addition and subtraction still work.
    - —if binary points are aligned
- **Example： 5-bit fraction**

fraction			Integer	
010.10	2.5	$(10/2^2)$	01010	10
+101.11	-2.25	$(-9/2^2)$	+ 10111	-9
000.01	0.25	$(1/2^2)$	00001	1

n	2^n
-4	0.0625
-3	0.125
-2	0.25
-1	0.5
0	1
1	2
2	4
3	8
4	16
5	32
6	64

- A n-bit binary fraction with k fraction bits is equivalent to the n-bit binary integer divided by 2^k

Very Large and Very Small Data

- The LC-3 use the 16bit 2’s complement data type,
- One bit to identify positive or negative, 15bits to represent the magnitude of the value. We can express values:

$- 2^{15}$ through $2^{15} - 1$

(– 32768 through 32767)

How can we represent very large and very small data?

Very Large and Very Small Data

Large values: 6.023 x 10^{23} — requires 79 bits

Small values: 6.626 x 10^{-34} — requires >110 bits

How can we represent

very large and very small data?

Very Large and Very Small: Floating-Point

Large values: 6.023 x 10^{23} — requires 79 bits
Small values: 6.626 x 10^{-34} — requires >110 bits

Use equivalent of "scientific notation": F x 2^E
Need to represent F (*fraction/mantissa*), E (*exponent*), and S(sign).

IEEE 754 Floating-Point Standard (32-bits):

$$N = (-1)^S \times 1.\text{fraction} \times 2^{\text{exponent}-127}, \quad 1 \le \text{exponent} \le 254$$

$$N = (-1)^S \times 0.\text{fraction} \times 2^{-126}, \quad \text{exponent} = 0$$

Normalized Form

$$N = (-1)^s \times 1.fraction \times 2^{exponent-127} \ , \ 1 \le exponent \le 254$$

$$exponent: 0b0000\_0000 < exponent < 0b1111\_1111$$

■Single-precision IEEE floating point number:

1 01111110 10000000000000000000000

sign | exponent | fraction

- **Sign is 1 – number is negative.**
- **Exponent field, unsigned integer, excess code, biased representations: 01111110 = 126 (decimal).**
- **Fraction is .100000000000… = .5 (decimal).**

Value = -1.5 x $2^{(126-127)}$ = -1.5 x 2^{-1} = -0.75.

Floating Point Example

■Example 2.12

0 01111011 000 0000 0000 0000 0000 0000

\+ 123 0

$$1.0 \times 2^{123-127} = 2^{-4} = \frac{1}{16}$$

■Example 2.13

$- 6\frac{5}{8}$

$- (6+\frac{4}{8} + \frac{1}{8}) = -110.101$

$- 1.10101 \times 2^{2}$

$- 1.10101 \times 2^{129-127}$

1 10000001 101 0100 0000 0000 0000 0000

Floating Point Example

■ **Example 2.14**

0 10000011 001 0100 0000 0000 0000 0000

+ 1.00101 $\times 2^{131-127}$ **=10010.1=18.5**

1 10000010 001 0100 0000 0000 0000 0000

- 1.00101 $\times 2^{130-127}$ **=-1001.01=-9.25**

0 11111110 111 1111 1111 1111 1111 1111

+ 1.1111... $\times 2^{254-127}$ **= 1.1111...** $\times 2^{127} \approx 2^{128}$

Very Small: Floating-Point

- **Normalized Form**

$$N = (-1)^s \times 1.fraction \times 2^{exponent-127} \ , 1 \leq exponent \leq 254$$

$$exponent: 0b0000\_0000 < exponent < 0b1111\_1111$$

- **The smallest positive number that can be represented in normalized form is**

$$\mathbf{N = 1.00000000000000000000000 \times 2^{-126}}$$

Very Small: subnormal numbers

$$N = (-1)^s \times 0.fraction \times 2^{-126}\ ,\ exponent = 0$$

- The **largest** subnormal number is

$$\mathbf{N = 0.11111111111111111111111 \times 2^{-126}}$$

- The **smallest** subnormal number is

$$\mathbf{N = 0.00000000000000000000001 \times 2^{-126}}$$

$$\mathbf{= 2^{-23} x 2^{-126} = 2^{-149}}$$

- Example

0 00000000 00001000000000000000000

2^{-5} x 2^{-126} = 2^{-131}

Infinities

■Normalized Form

$$N = (-1)^s \times 1.fraction \times 2^{exponent-127} \text{ , } 1 \le exponent \le 254$$

$$exponent: 0b0000\_0000 < exponent < 0b1111\_1111$$

■Subnormal numbers:

$$N = (-1)^s \times 0.fraction \times 2^{-126} \text{ , } exponent = 0$$

■So, what if the exponent is equal to 1111_1111?

- If the exponent field contains 1111_1111, we use the floating point data type to represent various things, among them the notion of infinity.
- Infinity is represented by the exponent field containing all 1s and the fraction field containing all 0s.
- We represent positive infinity if the sign bit is 0 and negative infinity if the sign bit is 1

Floating-Point Operations

- **Will regular 2's complement arithmetic work for Floating Point numbers?**

(*Hint*: In decimal, how do we compute $3.07 \times 10^{12} + 9.11 \times 10^{8}$?)

Other Data Types

Text strings

- sequence of characters, terminated with NULL (0)
- typically, no hardware support

Image

- array of pixels
  - monochrome: one bit (1/0 = black/white)
  - color: red, green, blue (RGB) components (e.g., 8 bits each)
  - other properties: transparency
- hardware support:
  - typically none, in general-purpose processors
  - MMX -- multiple 8-bit operations on 32-bit word

Sound

- sequence of fixed-point numbers

Within the Computer: Everything is a Number.

How do computers represent text ? -- ASCII Characters

ASCII: Maps 128 characters to 7-bit code.

- both printable and non-printable (ESC, DEL, …) characters

<table>
<tr><td>00</td><td>nul</td><td>10</td><td>dle</td><td>20</td><td>sp</td><td>30</td><td>0</td><td>40</td><td>@</td><td>50</td><td>P</td><td>60</td><td>`</td><td>70</td><td>p</td></tr>
<tr><td>01</td><td>soh</td><td>11</td><td>dc1</td><td>21</td><td>!</td><td>31</td><td>1</td><td>41</td><td>A</td><td>51</td><td>Q</td><td>61</td><td>a</td><td>71</td><td>q</td></tr>
<tr><td>02</td><td>stx</td><td>12</td><td>dc2</td><td>22</td><td>"</td><td>32</td><td>2</td><td>42</td><td>B</td><td>52</td><td>R</td><td>62</td><td>b</td><td>72</td><td>r</td></tr>
<tr><td>03</td><td>etx</td><td>13</td><td>dc3</td><td>23</td><td>#</td><td>33</td><td>3</td><td>43</td><td>C</td><td>53</td><td>S</td><td>63</td><td>c</td><td>73</td><td>s</td></tr>
<tr><td>04</td><td>eot</td><td>14</td><td>dc4</td><td>24</td><td>$</td><td>34</td><td>4</td><td>44</td><td>D</td><td>54</td><td>T</td><td>64</td><td>d</td><td>74</td><td>t</td></tr>
<tr><td>05</td><td>enq</td><td>15</td><td>nak</td><td>25</td><td>%</td><td>35</td><td>5</td><td>45</td><td>E</td><td>55</td><td>U</td><td>65</td><td>e</td><td>75</td><td>u</td></tr>
<tr><td>06</td><td>ack</td><td>16</td><td>syn</td><td>26</td><td>&</td><td>36</td><td>6</td><td>46</td><td>F</td><td>56</td><td>V</td><td>66</td><td>f</td><td>76</td><td>v</td></tr>
<tr><td>07</td><td>bel</td><td>17</td><td>etb</td><td>27</td><td>'</td><td>37</td><td>7</td><td>47</td><td>G</td><td>57</td><td>W</td><td>67</td><td>g</td><td>77</td><td>w</td></tr>
<tr><td>08</td><td>bs</td><td>18</td><td>can</td><td>28</td><td>(</td><td>38</td><td>8</td><td>48</td><td>H</td><td>58</td><td>X</td><td>68</td><td>h</td><td>78</td><td>x</td></tr>
<tr><td>09</td><td>ht</td><td>19</td><td>em</td><td>29</td><td>)</td><td>39</td><td>9</td><td>49</td><td>I</td><td>59</td><td>Y</td><td>69</td><td>i</td><td>79</td><td>y</td></tr>
<tr><td>0a</td><td>nl</td><td>1a</td><td>sub</td><td>2a</td><td>*</td><td>3a</td><td>:</td><td>4a</td><td>J</td><td>5a</td><td>Z</td><td>6a</td><td>j</td><td>7a</td><td>z</td></tr>
<tr><td>0b</td><td>vt</td><td>1b</td><td>esc</td><td>2b</td><td>+</td><td>3b</td><td>;</td><td>4b</td><td>K</td><td>5b</td><td>[</td><td>6b</td><td>k</td><td>7b</td><td>{</td></tr>
<tr><td>0c</td><td>np</td><td>1c</td><td>fs</td><td>2c</td><td>,</td><td>3c</td><td><</td><td>4c</td><td>L</td><td>5c</td><td>\</td><td>6c</td><td>l</td><td>7c</td><td>|</td></tr>
<tr><td>0d</td><td>cr</td><td>1d</td><td>gs</td><td>2d</td><td>-</td><td>3d</td><td>=</td><td>4d</td><td>M</td><td>5d</td><td>]</td><td>6d</td><td>m</td><td>7d</td><td>}</td></tr>
<tr><td>0e</td><td>so</td><td>1e</td><td>rs</td><td>2e</td><td>.</td><td>3e</td><td>></td><td>4e</td><td>N</td><td>5e</td><td>^</td><td>6e</td><td>n</td><td>7e</td><td>~</td></tr>
<tr><td>0f</td><td>si</td><td>1f</td><td>us</td><td>2f</td><td>/</td><td>3f</td><td>?</td><td>4f</td><td>O</td><td>5f</td><td>_</td><td>6f</td><td>o</td><td>7f</td><td>del</td></tr>
</table>

ASCII (American Standard Code for Information Interchange)

ASCII表

(American Standard Code for Information Interchange 美国标准信息交换代码)

高四位		ASCII控制字符												ASCII打印字符												
		0000 (0)						0001 (1)						0010 (2)		0011 (3)		0100 (4)		0101 (5)		0110 (6)		0111 (7)		
低四位		十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	Ctrl	代码	转义字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl
0000	0	0		^@	NUL	\0	空字符	16	►	^P	DLE		数据链路转义	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH		标题开始	17	◄	^Q	DC1		设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX		正文开始	18	↕	^R	DC2		设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX		正文结束	19	‼	^S	DC3		设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4		设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK		否定应答	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK		肯定应答	22	▬	^V	SYN		同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	\a	响铃	23	↨	^W	ETB		传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	\b	退格	24	↑	^X	CAN		取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	HT	\t	横向制表	25	↓	^Y	EM		介质结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	\n	换行	26	→	^Z	SUB		替代	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	\v	纵向制表	27	←	^[	ESC	\e	溢出	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	\f	换页	28	∟	^\	FS		文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS		组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO		移出	30	▲	^^	RS		记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI		移入	31	▼	^-	US		单元分隔符	47	/	63	?	79	O	95	_	111	o	127	⌂	^Backspace 代码：DEL

注：表中的ASCII字符可以用“Alt + 小键盘上的数字键”方法输入。

Interesting Properties of ASCII Code

- What is relationship between a decimal digit ('0', '1', ...)and its ASCII code?
- What is the difference between an upper-case letter ('A', 'B', ...) and its lower-case equivalent ('a', 'b', ...)?
- Given two ASCII characters, how do we tell which comes first in alphabetical order?
- Are 128 characters enough? (http://www.unicode.org/)

No new operations – integer arithmetic and logic.

How do computers represent image ?

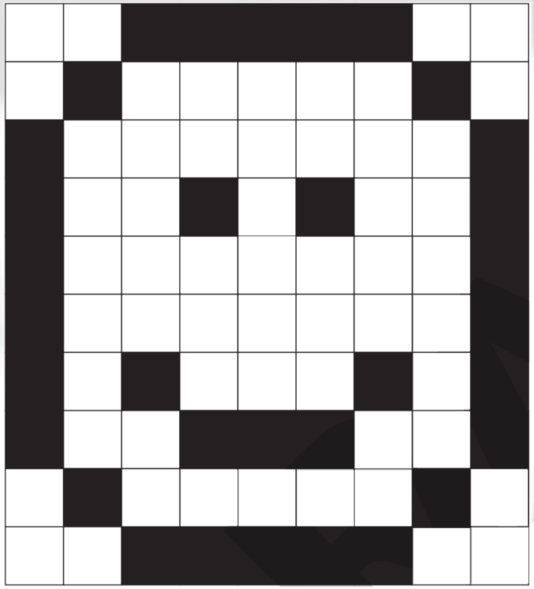
- Each image has a **resolution** and a **color depth**.
  - The **resolution** is the number of pixels wide and the number of pixels high that are used to create the image.
  - The **color depth** is the number of bits that are used to represent each color.

Figure 1.3: 8-bit colour 16-bit colour 32-bit colour

- For example, each color could be represented using 8-bit, 16-bit or 32-bit binary numbers.
- The greater the number of bits, the greater the range of colors that can be represented.

Converting images to binary

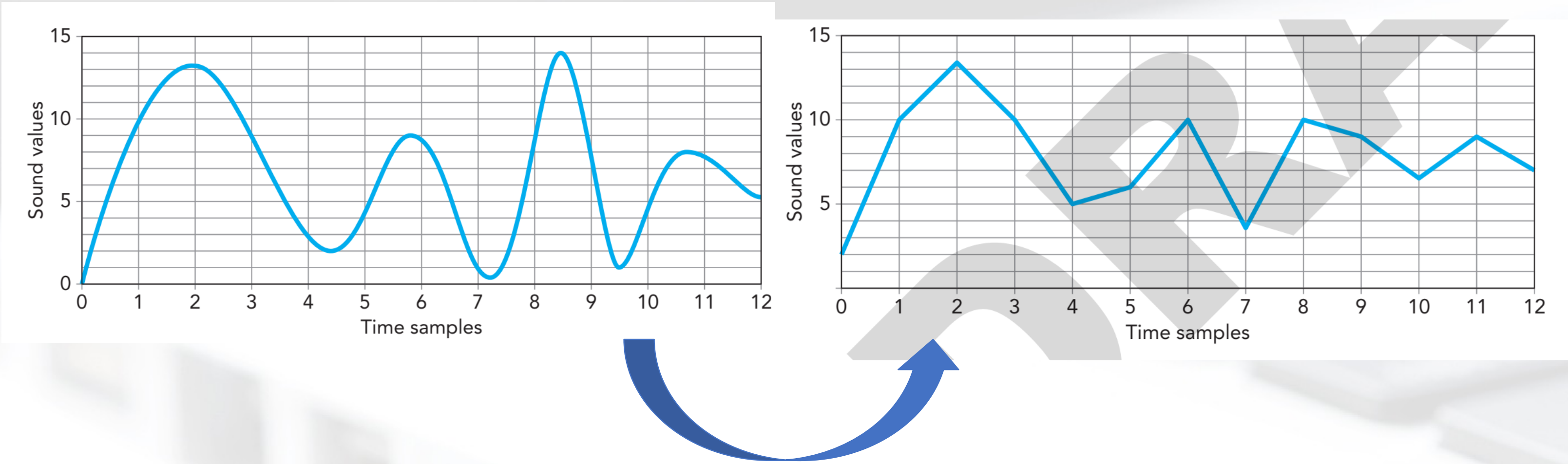
- If each pixel is converted to its binary value, a data set such as the following could be created:



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

How do computers represent sound ?

- Sound is made up of sound waves. When sound is recorded, this is done at set time intervals. This process is known as **sound sampling** :



Time sample	1	2	3	4	5	6	7	8	9	10	11	12
Sound value	9	13	9	3.5	4	9	1.5	9	8	5	8	5.5

LC-3 Data Types

- Some data types are supported directly by the instruction set architecture.
- For LC-3, there is only one supported data type:
  - `16-bit 2's complement signed integer`
  - `Operations: ADD, AND, NOT`
- Other data types are supported by interpreting 16-bit values as logical, text, fixed-point, etc., in the software that we write.

intel.

Intel® 64 and IA-32 Architectures Software Developer's Manual

Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4

NOTE: This document contains all four volumes of the Intel 64 and IA-32 Architectures Software Developer's Manual: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.

Order Number: 325462-084US
June 2024