FISF130020.01: Introduction to Computer Science

Lecture 1: Numbers and Computation

Hui Xu

xuh@fudan.edu.cn

Outline

- 1. Course Introduction
- 2. Binary System
- 3. Computation
- 4. In-class Practice

1. Course Introduction

Instruction Team

- Instructor: Xu, Hui
  - Ph. D. degree from CUHK
  - Research Interests: program analysis, software reliability
  - Email: xuh@fudan.edu.cn
  - Office: Room D6023, X2 Building, Jiangwan Campus
- Tutors:

Xingjian Zhao
zhaoxj24@m.fudan.edu.cn

Yiran Yang
23210240358@fudan.edu.cn

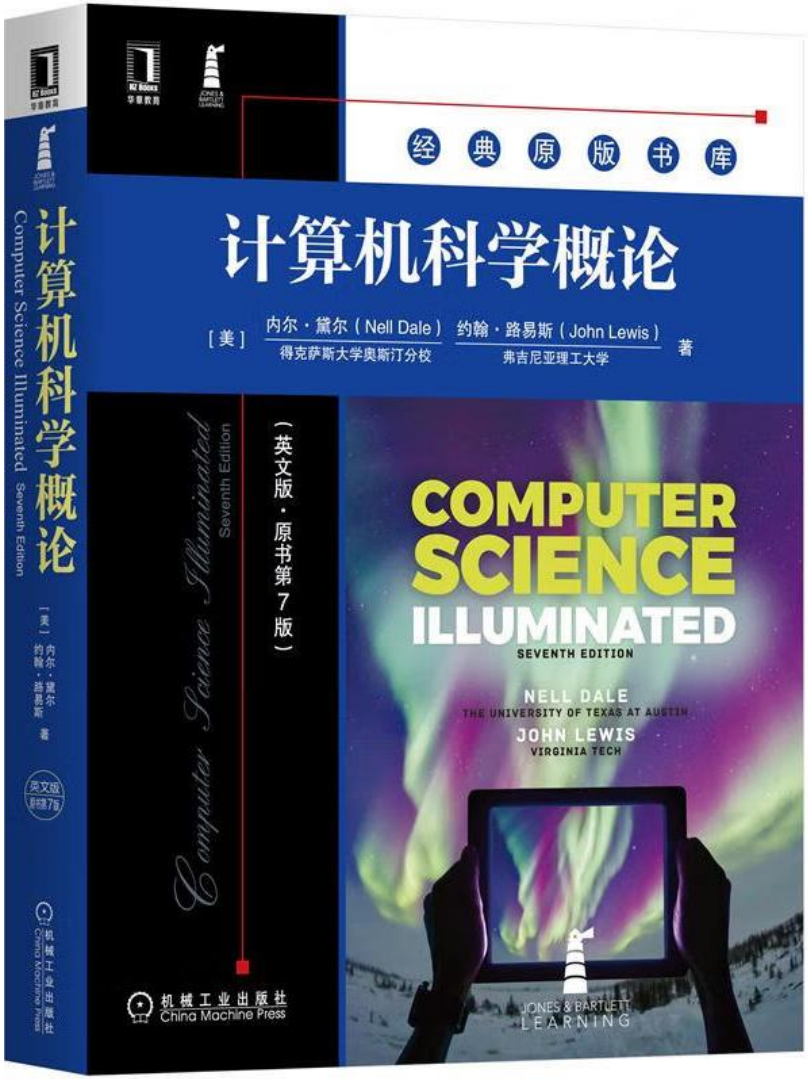
Course Objective

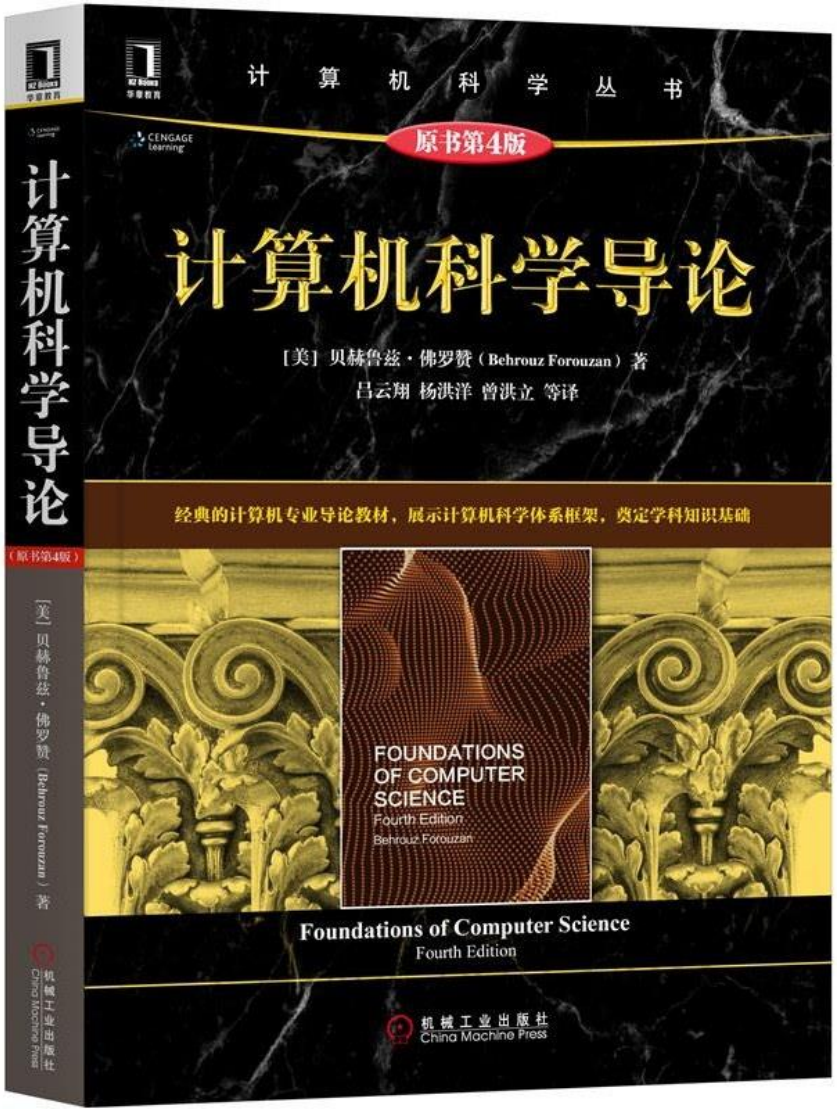
- Understand how computers work
- Learn to write simple programs
- Understand how real-world applications work
- Understand what AI can or cannot do
- Know common cybersecurity issues

Tentative Schedule

Week	Date	Course
1	Sep-5	Numbers and Computation
2	Sep-12	Data Structures
3	Sep-19	Algorithm I
4	Sep-26	Algorithm II
5	Oct-3	Holiday
6	Oct-10	Computer Architecture
7	Oct-17	Programming Language and Compiler
8	Oct-24	Operating System
9	Oct-31	Network
10	Nov-7	Database
11	Nov-14	B/S Applications
12	Nov-21	C/S Applications
13	Nov-28	Cryptography
14	Dec-5	Cybersecurity
15	Dec-12	Artificial Intelligence I
16	Dec-19	Artificial Intelligence II

Reference Book





Course Information

- Time: Thursday 6:30pm - 9:10pm
- Classroom: H6304
- Platform:
  - Couse Webpage: eLearning
    - Lecture notes
    - Assignment submission
  - Webpage: https://github.com/hxuhack/intro2cs
  - Notification & Discussion: WeChat Group

Grading

- In-Class Practice: 50%
  - Issued during the third class each week
  - Due one week later (before the class)
  - Any two practices can be waived (count n-2 best graded assignments)
- Final Exam: 50%
  - Closed book
  - Time: 2024-12-24 15:30~17:30

2. Binary System

All Data are Stored in Computers as Bits

- Natural Number
- Negative Integers
- Floating-point Numbers
- Text/Documents
- Multimedia
- Code
- Neural Networks
- …

Representing Natural Numbers as Bits

Decimal	Binary	Hex
0	**0**	0
1	**01**	1
2	**10**	2
3	**11**	3
4	**100**	4
5	**101**	5
6	**110**	6
7	**111**	7
8	**1000**	8
9	**1001**	9
10	**1010**	A
11	**1011**	B
12	**1100**	C
13	**1101**	D
14	**1110**	E
15	**1111**	F
16	**10000**	10
...	...	...

EXPLICATION
DE L'ARITHMÉTIQUE
BINAIRE,

Qui ſe ſert des ſeuls caracteres 0 *&* 1 *; avec des Remarques ſur ſon utilité, & ſur ce qu'elle donne le ſens des anciennes figures Chinoiſes de Fohy.*

PAR M. LEIBNITZ.

1703. 5. Mai.

LE calcul ordinaire d'Arithmétique ſe fait ſuivant la progreſſion de dix en dix. On ſe ſert de dix caracteres, qui ſont 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, qui ſignifient zero, un, & les nombres ſuivans juſqu'à neuf inclusivement. Et puis allant à dix, on recommence, & on écrit *dix* ; par 10 ; & dix fois dix, ou *cent*, par 100 ; & dix fois cent, ou *mille*, par 1000 ; & dix fois mille, par 10000. Et ainſi de ſuite.

Mais au lieu de la progreſſion de dix en dix, j'ai employé depuis pluſieurs années la progreſſion la plus ſimple de toutes, qui va de deux en deux ; ayant trouvé qu'elle ſert à la perfection de la ſcience des Nombres. Ainſi je n'y employe point d'autres caracteres que 0 & 1, & puis allant à deux, je recommence. C'eſt pourquoi *deux* s'écrit ici par 10, & deux fois deux ou *quatre* par 100 ; & deux fois quatre ou *huit* par 1000 ; & deux fois huit ou *ſeize* par 100000, & ainſi de ſuite. Voici *la Table des Nombres* de cette façon, qu'on peut continuer tant que l'on voudra.

On voit ici d'un coup d'œil la raiſon d'une *propriété célébre de la progreſſion Géométrique double* en Nombres entiers, qui porte que ſi on n'a qu'un de ces nombres de chaque degré, on en peut compoſer tous les autres nom-

Question

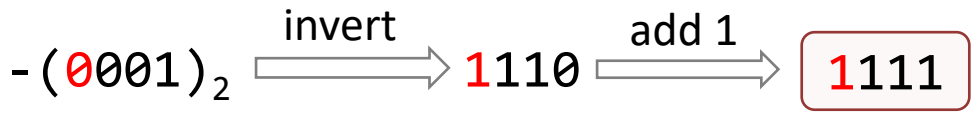
- What is the binary representation for:
  - 100
  - 1024 (also known as 1 kilo)
- What is the decimal value of:
  - 1111 1111
  - 1 0000 0000 0000 0000 0000 (1 mega)
- More quantifiers:
  - 1 giga: 2^{30}
  - 1 tera: 2^{40}
  - 1 peta: 2^{50}
  - …

How to Represent Negative Numbers?

-1 = ?

Representing Negative Integers with Two's Complement

- Employ the leftmost bit as the sign bit
  - 0 for positive numbers
  - 1 for negative numbers
  - Padding 0 or 1 to a fixed size, e.g., 4 bits
- Two's complement for negative numbers
  - Invert all the bits of the positive number
  - Add one



Decimal	Binary
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Question

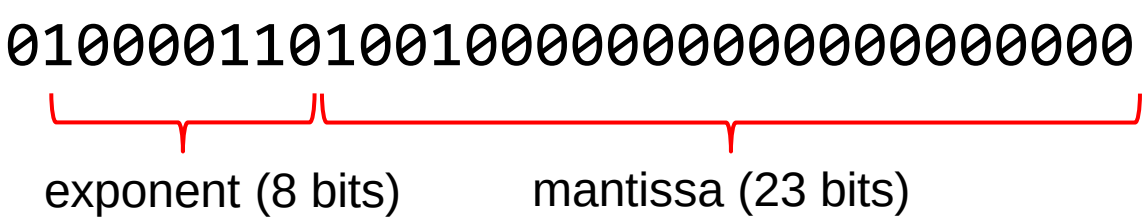
- What is the largest signed integer that 32 bits can represent?
- What is the smallest signed integer that 32 bits can represent?

How to represent larger numbers?

How to represent decimals or fractions?

Representing Floating-point Numbers with IEEE 754

- Meaning of bits for 32-bit single-precision floating-point numbers:
  - 32 (leftmost): sign bit
  - 24-31: exponent bits, to represent a wide range of values.
  - 1-23: mantissa bits
- Evaluation: $2^{exp} * mantissa$



$2^7 + 2^2 + 2^1 - 127 = 7$

$1 + 2^{-1} + 2^{-4} = 1.5625$

$$2^7 * 1.5625 = 200$$

Why do we substract a bias 127?

Convert Numbers to Floating-point Representation

- E.g., 11.25 => 01000001011010000000000000000000

Integer Part

```
11/2 = 5 + 1
5/2  = 2 + 1
2/2  = 1 + 0
1/2  = 0 + 1
```

Decimal Part

```
0.25 * 2 = 0.5 + 0
0.50 * 2 = 0.0 + 1
```

1011.01

1+.01101 with exp = 3

0 10000010 01101000000000000000000

3+127

Question

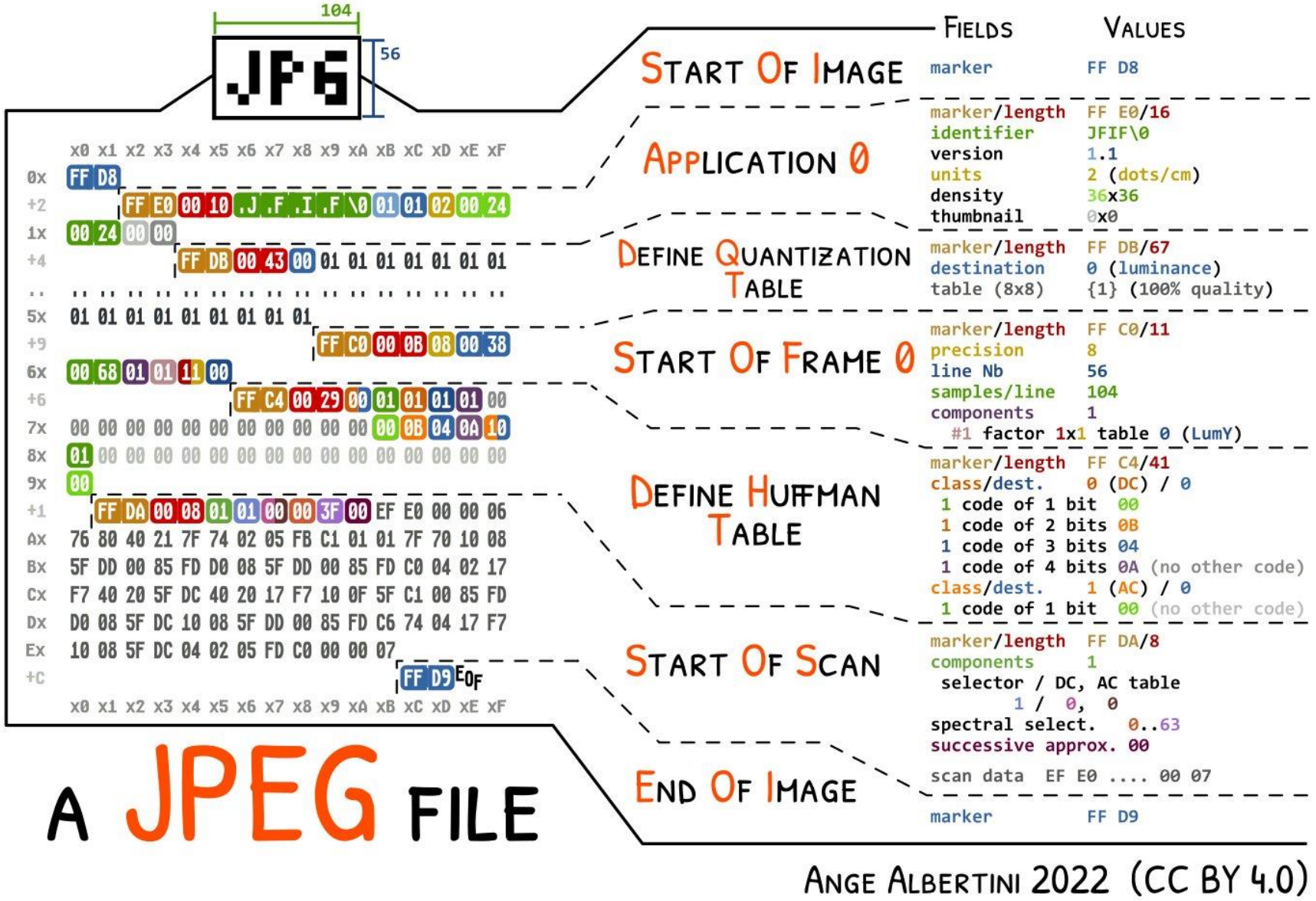
- Which of the following numbers can be represented by floating-point numbers without precision loss?
  - 0.1
  - 0.2
  - 0.3
  - 0.4
  - 0.5

Encoding Characters in Bits (as Bytes)

- ASCII (American Standard Code for Information Interchange)
  - Using 8 bits (7 useful bits) to represent a character
  - *e.g.,* 0 can be represented as 0011 0000
- 1 byte = 8 bits, or 1 B = 8 b

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Images

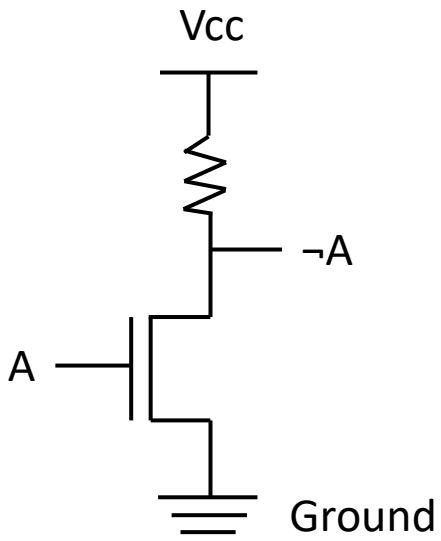

104
JPG
56
x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xA xB xC xD xE xF
0x FF D8
+2 FF E0 00 10 .J .F .I .F \0 01 01 02 00 24
1x 00 24 00 00
+4 FF DB 00 43 00 01 01 01 01 01 01 01
.. ..
5x 01 01 01 01 01 01 01 01
+9 FF C0 00 0B 08 00 38
6x 00 68 01 01 11 00
+6 FF C4 00 29 00 01 01 01 01 00
7x 00 00 00 00 00 00 00 00 00 00 00 00 0B 04 0A 10
8x 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9x 00
+1 FF DA 00 08 01 01 00 00 3F 00 EF E0 00 00 06
Ax 76 80 40 21 7F 74 02 05 FB C1 01 01 7F 70 10 08
Bx 5F DD 00 85 FD D0 08 5F DD 00 85 FD C0 04 02 17
Cx F7 40 20 5F DC 40 20 17 F7 10 0F 5F C1 00 85 FD
Dx D0 08 5F DC 10 08 5F DD 00 85 FD C6 74 04 17 F7
Ex 10 08 5F DC 04 02 05 FD C0 00 00 07
+C FF D9 EOF
x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xA xB xC xD xE xF
A JPEG FILE
FIELDS
VALUES
START OF IMAGE
marker FF D8
APPLICATION 0
marker/length FF E0/16
identifier JFIF\0
version 1.1
units 2 (dots/cm)
density 36x36
thumbnail 0x0
DEFINE QUANTIZATION TABLE
marker/length FF DB/67
destination 0 (luminance)
table (8x8) {1} (100% quality)
START OF FRAME 0
marker/length FF C0/11
precision 8
line Nb 56
samples/line 104
components 1
#1 factor 1x1 table 0 (LumY)
DEFINE HUFFMAN TABLE
marker/length FF C4/41
class/dest. 0 (DC) / 0
1 code of 1 bit 00
1 code of 2 bits 0B
1 code of 3 bits 04
1 code of 4 bits 0A (no other code)
class/dest. 1 (AC) / 0
1 code of 1 bit 00 (no other code)
START OF SCAN
marker/length FF DA/8
components 1
selector / DC, AC table
1 / 0, 0
spectral select. 0..63
successive approx. 00
scan data EF E0 .... 00 07
END OF IMAGE
marker FF D9
ANGE ALBERTINI 2022 (CC BY 4.0)

3. Computation

High-level Idea

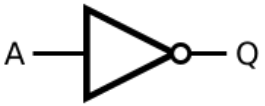
- Compose **logic gates** based on **transistors**
- Compose **artithmatic computation units** based on logic gates
- Manufacture **CPUs** with ALU, control unit, *etc*.
- Build **computers** with CPU, memory, input, and output

Transistor => Logical Gate: NOT



NMOS NOT

Simplified Representation:



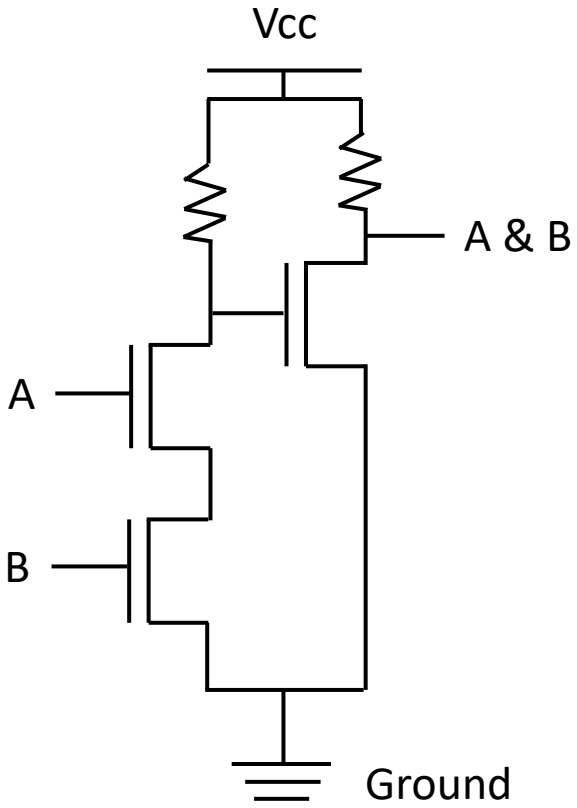
Operator Symbol: ¬

Truth Table:

A	¬ A
0	1
1	0

Vcc: Voltage Common Collector (+ terminal)

Transistor => Logical Gate: AND



NMOS AND

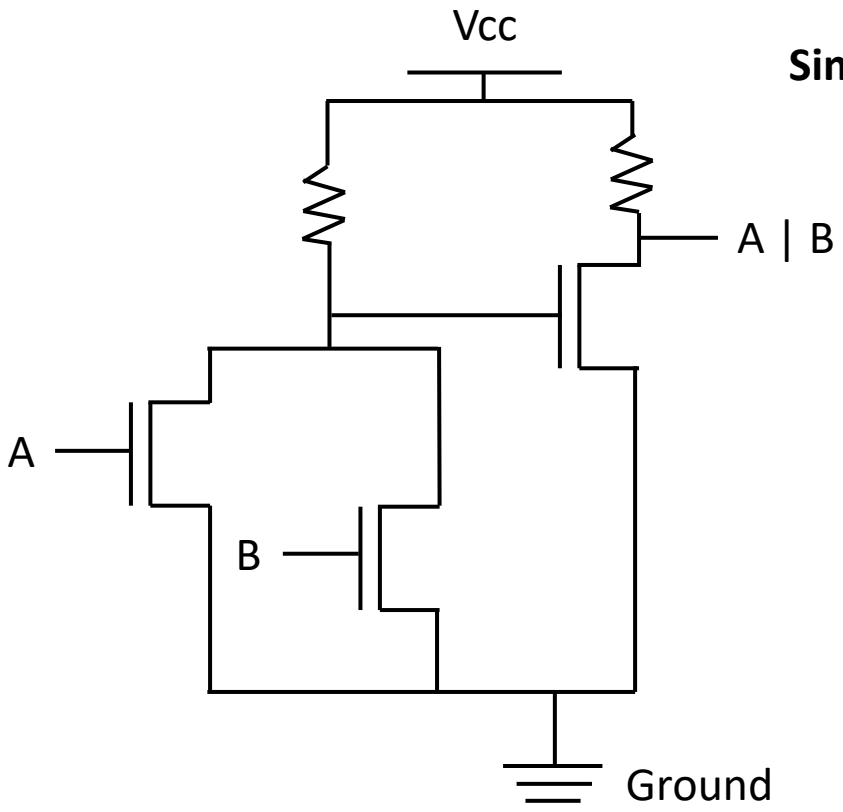
Simplified Representation:

Operator Symbol: **&**

Truth Table:

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Transistor => Logical Gate: OR



NMOS OR

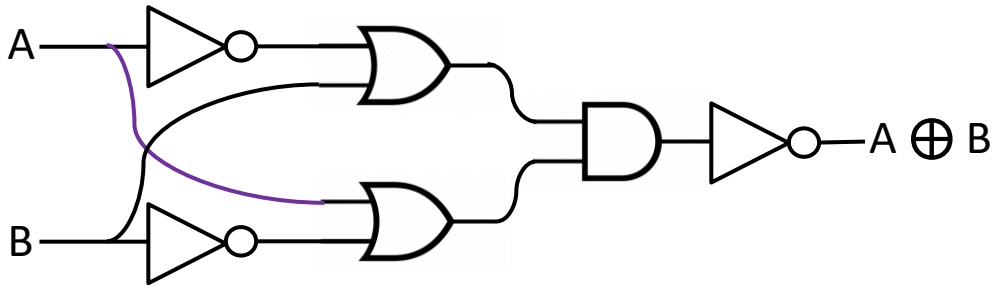
Simplified Representation:

Operator Symbol: |

Truth Table:

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

Transistor => Logical Gate: XOR



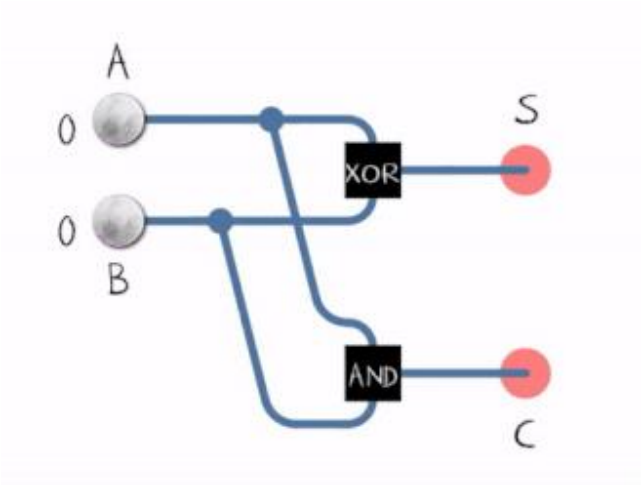
Simplified Representation:

Operator Symbol: ⊕

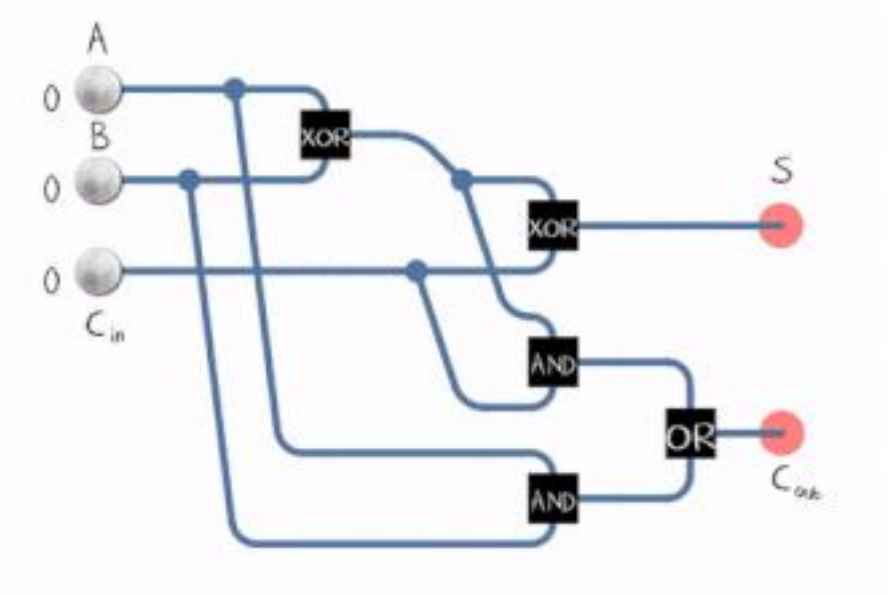
Truth Table:

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Compose an Adder with Logical Gates



Half Adder

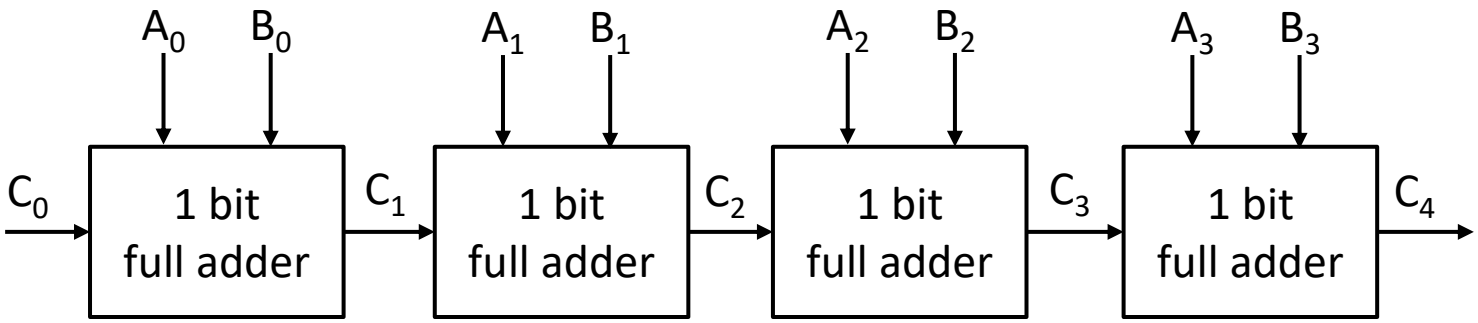


Full Adder (with carrier input)

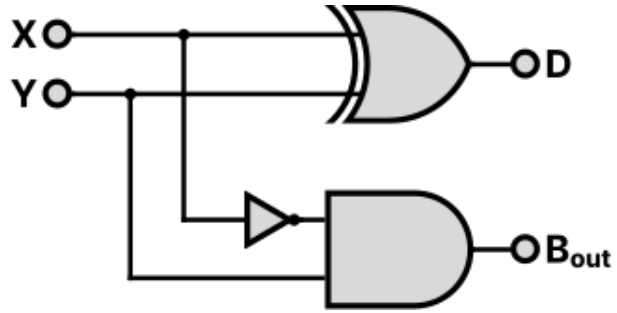
Inputs		Outputs	
A	**B**	**C_{out}**	**S**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Inputs			Outputs	
A	**B**	**C_{in}**	**C_{out}**	**S**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Add 4-bits with Full Adder

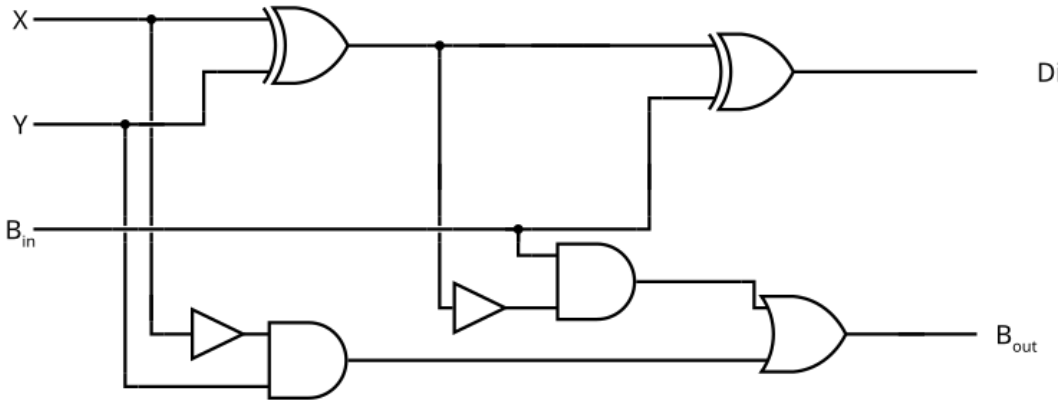


Combinational Circuits: Subtractor



Half Subtractor

Inputs		Outputs	
X	Y	D	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

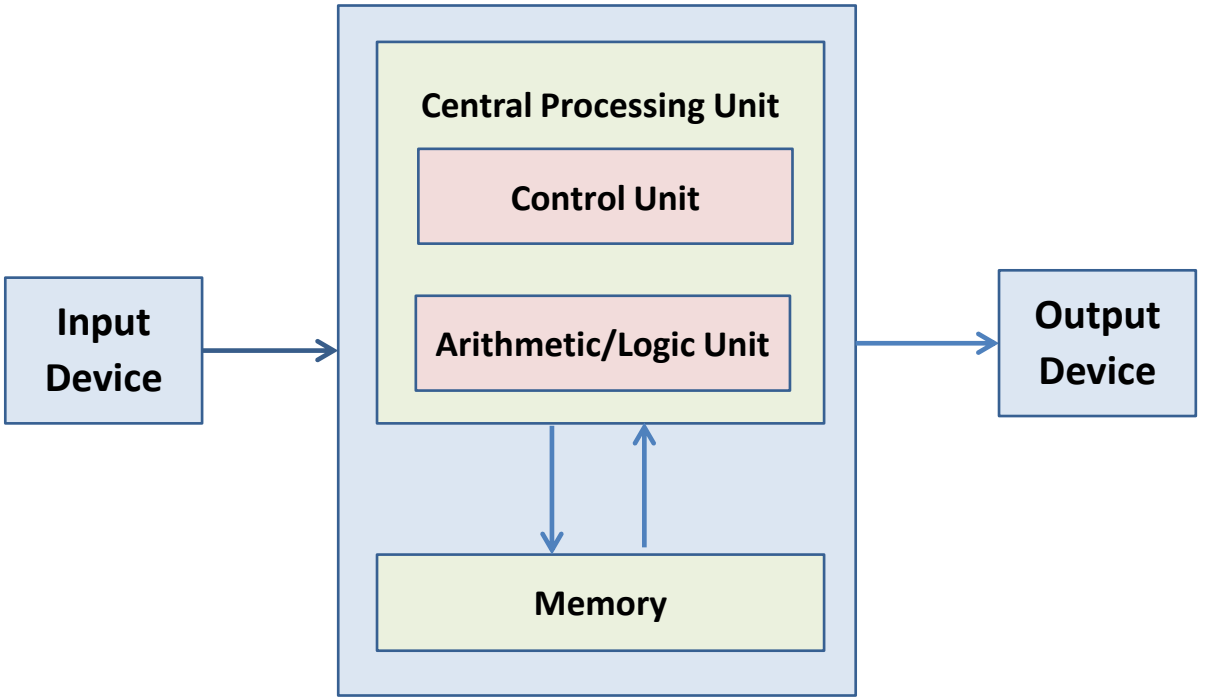


Full Subtractor (with borrow input)

Inputs			Outputs	
X	Y	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

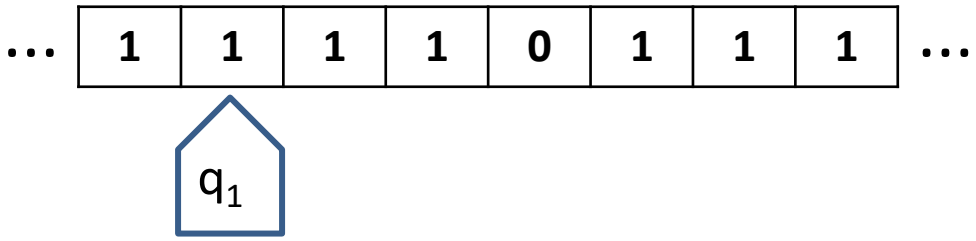
CPU and Von Neumann Model

- Control unit: fetch instructions from memory at the address specified by the program counter.
- ALU: perform the operation specified by the instruction, write the results to memory or registers.



General Turing Machine

- Infinite tape: composed on 0/1s
- Head: read/write symbos
- State register: finite number of values
- Table of instructions



	0	1
q_1	1 R q_2	1 R q_1
q_2	0 L q_3	1 R q_2
q_3	0 H q_3	0 H q_3

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.

Summary

- All data are stored in computers as bits
  - Integers => negative integers => floating-point numbers => files
- Computers are made of transistors that accept bit inputs
  - Transistors => logic gates => arithmatic units => CPU => computer

4. In-class Practice

Setup Rust Programming Environment

1) Install Rust
   - https://www.rust-lang.org/tools/install
2) Or use the web version
   - https://play.rust-lang.org/

Option 1: Analyze Floating-point Numbers

1) Write a program that output the bits of a floating-point number
2) Explain the bit representation

```
fn main() {
    let number: f32 = 200.1;
    let bits = number.to_bits();
    println!("Bit representation of {}: {:032b}", number, bits);
}
```

Option 1: Analyze Floating-point Numbers

3) Write a program that evaluate the bits of a floating-point number
4) Manually verify the correctness

```
use std::num::ParseIntError;

fn bits_to_f32(bits: &str) -> Result<f32, ParseIntError> {
    let int_value = u32::from_str_radix(bits, 2)?;
    let float_value = f32::from_bits(int_value);
    Ok(float_value)
}

fn main() {
    let bit_sequence = "01000011010010000000000000000000";
    match bits_to_f32(bit_sequence) {
        Ok(float) => println!("The f32 value is: {}", float),
        Err(e) => println!("Failed to parse bit sequence: {}", e),
    }
}
```

Option 2: Analyze the Bit Representation of Files

1) Change the file path of the following program and execute it;
2) Try to interpret the output;
3) Modify the content of the output and create a new valid jpg file;
   - *e.g.,* via hex editor tools like https://hexed.it/
4) Discuss whether your changes to the image are visible?

```
use std::fs::File;
use std::io::{self, Read};

fn main() -> io::Result<()> {
    let mut file = File::open("/home/aisr/john.jpg")?;
    let mut buffer = Vec::new();
    file.read_to_end(&mut buffer)?;
    for byte in buffer {
        print!("{:02x} ", byte);  // Prints in hexadecimal format
    }
    Ok(())
}
```