MF20006: Introduction to Computer Science

# Lecture 1: Numbers and Computation

Hui Xu

xuh@fudan.edu.cn

# Outline

1. Binary System

2. Computation

# 1. Binary System

# How to Represent Numbers?

- **Decimal numbers are commonly used in daily life.**
  - Each digit represents a power of 10
  - e.g., $123 = 10^3 + 2 * 10^2 + 3 * 10^1$
- **In computer, we represent numbers with binary (base-2) numbers.**
  - Each binary digit, or bit, represents a power of 2
  - e.g., $(1011)_2 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 11$
- **We can generalize this idea to any base *n*.**
  - Each position represents a power of *n*.
  - e.g., $(1011)_n = 1 * n^3 + 0 * n^2 + 1 * n^1 + 1 * n^0$

# Binary-based Numbers in Computer

- ❑ **Computer storage consists of consecutive bits.**
- ❑ **Which position is the start of a number?**
  - ➢ Generally via the memory address.
  - ➢ Granularity: 1-byte or 8 bit.
- ❑ **Which position is the end of a number?**
  - ➢ We need a fixed number of bits of to represent a number.
  - ➢ E.g., 8-bit, 32-bit, *etc.*

```
...000000000000000100000010000000110000010000000101...
```

Memory Address: p

# Represent Natural Numbers with 8 Bits (1 Bytes)

| **Decimal** | **Fixed-length** (8 bit/1 byte) |
|---|---|
| 0 | 00000000 |
| 1 | 00000001 |
| 2 | 00000010 |
| 3 | 00000011 |
| 4 | 00000100 |
| 5 | 00000101 |
| 6 | 00000110 |
| 7 | 00000111 |
| 8 | 00001000 |
| 9 | 00001001 |
| 10 | 00001010 |
| 11 | 00001011 |
| 12 | 00001100 |
| 13 | 00001101 |
| 14 | 00001110 |
| 15 | 00001111 |
| 16 | 00010000 |
| … | … |

# Simplified Representations in Hexadecimal Format

| **Decimal** | **Fixed-length** (8 bit/1 byte) | **Hex** (base 16) |
|---|---|---|
| 0 | 00000000 | 0 |
| 1 | 00000001 | 1 |
| 2 | 00000010 | 2 |
| 3 | 00000011 | 3 |
| 4 | 00000100 | 4 |
| 5 | 00000101 | 5 |
| 6 | 00000110 | 6 |
| 7 | 00000111 | 7 |
| 8 | 00001000 | 8 |
| 9 | 00001001 | 9 |
| 10 | 00001010 | A |
| 11 | 00001011 | B |
| 12 | 00001100 | C |
| 13 | 00001101 | D |
| 14 | 00001110 | E |
| 15 | 00001111 | F |
| 16 | 00010000 | 10 |
| … | … | … |

# Exercise

- **What is the binary representation for:**
  - $(100)_{10}$
  - $(1024)_{10}$ (also known as 1 kilo)

# More Quantifiers

- 1 mega (M): $2^{20}$
- 1 giga (G): $2^{30}$
- 1 tera (T): $2^{40}$
- 1 peta (P): $2^{50}$
- 1 exa (E): $2^{60}$
- 1 zetta (Z): $2^{70}$
- 1 yotta (Y): $2^{80}$
- ...

# How to Represent Negative Numbers?

- **Option 1: Simply employ the first bit as the sign bit.**
  - $(0011)_2 = 3, (1011)_2 = -3$
  - Issue: 0 has two representations.
- **Option 2: Complement-based approach by flipping all bits.**
  - e.g., one's complement: $x + y = 2^n - 1$
  - $(1000)_2 = -(0111)_2 = -7$
  - $(1011)_2 = -(0100)_2 = -4$
  - $(1111)_2 = -(0000)_2 = -0$
  - Issue: 0 still has two representations.
- **Two's complement:** $x + y = 2^n$
  - $(1000)_2 = -(0111 + 1)_2 = -(1000)_2 = -8$
  - $(1111)_2 = -(0000 + 1)_2 = -1$

# 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.
  - Then add one.

$$-(0001)_2 \xrightarrow{\text{invert}} 1110 \xrightarrow{\text{add 1}} \boxed{1111}$$

| 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** |

# Benefits of Two's Complement

- Turn all subtractions in to adds => Reuse the adders.

$$7 - 5 = 7 + (-5)$$

```
    0111
+   1011
--------
   ~~1~~0010
```

$$2 - 5 = 2 + (-5)$$

```
    0010
+   1011
--------
    1101
```

# Why?

**To Prove:** $x + (-y) \bmod 2^n = x - y \bmod 2^n$

According to the definition of two's complement:

$$-y = 2^n - y$$

Therefore,

$$x + (-y) \bmod 2^n$$

$$= x + 2^n - y \bmod 2^n$$

$$= x - y \bmod 2^n$$

# Exercise

- What is the largest signed integer that 16 bits can represent?
- What is the smallest signed integer that 16 bits can represent?

# Questions

- What if the the number is very large?
- How to represent fractional numbers?

# Representing Decimals

- **Fixed-point method: Allocate a fixed number of bits for the integer part and the fractional part.**
  - e.g., $(1000.0100)_2 = 2^3 + 2^{-2} = 8.25$
  - Limitation: limited range and fixed precession
- **Floating-point method: shift the decimal point based on the number.**
  - If the number is large, use more bits for the integer part
  - If the number is small, use more bits for the fractional part

# Floating-point Numbers Defined in 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: $2^{exp-bias} * mantissa$

$$0\underbrace{10000110}_{\text{exponent (8 bits)}}\underbrace{10010000000000000000000}_{\text{mantissa (23 bits)}}$$

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

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

$$2^7 * 1.5625 = 200$$

Why do we subtract a bias 127?

# Exercise

- What is the value of the following floating-point numbers
  - 01111111100000000000000000000000
  - 01000000001100000000000000000000
  - 11000000101100000000000000000000

# Conversion to Floating-point Numbers

- What is the floating-point representation for 11.25?

| Integer Part | Fractional Part |
|---|---|
| 11/2 = 5 + 1 | 0.25 * 2 = 0.5 + 0 |
| 5/2 = 2 + 1 | 0.50 * 2 = 0.0 + 1 |
| 2/2 = 1 + 0 | |
| 1/2 = 0 + 1 | |

1011.01

1+.01101 with exp = 3

$$0\underbrace{10000010}_{3+127}01101000000000000000000$$

# Exercise

- 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

# Beside Numbers, All Data are Stored in Computers as Bits

- ❑Text/Documents
- ❑Images
- ❑Videos
- ❑Sounds
- ❑Code
- ❑Neural Networks
- ❑…

# Encoding Characters in Bits (Bytes)

## ❑ASCII (American Standard Code for Information Interchange)

- Using 8 bits (7 useful bits) to represent a character.
- 0 can be represented as 0x30 in hex or 0011 0000 in binary, or 48.
- A can be represented as 0x41 in hex or 0100 0001 in binary, or 65.

| | 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 |

# 2. Computation

# Computation Unit

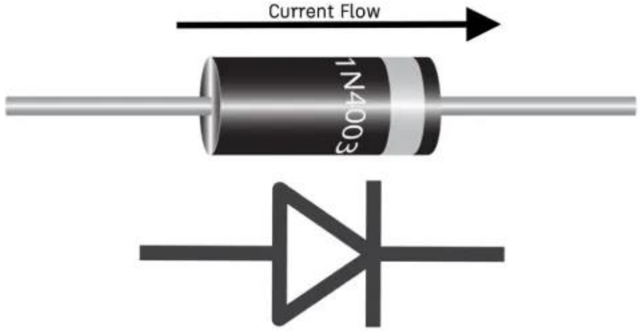
- CPUs are made with ALU (Arithmetic Logic Unit) and others.
- The ALU is built using logic gates.
- Logic gates are implemented with transistors.
- Transistors are made from semiconductors, primarily silicon.

# Semiconductor

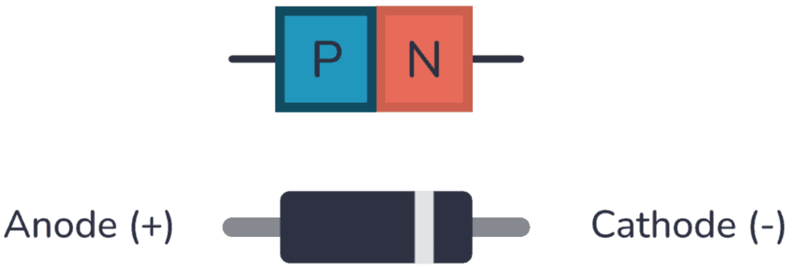
❑ **Pure materials like silicon (Si) with limited conductivity.**

❑ **Doping: Adding small amounts of impurities to change conductivity.**

- P-type: Add atoms with 3 valence electrons *e.g.,* boron.
  - Create holes (positive charge carriers).
- N-type: Add atoms with 5 valence electrons (*e.g.,* phosphorus)
  - Add free electrons (negative charge carriers).

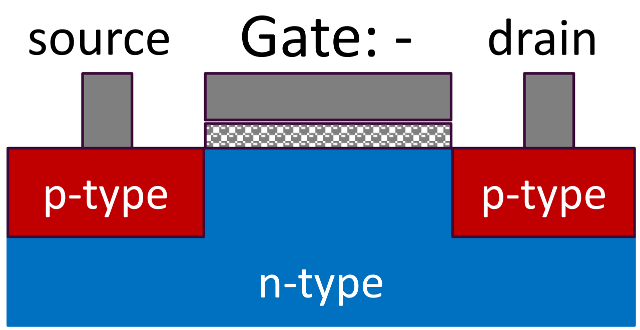



**Diode**
**(one-direction)**

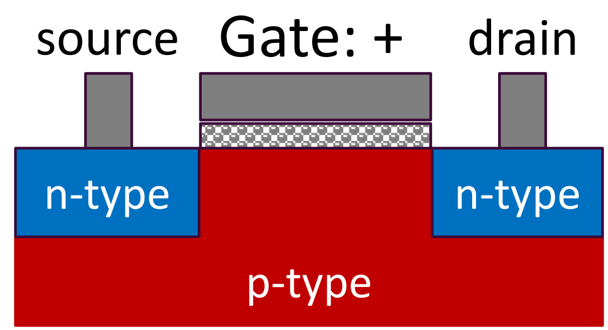


# Transistors

- ❑ **A transistor is an electronic switch that controls the flow of current.**
- ❑ **Types of Transistors:**
  - ➢ PMOS (P-type MOSFET): Conducts when gate is low.
  - ➢ NMOS (N-type MOSFET): Conducts when gate is high.


PMOS



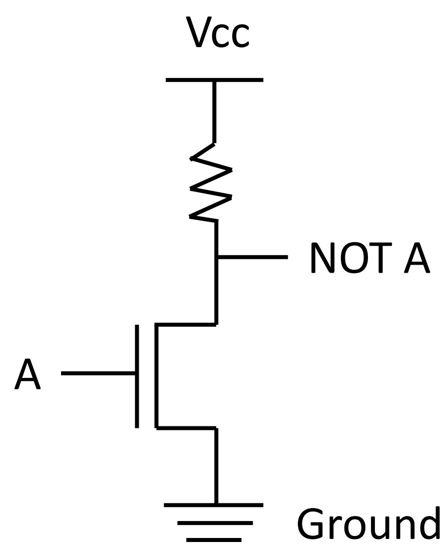

NMOS

# Transistor => Logical Gate: NOT

- When a voltage is applied to a semiconductor, it can become conductive.

**Truth Table:**

| A | Not A |
|---|---|
| 0 | 1 |
| 1 | 0 |



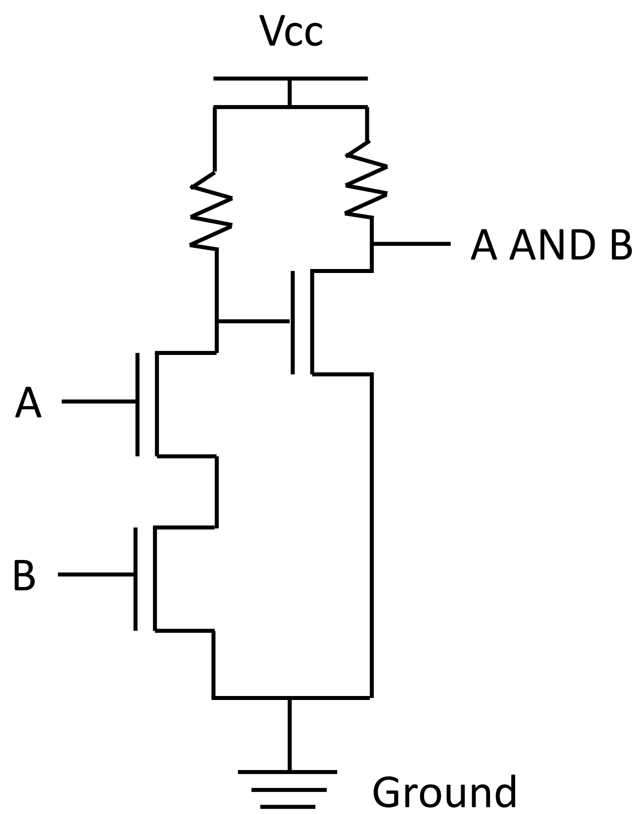

**NMOS NOT**

Vcc: Voltage Common Collector (+ terminal)

# Transistor => Logical Gate: AND

**Truth Table:**

| A | B | A AND B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

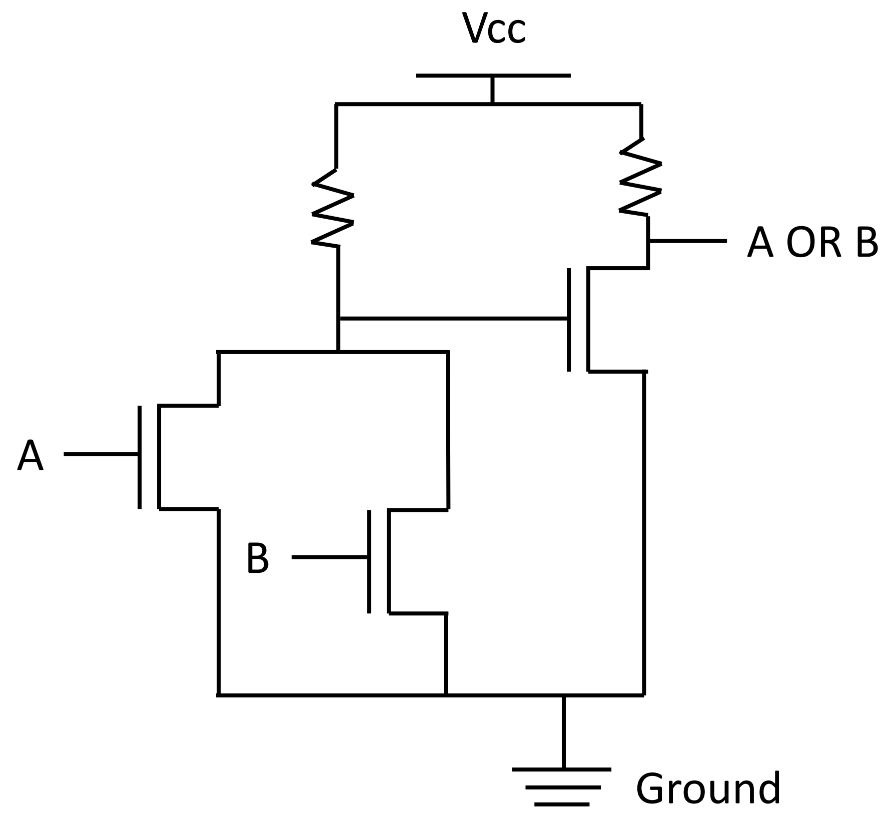



**NMOS AND**

# Transistor => Logical Gate: OR

**Truth Table:**

| A | B | A OR B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

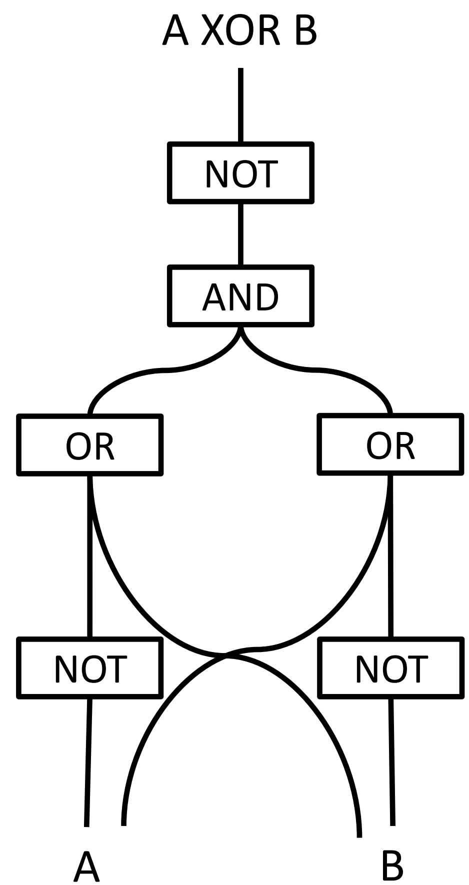



**NMOS OR**

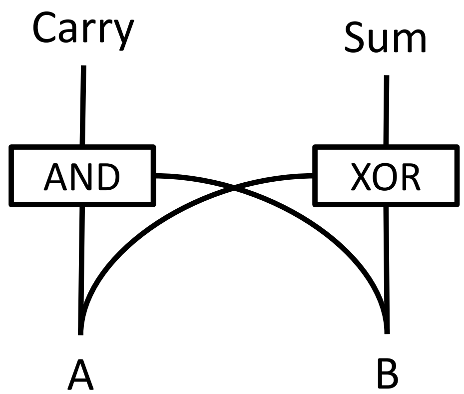
# Transistor => Logical Gate: XOR

**Truth Table:**

| A | B | A XOR B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

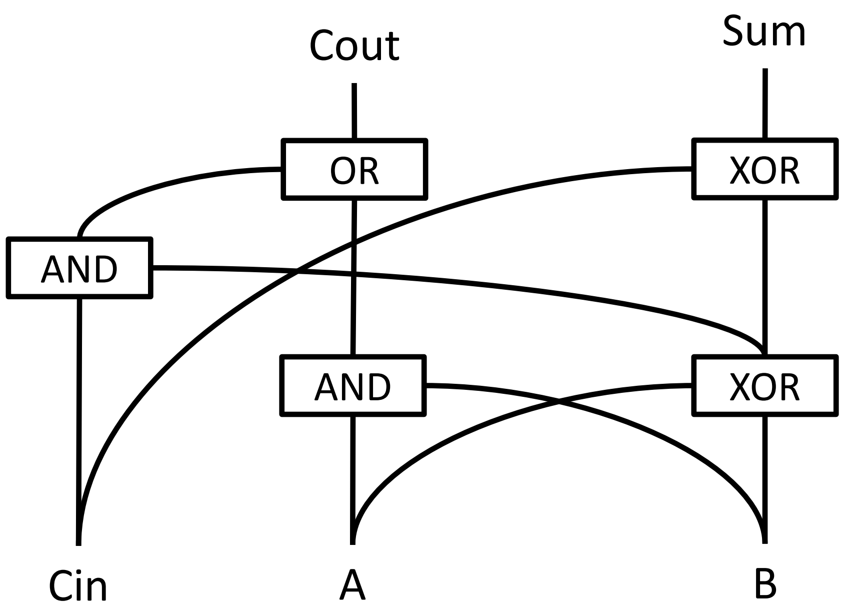


# Compose an Adder with Logical Gates




**Half Adder**

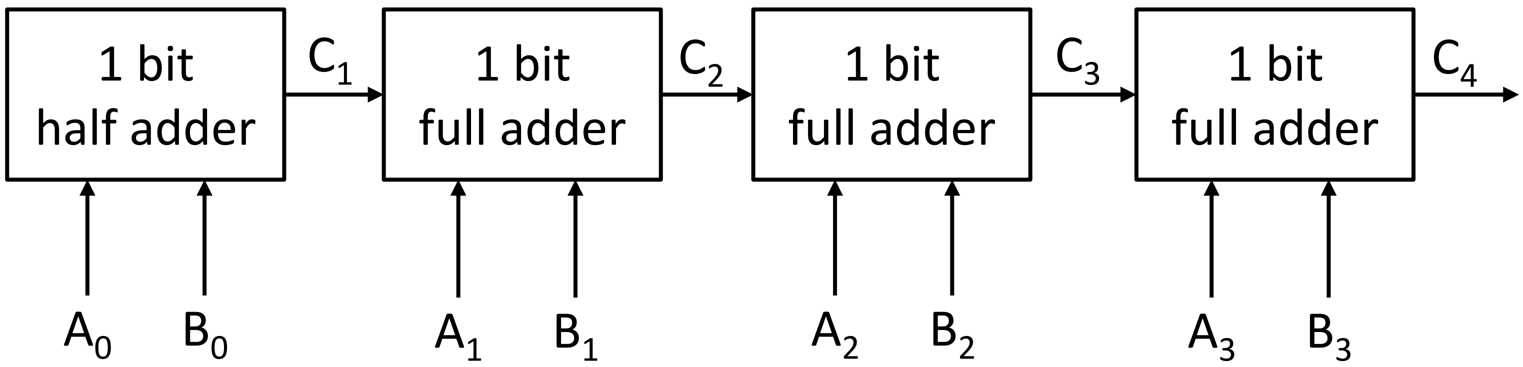
| Inputs | | Outputs | |
|---|---|---|---|
| **A** | **B** | **$C_{out}$** | **S** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |




**Full Adder (with carrier input)**

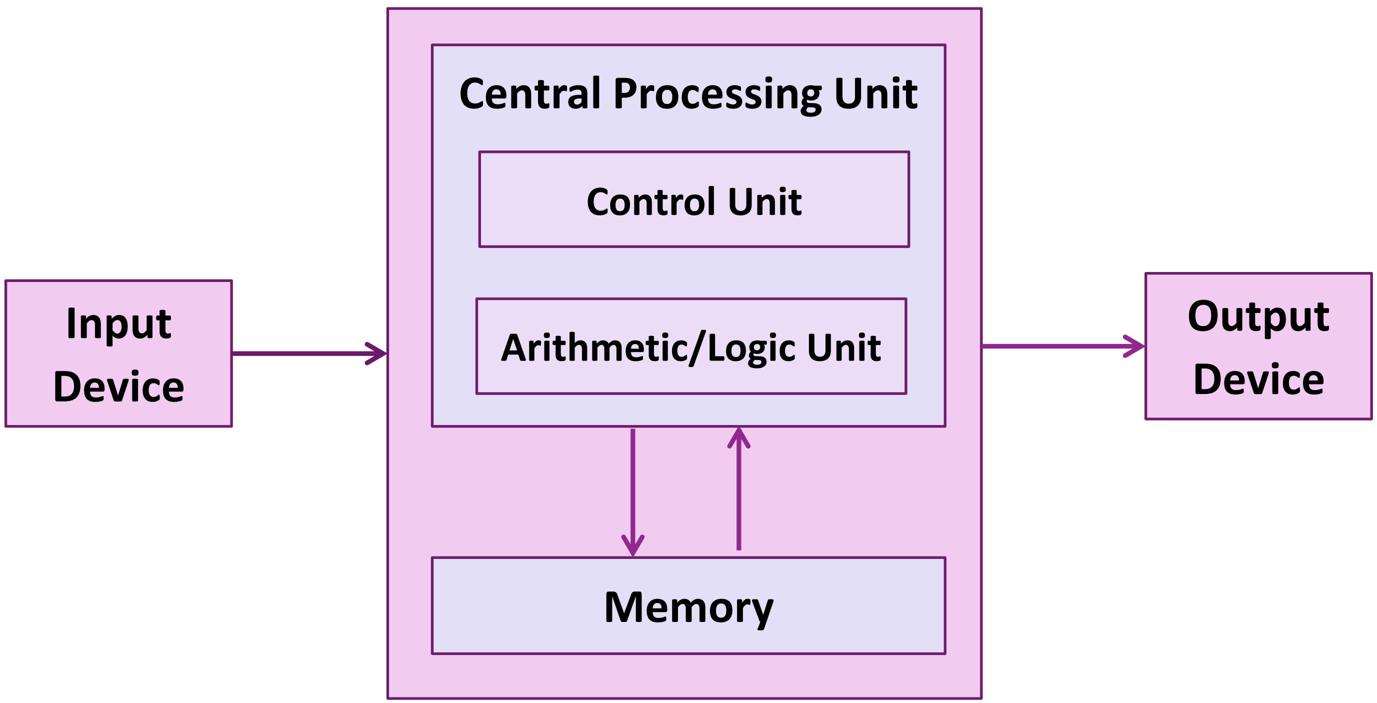
| Inputs | | | Outputs | |
|---|---|---|---|---|
| **A** | **B** | **$C_{in}$** | **$C_{out}$** | **S** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

# Add 4 Bits with Full Adder



# 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.



# Summary

- ❑ **All data are stored in computers as bits.**
  - ➢ Integers: Base-2 system.
  - ➢ Negative integers: Two's complement.
  - ➢ Decimals: Floating-point numbers with IEEE 754.
- ❑ **Computers are made of transistors that accept bit inputs.**
  - ➢ Composing of logic gates with transistors.
  - ➢ Composing arithmetic units with logical gates.