Boolean Algebra
Digital Logic
Computer Org. & Arch.
Computer Operating Systems

Objectives

In this lecture, you will learn basic concepts about

- Boolean algebra
- Digital logic gates
- Digital logic circuits
- Computer organization & architecture
- Computer operating systems

Boolean Algebra

Boolean Algebra

- Introduced by George Boole (1815-1864), British mathematician and philosopher
- In general, an algebra over finite sets of discrete values
- Variables can have values of either 0 (false) or 1 (true).
- Fundamental in today's digital computing.

Boolean Variables/Values

- A Boolean variable can be either 1 (true) or 0 (false).
- Boolean algebra is operating over these Boolean values.

Boolean Operators

- Relational operators (<, <=, ==, >, >=, and !=),
- Logic AND, OR, NOT
  - Notations used for AND is a period (.) or nothing like $x \cdot y$ xy
  - Notations used for OR is a plus sign (+) like $x + y$
  - Notations for NOT like $\sim x, \neg x, x', !x$

Operator Precedence

- Dictate which operation will perform first

NOT	*logical not*
<, <=	*less than, less than or equal to*
>, >=	*greater, greater than or equal to*
==, !=	*equal to, not equal to*
AND	*logical and*
OR	*logical or*

Precedenc
Order
Decreases

Boolean Expression

A Boolean expression can be the following

- A Boolean variable,
- A relational expression, or
- Any combination of Boolean variables, relational expressions, and Boolean expressions with logical operators AND, OR, NOT, and parentheses.

Boolean Functions

- A Boolean function maps a set of Boolean variables to a Boolean value.
- Consider $f(A,B) = A + B$

- Boolean variables A, and B can only hold 0 or 1. The operator "+" denotes an OR.
- So, the Boolean function represents logical OR.

$$f(0,0) = 0$$

$$f(0,1) = 1 \qquad f(1,0) = 1 \qquad f(1,1) = 1$$

Truth Table

- A visual representation of a Boolean function.
- Input variables to the left, and an output to the rightmost
- All possible input values (combinations) are listed.
- Normal ing order.

A	B	f(A,B)
0	0	0
0	1	1
1	0	1
1	1	1

Laws of Boolean Algebra

- Like regular algebra, there are laws in the Boolean algebra that ease the computation, or simplify Boolean expressions.
- Some of the laws from regular algebra may NOT apply to Boolean algebra.
- Examples:
  - Distribution law

    $x(y+z) = xy + xz$

    $x + (yz) \equiv (x + y)(x + z)$
  - Inverse law

    $x + \bar{x} = true$

    $x\bar{x} = false$
  - Commutative law

    $x + y = y + x$

    $xy = yx$

Laws of Boolean Algebra

- More example laws:
  - Associative law

    $x + y + z = (x + y) + z = x + (y + z)$

    $xyz = (xy)z = x(yz)$

  - Identity law

    $x + x = x$

    $xx = x$

  - Redundance law

    $x + xy = x$

    $x(x + y) = x$

  - False law

    $0 + x = x$

    $0x = 0$

  - True law

    $1 + x = 1$

    $1x = x$

  - De Morgan's law

    $\neg(xy) = \neg x + \neg y$

    $\neg(x + y) = \neg x \neg y$

Boolean Expression Simplification

- Consider A+AB
- Actually, A+AB=A

- Consider $\bar{A}BC + A\bar{B}C$
- Actually, $\bar{A}BC + A\bar{B}C = \bar{A}C$

- We can apply Boolean Algebra laws to simplify expressions.
  - Key to design efficient digital circuits

Digital Logic Gates

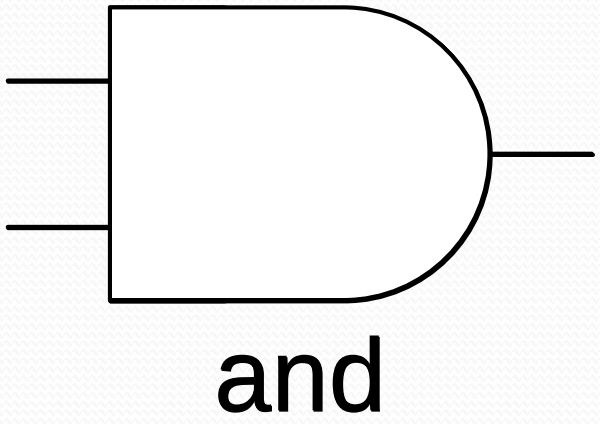
2-Input Boolean Functions

A	B	f(A,B)
0	0	?
0	1	?
1	0	?
1	1	?

AND

- Boolean function *f(A, B)* is true only if both *A* and *B* are true. *f(A, B)=AB*

A	*B*	*f(A,B)*
0	0	0
0	1	0
1	0	0
1	1	1

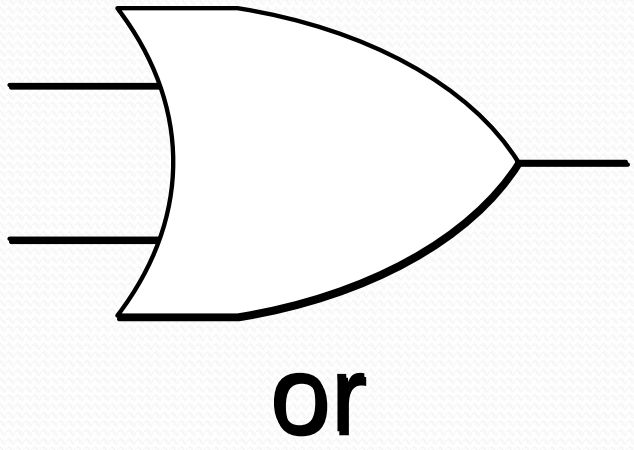


- AND function is also called Boolean product.

OR

- Boolean function *f(A, B)* is false only if both *A* and *B* are false. *f(A, B)=A+B*

A	*B*	*f(A,B)*
0	0	0
0	1	1
1	0	1
1	1	1



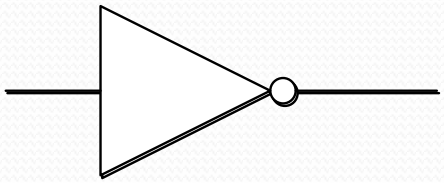
- OR function is also called Boolean sum.

NOT

- Boolean function *f(A)* is true only if *A* is false.

$$f(A) = \overline{A}$$

A	*f(A)*
0	1
1	0



not

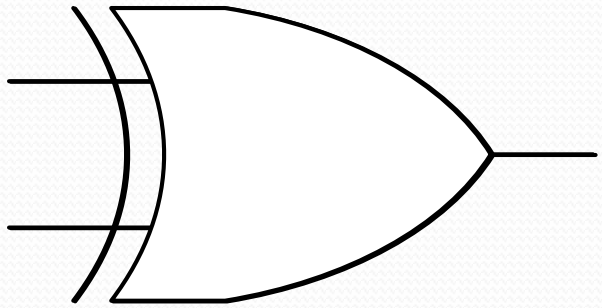
- NOT function is also called Boolean inverse.

Exclusive OR (XOR)

- Boolean function *f(A, B)* is true only if one of *A* and *B* is true, not both

$$f(A,B) = A \oplus B$$

A	*B*	*f(A,B)*
0	0	0
0	1	1
1	0	1
1	1	0

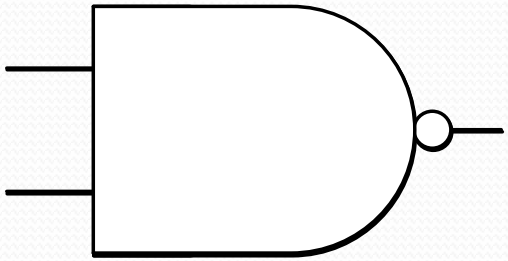


NAND

- Not AND

$$f(A,B) = \overline{AB}$$

A	*B*	*f(A,B)*
0	0	1
0	1	1
1	0	1
1	1	0



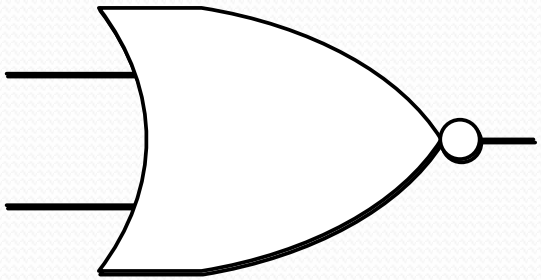
nand

NOR

- Not OR

$$f(A,B) = \overline{A + B}$$

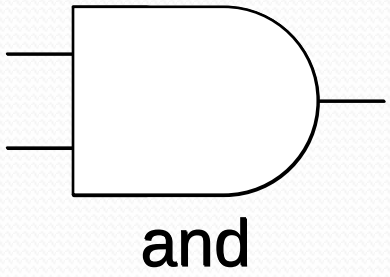
A	*B*	*f(A,B)*
0	0	1
0	1	0
1	0	0
1	1	0

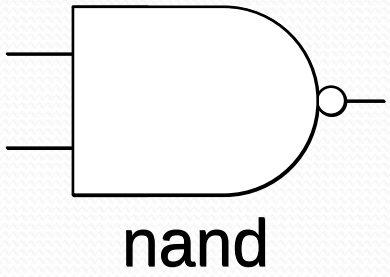


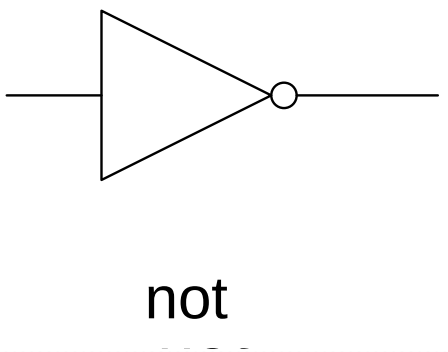
nor

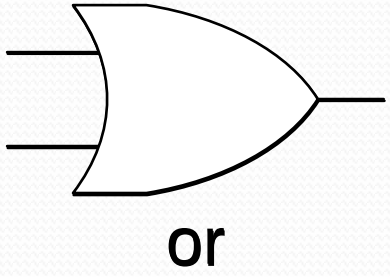
Gate Symbols

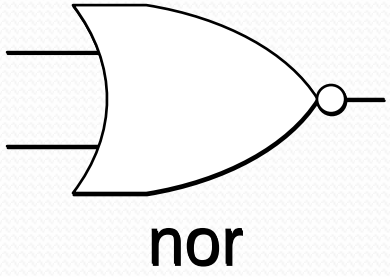
- Symbols that represent electronic circuits for gates

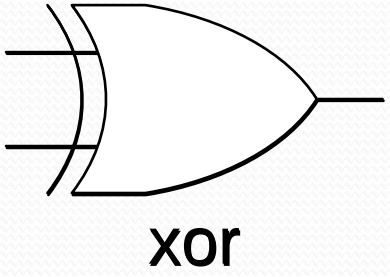






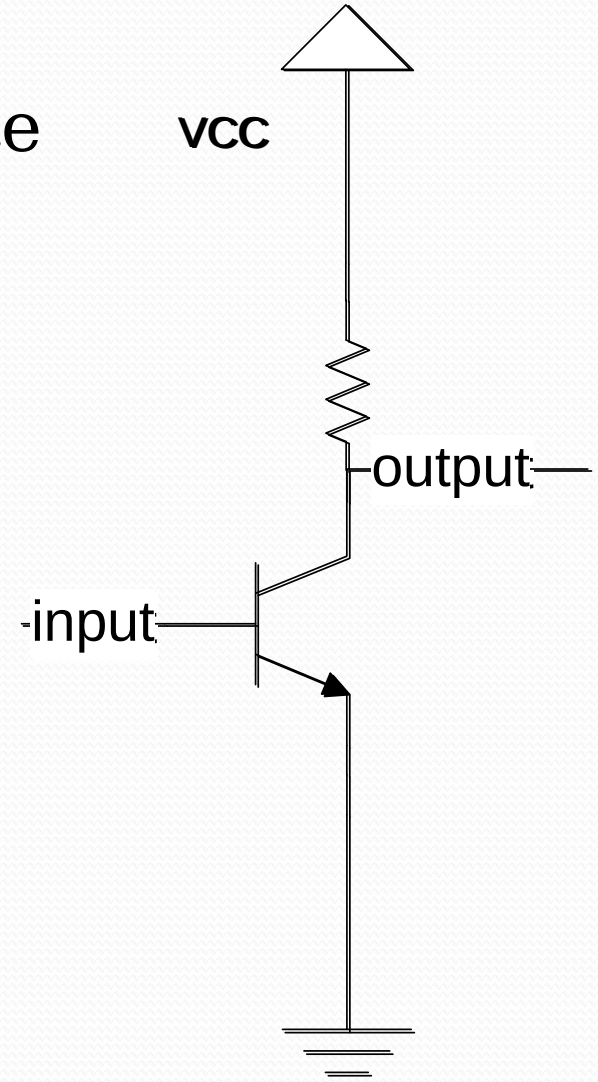






Transistor Implementation

- Transistors are used to implement logic gates.
- E.g., NOT gate

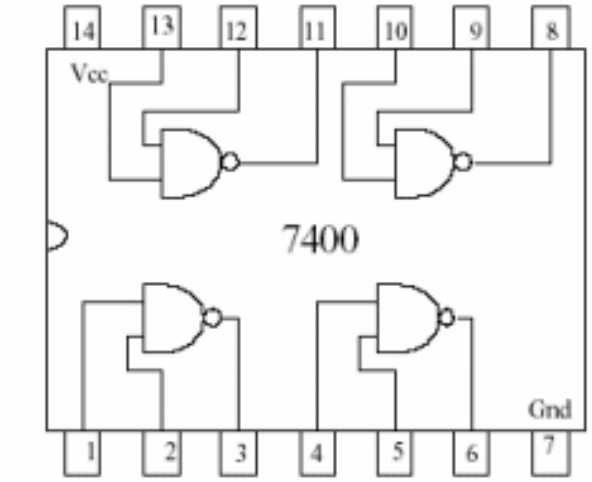


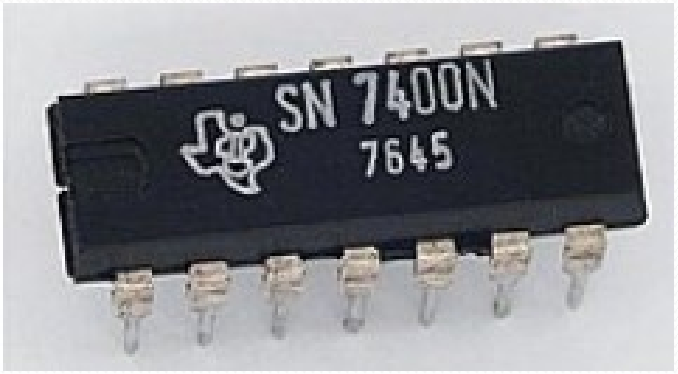
Characteristics of Gates

- Outputs are subject to inputs only.
- Switching time form on to off, or off back to on, takes a few nanoseconds in transistors.
- Propagation delay from inputs to outputs refers to the time between the inputs change and the new reflected outputs.
- Gate delay refers to the propagation delay for a gate.

Completeness

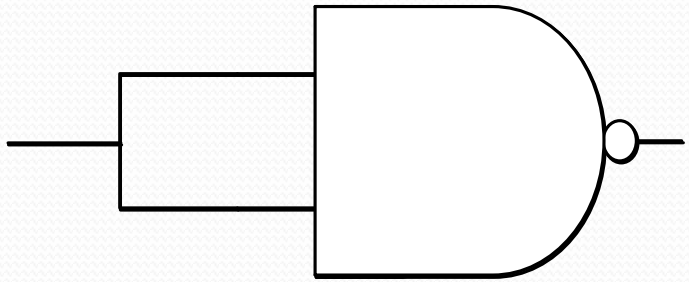
- Any Boolean function can be computed using one type of computationally complete gates.
- NOR and NAND are computationally complete gates.
- They are easily to be manufactured than others.
- Normally, you have a pack of NANDs in an IC. You want to reuse them as many as possible to sa



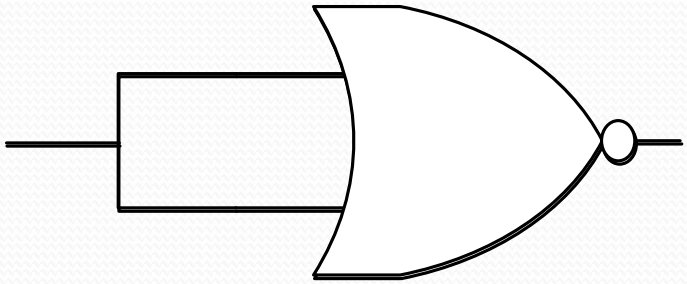


Equivalent NOT

$$\overline{A} = \overline{A + A} = \overline{AA}$$



nand

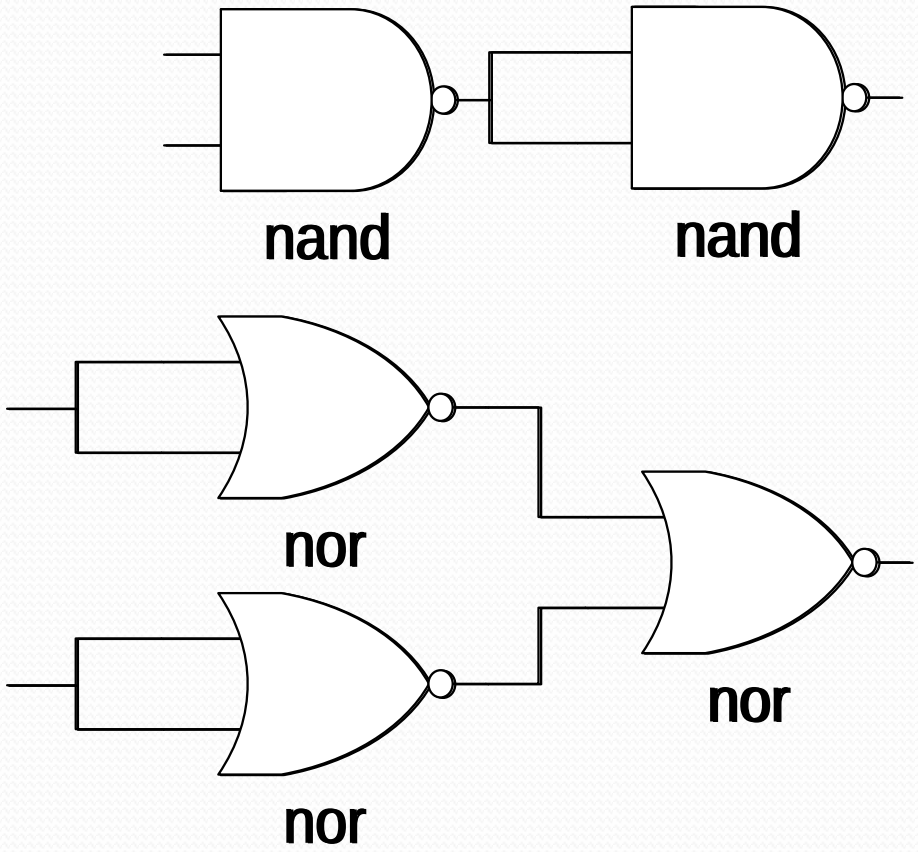


nor

Equivalent AND

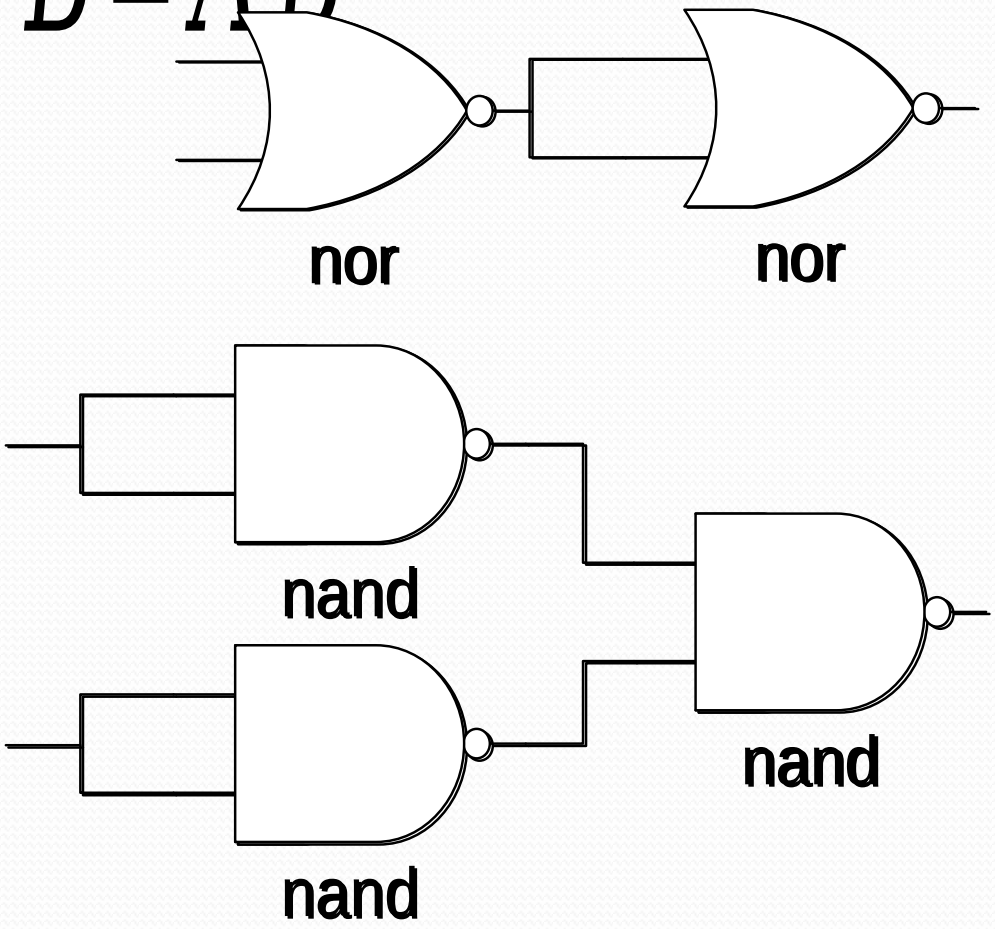
$$AB = \overline{\overline{AB}}$$

$$AB = \overline{\overline{AB}} = \overline{\overline{A} + \overline{B}}$$



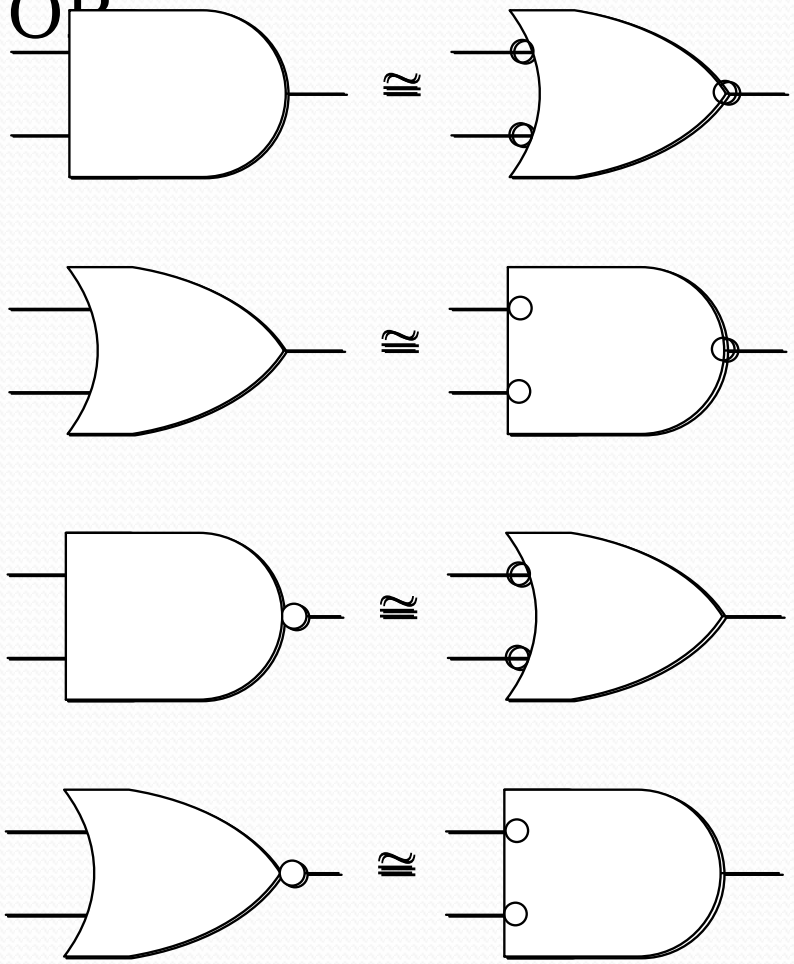
Equivalent OR

$$A + B = A + B = \overline{\overline{A}\,\overline{B}}$$



Logic Gate Equivalence

- Invert the inputs
- Exchange AND and OR
- Invert the output



Digital Logic Circuits

Digital Computation

- Boolean algebra deals with operations over binary digits and generates binary results.
- Arithmetic operations can be described by truth tables.
- Truth tables can be converted to Boolean expressions.
- Digital circuits can be built based on Boolean expressions (sum or product).

Combinational Logic

- The output of a logic circuit depends only on its inputs.
- E.g., the full adder
- It is called combinational logic, combinational circuit, or combinatorial logic.
- Combinational logic does not remember its previous state.

Combinational Logic Design

- Find out what are inputs and what are outputs
- Build truth tables
- Derive Boolean functions
- Simplify Boolean functions
- Create the circuit

Example: Half Adder

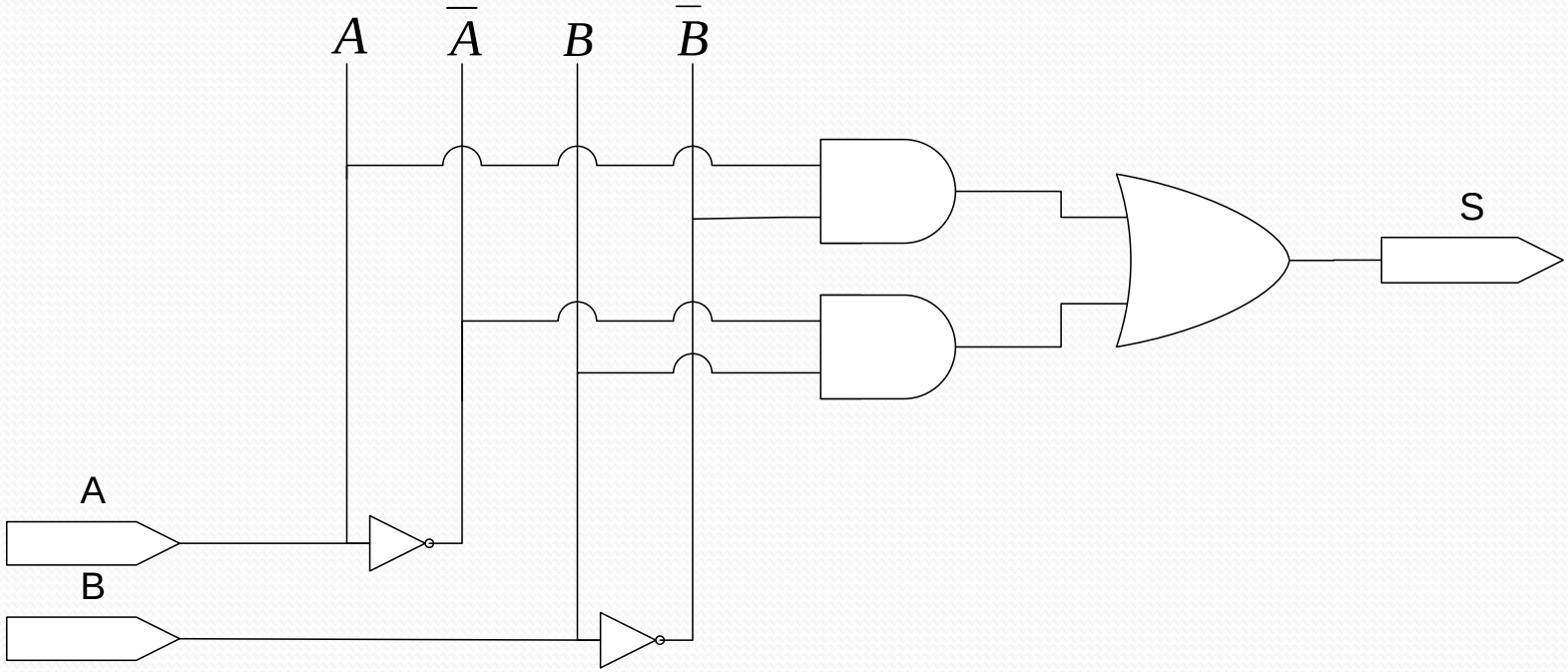
- Truth table and Boolean function

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

$$S = \overline{A}B + A\overline{B}$$

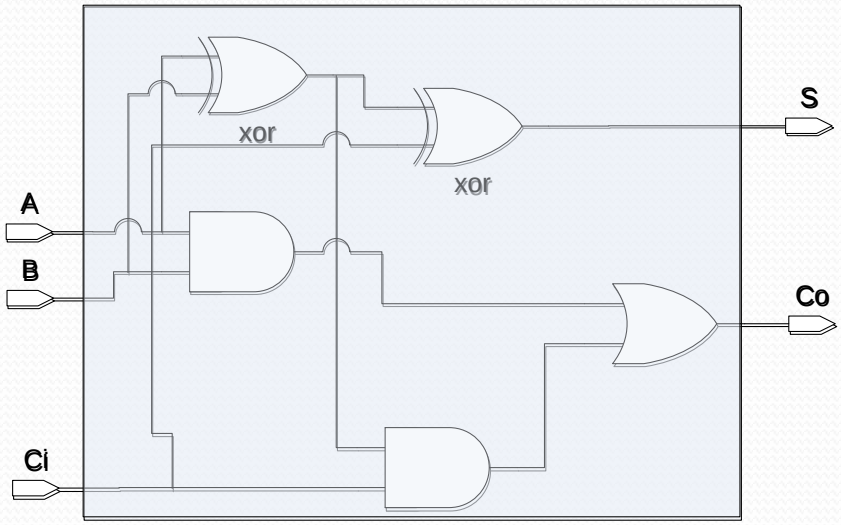
Example: Half Adder

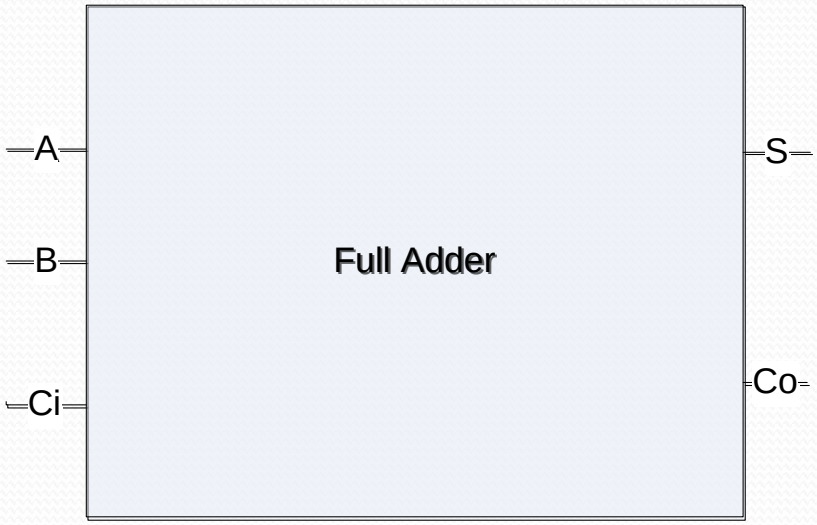
- Logic circuit



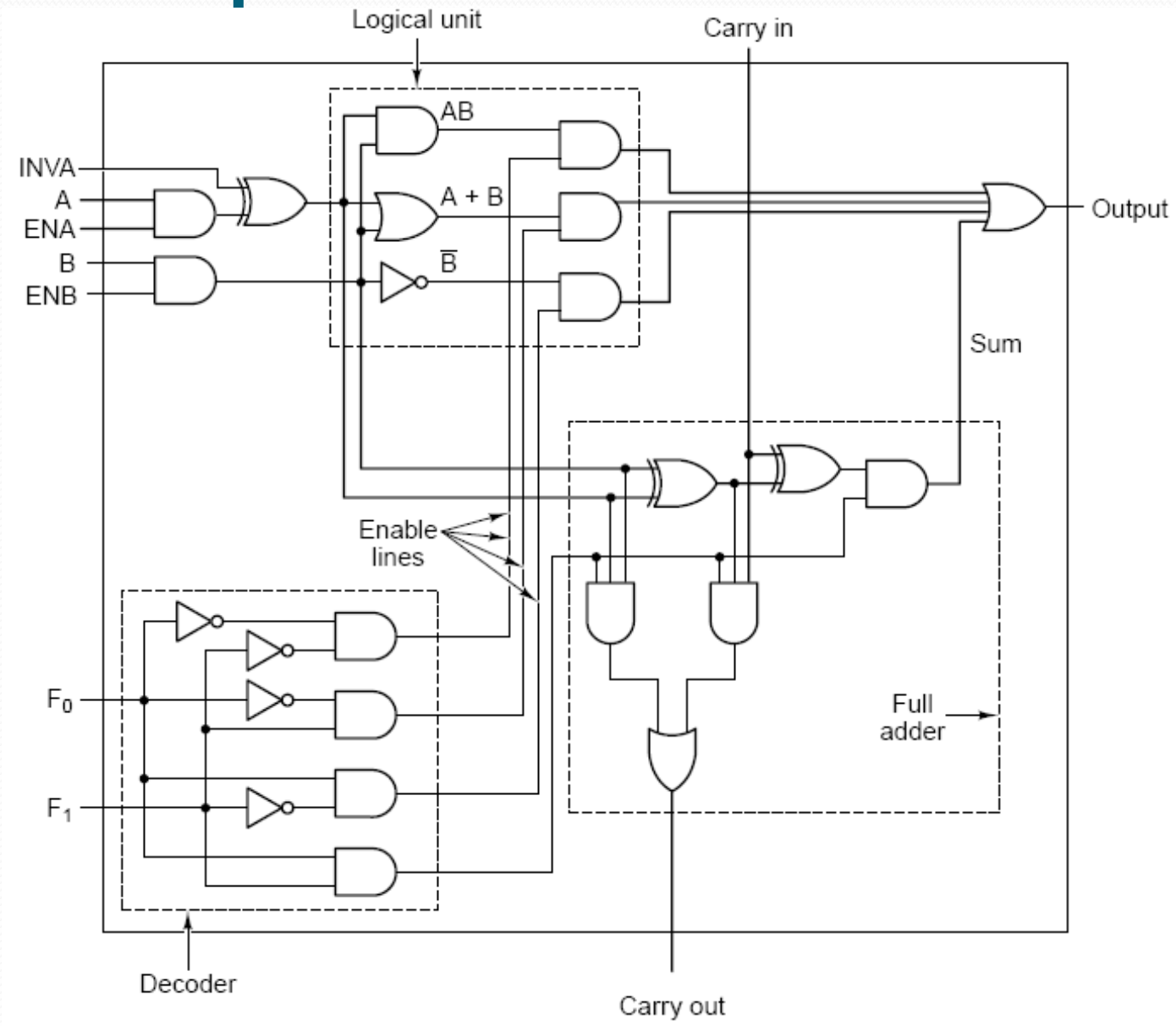
$$S = \overline{A}B + A\overline{B}$$

Example: Full Adder





Example: 1-Bit ALU

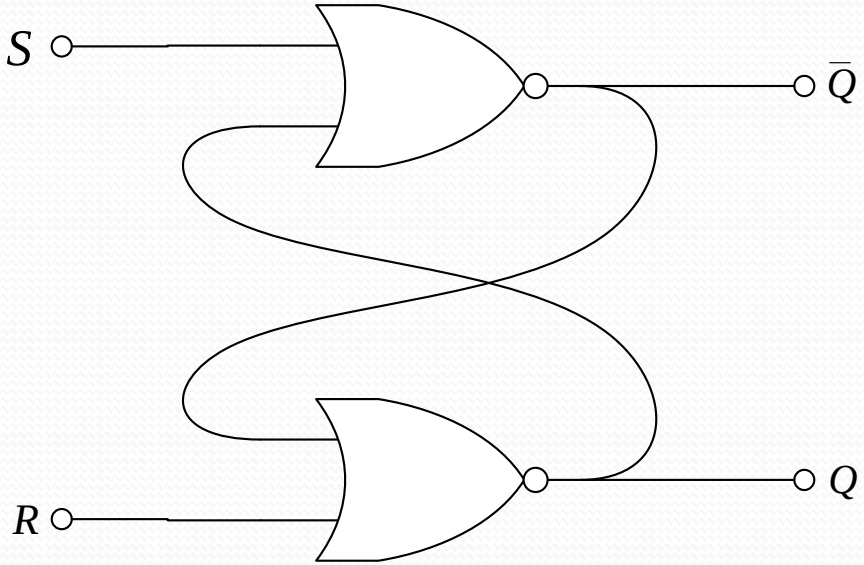


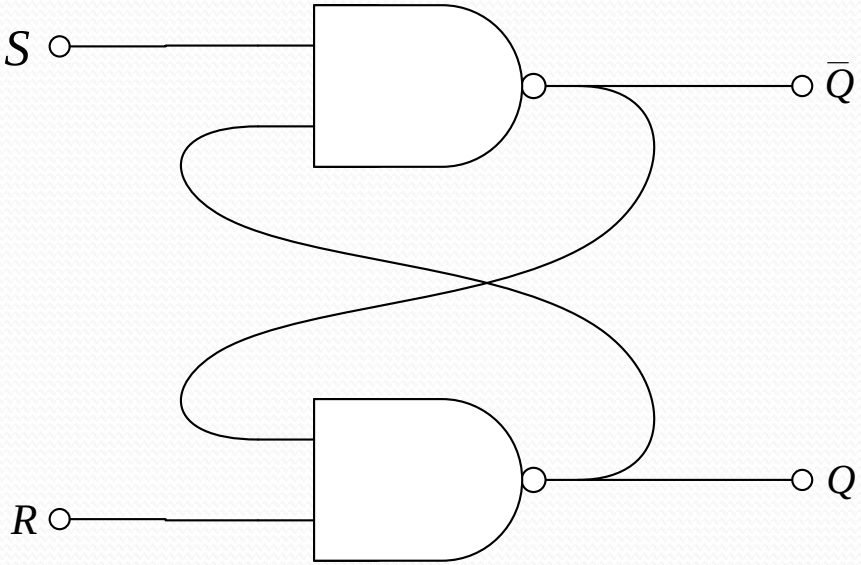
Sequential Logic

- Outputs of sequential logic are based on inputs and its current internal state.
- Usually, sequential logic circuit involves clock, which triggers state transition.
- Sequential logic: Latch, Flip-Flop, Registers, Memory, etc.

Example: Latch

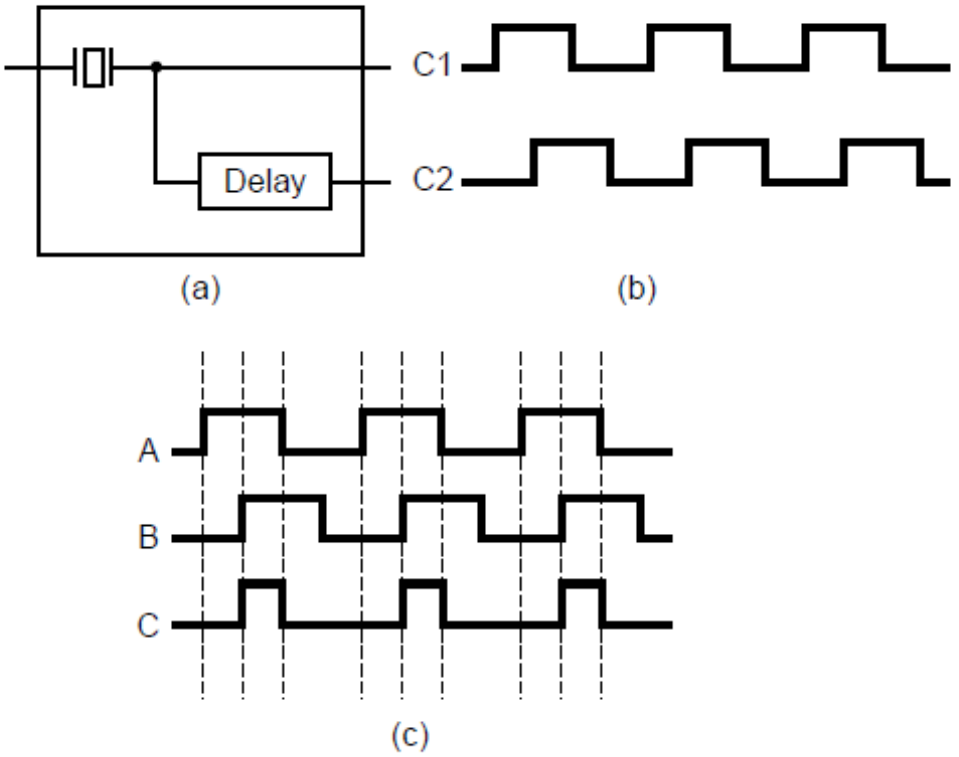
- The most basic sequential logic
- Can keep (or, remember) one bit of information
- No clock involved





Clocks

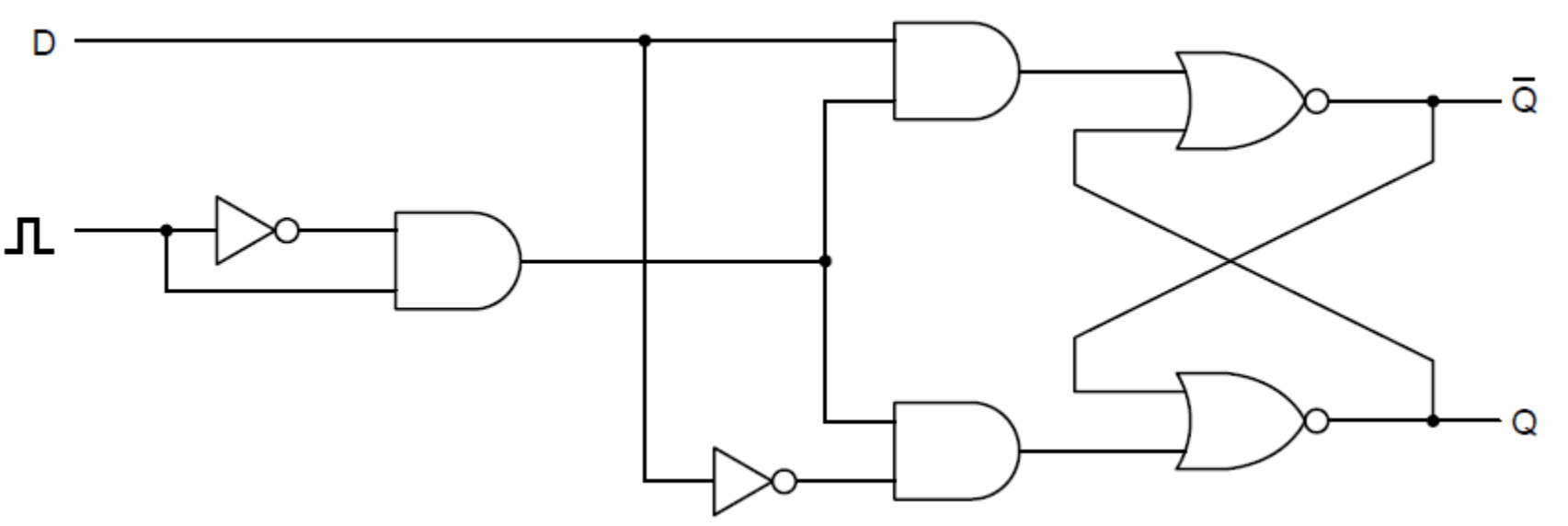
- Every computer contains at least one clock that:
  - Regulates how quickly instructions can be executed
  - Synchronizes the activities of its components.
- A fixed number of clock cycles are required to carry out each data movement or computational operation.

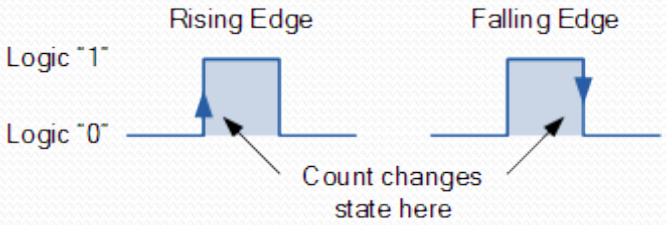


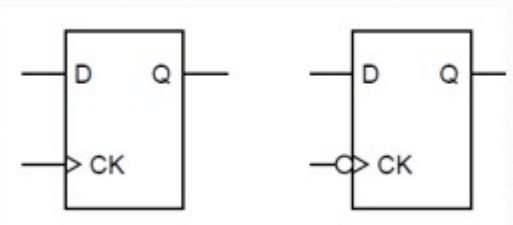
(a) A clock. (b) The timing diagram for the clock.
(c) Generation of an asymmetric clock.

Example: D Flip-Flop

- A flip-flop is a latch triggered by a rising edge or a falling edge of the clock.





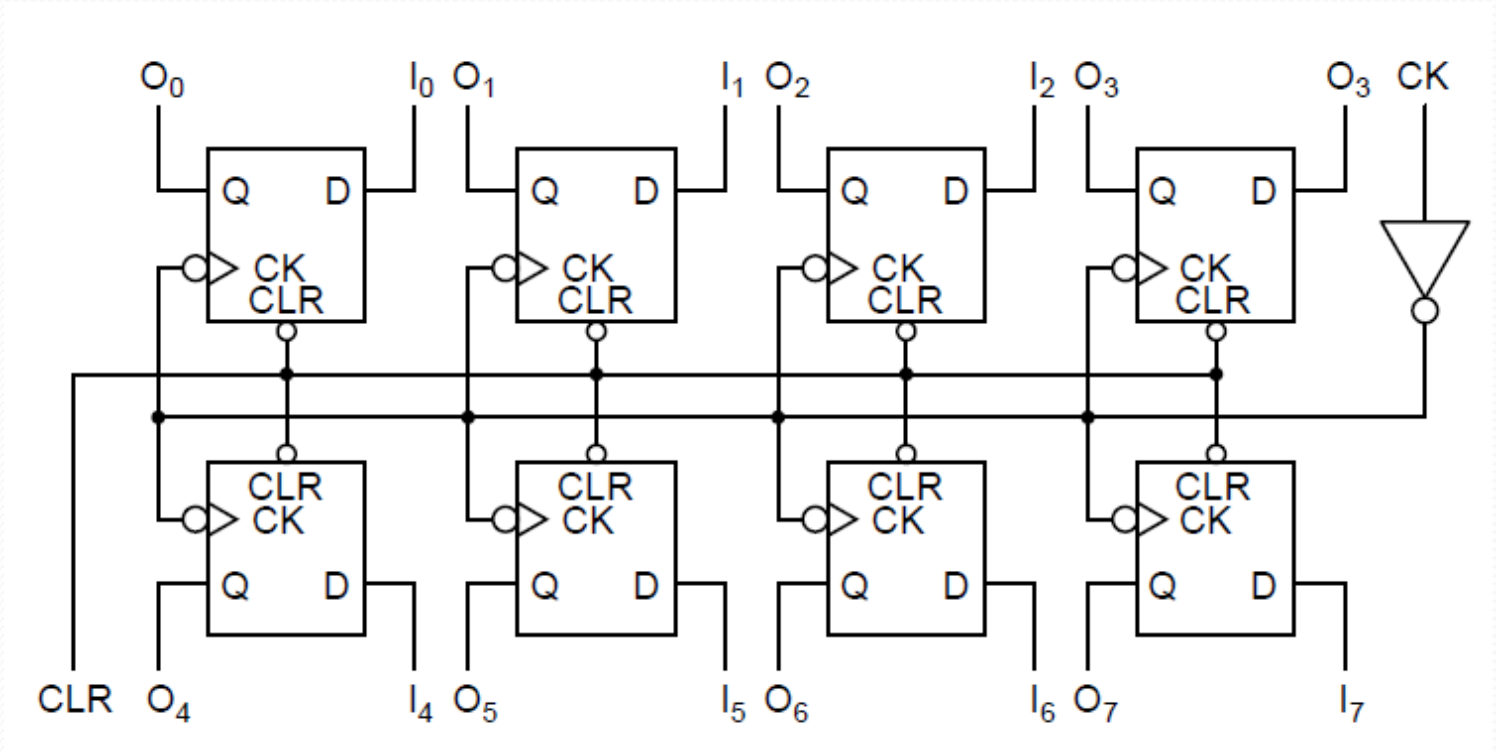


Symbols for D flip-flops

Example: Registers

- Registers are groups of flip-flops, where each flip-flop is capable of storing one bit of information.
- Normally, 8-bit/16-bit/32-bit/64-bit registers
- Store data in CPU for computation
- Can also be used to build memories

Example: Registers



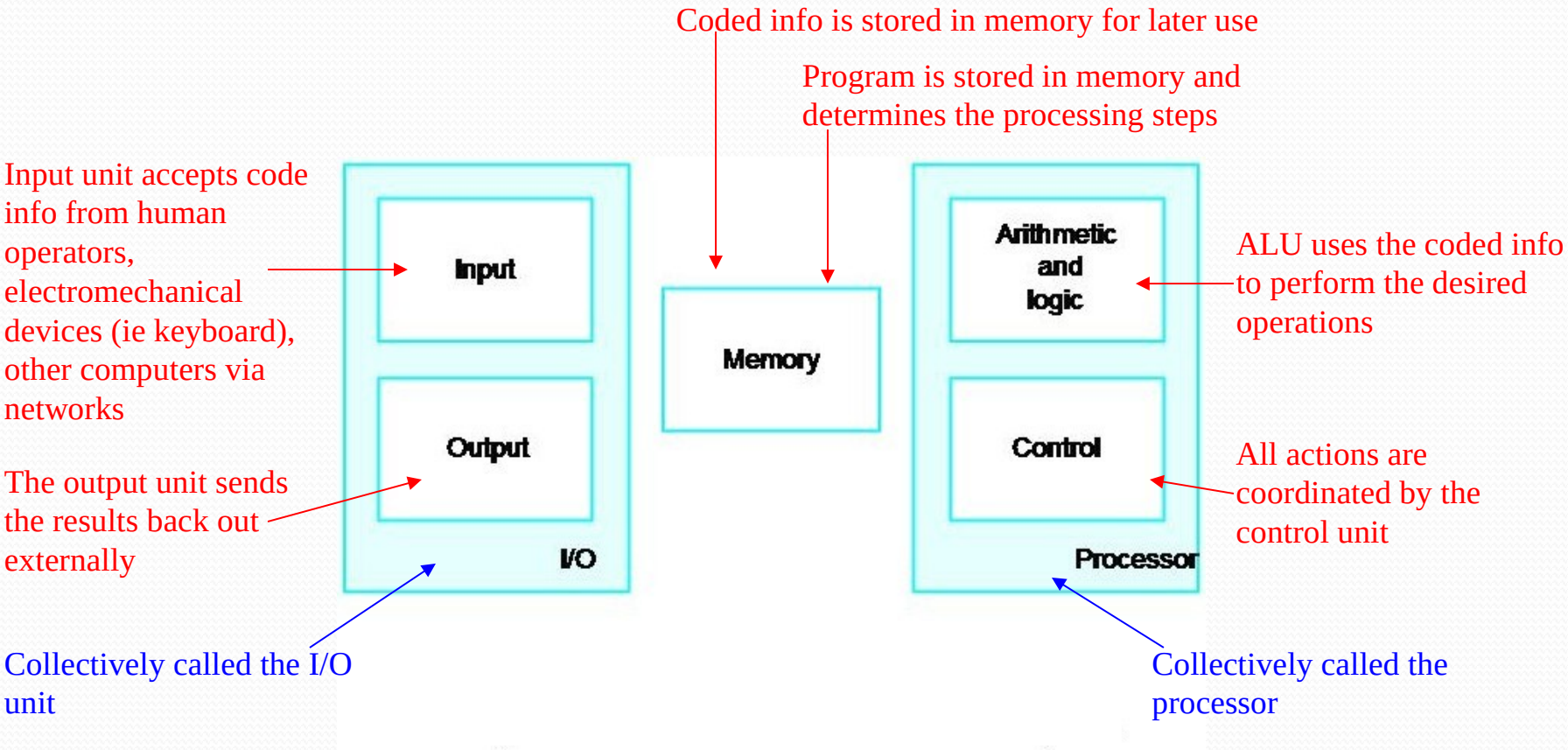
An 8-bit register constructed from single-bit flip-flops

Density of Integrated Circuit

- SSI
  - Small-scale integration packages gates
  - Dozens of gates per chip
- MSI
  - Medium-scale integration packages functions, *e.g. adders, multiplexors*
  - Hundreds of gates per chip
- LSI
  - Large-scale integration packages complete units, e.g. a calculator chip
  - Thousands of gates per chip
- VLSI
  - Very large-scale integration packages complex units with millions of transistors.

Computer Organization & Architecture

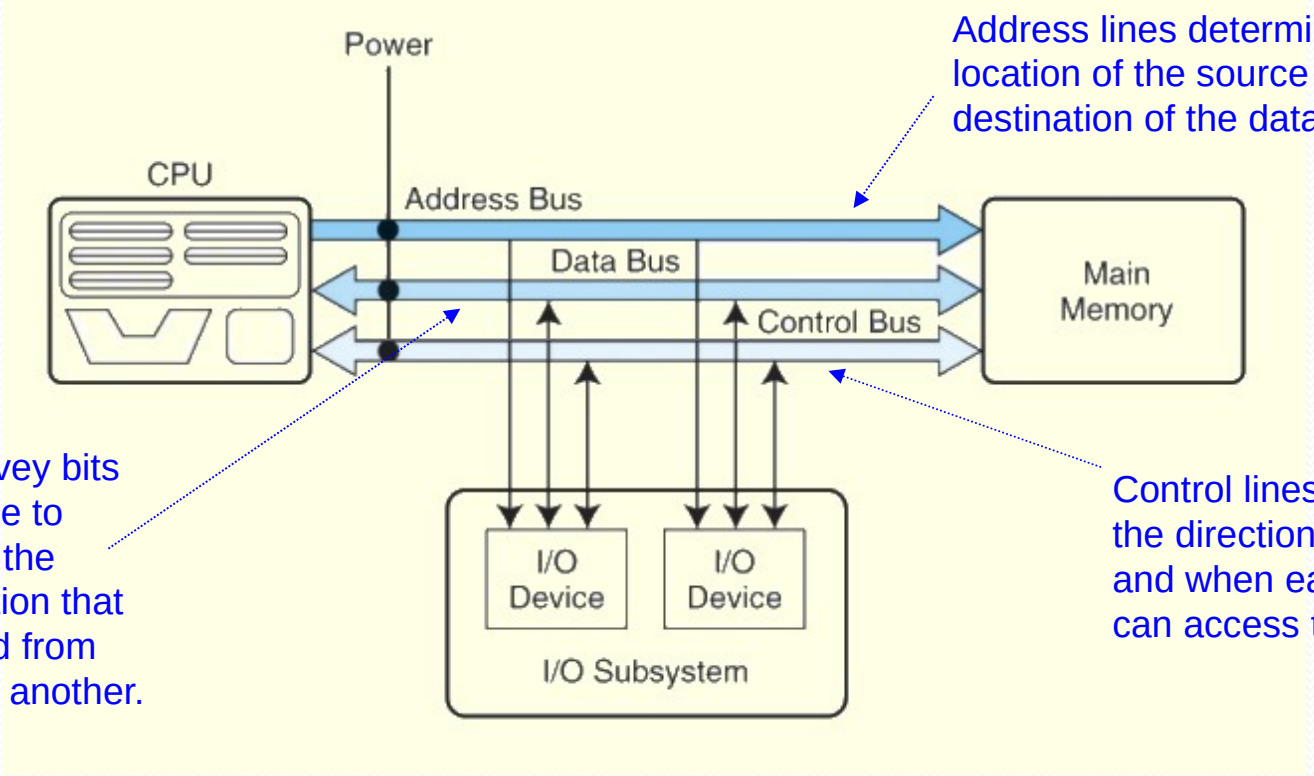
Basic Structure of Computers



The von Neumann Architecture

The Bus

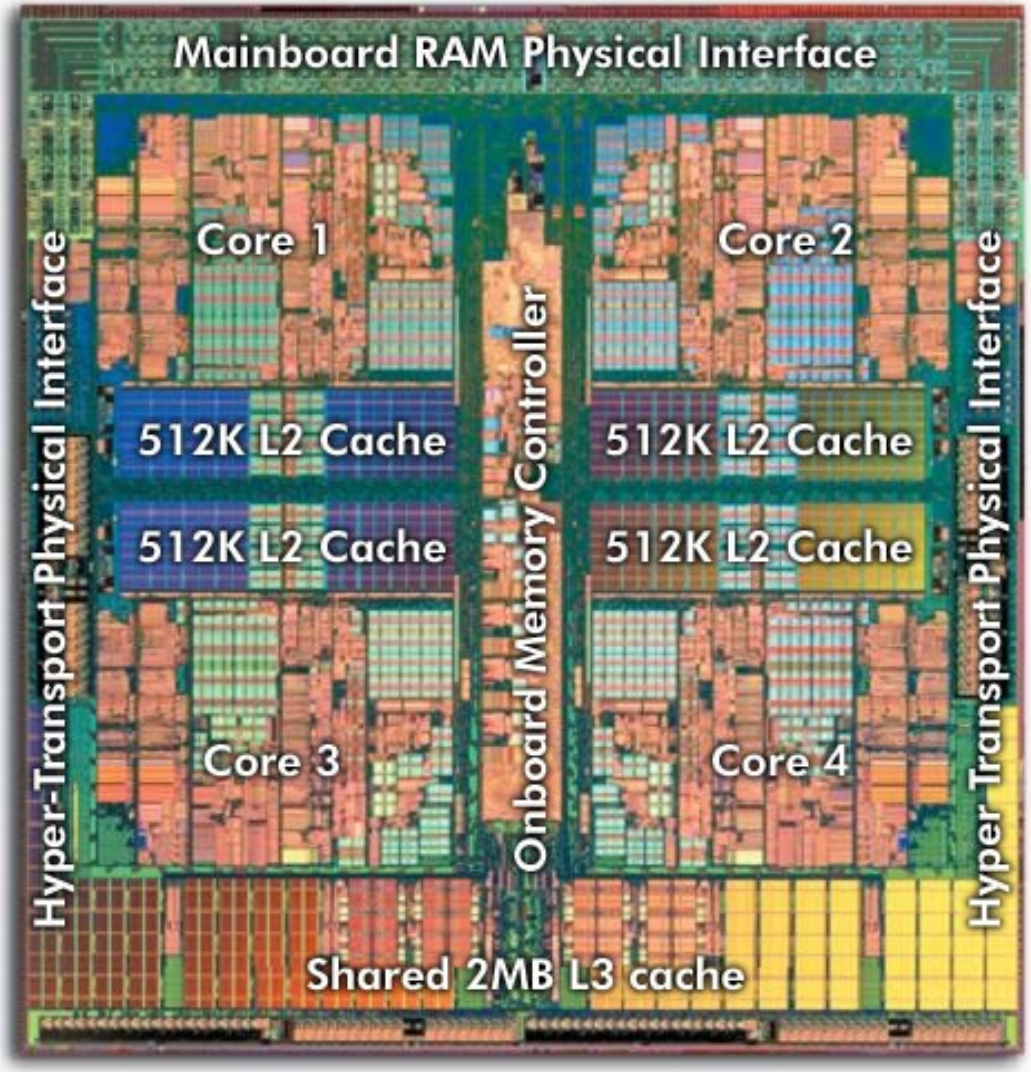
- Buses consist of data lines, control lines, and address lines.



CPU

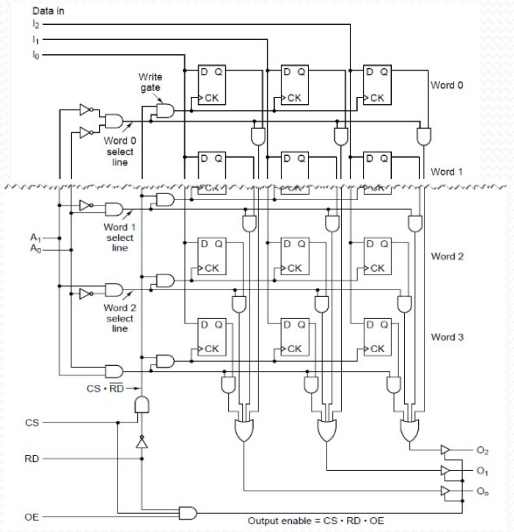
Outside Appearance (Top & Bottom)

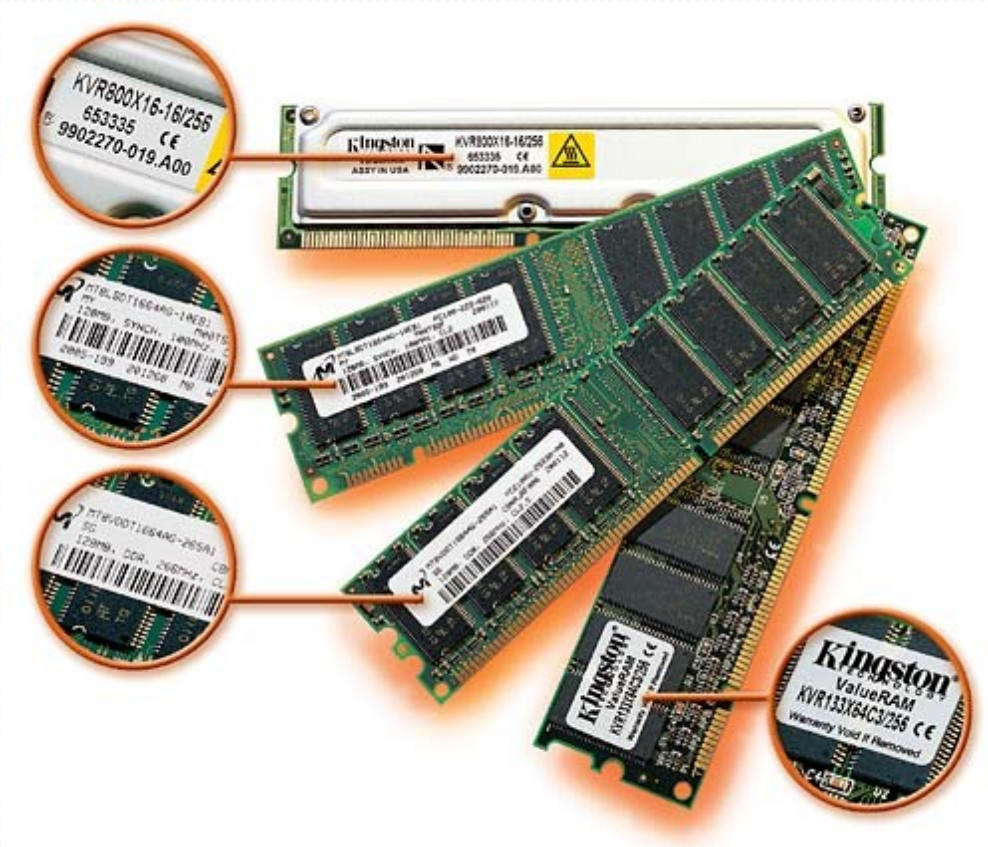
Inside CPU

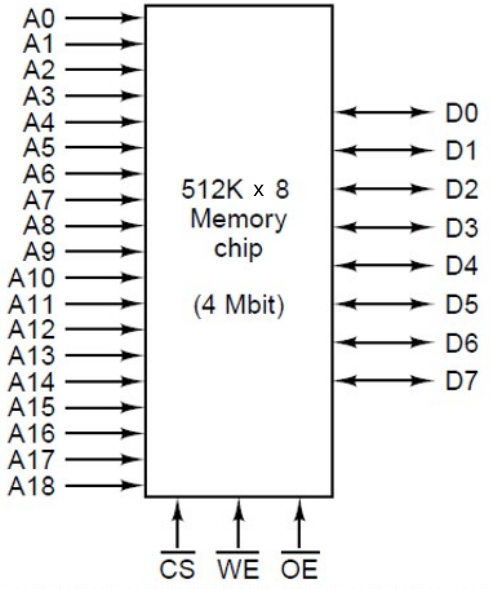


Memory

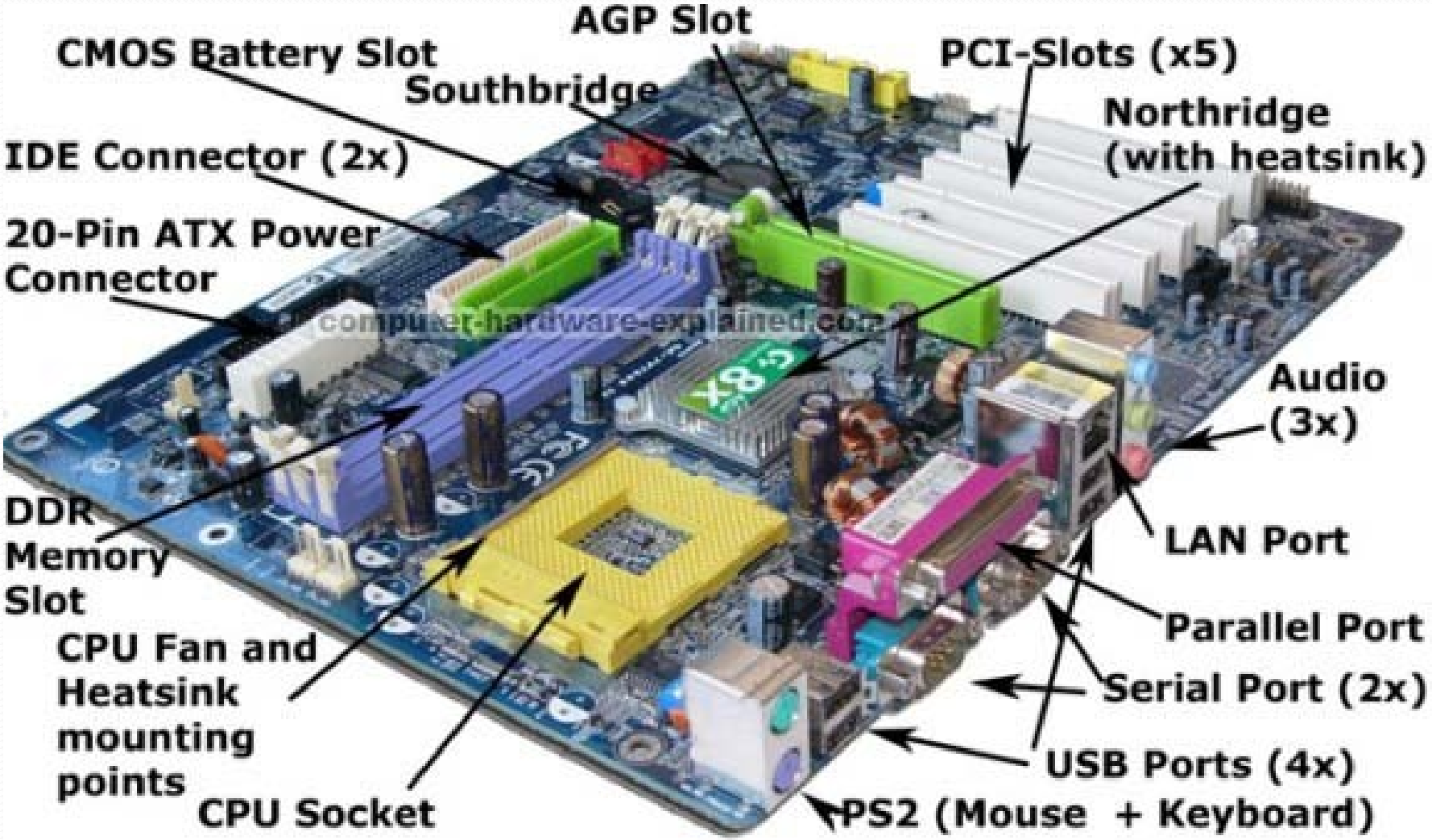
- Computer memory consists of a linear array of addressable storage cells that are similar to registers.







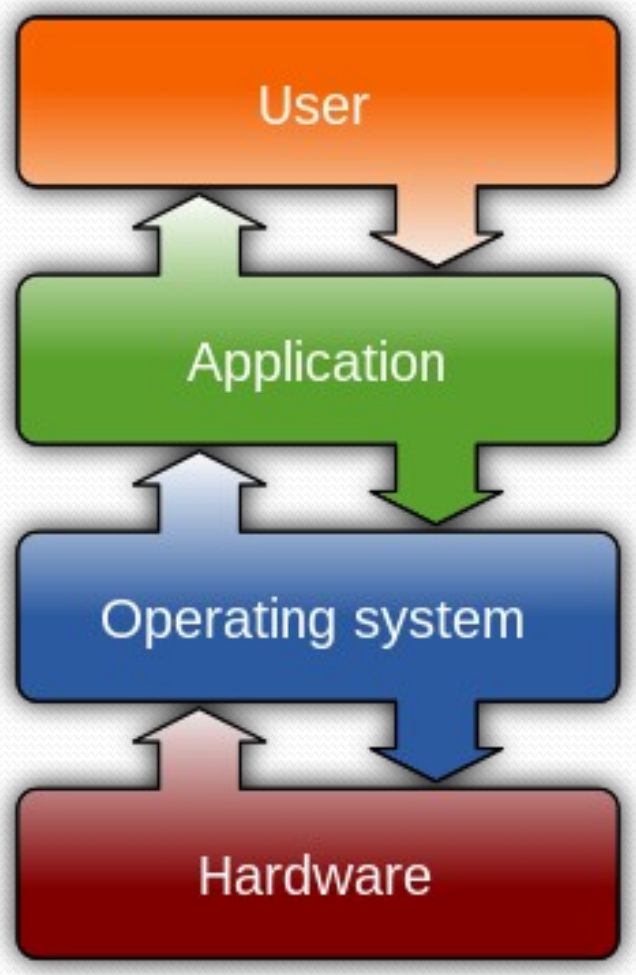
Motherboard



Computer Operating Systems

Definition

- An Operating System (OS) is low-level system software that
  - manages computer hardware and software resources;
  - provides common services for computer programs to interact with computer hardware and other programs.



https://en.wikipedia.org/wiki/File:Operating_system_placement.sv

OS Services

- Program execution
  - OS provides an environment where a user can run programs without having to worry about memory allocation or CPU scheduling.
- I/O operations
  - OS helps a user/program utilize input/output devices without knowing hardware details of the I/O devices.
- File system manipulation
  - OS provides an interface for a user/program to access and manage files and directorys.
- Communication
  - OS facilitates communication between processes to exchange information, whether the processes are running on the same computer or on different computers.

OS Services

- Error detection
  - OS constantly checks for errors in CPU, I/O, memory, etc., and takes actions to ensure correct and consistent computing.
- Resource allocation
  - OS manages resources (memory, CPU, file storage, etc.) using schedulers for multi-user or multi-tasking environments.
- Protection
  - OS ensures that all access to system resources is controlled and authenticated.

Common Operating Systems

- Linux
- Microsoft Windows
- Apple macOS
- Google Android
- Apple iOS

Summary

- Boolean algebra
  - Boolean variable, operators, function, truth table
- Digital logic gates
  - Six basic gates, symbols, and truth tables
- Digital logic circuits
  - Combinational logic, sequential logic
- Computer Organization & Architecture
  - The von Neumann architecture
- Computer Operating Systems
  - Functions, services

Thank You!