信息科学与技术学院
School of Information Science and Technology

# CS 110
# Computer Architecture
# Digital Circuits and Systems

**Instructors:**
**Siting Liu & Chundong Wang**
Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2024/index.html
**School of Information Science and Technology (SIST)**
**ShanghaiTech University**

2024/3/26

# Administratives

- Be on time! Only those submissions before ddl will receive marks, otherwise you got 0. So START EARLY!
- You are responsible for your submissions. Make sure that we are able to mark it. Do follow the instructions for each assignments.
- Lab 4 available, please prepare in advance!
- Proj1.1 released, individual work, ddl April 8th
- Discussion this week on CALL/RISC-V, useful for Proj1.1, covered by TA Chen Suting at teaching center 301.

# Outline

- **Digital system**
- **Combinational logics**
  - From transistors to basic logic gates
  - From logic gates to combinational circuits
    - Boolean algebra
    - Boolean expression
    - Truth table
- State elements
- Useful building blocks

# Where are we?

**High Level Language Program (e.g., C)**

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler

Assembly Language Program (e.g., RISC-V)

```
lw   t0, 0(s2)
lw   t1, 4(s2)
sw   t1, 0(s2)
sw   t0, 4(s2)
```

**Assembler**

Machine Language Program (RISC-V)

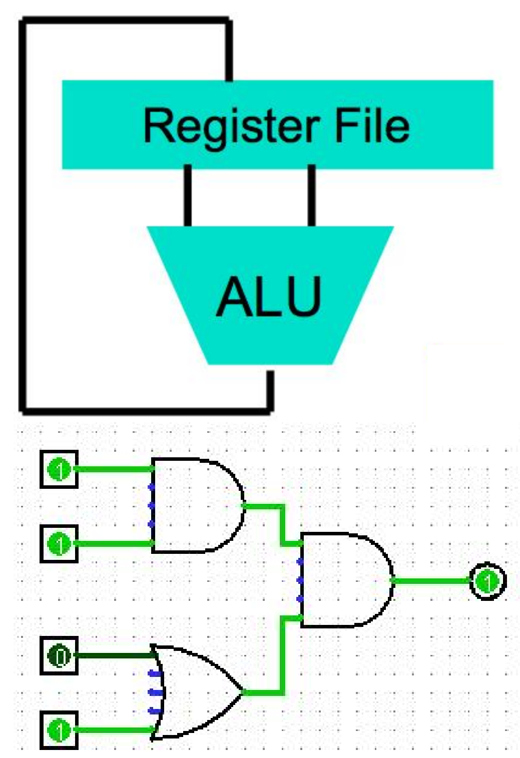
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)



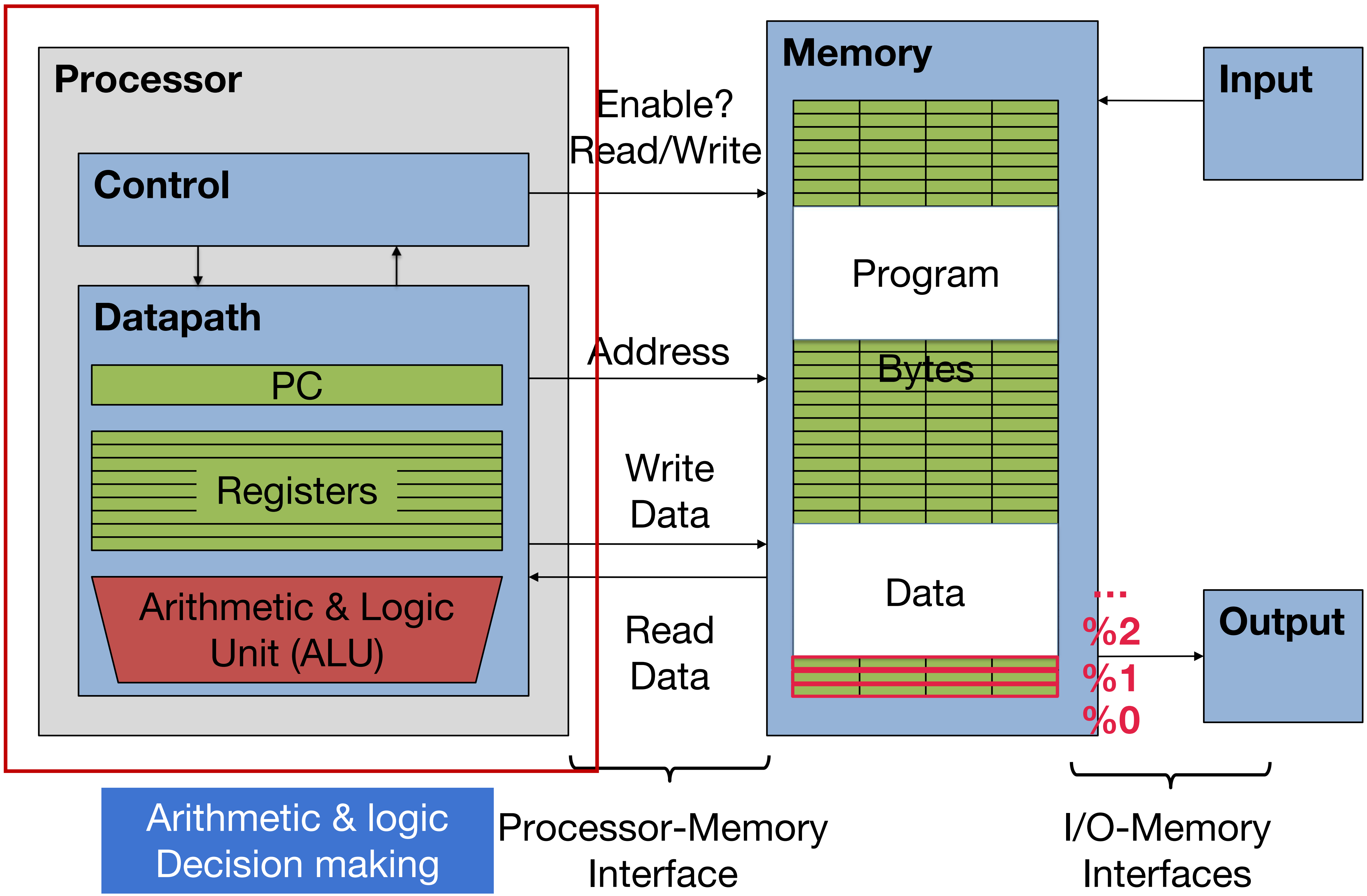

*We are here!*

*Bottom-up*

# Hardware (HW) Design

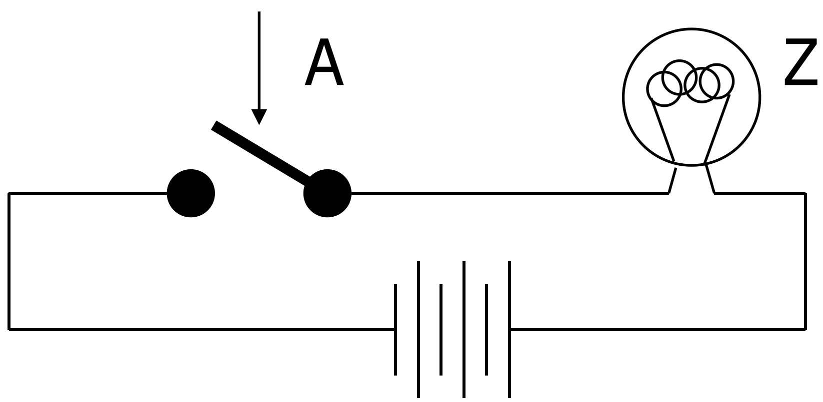
- Next several weeks: how a modern processor is built, starting with basic elements (transistors) as building blocks
- Why study hardware design?
  - Understand capabilities and limitations of HW in general and processors in particular
  - What processors can do fast and what they can't do fast (avoid slow things if you want your code to run fast!)
  - Background for more in-depth HW courses (Digital circuit/VLSI/AI computing system, etc.)
  - There is only so much you can do with standard processors: you may need to design own custom HW for extra performance
    - Even some commercial processors today have customizable hardware!
    - E.g. Google Tensor Processing Unit (TPU)

# Components of Computers

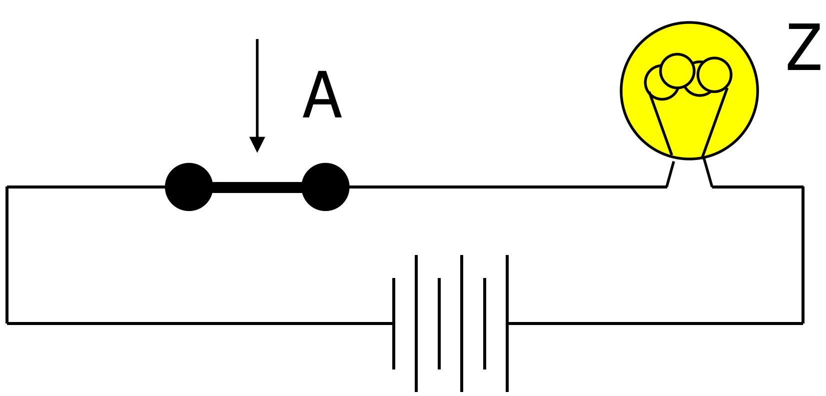


# Switches: Basic Element of Physical Implementations

- Implementing a simple circuit (arrow shows action if wire changes to “1” or is *asserted*):




*Off*-switch (if A is “0” or unasserted) turns-off light bulb (Z)

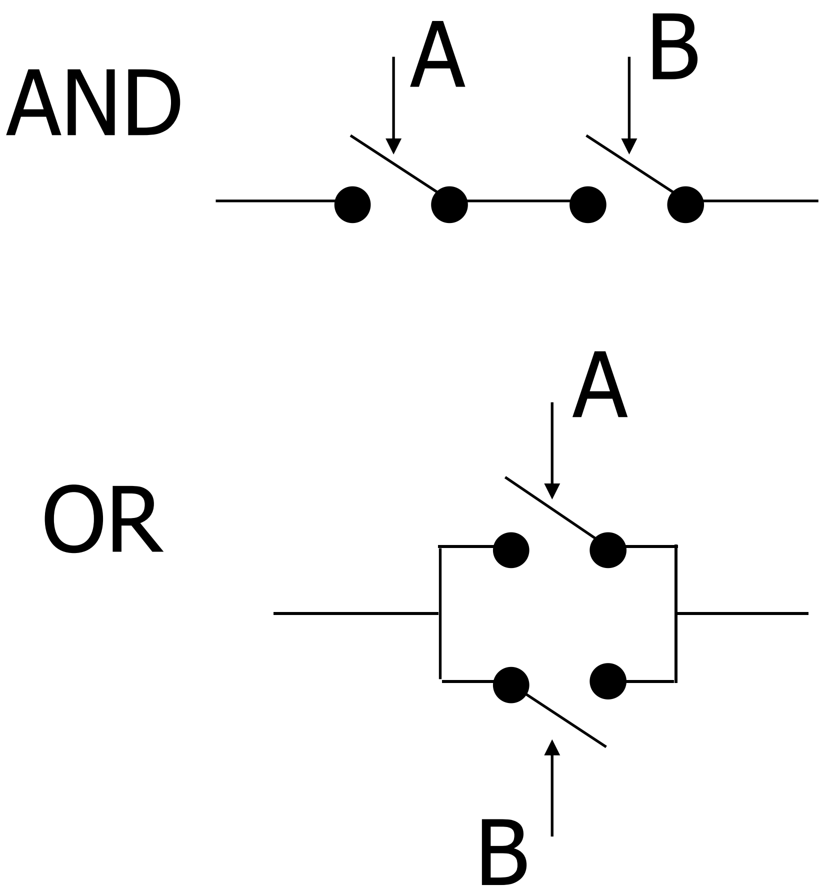



*On*-switch (if A is “1” or asserted) turns-on light bulb (Z)

$$Z \equiv A$$

# Switches

- Compose switches into more complex ones (Boolean functions):

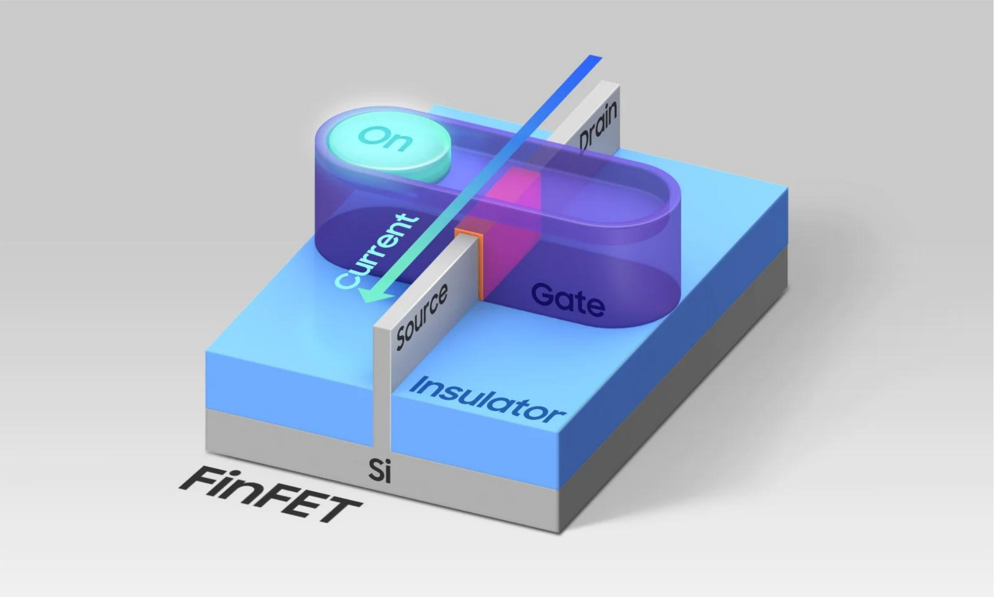



Z ≡ A <u>and</u> B

Z ≡ A <u>or</u> B

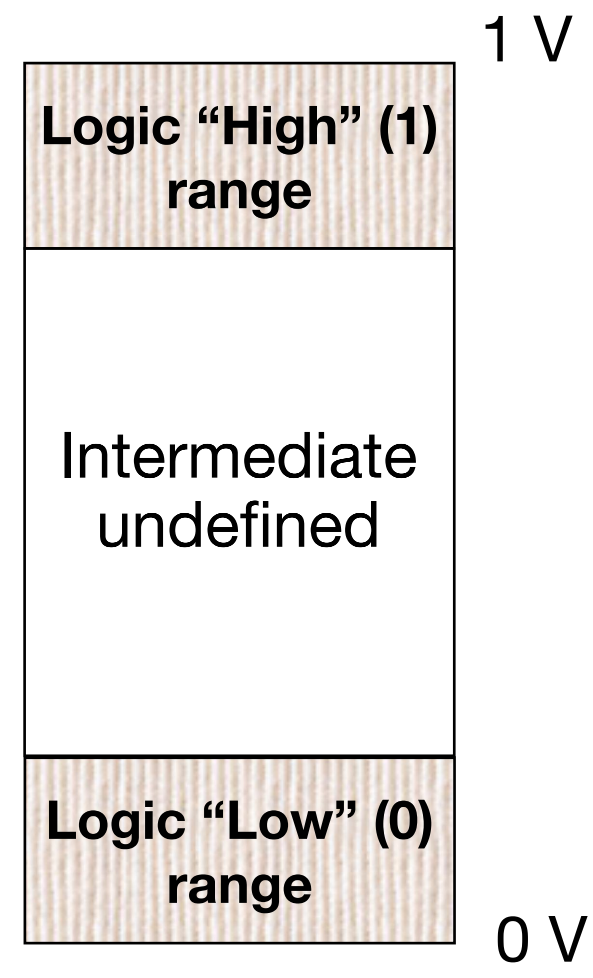
# Revisit: Binary System

- 0 and 1 (binary digit or bit, unit of information entropy)
- Decided by the characteristic of semiconductor devices (bi-stable states)
  - **They can also be considered as voltage-controlled switches**
- Resilient to noise (threshold)
- Supported by Boolean algebra theory (George Boole, 1854)
- Basic operations: ^, |, ~



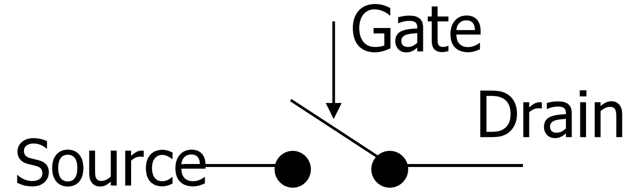
# Binary Representation of Signals

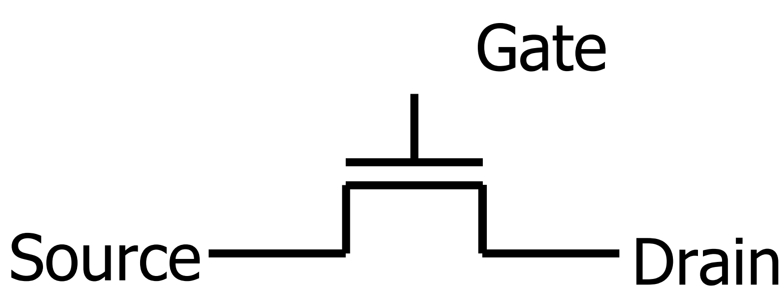
- High voltage ($V_{dd}$) represents 1, or true
  - In modern microprocessors, Vdd ~ 1.0 Volt
- Low voltage (0 Volt or Ground) represents 0, or false
- **Digital**: discretize signal/voltage to a 0 or a 1
  - This removes noise as signals propagate – a big advantage of digital systems over analog systems
  - Circuits to discriminate between two possible inputs are simple to implement and have scaled well with Moore's Law.
- If one switch can control another switch with digital signal, we can build a computer!
- Our switches: CMOS transistors

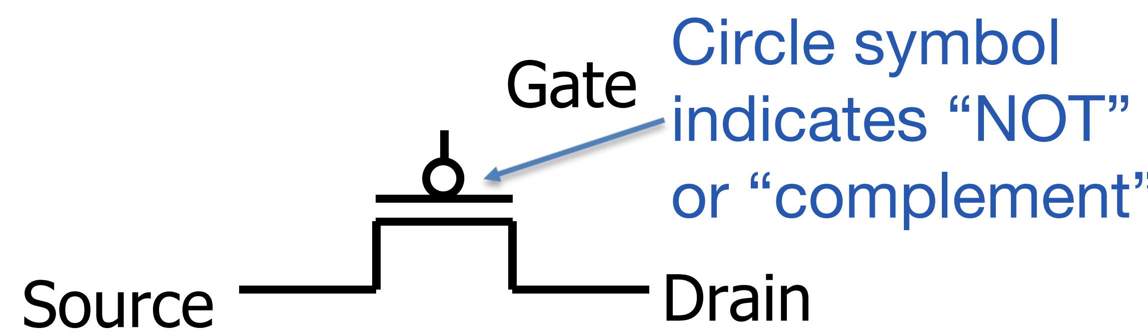


# NMOS & PMOS Transistors

- Three terminals: source, gate, and drain
  - Basic model




n-channel transitor
off when voltage at Gate is low
on when:
voltage (Gate) > voltage (Threshold)

(**High** resistance when gate voltage **Low**,
**Low** resistance when gate voltage **High**)

p-channel transitor
on when voltage at Gate is low
off when:
voltage (Gate) > voltage (Threshold)

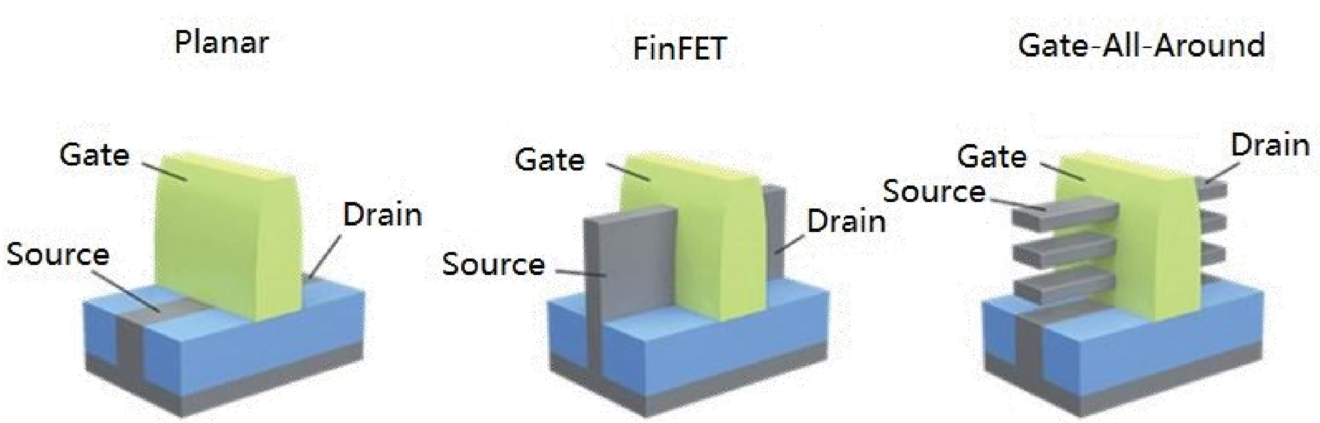
(**Low** resistance when gate voltage **Low**,
**High** resistance when gate voltage **High**)

# NMOS & PMOS Transistors: Clarifications

- Transistors can be modeled by resistors and capacitors, i.e., they can have non-ideal effects such as leakage and delay

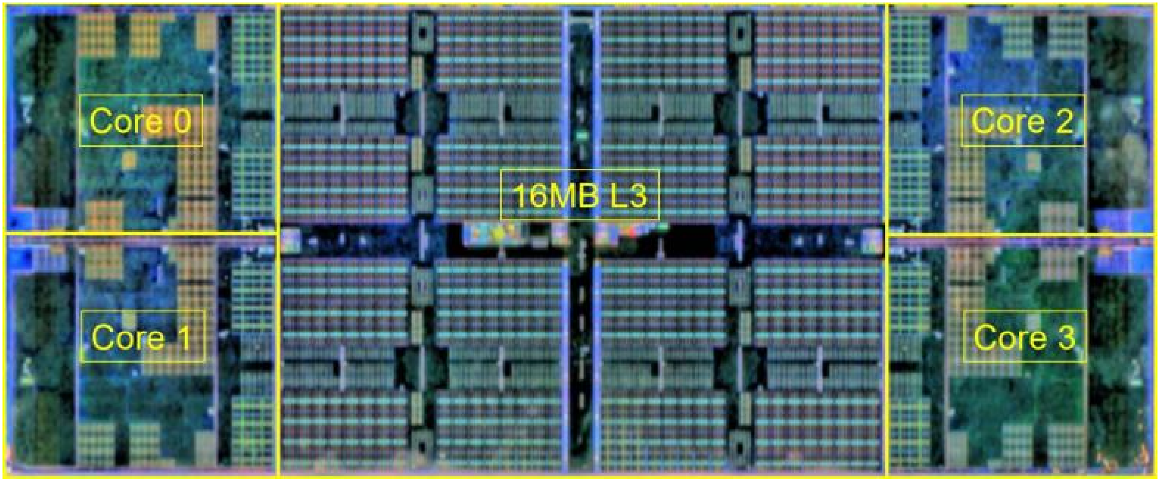
- Recent trend of transistors

Planar → FinFET → GAAFET




From Utmel.com

- Real stuff: AMD Zen 2




475M-transistor core slice is 7.83mm 2 with a 0.5MB L2 cache and 4MB of shared L3 cache

# Synchronous Digital System (SDS)

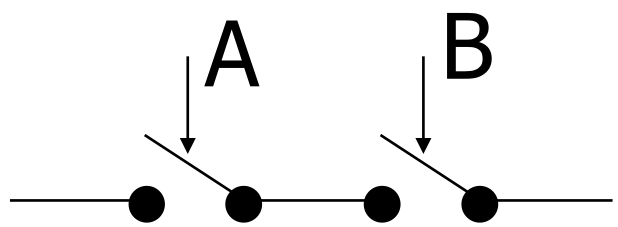
- A system that processes digital signals (0s and 1s)
- Synchronous digital systems consist of two basic types of circuits.
  - Combinational logic circuits (**this lecture**)
    - The outputs sorely depend on the input
    - No way to store information
  - State Elements (next time)
    - Circuits that store information
    - E.g., registers and memory
- CPU cores are SDS's

- Our Goal: Implement a RISC-V processor as a synchronous digital system.
- This SDS should have the capabilities to execute RISC-V instructions.

# From Transistors to Logic Gates

- Complementary MOS (CMOS)

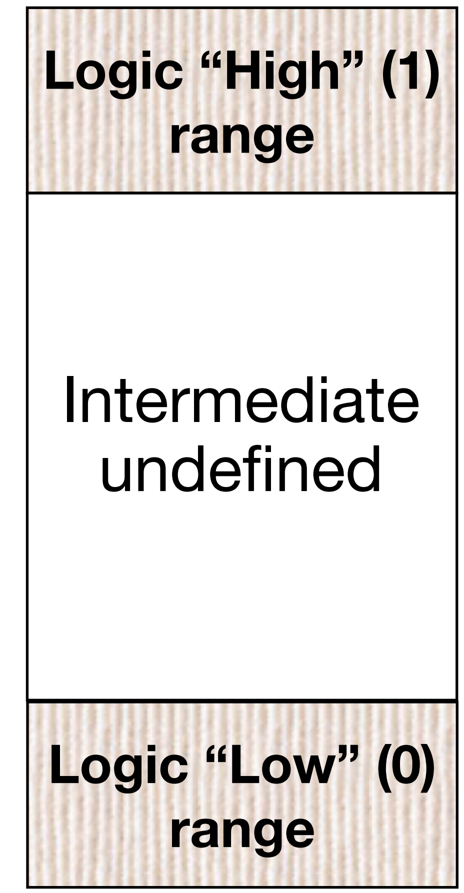
AND




$Z \equiv$ A <u>and</u> B

Similarly

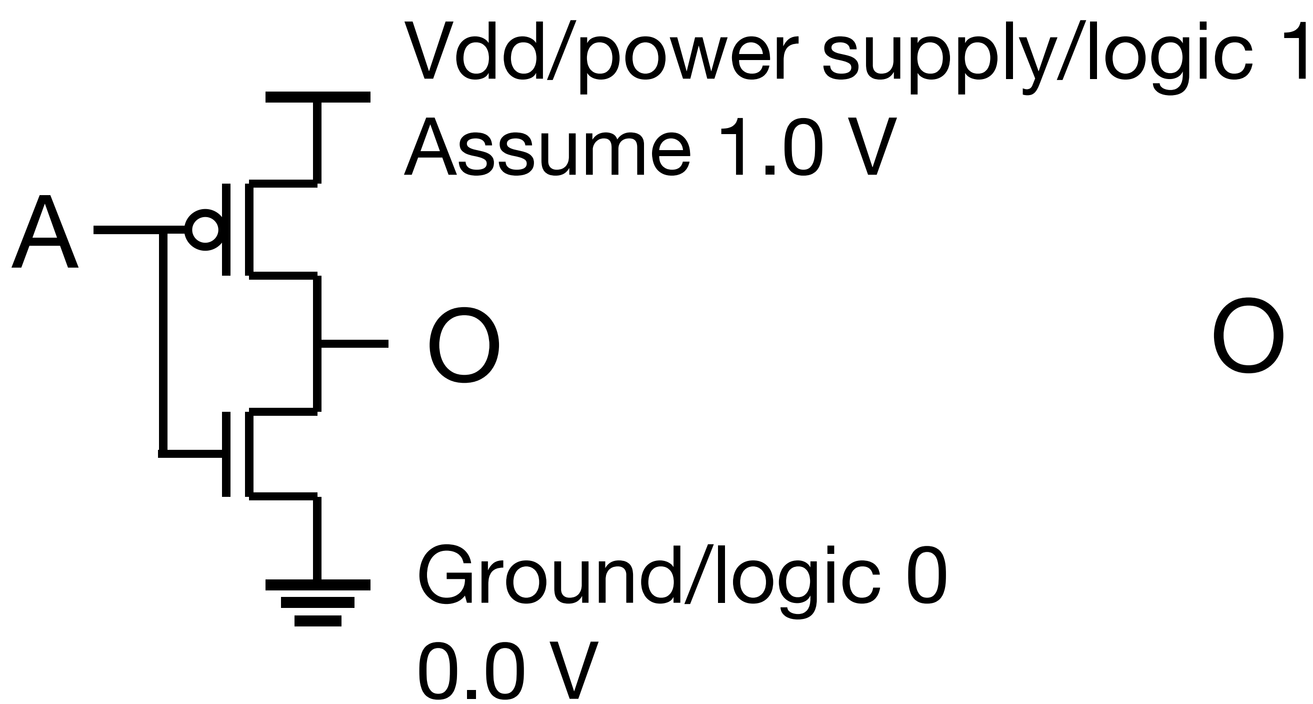
1V




- N-type transistors (NMOS) pass weak 1 (Vdd - Vth) and strong 0
- P-type transistors (PMOS) pass weak 0 (Vth) and strong 1
- Pairs of N/P-type transistors to pass strong 0 and strong 1

# The Simplest CMOS Circuits

- Inverter/Not gate

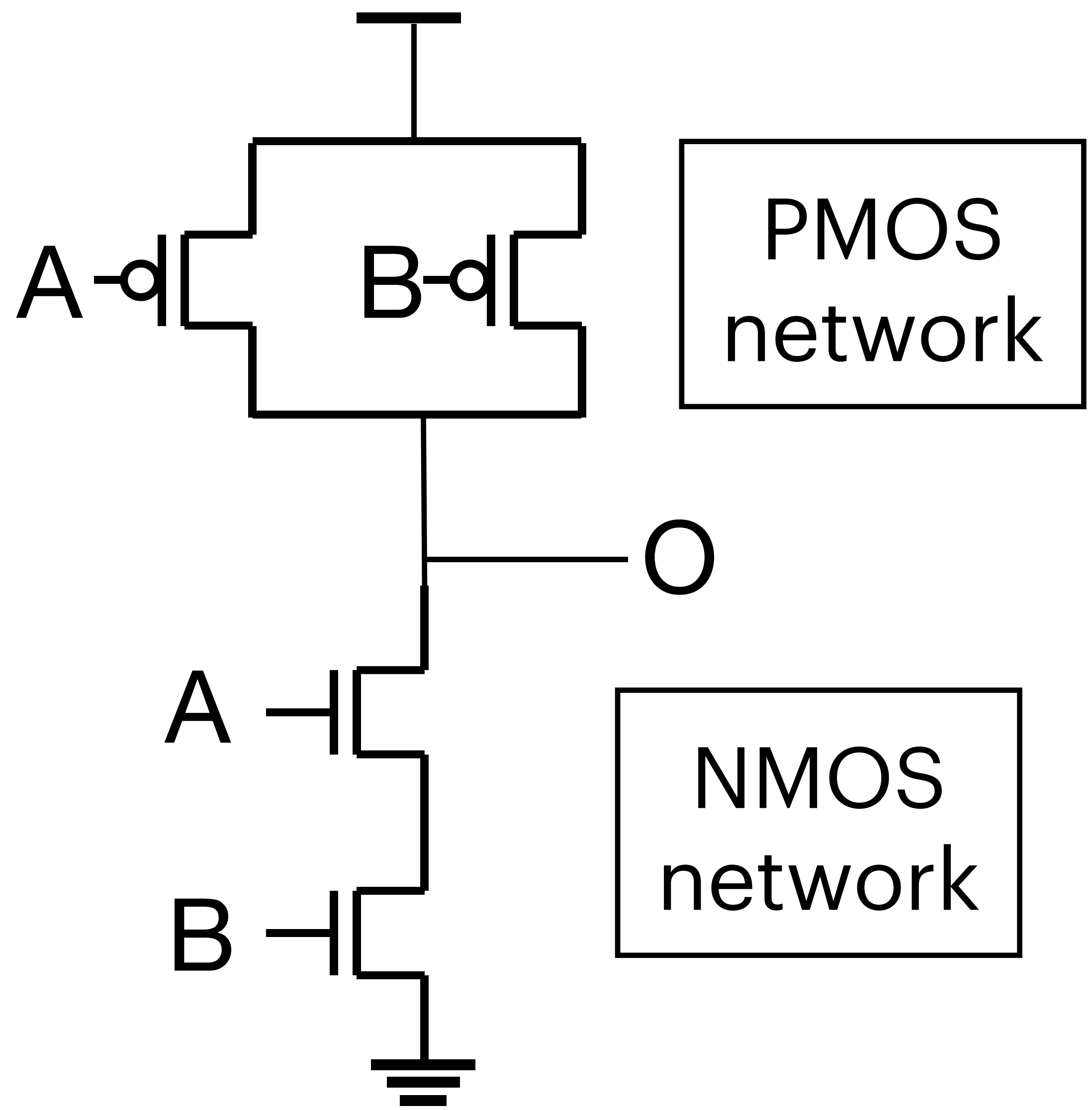



O = not A

# NAND Gate

Truth table

| A | B | O |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



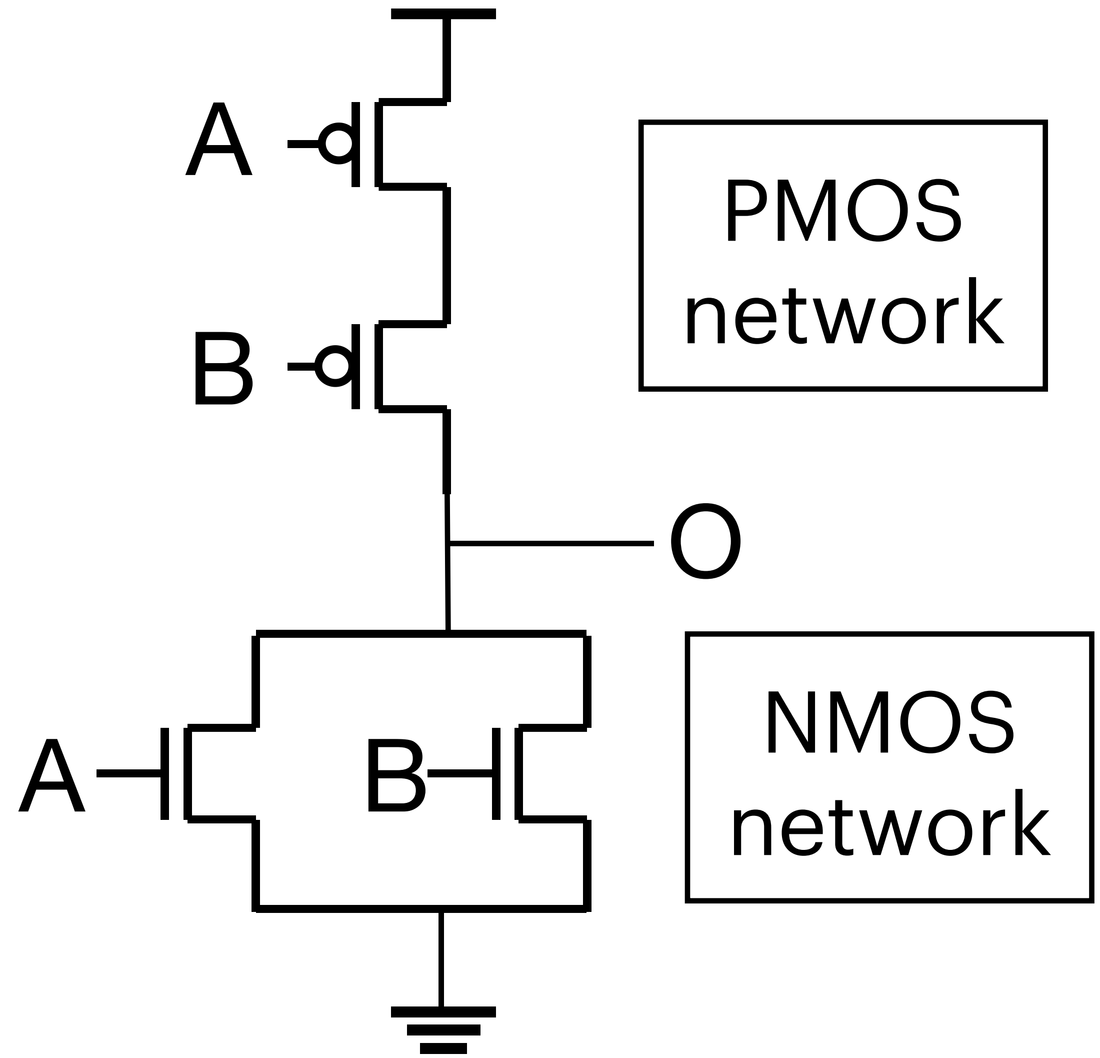

What about 3-input NAND?

What about 2-input AND?

# NOR Gate

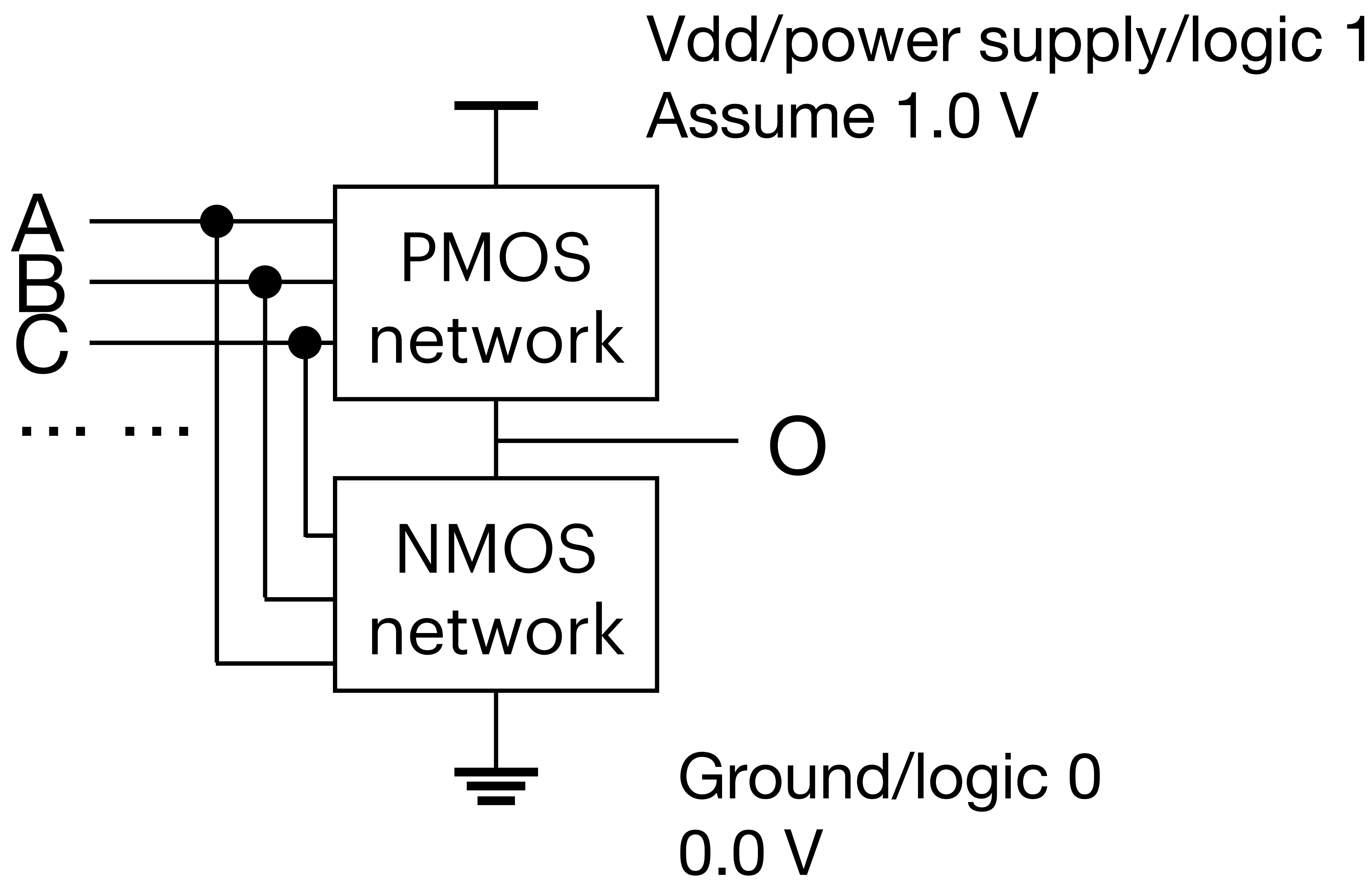
Truth table

| A | B | O |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



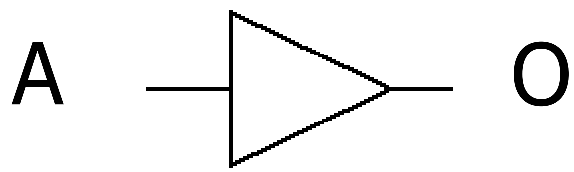

What about 3-input NOR?

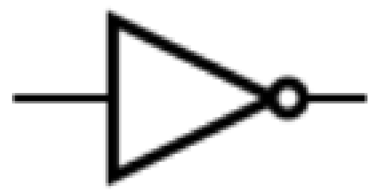
# General CMOS Logic Gates



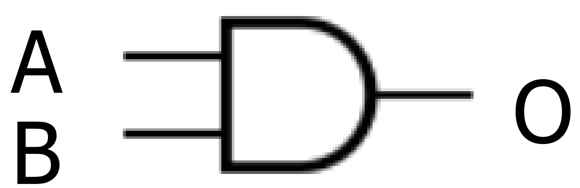
# Basic Symbols

- Standard symbols for logic gates
  - Buffer, NOT

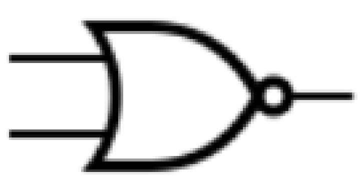


  - AND, NAND




  - OR, NOR

- Universal sets
  - NOT, AND, OR

Can be combined to implement any logics

  - NAND
  - NOR

Through Boolean algebra!

# From Logic Gates to Building Blocks

- Method 1: through boolean expressions (sum-of-minterm)
- Method 2: Karnauph Map

# Boolean Algebra

- Use plus "+" for OR
  - "logical sum" 1+0=0+1=1 (True); 1+1=2 (True); 0+0=0 (False)
- Use product for AND (a•b or implied via ab)
  - "logical product" 0•0 = 0•1 = 1•0 = 0 (False); 1•1 = 1 (True)
- "Bar" to mean complement (NOT)
- Thus

$ab + a + \bar{c}$

$= a \cdot b + a + \bar{c}$

= (a AND b) OR a OR (NOT c )

# Build Combinational Circuits with Basic Logic Gates

- Combinational circuits: the ones that the output of the digital circuits depends solely on its inputs; usually built with logic gates without feedback
  - Step 1: Write down truth table of the desired logic

For example build an XOR with AND/OR/NOT

| A | B | O |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Build Combinational Circuits with Basic Logic Gates

- Combinational circuits: the ones that the output of the digital circuits depends solely on its inputs; usually built with logic gates without feedback
  - Step 2: Pick the lines with 1 as the output; write them down in *Sum of Minterms (Product)* form;

For example build an XOR with AND/OR/NOT

| A | B | O |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Minterms*

| | |
|---|---|
| $\overline{AB}$ | $m_0$ |
| $\overline{A}B$ | $m_1$ |
| $A\overline{B}$ | $m_2$ |
| $AB$ | $m_3$ |

# Build Combinational Circuits with Basic Logic Gates

- Combinational circuits: the ones that the output of the digital circuits depends solely on its inputs; usually built with logic gates without feedback
  - Step 3: Simplify using Laws of Boolean algebra;

For example build an XOR with AND/OR/NOT

| A | B | O |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$$O = m_1 + m_2$$

*Minterms*

| | |
|---|---|
| $\overline{A}\overline{B}$ | $m_0$ |
| $\overline{A}B$ | $m_1$ |
| $A\overline{B}$ | $m_2$ |
| $AB$ | $m_3$ |

# Laws of Boolean Algebra

| AND form | OR form | |
|---|---|---|
| $X\overline{X} = 0$ | $X+\overline{X} = 1$ | Complementarity |
| $X0 = 0$ | $X+1 = 1$ | Laws of 0's and 1's |
| $X1 = X$ | $X+0 = X$ | Identities |
| $XX = X$ | $X+X = X$ | Idempotent Laws |
| $XY = YX$ | $X+Y = Y+X$ | Commutativity |
| $(XY)Z = X(YZ)$ | $(X+Y)+Z = X+(Y+Z)$ | Associativity |
| $X(Y+Z) = XY+XZ$ | $X+YZ = (X+Y)(X+Z)$ | Distribution |
| $XY+X = X$ | $(X+Y)X = X$ | Absorption |
| $\overline{XY} = \overline{X}+\overline{Y}$ | $\overline{X+Y} = \overline{X}\,\overline{Y}$ | DeMorgan's Law |

# Your turn!

- Build a half adder:
  - Sum Carry
  - 0 + 0 = 0 0
  - 0 + 1 = 1 0
  - 1 + 0 = 1 0
  - 1 + 1 = 0 1
- Build a 2-bit adder:
  - Sum Carry
  - 00 + 00 = 00 0
  - 00 + 01 = 01 0
  - 00 + 10 = 10 0
  - 00 + 11 = 11 0
  - 01 + 00 = 01 0
  - 01 + 01 = 10 0
  - 01 + 10 = 11 0
  - 01 + 11 = 00 1

  AB CD

Sum Carry

10 + 00 = 10 0

10 + 01 = 11 0

10 + 10 = 00 1

10 + 11 = 01 1

11 + 00 = 11 0

11 + 01 = 00 1

11 + 10 = 01 1

11 + 11 = 10 1

# Another Method—Karnauph Map
(optional)

AB Gray coded

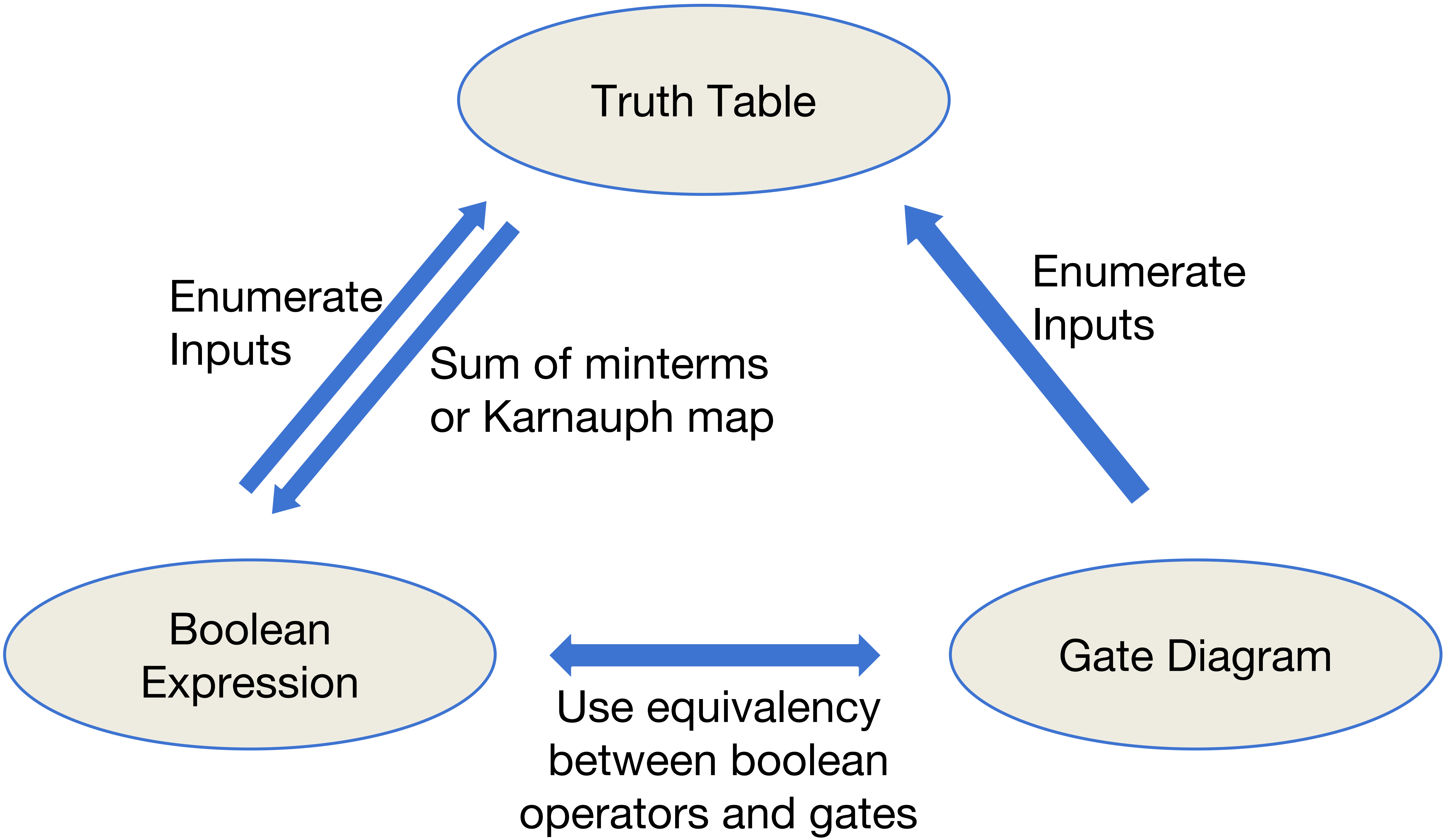
| CD \ AB | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | | 1 | |
| 11 | | 1 | 1 | 1 |
| 10 | | | 1 | 1 |

Gray coded

Each cell corresponds to a minterm

Online Karnauph map solver: http://www.32x8.com/index.html

# Representations of Combinational Logic

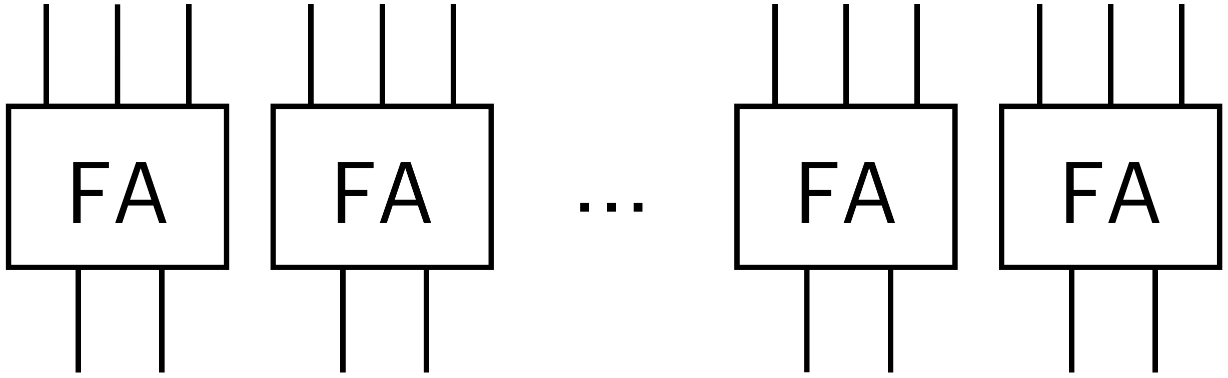


# Build Larger Blocks—like LEGO®

  01010101
+ 01110011

Build a full adder (FA): truth table

| Carry in | A | B | Sum | Carry out |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

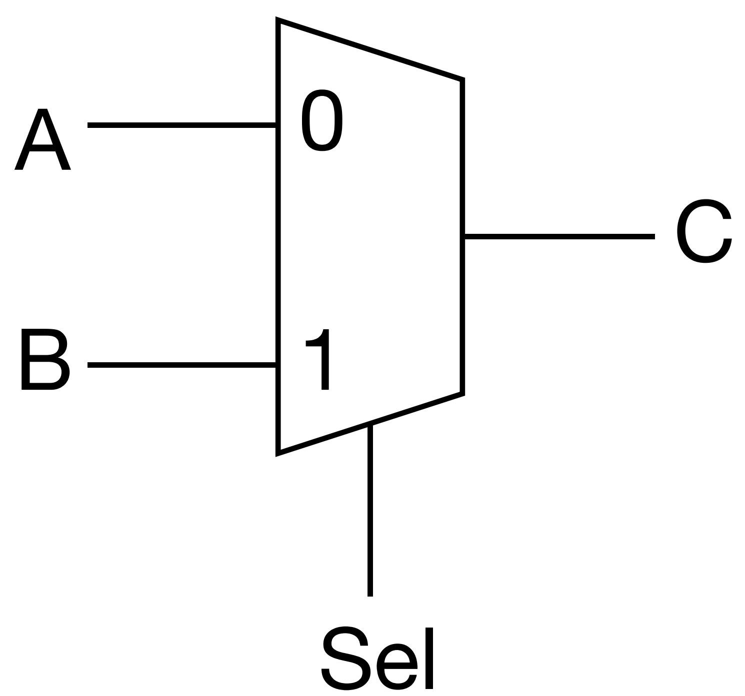


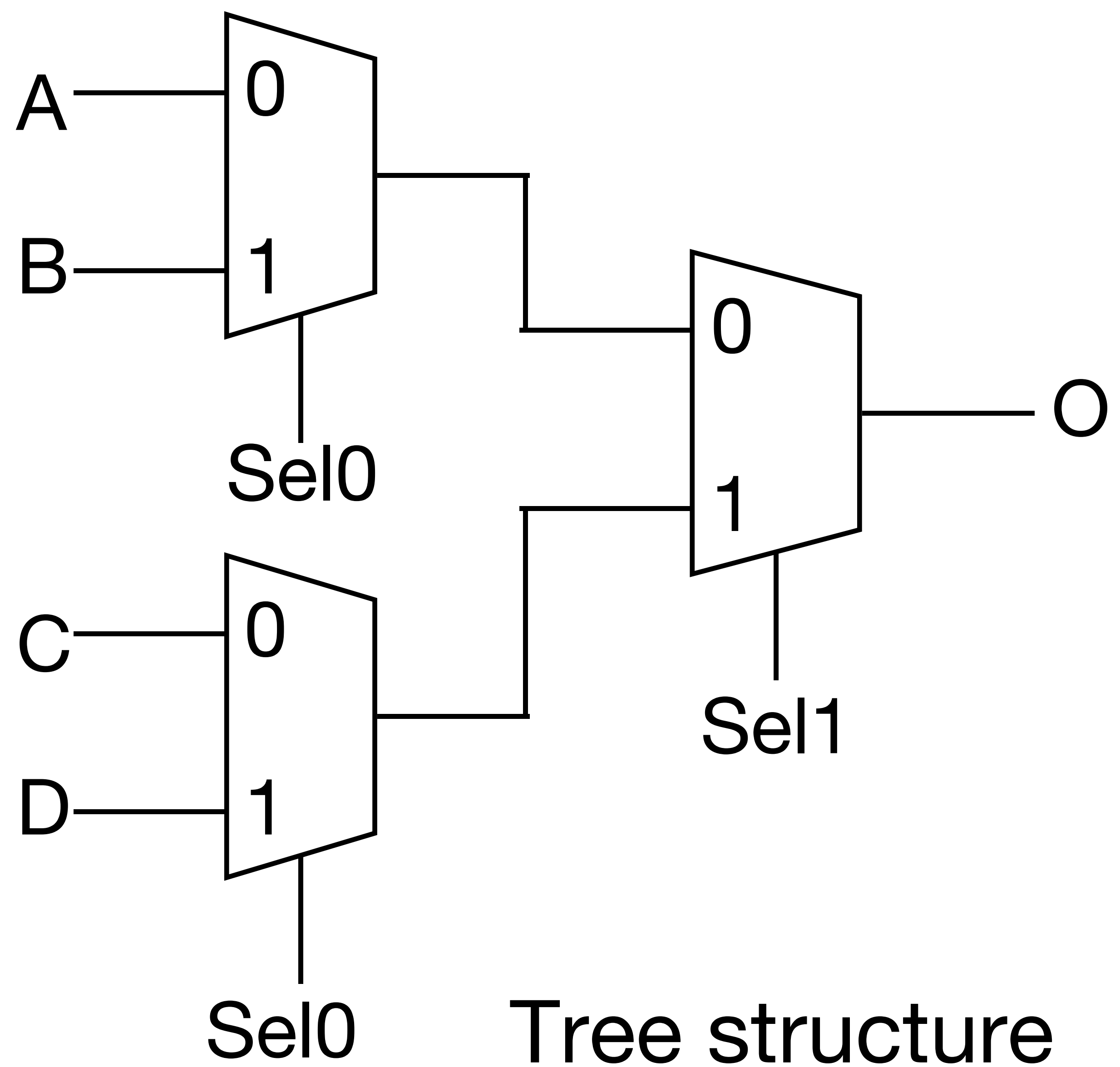
# Exercise

- Recall beq instruction. Build a comparator that makes the decision. 1 indicates “equal”, 0 indicates “not equal”

# Other Useful Combinational Circuits

- Multiplexer ($2^n$-to-1)




Tree structure