Class Topics (클래스 홈페이지 참조)

- Part 1: Fundamental concepts and principles
  1) Invention of computers and digital logic design
  2) Abstractions to deal with complexity
  3) Data (versus code)
  4) Machines called computers
  5) Underlying technology and evolution since 1945
- Part 2: 빠른 컴퓨터를 위한 설계 (ISA design)
- Part 3: 빠른 컴퓨터를 위한 구현 (ISA implementation)

Machines Called Computers
(복잡한 자동장치: 어떻게 만들 수 있었나)

Part 1

- Invention of computers
- Digital Logic Design
- Notion of "Abstraction"

References:

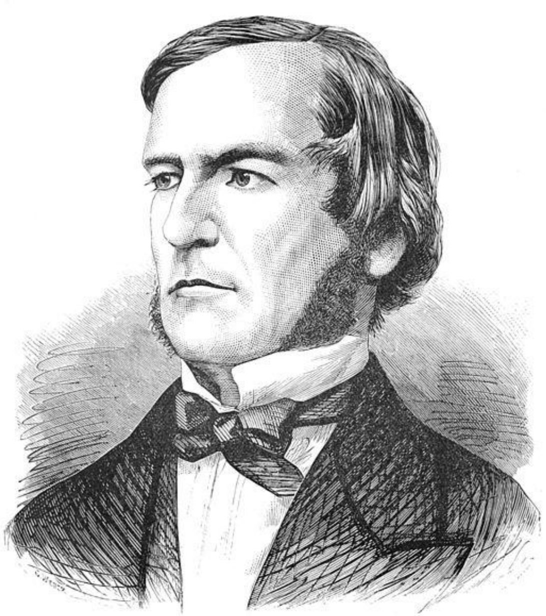
1. Computer Organization and Design & Computer Architecture, Hennessy and Patterson (slides are adapted from those by the authors)

Invention of Computers

- 컴퓨터의 탄생
  - 과학적 성취 (새로운 지식의 창조)
    - Boole in 19C
  - 실용적 도구 개발 – 산업혁명의 맥
    - Automata (자동장치) 개발
      - Ultimate form of automata: computers
  - 구현 기술 발전
    - Transistor 발명 (wheel/shaft/cam, relays, 진공관)
- Digital logic design
- Abstraction: fundamental engineering concept

George Boole

- 19C English mathematician, philosopher, logician



- "The Laws of Thought" in 1854
  - Propositions (명제)
    - Binary (True, False; "1", "0")
  - AND, OR, NOT, IF

- 인간의 논리적 생각은 명제 그리고 명제들을 AND, OR, NOT, IF 로 결합함으로써 표현할 수 있다
  - 철학, 논리학에서 다룬 것을 수학으로 깨끗하게 정립

Propositional Logic (명제논리학)

- Proposition: basic building block
  - Declarative sentence that is either true or false
    - 2014/10/25 is Monday, 2 + 3 = 6
    - x + 3 = 5, what time is it?
- Compound propositions – apply recursively

  p: 2014/10/25 is Monday, q: 2 + 3 = 6

  - $p \cdot q$ (AND operation), $p + q$ (OR)
  - $\bar{p}$ (NOT), $p \rightarrow q$ (IF)
- Extended to first-order logic (mathematical logic)

Truth Tables

- 1 = True; 0 = False

AND

p	q	$p \cdot q$
1	1	1
1	0	0
0	1	0
0	0	0

OR

p	q	p + q
1	1	1
1	0	1
0	1	1
0	0	0

NOT

p	$\bar{p}$
1	0
0	1

IF

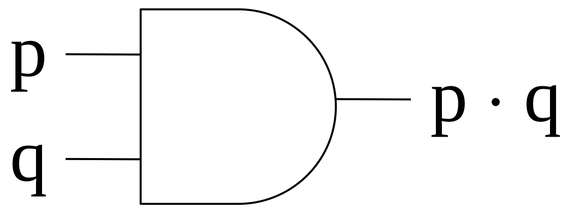
p	q	$p \rightarrow q$
1	1	1
1	0	0
0	1	1
0	0	1

In Your Mind - Digital Logic Gates

- Can implement AND, OR, NOT with electronic circuits

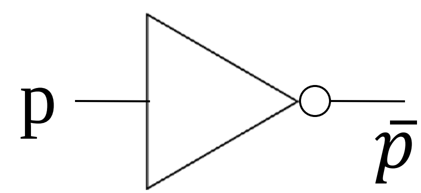
AND

p	q	p · q
1	1	1
1	0	0
0	1	0
0	0	0



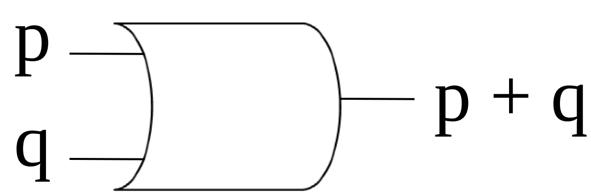
NOT

p	$\bar{p}$
1	0
0	1



OR

p	q	p + q
1	1	1
1	0	1
0	1	1
0	0	0

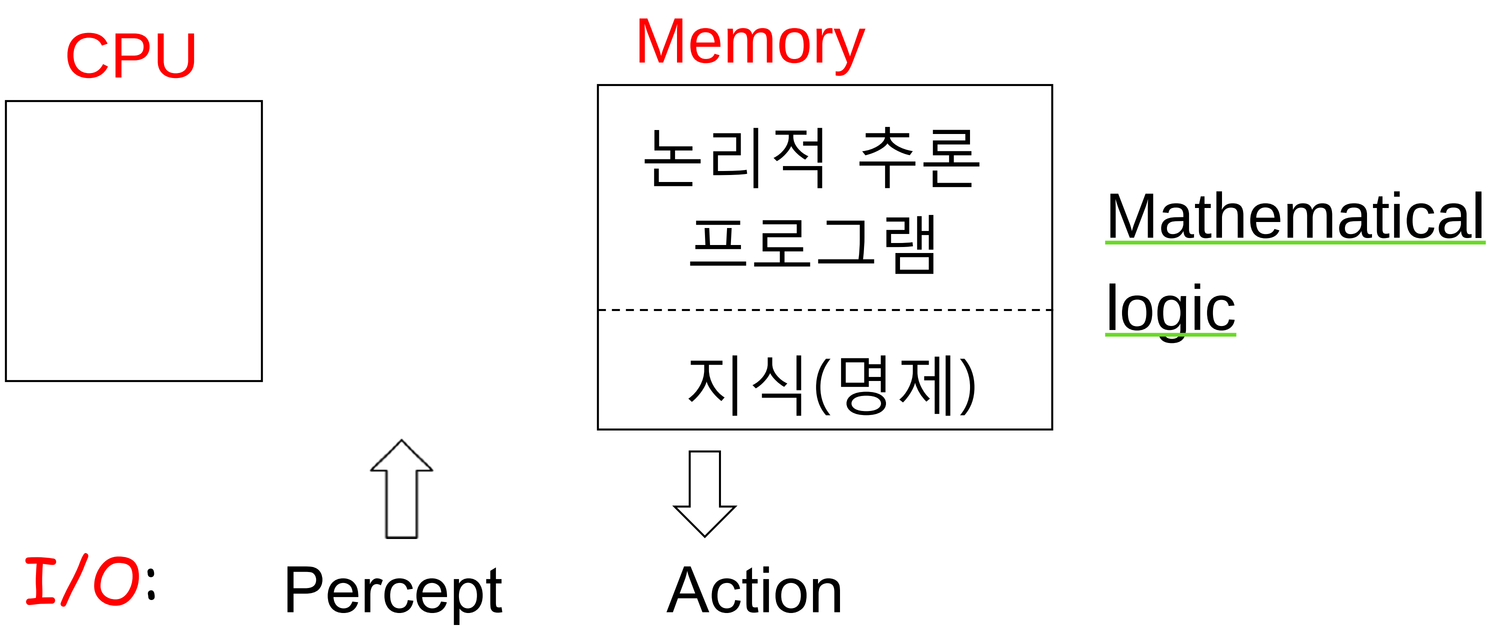


Impact of George Boole

- 인간의 논리적 생각은 명제 그리고 명제들을 AND, OR, NOT, IF 로 결합함으로써 표현할 수 있다
- Mathematical logic 분야 열다 (빈틈없는 논리의 전개)
  - AI (생각하는 기계): 지식(명제)의 저장 및 논리적 추론
    - Expert systems, planning, …
- Boolean Algebra
  - Digital logic design
    - AND, OR, NOT $\rightarrow$ CPU, memory, I/O
- Conceptual foundation of Computer Science

AI: Knowledge-Based Approach

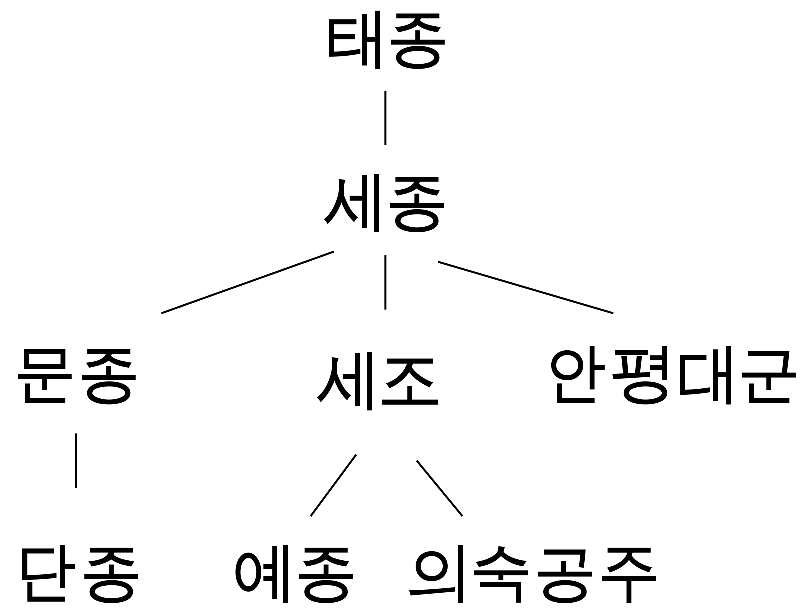
- Given computers, can we develop intelligent machines?
- Knowledge representation and reasoning (bio-inspired)



- 70년 동안 많은 연구 및 진화 (그 중의 하나 소개하면)
- What else in AI? Are they thinking?

Logic Programming

(fact (parent 세조 예종))
(fact (parent 세조 의숙공주))
(fact (parent 문종 단종))
(fact (parent 세종 세조))
(fact (parent 세종 안평대군))
(fact (parent 세종 문종))
(fact (parent 태종 세종))

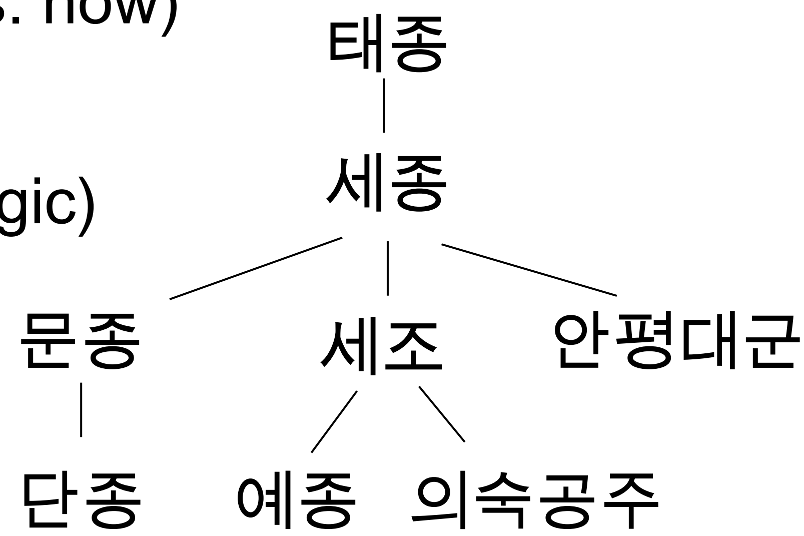


(query (parent 세조 ?c)) // 예종, 의숙공주

(fact (child ?c ?p) (parent ?p ?c)) // rule

(query (child ?c 세종)) // 문종, 세조, 안평대군

- Declarative programming (what vs. how)
  - Fact (relation declared or rule)
  - Query (추론; subset of math. logic)



(query (child 단종 문종))   // True

(query (child 태종 ?p))   // False

(fact (ancestor ?a ?y) (parent ?a ?y))   // recursive rule
(fact (ancestor ?a ?y) (parent ?a ?z) (ancestor ?z ?y))

(query (ancestor ?a 단종) (ancestor ?a 의숙공주)) // 태종, 세종

Impact of George Boole (반복)

- 인간의 논리적 생각은 명제 그리고 명제들을 AND, OR, NOT, IF 로 결합함으로써 표현할 수 있다
- Mathematical logic 분야 열다 (빈틈없는 논리의 전개)
  - AI: knowledge-based approach
    - Expert systems, planning, …
- Boolean Algebra
  - Digital logic design
    - AND, OR, NOT logic gate $\rightarrow$ CPU, memory, I/O
- Conceptual foundation of Computer Science

To Think about

- 인간의 논리적 생각은 명제 그리고 명제들을 AND, OR, NOT, IF 로 결합함으로써 표현할 수 있다

- 확인된 것은: computers (계산 및 논리적 처리)
  - 미확인: creativity? thinking?

- Scale of claims
  - 만물의 근원: 물, 불, 흙, 공기 (4원소론)
  - 모든 것은 수 (numbers)
  - 인간의 생각은 …

Boolean Algebra (불 대수)

Set of values {0, 1}

Set of operations: AND, OR, NOT

(복잡한 자동장치 설계 위한 수학적 기반 제공)

What is Algebra?

- What is Arithmetic?
  - Numbers
  - Four basic operators: +, -, ×, /
- What is Algebra?
  - Use rules of arithmetic
  - Additional concept: unknowns (or variables)
    - Use symbols like x, y, …
  - Algebraic equations: $x + 3 = 1$, $y^2 + 2y + 5 = 0$, …
    - Important for modeling real world

Elementary Algebra

- Elementary algebra
  - Use of symbols to represent values and their relations, especially for solving equations

$$2 + 3 = 5$$

$$x^2 + 2x + 1 = 0$$

  - 실수값, 변수 (variable), 사칙연산

Boolean Algebra

- Operations and rules for working with the set {0, 1}
  - Operations
    - AND, OR, NOT
  - Boolean expressions
    - 0, 1, x_1, x_2, …, x_n are Boolean expressions
    - If E_1 and E_2 are Boolean expressions, then
      $\overline{E_1}$, $E_1 \cdot E_2$, E_1+E_2 are Boolean expressions

† Arithmetic expressions 대비: 이진값, 변수, AND/OR/NOT

† Why the name ALU (Arithmetic and Logic Unit)?

Basic Identities of Boolean Algebra

- Axioms → conjectures → theorems (or knowledge)

$x + 0 = x$ | $x \bullet 0 = 0$

$x + 1 = 1$ | $x \bullet 1 = x$

$x + x = x$ | $x \bullet x = x$

$x + x' = 1$ | $x \bullet x' = 0$

$x + y = y + x$ | $xy = yx$

$x + (y + z) = (x + y) + z$ | $x(yz) = (xy)z$

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

$(x + y)' = x'y'$ | $(xy)' = x' + y'$

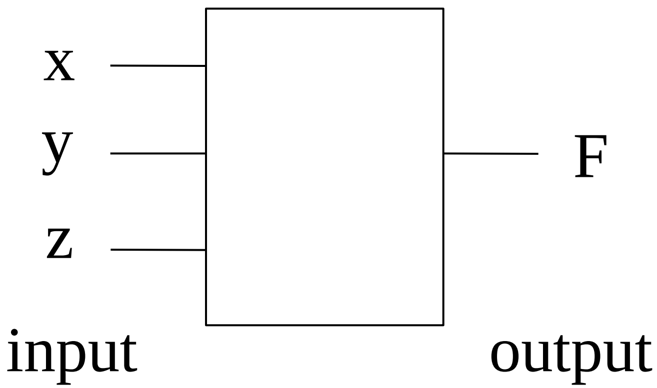
$(x')' = x$

Boolean Algebra

- Given truth table, find Boolean expression for function?
  - F = f(x,y,z)

x	y	z	F
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

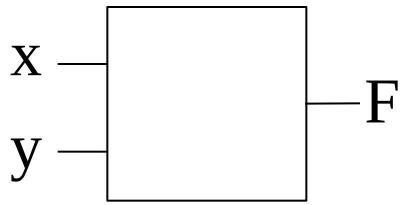
- Practical view



Exercises - Boolean Expressions

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

$\mathbf{F = x \cdot y}$



x	y	F
0	0	0
0	1	1
1	0	0
1	1	1

$\mathbf{F} = (\boldsymbol{x} \cdot \boldsymbol{y}) + (\overline{\boldsymbol{x}} \cdot \boldsymbol{y})$

Multiple outputs:

x	y	F1	F2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

$\mathbf{F1} = \boldsymbol{x} + \boldsymbol{y}$

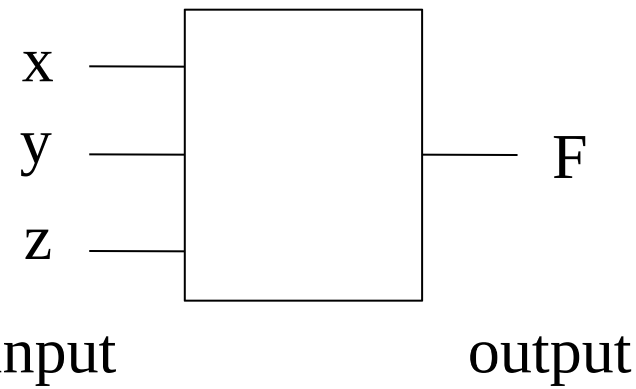
$\mathbf{F2} = \overline{\boldsymbol{x} + \boldsymbol{y}}$

Boolean Algebra

- Given truth table, systematically find Boolean expression for F?
  - F = $x \cdot y + \bar{z}$
  - Simplest form?

x	y	z	F
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

† Practical view

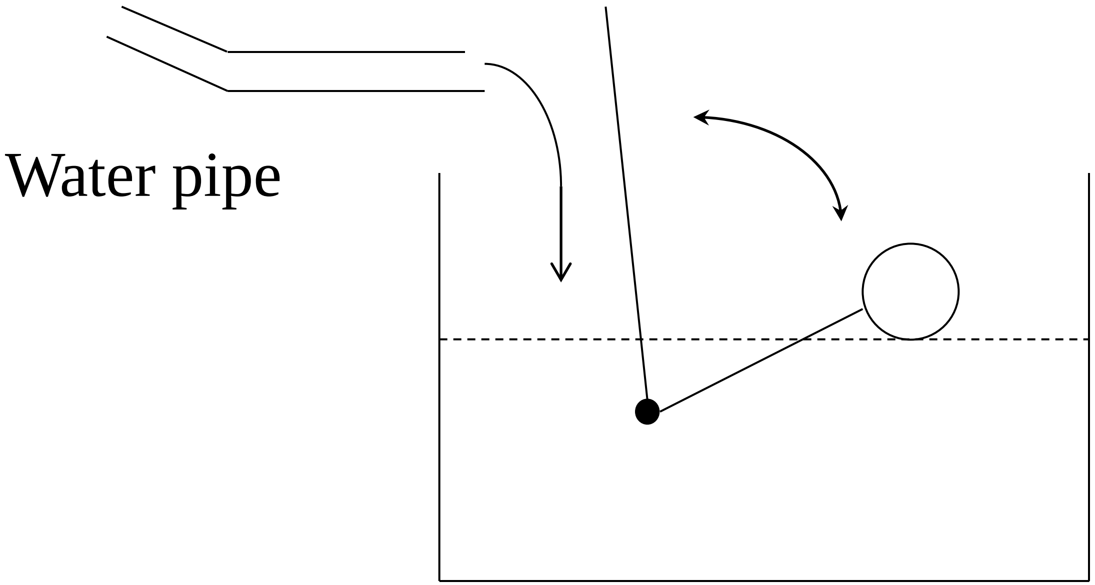


Automata (자동장치) Design and Boolean Algebra

(복잡한 자동장치의 체계적 설계)

Simple Automaton

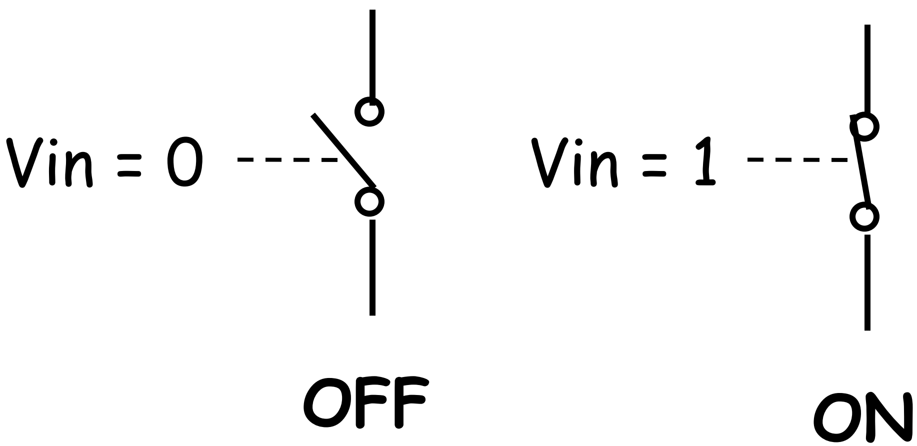
- Self-operating machine (water-level controller)



- More useful/complex automata: "sensors" and "switches"

Electronics and Digital Switches

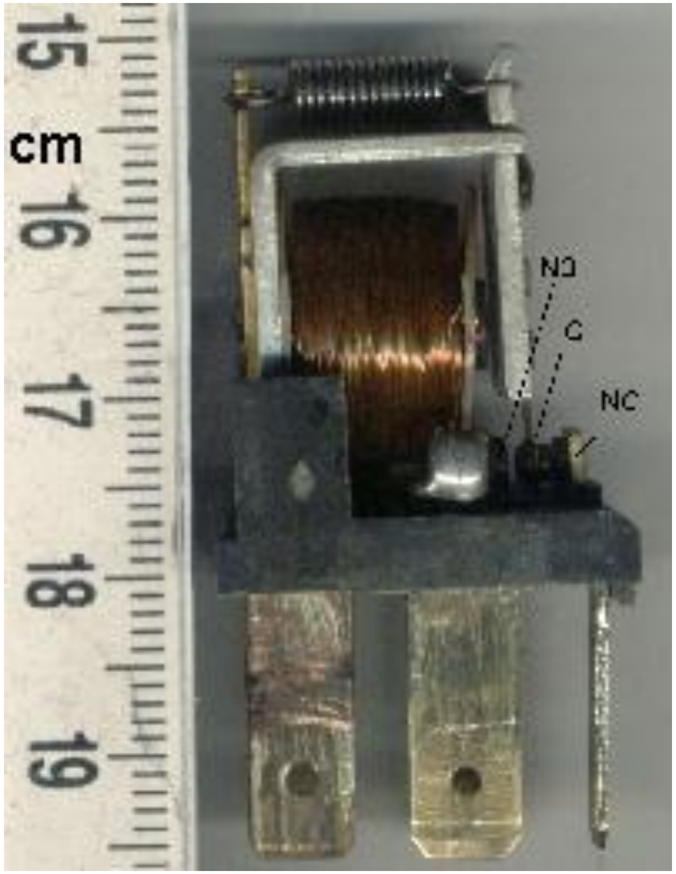
- Need for three-terminal switching devices
  - Control signal, flow between the remaining two
    - Digital switch (ON, OFF)
  - Mechanical, electromechanical, electronic

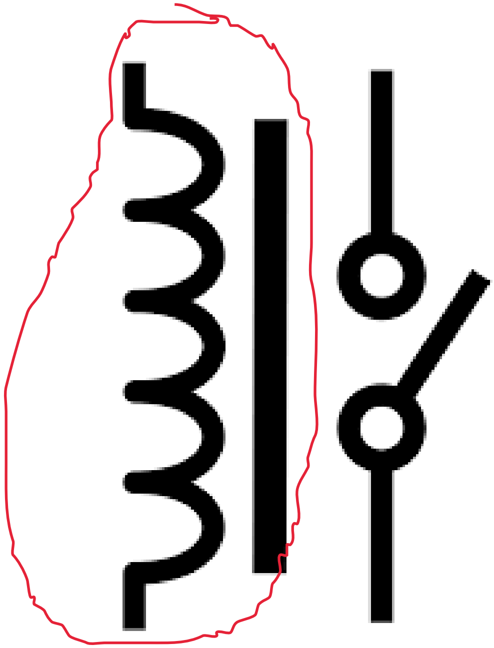


- High = 2^V = "1" = True, Low = 0^V = "0" = False

Electro-Mechanical Relay

- Invented in 1835, switching speed: order of milliseconds



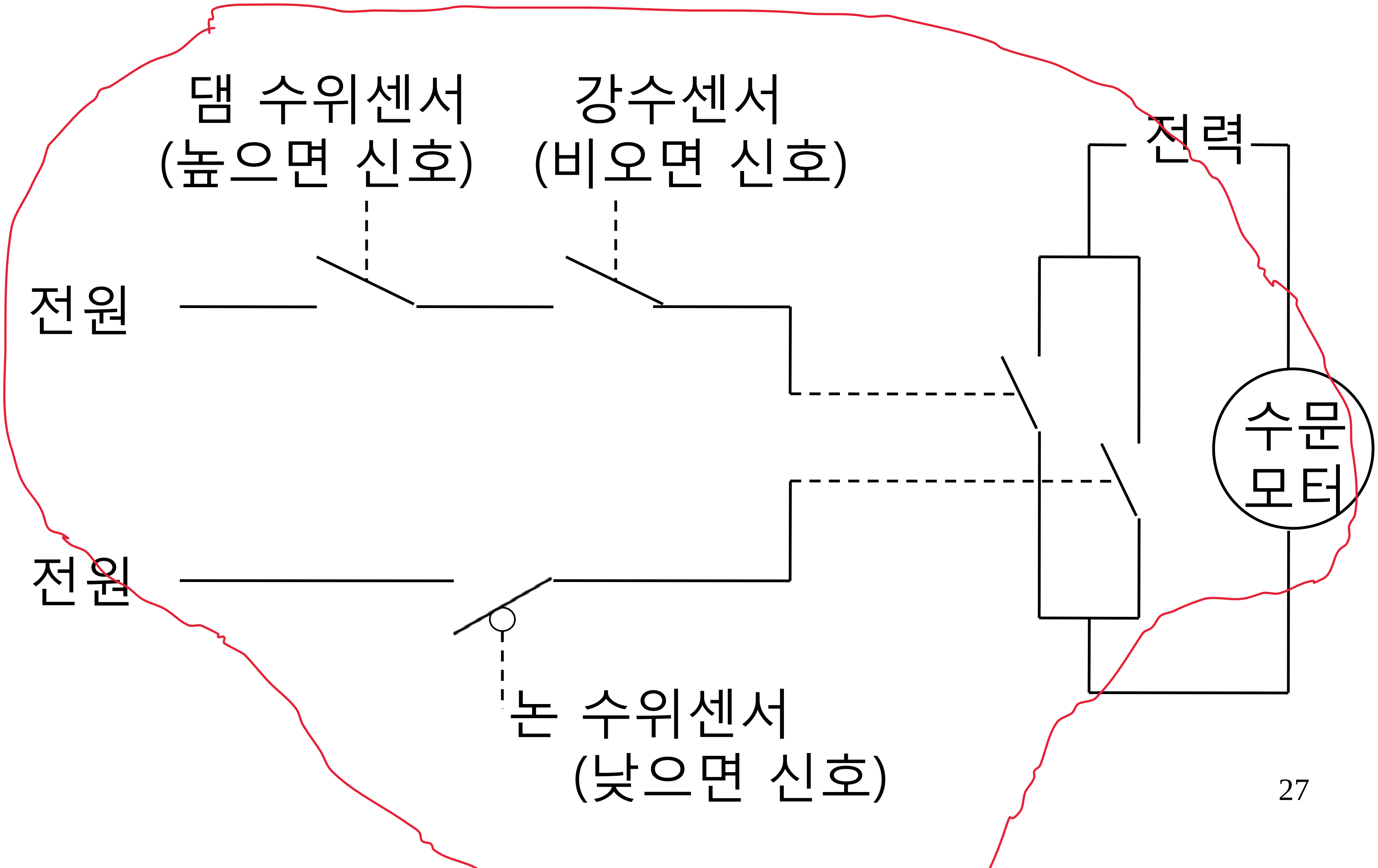


Electron or Vacuum Tube

- Invented in 1906 (speed: order of microseconds)
- First commercial electron tube by RCA in 1920
  - Radio, TV, Audio, telephone networks, ENIAC

More Meaningful Automaton

- Self-operating machine ("sensors" and "digital switches")



Automata Design (Shannon, 1938)

- Real world example
  - 댐 수위 높은데 비가 오면 수문 연다
  - 또는 논에 물이 적으면 수문 연다

    x: 댐의 수위가 높다

    y: 비가 온다

    z: 논에 물이 충분하다

    F: 댐의 수문을 연다 (output)

- F = $x \cdot y + \bar{z}$
  - Simplest form?

Truth Table

x	y	z	F
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

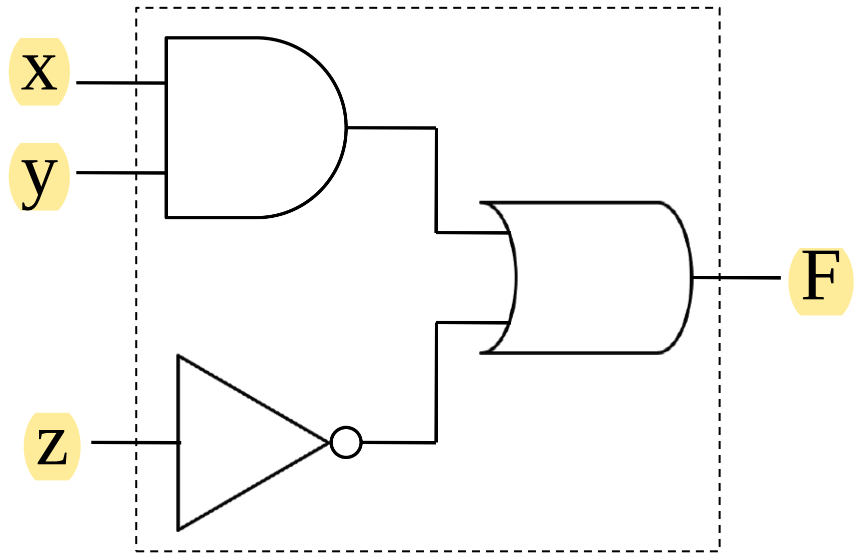
Digital Logic Design

- Real world example
  - x: 댐의수위가 높다, y: 비가 온다, z: 논에 물이 충분하다
  - F: 댐의 수문을 연다

x	y	z	F
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

Truth Table

- F = $x \cdot y + \bar{z}$
  - Simplest form?



Logic Diagram

Automata Design

- Empirical automata design
  - Ad hoc approach using 3-terminal digital switches
  - Underlying notion of AND, OR, NOT
- Shannon: relate automata design and Boolean Algebra
- Systematic design of automata
  - Desired function
  - Think about inputs, outputs (명제)
  - Build truth table
  - Reduce to Boolean logic function (and implement)
- Facilitate design automation (VLSI CAD tools)
  - Ultimate automata: ALUs, processors, computers

Invention of Computers (반복)

- 컴퓨터의 탄생
  - 과학적 성취 (새로운 지식의 창조)
    - Boole in 19C
  - 실용적 도구 개발 – 산업혁명의 맥
    - Automata (자동장치) 개발
      - † Ultimate form of automata: computers
  - 구현 기술 발전
    - Transistor 발명 (wheel/shaft/cam, relays, 진공관)
- Boolean Algebra 에서 Digital logic design 으로
- Abstraction: fundamental engineering concept

Digital Logic Design

Binary (0/1)

AND, OR, NOT 으로 유용한 자동장치 만듬

(체계적인 설계 방법)

Combinational Logic Design

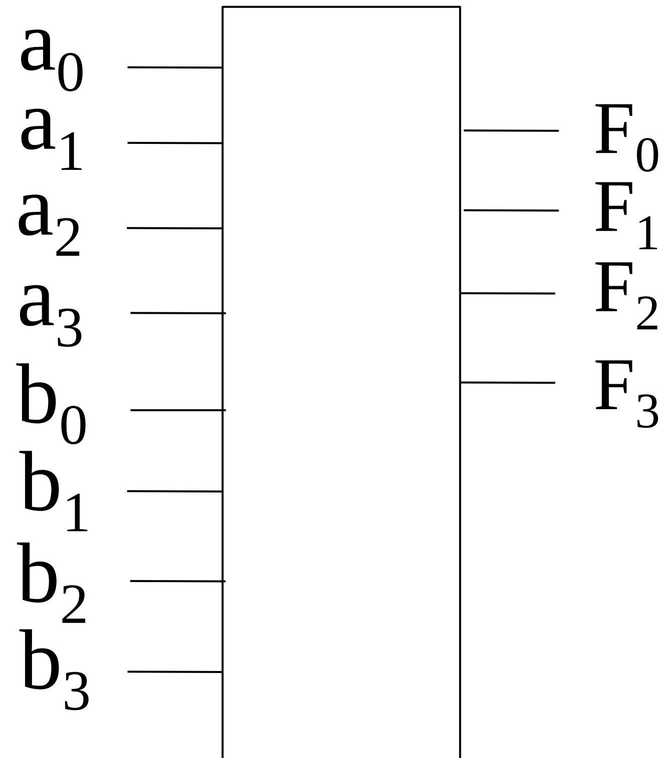
- Combinational logic
  - Outputs completely determined by inputs
- Combinational logic design
  - Given: AND, OR, NOT gates
  - Paradigm
    - Desired function
    - Determine input and output variables
    - Build truth table
    - Outputs: Boolean functions of input variables
      - † VLSI CAD tools

Imagine ALU design

- <u>4-bit adder</u>
  - Input: $a_3, a_2, a_1, a_0,$ and b_3, b_2, b_1, b_0
    - Output: F_3, F_2, F_1, F_0

$$
\begin{array}{rcl}
9_{10} & = & 1\ 0\ 0\ 1 \\
4_{10} & = & \underline{0\ 1\ 0\ 0} \\
 & & 1\ 1\ 0\ 1 \quad = \ 13_{10}
\end{array}
$$

$$
\begin{array}{cccc}
a_3 & a_2 & a_1 & a_0 \\
b_3 & b_2 & b_1 & b_0 \\
\hline
F_3 & F_2 & F_1 & F_0
\end{array}
$$



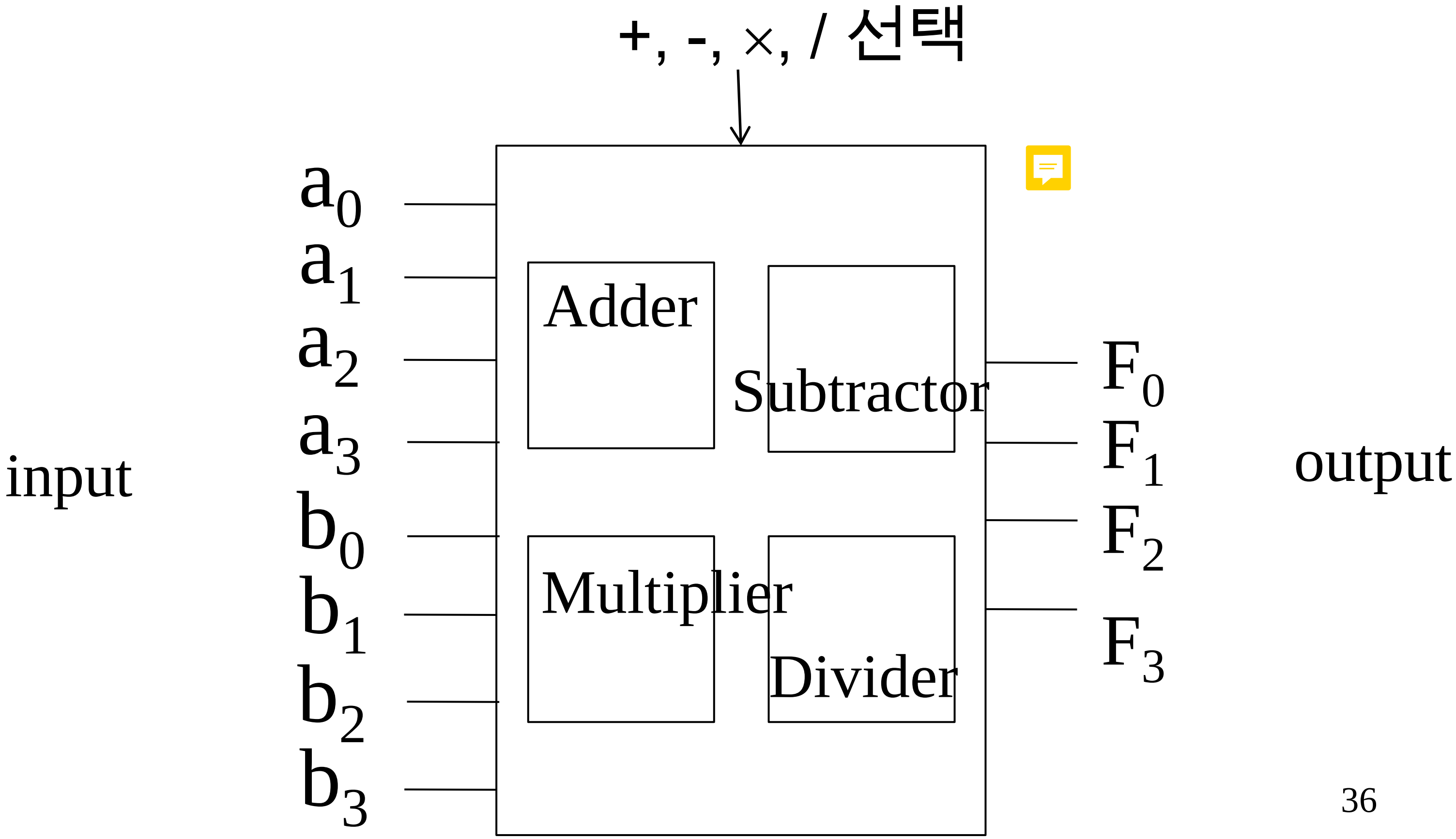
4-bit Adder

- Generate (large) truth table (with 2^8 rows)
- Find minimum Boolean expressions
  - F3 = f (a3, a2, a1, a0, b3, b2, b1, b0), F2 = …, F1 = …,
- Implement F3, F2, F1, F0

a_3	a_2	a_1	a_0	b_3	b_2	b_1	b_0	F_3	F_2	F_1	F_0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	1	0	1	0	0	1	1	0	1

Imagine ALU Design without Abstraction

- What about 4-bit ALU?
- What about 32-bit ALU?



Abstraction

(Fundamental Engineering Concept)

How to deal with complexity

(Textbook chapter 3 참조)

Abstraction

- 추상, 추상화
  - 자연어
  - Selective ignorance: 그림
  - Abstractions in engineering and Computer Science

Abstraction

- Fundamental concept of abstraction
  1) 모든 공학 제품
     - "Interface" (사용법)
     - "Implementation" (구현; 설계/구조/동작)
  2) Computer (CPU, SW) 를 포함한 모든 공학 도구/물건
     - Implementation 몰라도 interface 알면 사용 가능
       (selective ignorance)
- Examples
  - Automobiles, smartphones

Abstraction

- Machine-level programming
  - "Interface" (사용법): machine instructions
  - Implementation (구현): machines (CPUs)
- High-level programming
  - Interface (사용법): programming languages
  - Implementation: compilers (or interpreters)
- 제품이 제공하는 I/F (or 사용법 or service or abstraction)
- 무엇을 먼저: I/F design or implementation?
  - 우리 수업 3대 세부 목표

Two Major Interfaces in CS

Developers

High-level language — C, C++, Java

Compilers

(executable) Machine-level language — Machine instructions (Core, PowerPC)

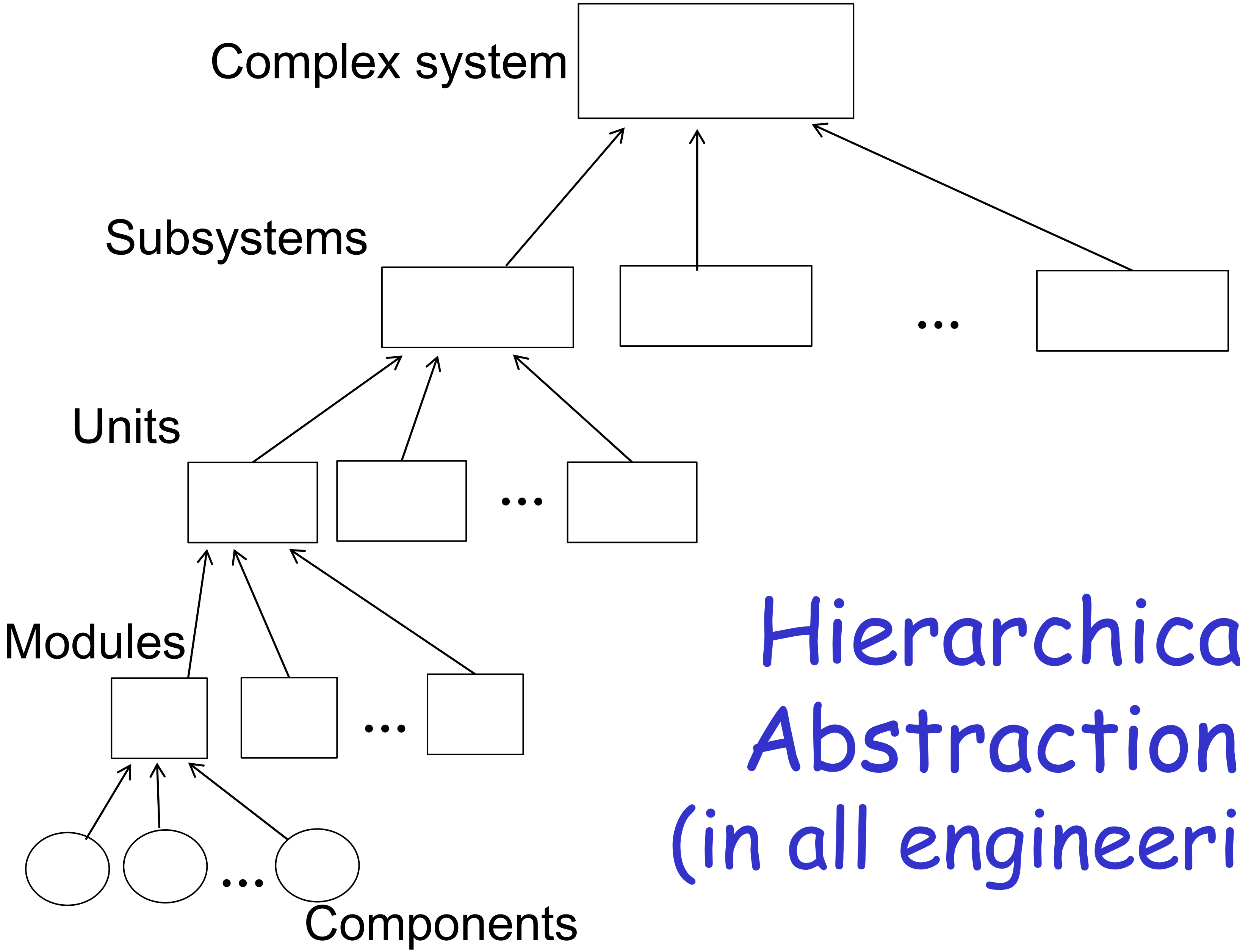
Machines (CPUs)

Abstraction (반복)

- Machine-level programming
  - "Interface" (사용법): machine instructions
  - Implementation (구현): machines (CPUs)
- High-level programming
  - Interface (사용법): programming languages
  - Implementation: compilers (or interpreters)
- 제품이 제공하는 I/F (or 사용법 or service or abstraction)
- 무엇을 먼저: I/F design or implementation?
  - 우리 수업 3대 세부 목표

Engineering: Building Abstractions

- Complex engineering products (예: SW, 자동차, 건물)
  - 작은 부품들로 복잡한 모듈 만듬, 이들로 더 복잡한 ...
    - Hierarchical abstraction building



Hierarchical Abstractions (in all engineering)

Engineering: Building Abstractions

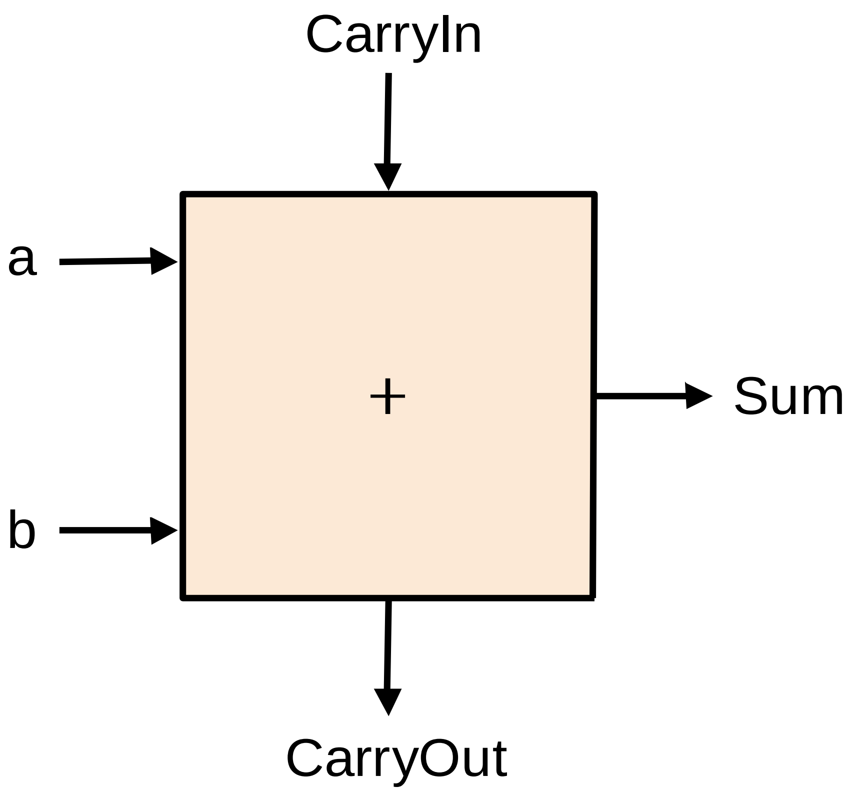
- Computer Science example
  - Hierarchical abstractions in ALU design
    - Textbook Chapter 3
  - Digital logic design
    - AND, OR, NOT 이용하여 유용한 자동장치 만듬

Add Binary Numbers

- Multiple 1-bit full adders
  - Inputs: two bits to add, carry from right
  - Output: sum, carry to left

$$
\begin{array}{rcl}
 & & 0\,0\,0\,1\;\;0\,0\,0 \qquad \text{carry} \\
9_{10} & = & 0\,0\,0\,0\;\;1\,0\,0\,1 \\
12_{10} & = & 0\,0\,0\,0\;\;1\,1\,0\,0 \\
\hline
 & & 0\,0\,0\,1\;\;0\,1\,0\,1 \quad = \; 21_{10}
\end{array}
$$

1-bit Full Adder Design



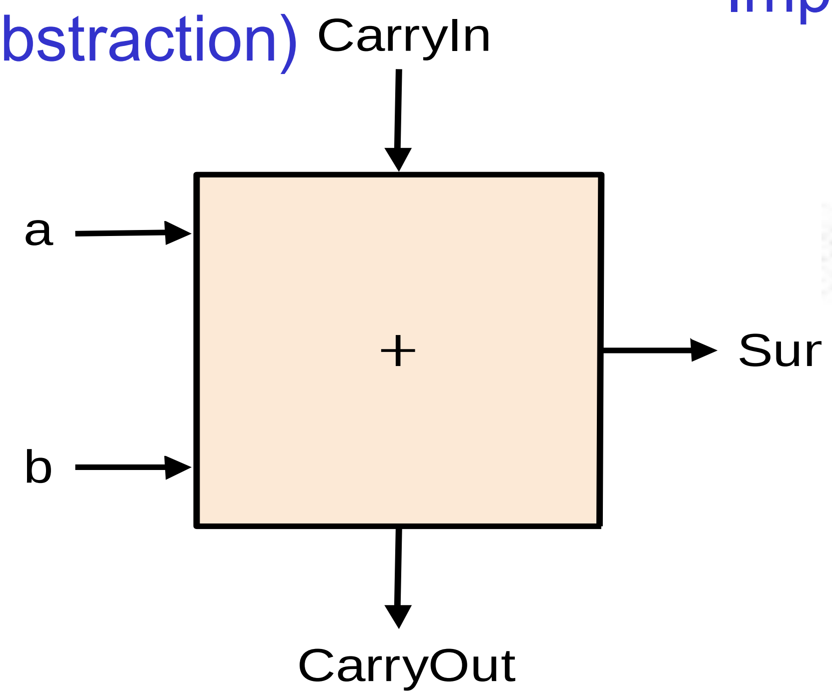
a	b	c_{in}	S	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$c_{out} = a \cdot b + a \cdot c_{in} + b \cdot c_{in}$$

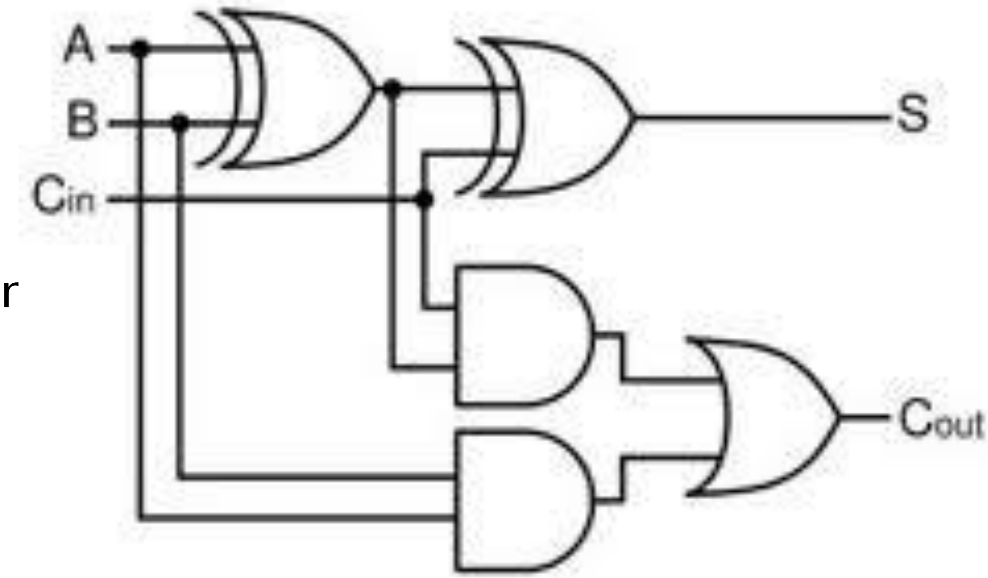
$$sum = a \text{ xor } b \text{ xor } c_{in}$$

1-bit Full Adder Design

Interface
(abstraction)



Implementation



$$c_{out} = a \cdot b + a \cdot c_{in} + b \cdot c_{in}$$
$$sum = a \text{ xor } b \text{ xor } c_{in}$$

- 사용 가능 primitives: AND, OR, NOT

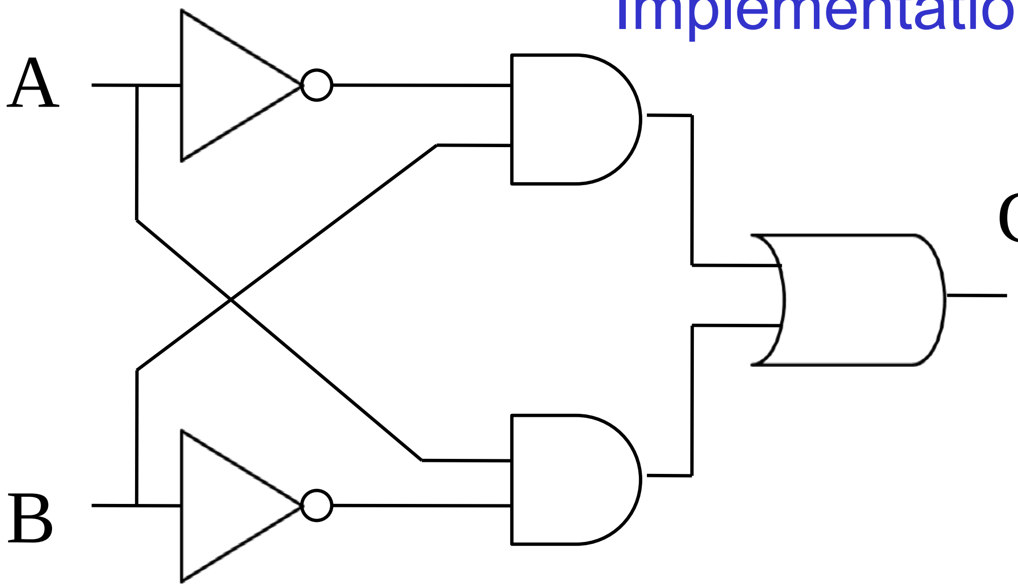
XOR (Exclusive-OR) Gate

- C = A XOR B = A ⊕ B

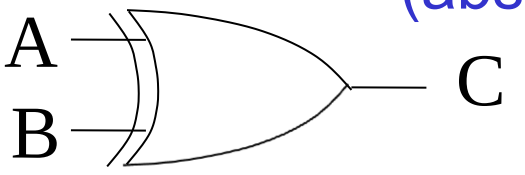
A	B	A ⊕ B
1	1	0
1	0	1
0	1	1
0	0	0

$$C = A \cdot \bar{B} + \bar{A} \cdot B$$

Implementation



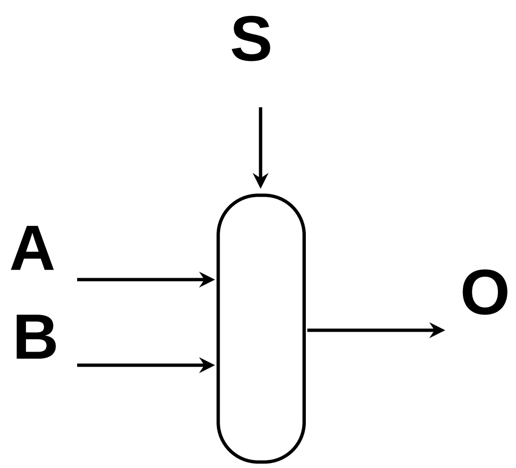
Interface (abstraction)

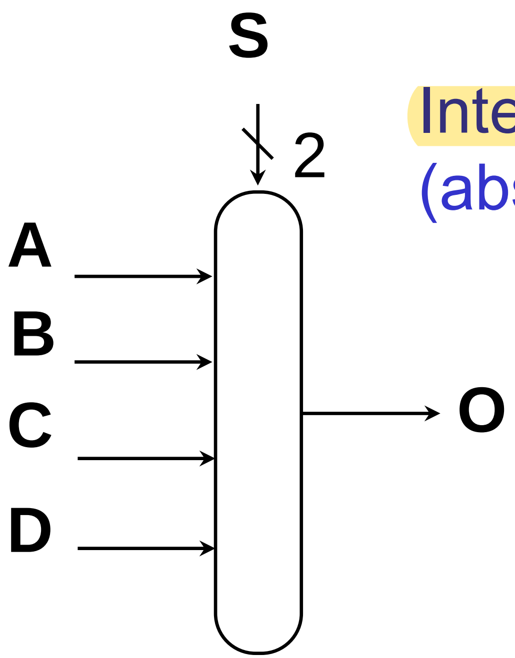


- 사용 가능 primitives: AND, OR, NOT, XOR

Multiplexors (Data Selectors)

- 2-to-1 MUX, 4-to-1 MUX

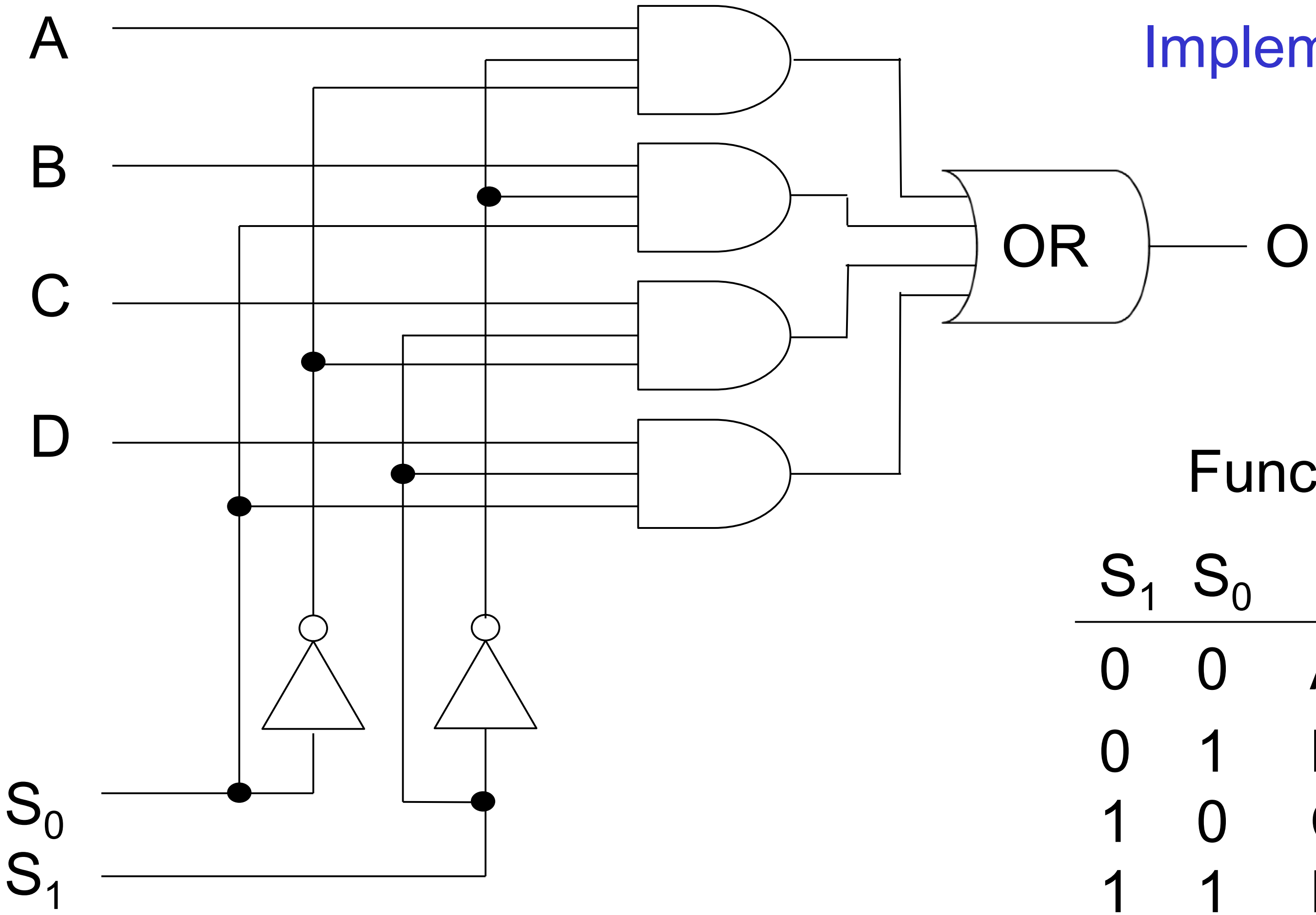




Interface
(abstraction)

- Primitives: AND, OR, NOT, XOR, MUX

4-to-1 Multiplexor



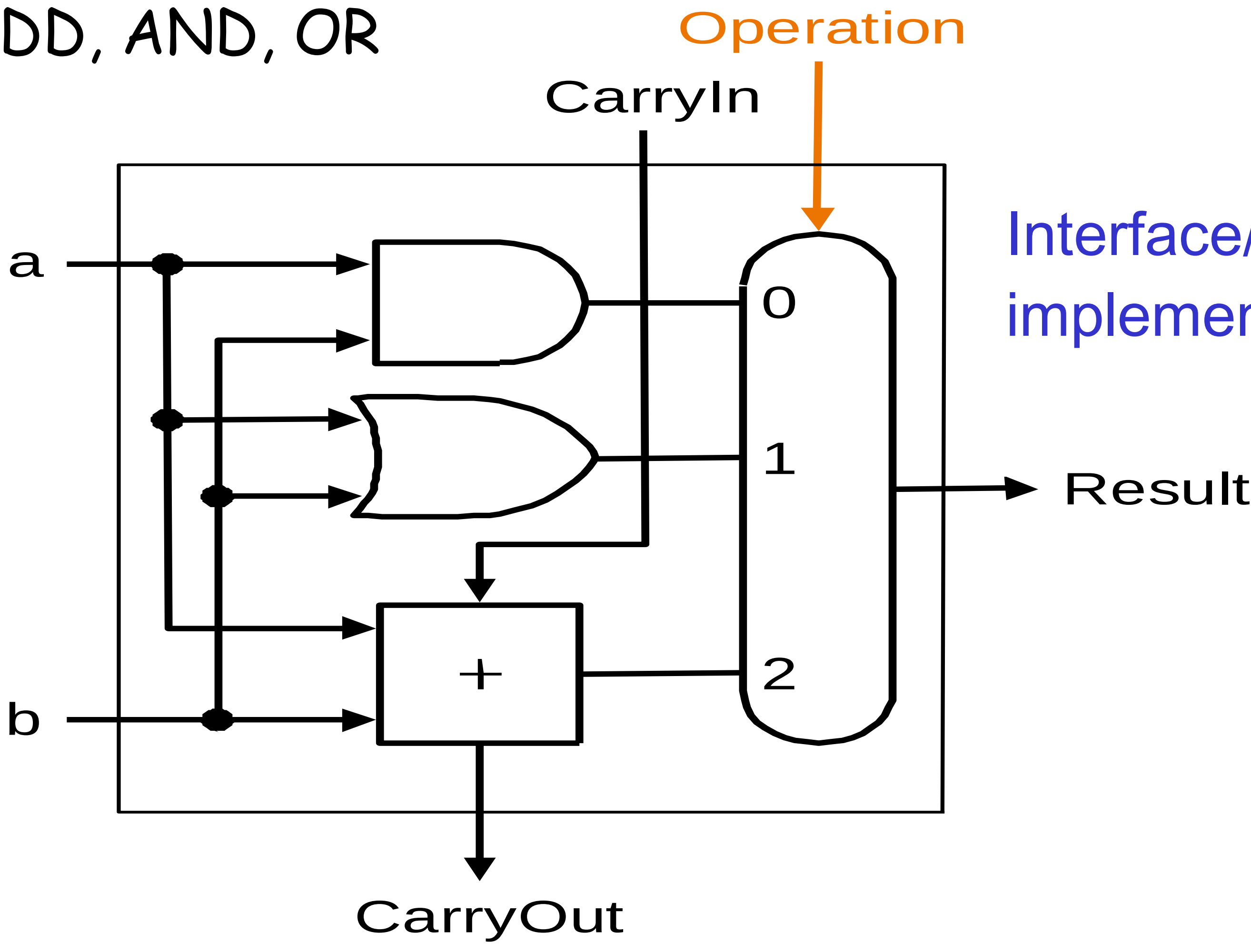
Implementation

Function Table

S_1	S_0	O
0	0	A
0	1	B
1	0	C
1	1	D

1-bit ALU Design

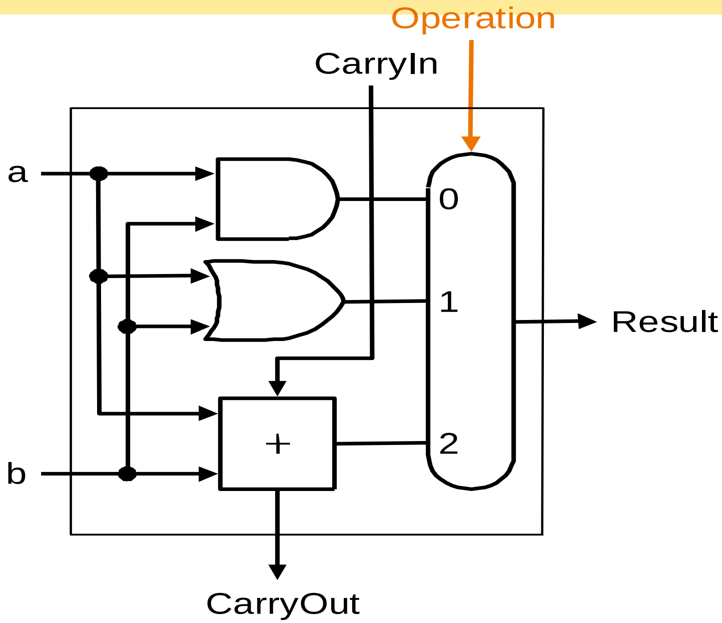
- 1-bit ADD, AND, OR

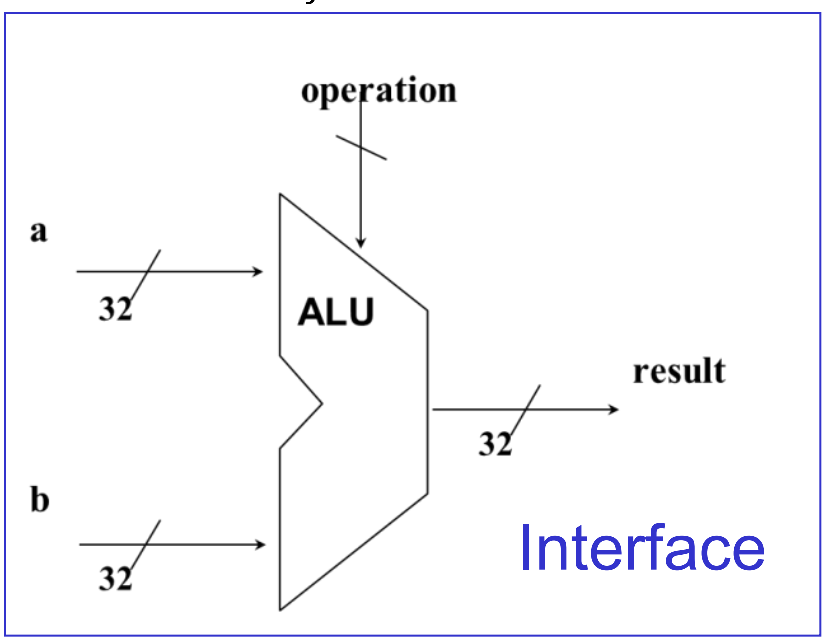


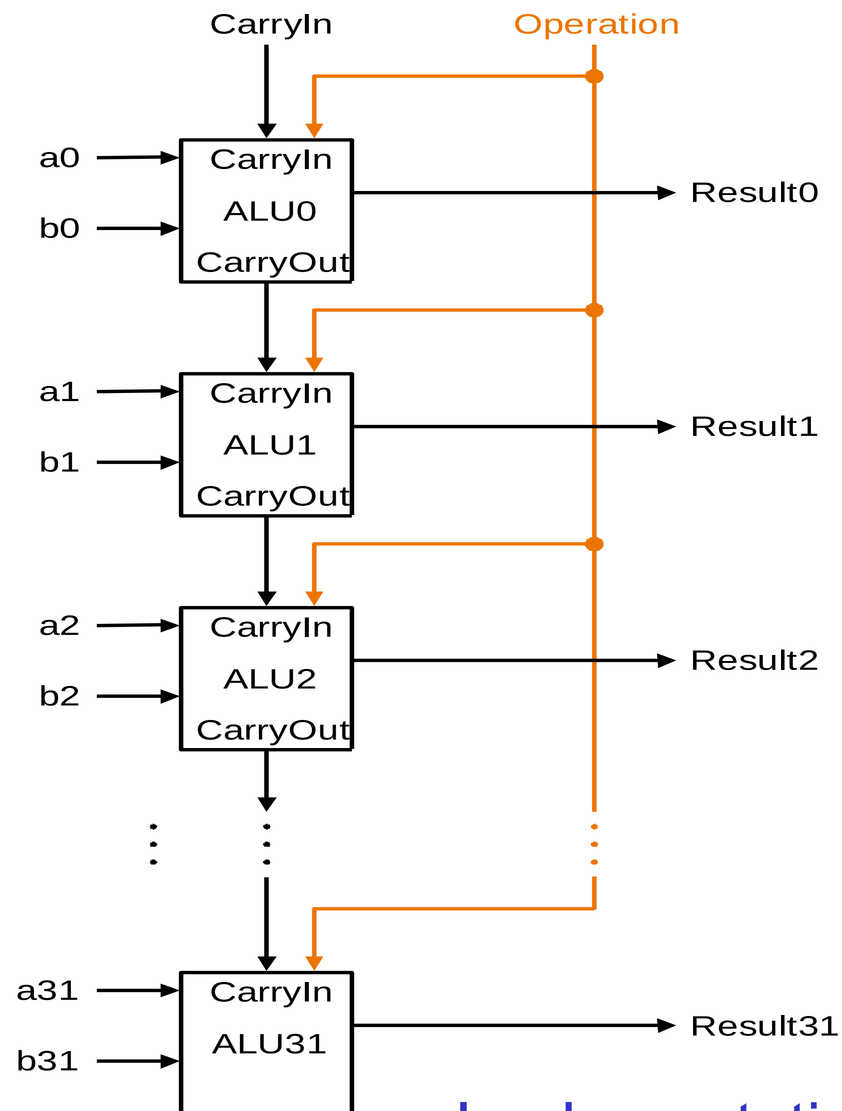
Interface/
implementation

- Primitives: AND, OR, NOT, XOR, MUX, 1-bit ALU

32-bit ALU Design







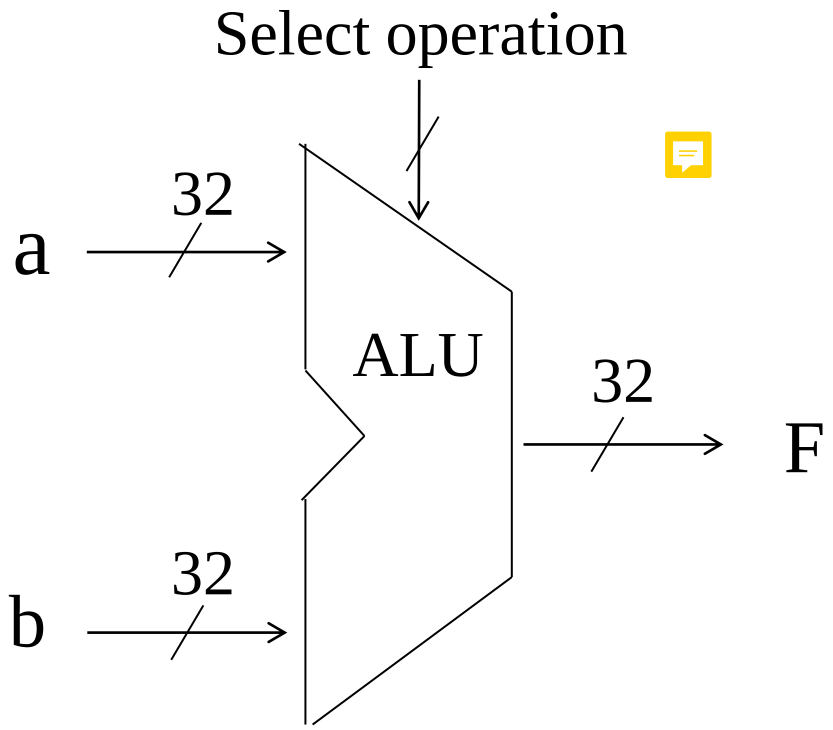
Implementation

Primitive-Composition-Abstraction

- Fundamental paradigm of digital logic design
  - Primitives: AND, OR, NOT
  - Composition: build function unit (FU) using gates
  - Abstraction
    - Given its interface, can use FU
    - Functional unit (FU) become primitive
- What is hardware design
  - Hierarchically build (more & more complex) abstractions
    - True in all engineering

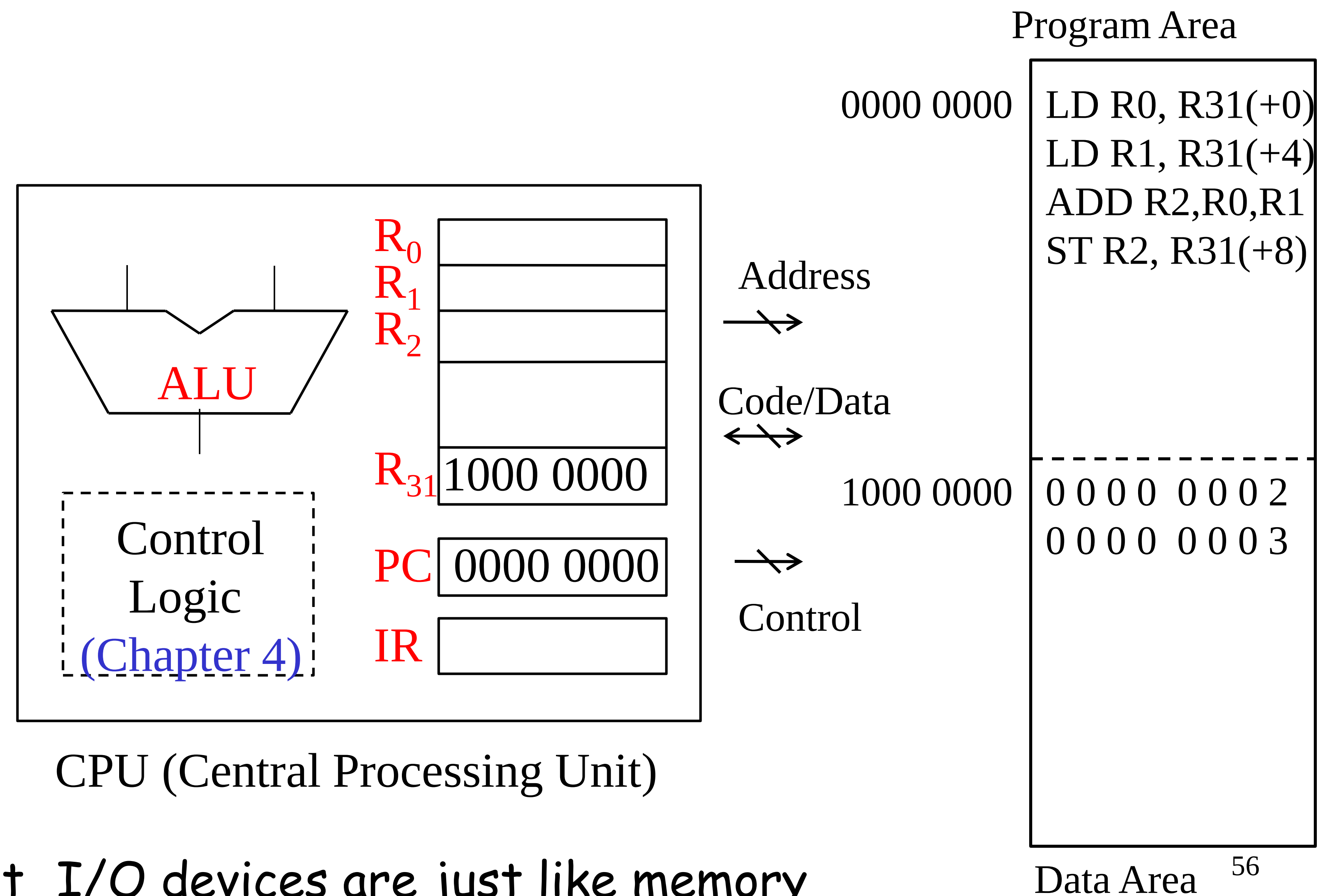
32-bit ALU

- Operations
  - Arithmetic: add, subtract, multiply, divide
  - Logical: bitwise AND, OR, NOT



Interface (abstraction; primitive)

Machines Called Computers (미리보기)



† I/O devices are just like memory

Inside a CPU (미리보기)

- ALU (arithmetic and logic unit)
  - Add, subtract, multiply, divide, AND, OR, NOT
- Registers: storage of temporary data
- PC (program counter)
  - Address of the next instruction to execute
- IR (instruction register)
  - Instruction being executed
- Control logic
  - The rest of CPU for "fetch-decode-execute"

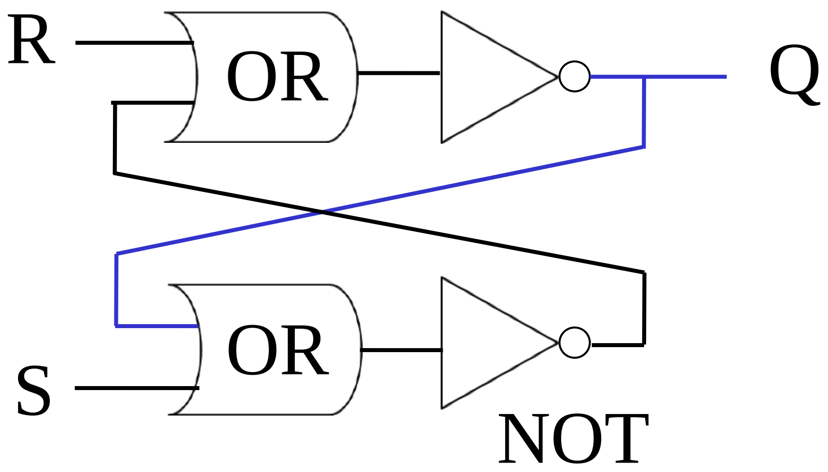
Storage (Registers and Memory)

- Notion of "Address"

(메모리: AND, OR, NOT 기반의 자동장치)

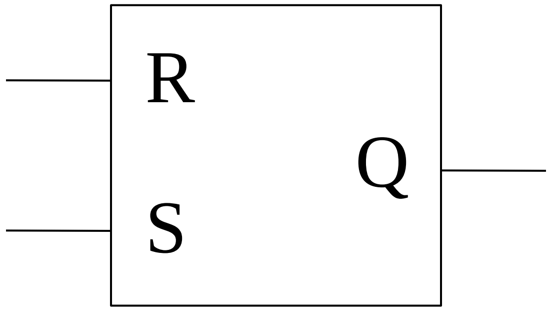
SR Flip-Flop (구조나 동작을 암기할 필요 없음)

- Two stable states (invention of flip-flop in 1918)



Implementation

S	R	Q	action
0	0	Q	hold
1	0	1	set
0	1	0	reset
1	1	not allowed	

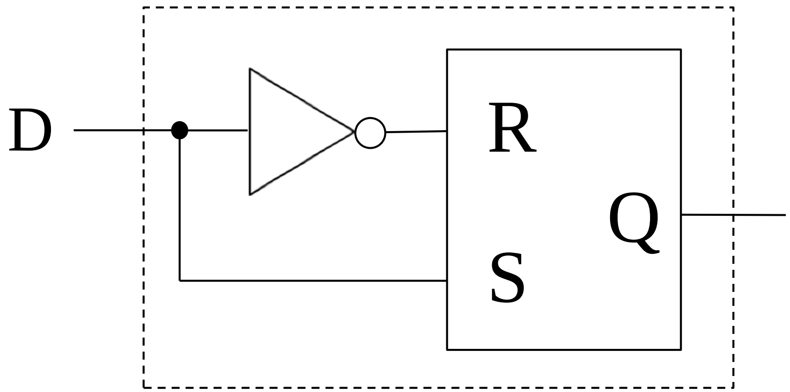


Interface

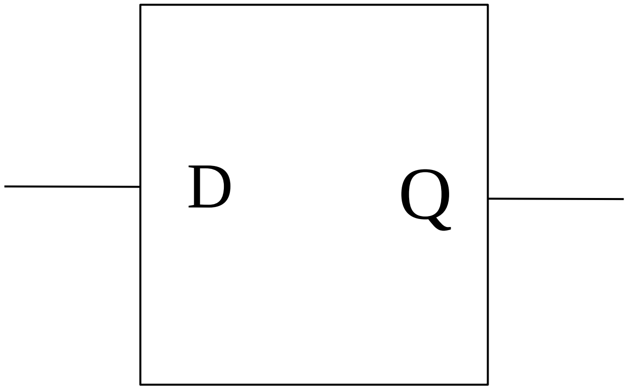
- Read Q with S=0, R=0
- Write to Q
  - To store 1: S=1, R=0
  - To store 0: S=0, R=1

D Flip-Flop (skip)

- Different types of flip-flops



implementation

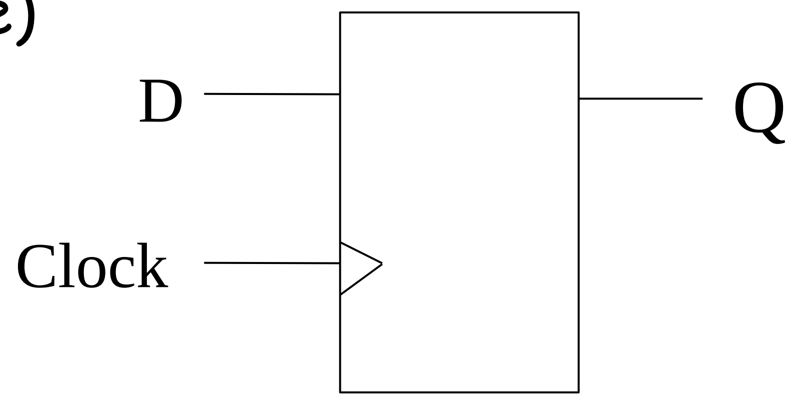


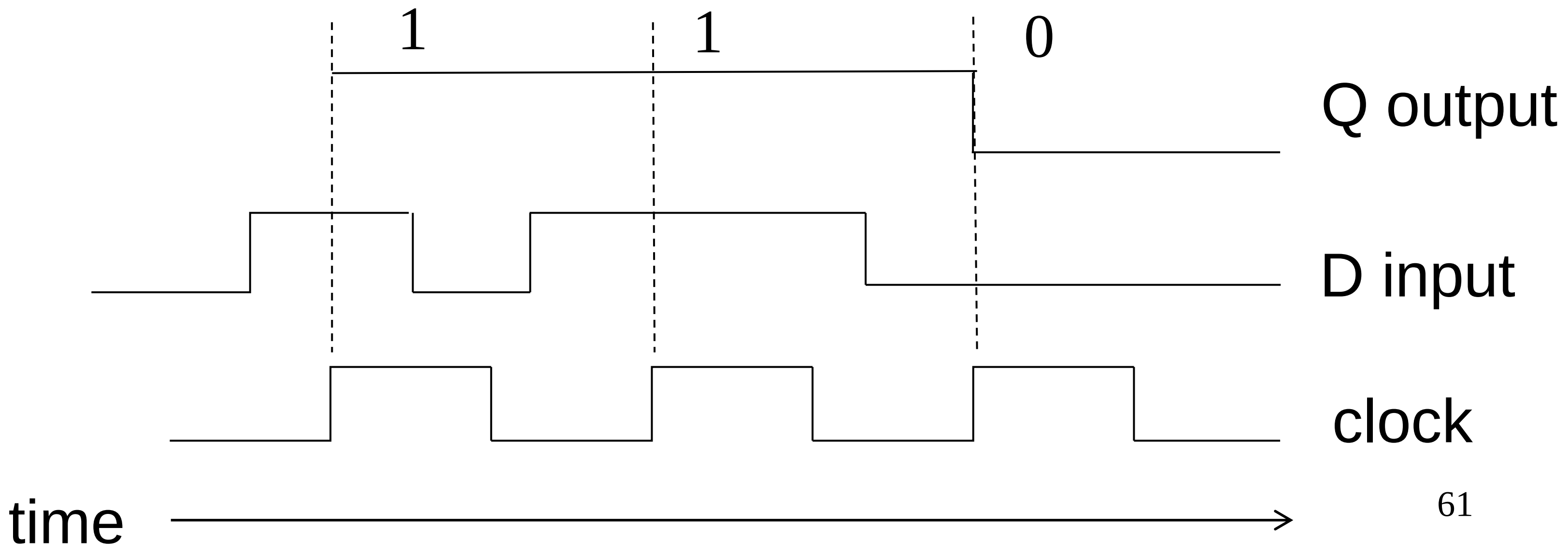
Interface

D	Q	action
0	0	reset
1	1	set

Clocked D Flip-Flop (skip)

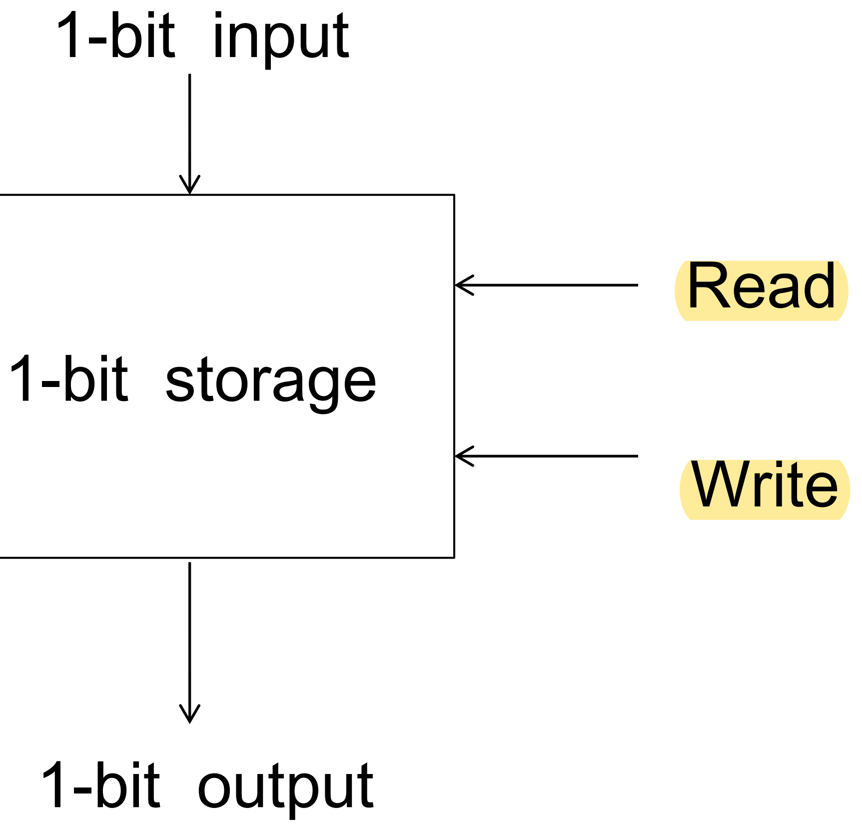
- Edge-trigger (rising edge)





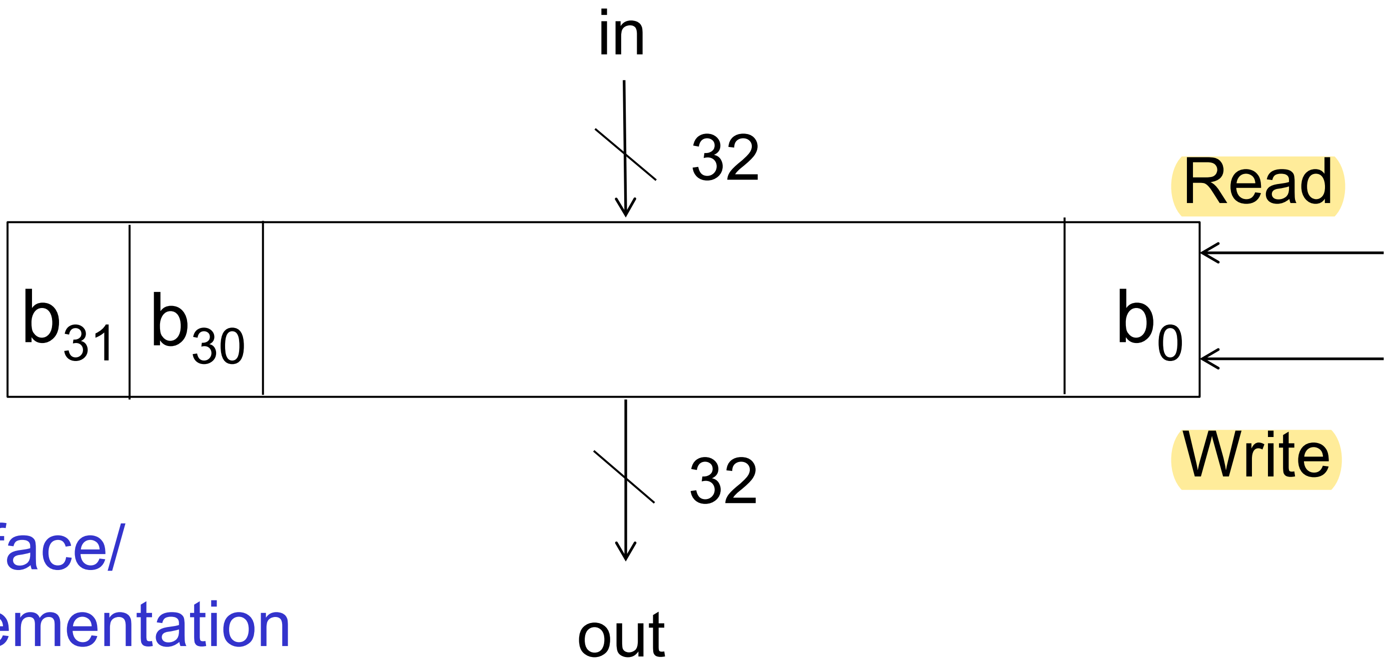
1-bit Storage (Clock 생략)

- Abstraction



32-bit Storage

- Use 32 of 1-bit storages in parallel (share "address")

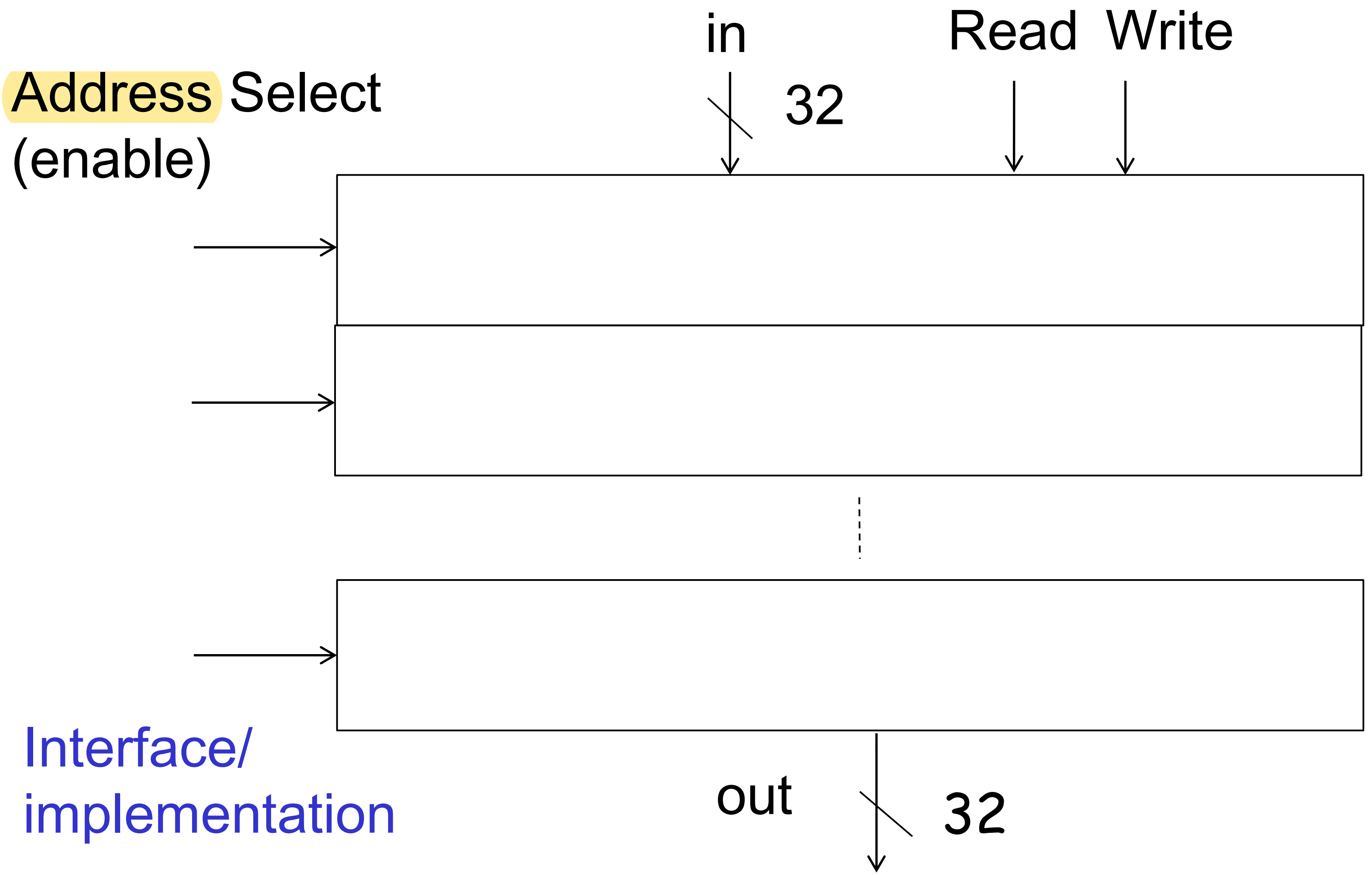


Interface/
implementation

- Registers: 32-bit storage in a processor

Main Memory in 32-bit Computers

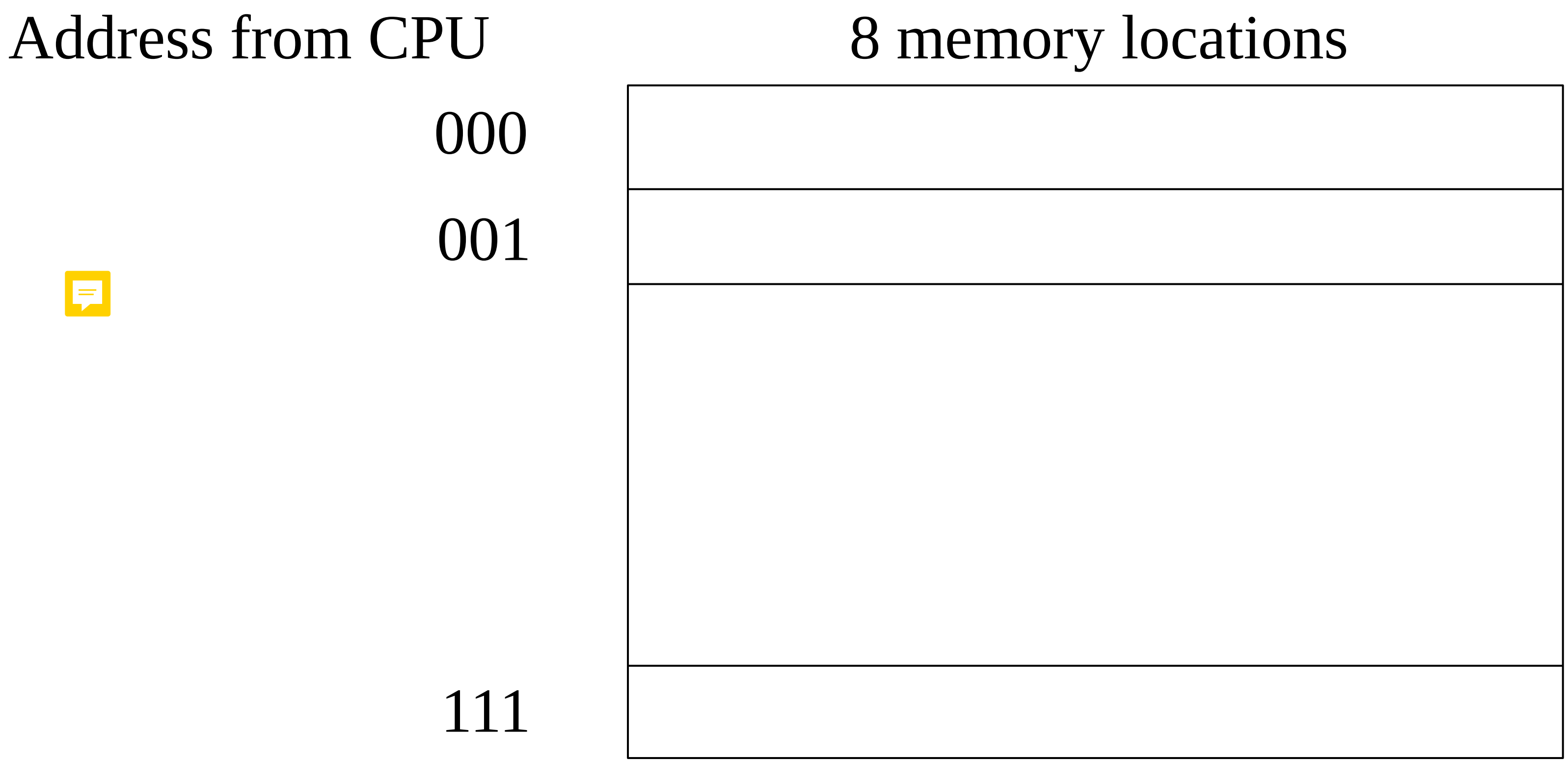
- Many locations – each has distinct address



Interface/
implementation

Meaning of Address

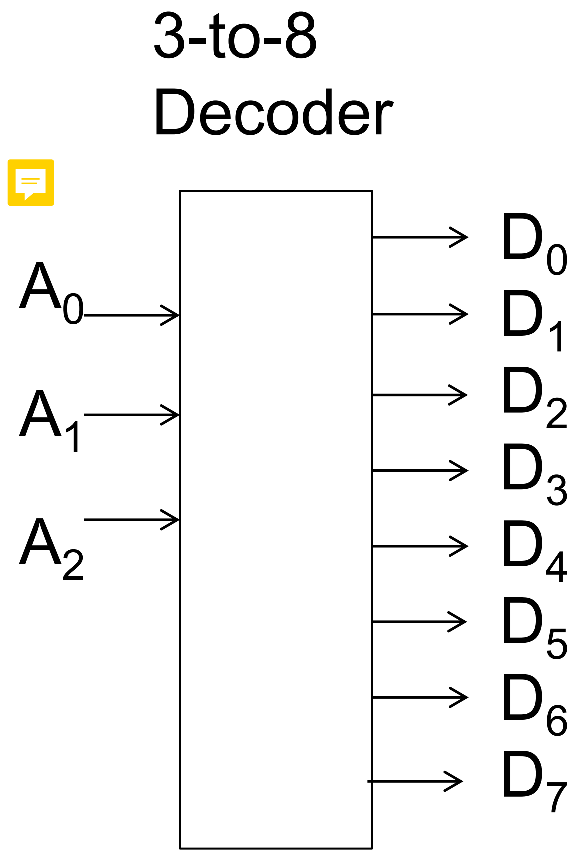
- Unique identifier for locations



- 3-to-8 decoder

3-to-8 Decoder (복습)

A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

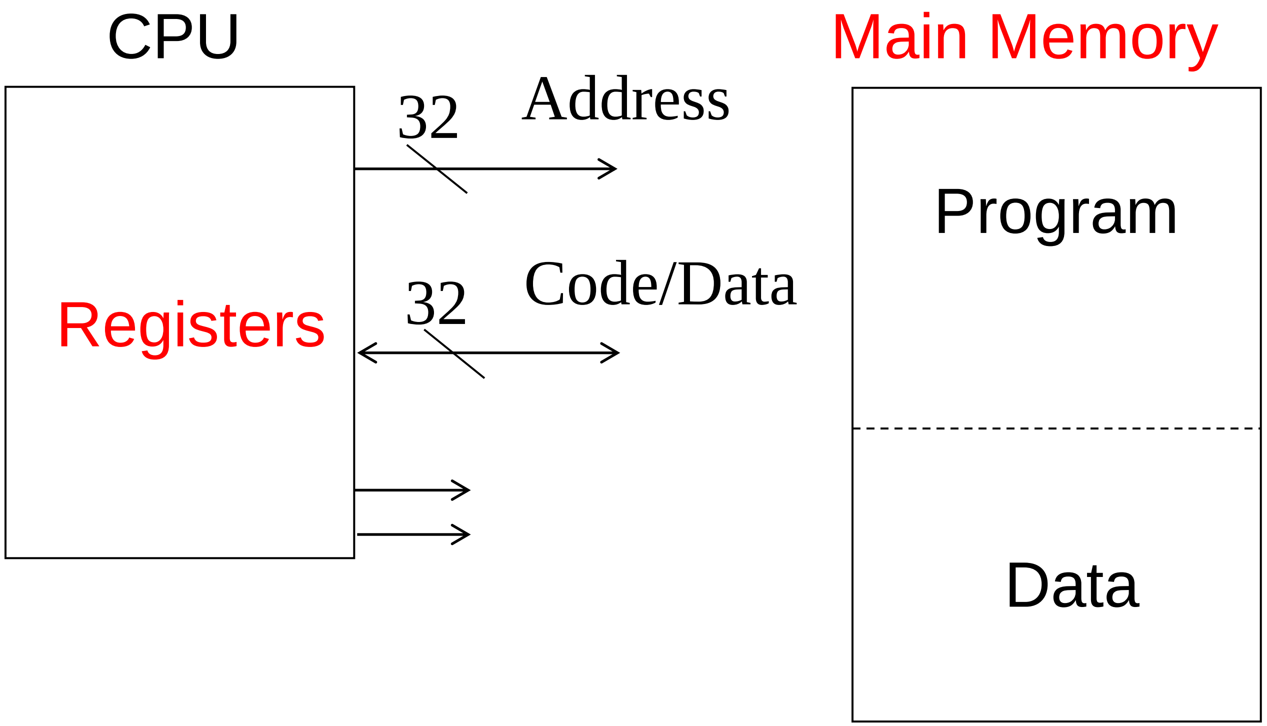


- 2-to-4 decoder, 4-to-16 decoder, …

Number of Address Bits

- 256 = 2^8 memory locations: 8-bit address
- 64K = 2^{16} memory locations: 16-bit address
  - 8-bit microprocessor
- 4G = 2^{32} memory locations: 32-bit address
  - 32-bit processor
- What is a pointer?
  - Indirection, machine-level concept

32-bit Computers



I/O device 0 (e.g., disk)
I/O device 1 (e.g., monitor)

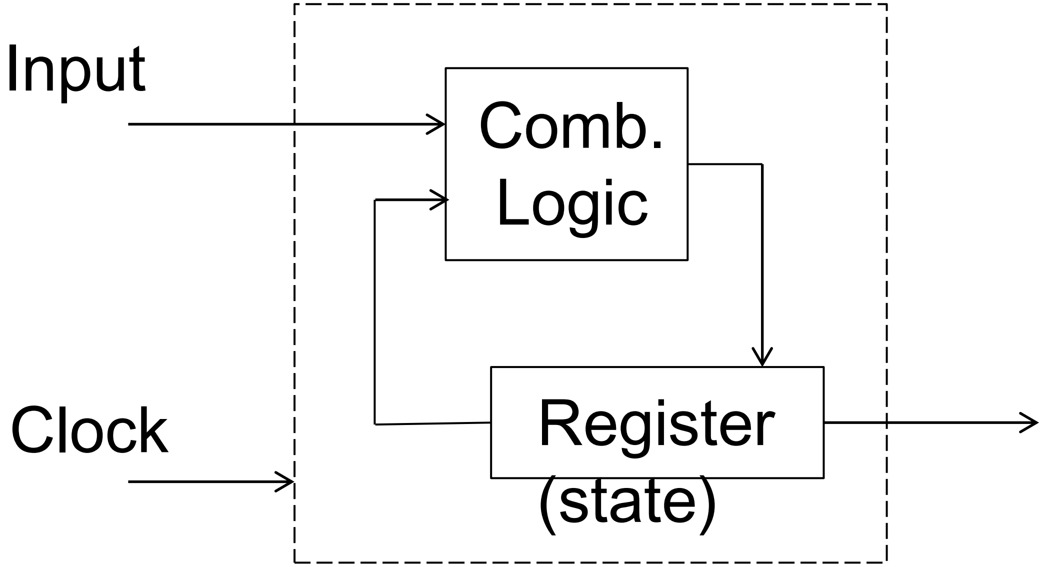
- 4G = 2^{32} memory and I/O locations
- Given address, enable corresponding location

Sequential Logic Design

CPU, Memory
(AND, OR, NOT 기반의 자동장치)
(IF 개념은 곧 다시 나옴)

Sequential Logic Design

- Registers (simplest ones), binary counters

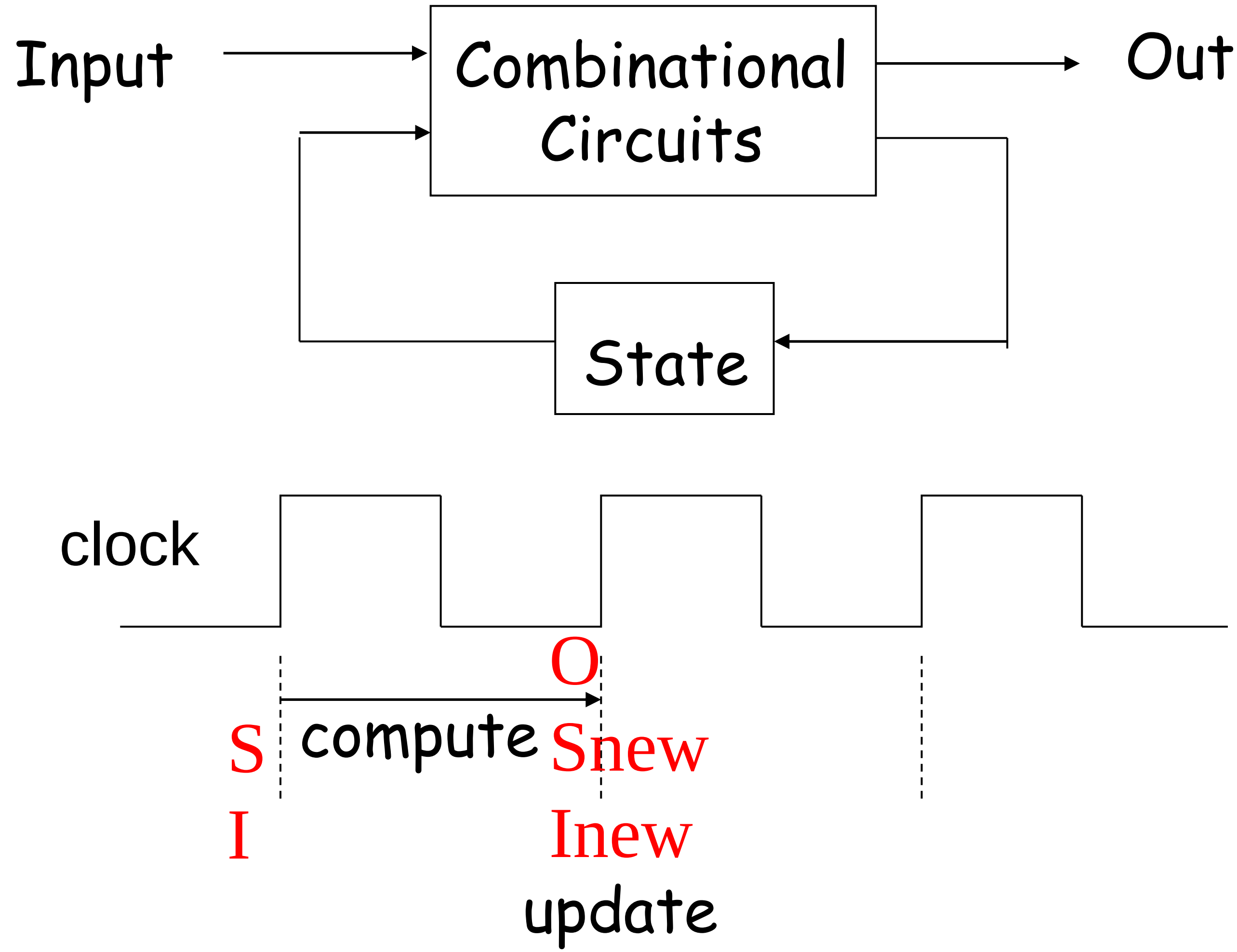


Out = f(in, state)

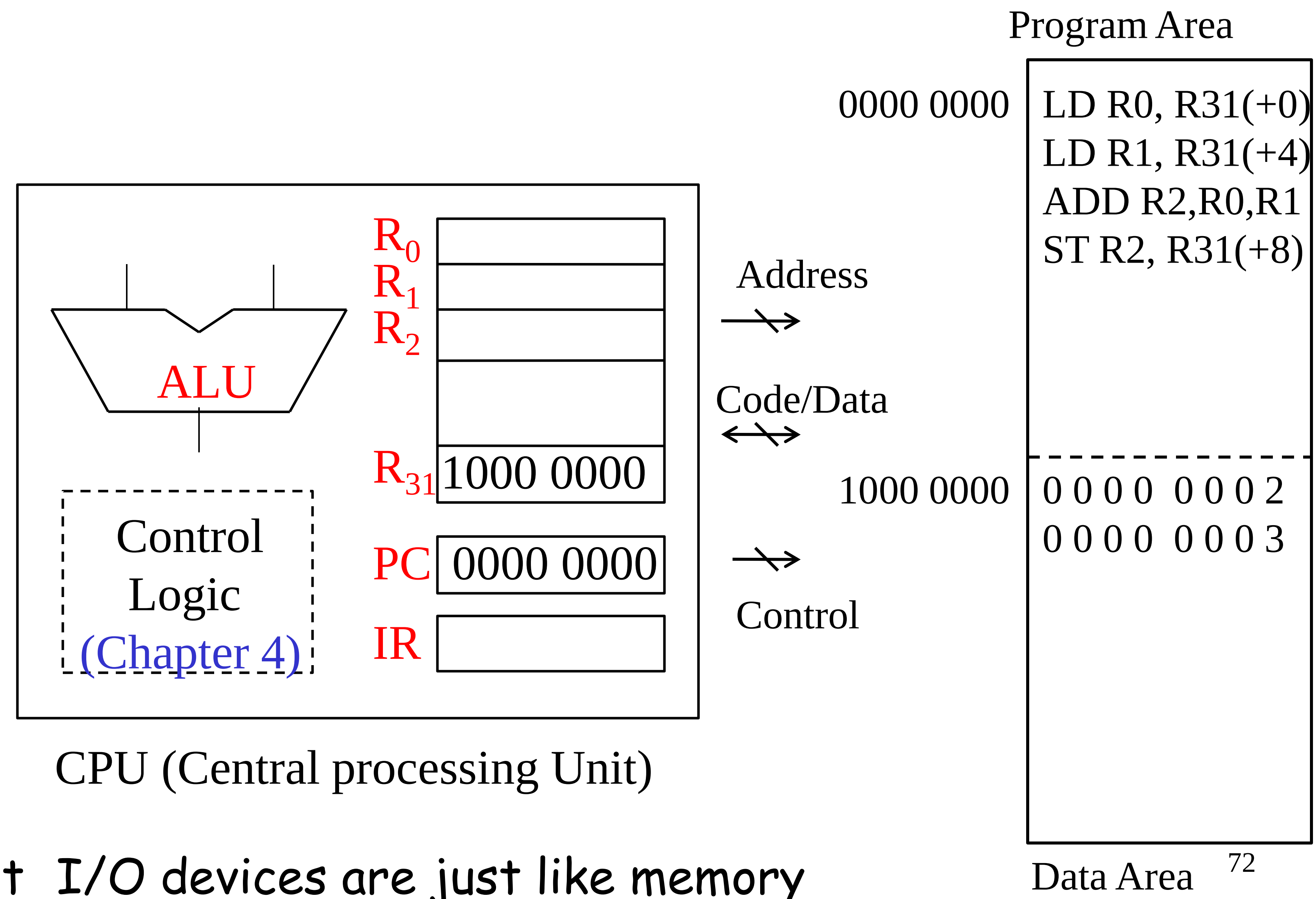
Next state = g(in, state)

- Sequential logic design more complex than comb. Logic
  - Truth table vs. state diagram

Synchronous Sequential Logic Circuits



Machines Called Computers



- I/O devices are just like memory

Class Topics (클래스 홈페이지 참조)

- Part 1: Fundamental concepts and principles
  1) Invention of computers and digital logic design
  2) Abstractions to deal with complexity
  3) Data (versus code)
  4) Machines called computers
  5) Underlying technology and evolution since 1945
- Part 2: 빠른 컴퓨터를 위한 설계 (ISA design)
- Part 3: 빠른 컴퓨터를 위한 구현 (ISA implementation)