Machine Called Computer

Part 2

- Invention of computer

- 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)

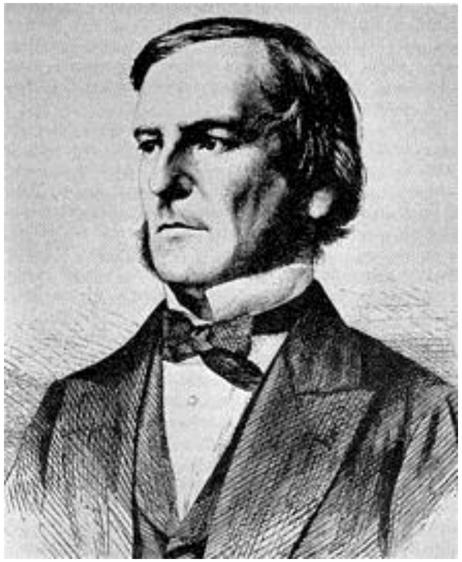
Big Picture

- 컴퓨터의 탄생
  - 과학적 성취 (수학적 개념의 창조)
    - Boole in 19C
  - 실용적 도구 개발 - 산업혁명의 맥
    - Automata (자동장치) 개발
    - 자동 계산기 (calculator) 개발
  - 구현 기술 발전
    - Transistor 발명 (wheel/shaft/cam, relays, 진공관)
- Abstraction: fundamental engineering concept

George Boole

- 19C English mathematician, philosopher, logician
- "The Laws of Thought" in 1854
  - Proposition (명제)
    - Binary (True, False; "1", "0")
  - AND, OR, NOT, IF
  - † To throw light on nature of human mind?

명제들을 AND,OR,NOT,IF로 표현할 수 있다.



Public domain image

- Impact on computer science and mathematics
  - Foundation of mathematical logic
    - vs. traditional logic (syllogism by Aristotle, B.C. 4C)
  - Boolean Algebra

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 · q (AND operation)
  - p + q (OR)
  - $\underline{P}$ (NOT; usually overline or bar)
  - p -> q (IF)

Truth Table

- 1 = True; 0 = False

AND

p	q	p · 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	$\underline{p}$
1	0
0	1

IF

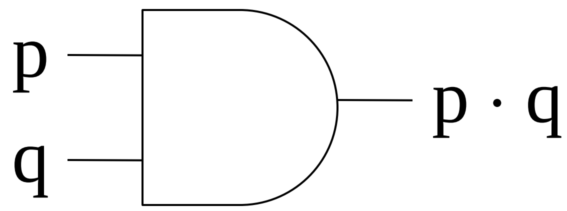
p	q	$p \rightarrow q$
1	1	1
1	0	0
0	1	1
0	0	1

In Your Mind - Digital Logic Gates

- Can implement AND, OR, NOT with electronic circuits

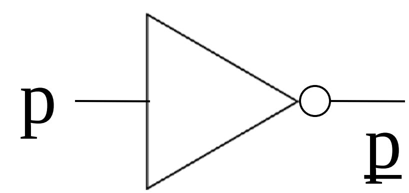
AND

p	q	p · q
1	1	1
1	0	0
0	1	0
0	0	0



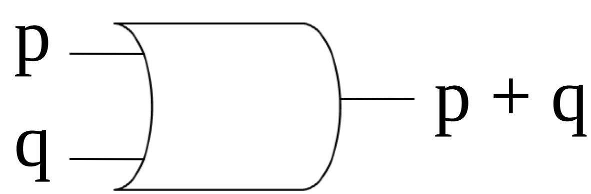
NOT

p	$\underline{p}$
1	0
0	1



OR

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



Boolean Algebra (불 대수)

Set of values {0, 1}

Set of operations: AND, OR, NOT

(복잡한 자동장치 설계 위한 수학적 기반 제공)

Algebra

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

$$2 + 3 = 5$$

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

- Modern algebra (e.g., Boolean Algebra)
  - Operations and relations that are defined on set of mathematical objects

Boolean Algebra

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

      $\underline{E_1}$, $E_1 \cdot E_2$, $E_1 + E_2$ are Boolean expressions

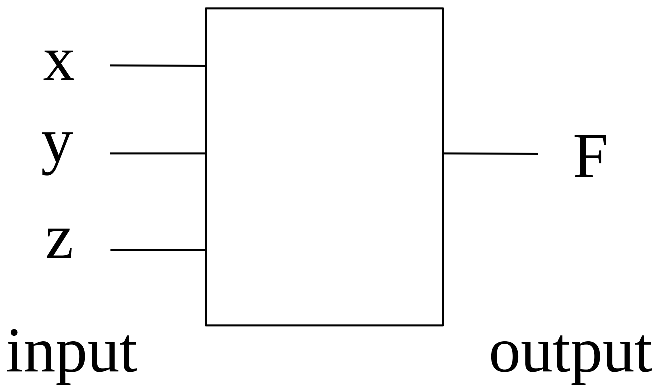
† Compare with arithmetic expressions

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

Boolean Algebra

- Axioms → Theorems (or knowledge)
- Given truth table, find Boolean expression for function?
  - F = f(x,y,z)

† Practical view

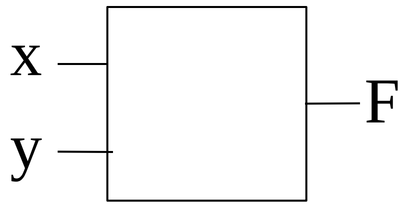


x	y	z	F
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

Exercises - Boolean Expression

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

$F = x \cdot y$



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

$F = (x \cdot y) + (\underline{x} \cdot y)$

Multiple outputs:

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

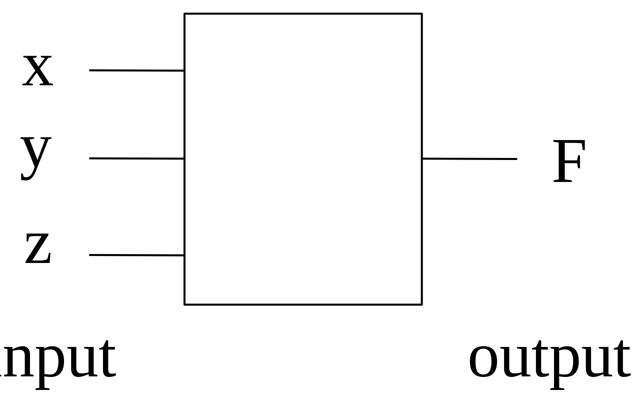
$F1 = x + y$

$F2 = \underline{x + y}$

Boolean Algebra

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

† Practical view



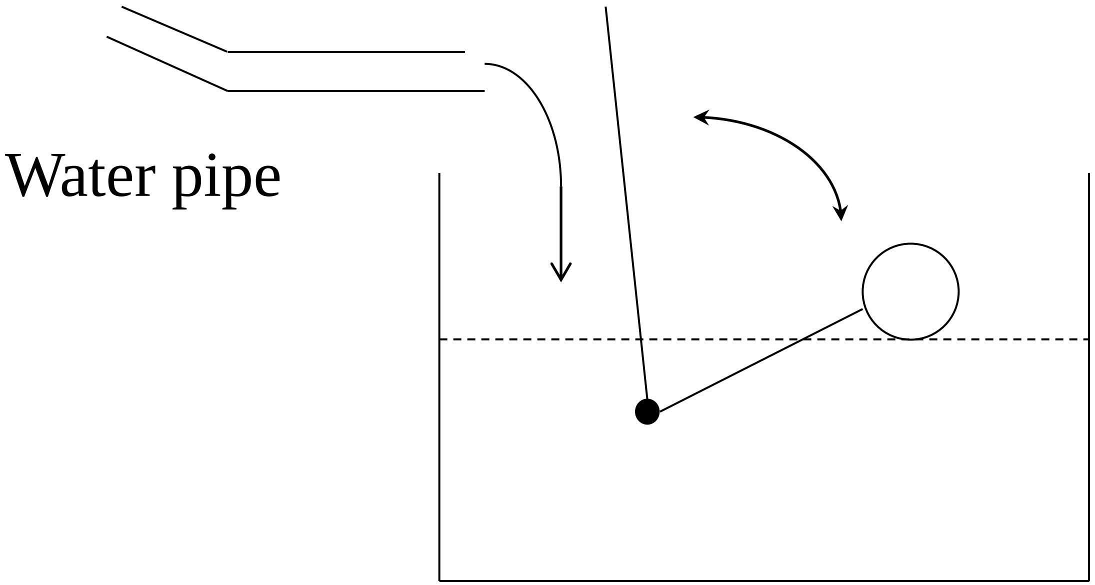
x	y	z	F
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

Automata (자동장치) Design and Boolean Algebra

(복잡한 자동장치의 체계적 설계)

Simple Automaton

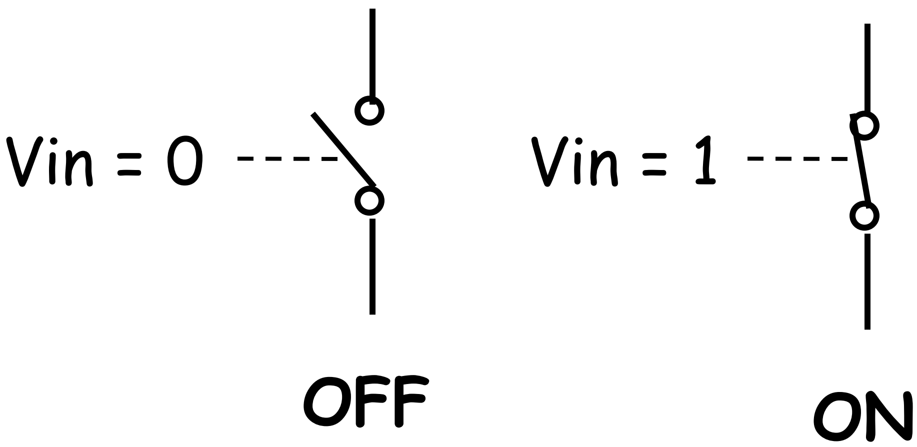
- Self-operating machine (water-level controller)



- More useful/complex automata: “sensors” and “switches”

Electronics and Digital Switch

- Need for three-terminal switching device
  - 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

Image of electromagnetic relays:
http://en.wikipedia.org/wiki/File:Relay.jpg
Image of electromagnetic relays:
http://en.wikipedia.org/wiki/File:Relay_symbols.svg

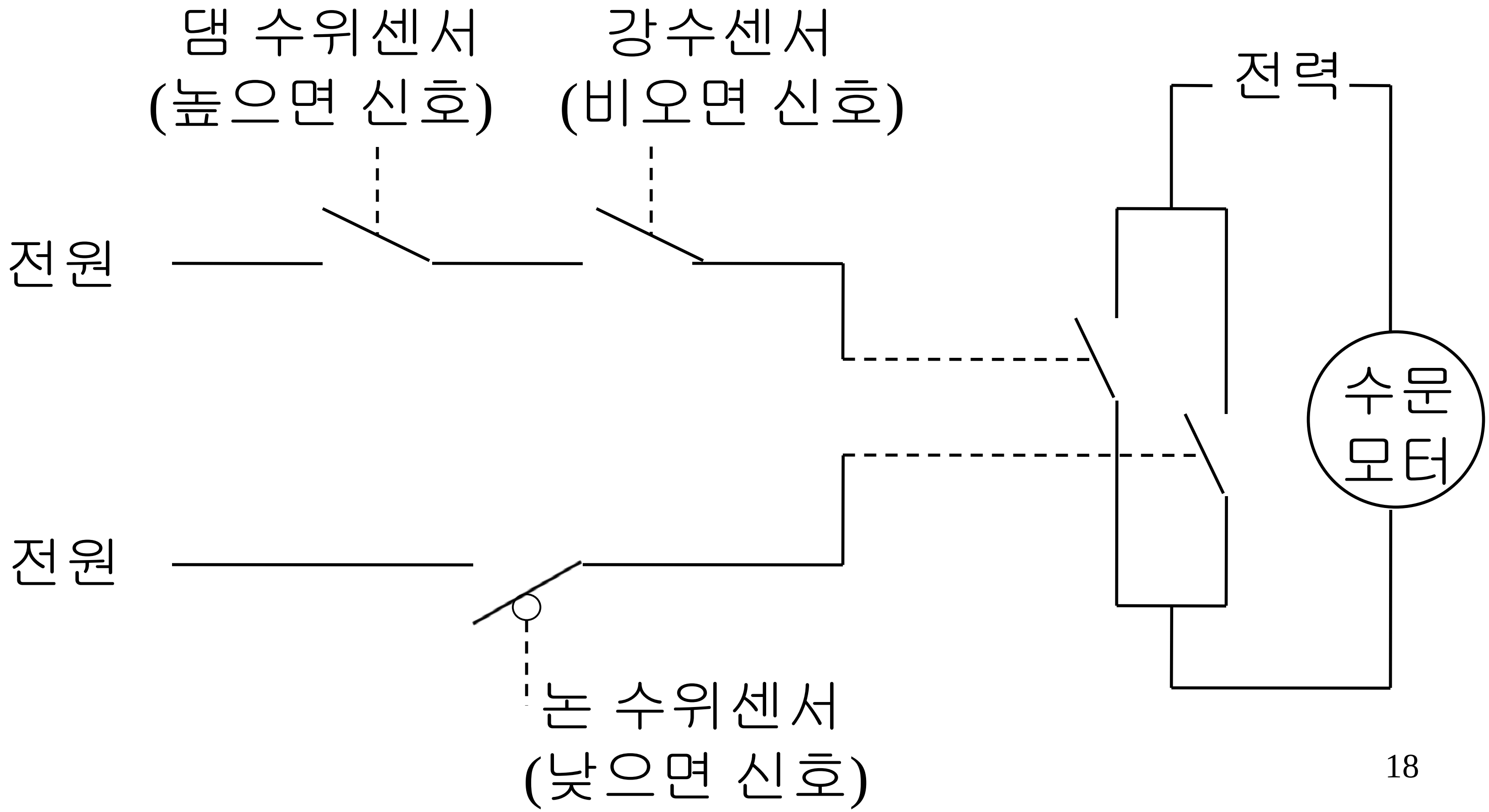
Electron or Vacuum Tube

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

Image of electronic vacuum tubes:
http://en.wikipedia.org/wiki/File:SE-300B-70W.jpg
Image of electronic vacuum tubes:
http://en.wikipedia.org/wiki/File:Triode_tube_schematic.svg

More Meaningful Automaton

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



Automata Design

- 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)

- $F = x \cdot y + \underline{z}$
  - 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

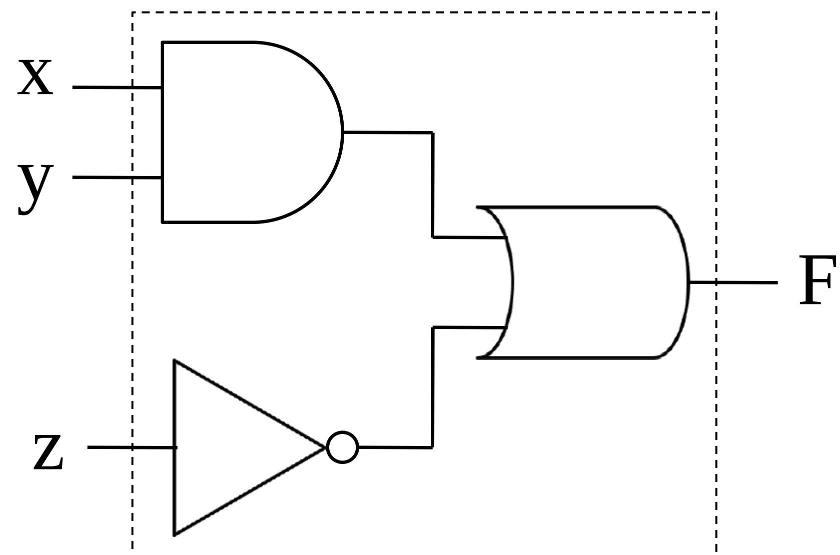
Digital Logic Design (Shannon, 1938)

- 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

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



Logic Diagram

Automata Design

- Empirical automata design
  - Ad hoc approach using 3-terminal digital switches
  - Underlying notion of AND, OR, NOT
- Relating automata design and Boolean Algebra
  - Shannon's M.S. Thesis in 1938
- Systematic design of automata
  - Think about inputs, outputs
  - Build truth table
  - Reduce to Boolean logic function
    - Readily be implemented with hardware
- Facilitate design automation (VLSI CAD tools)
  - Ultimate form of automata: ALU, processor, computer

Digital Logic Design

(Combinational Logic Design)

ALU: Complex Automaton

Combinational Logic Design

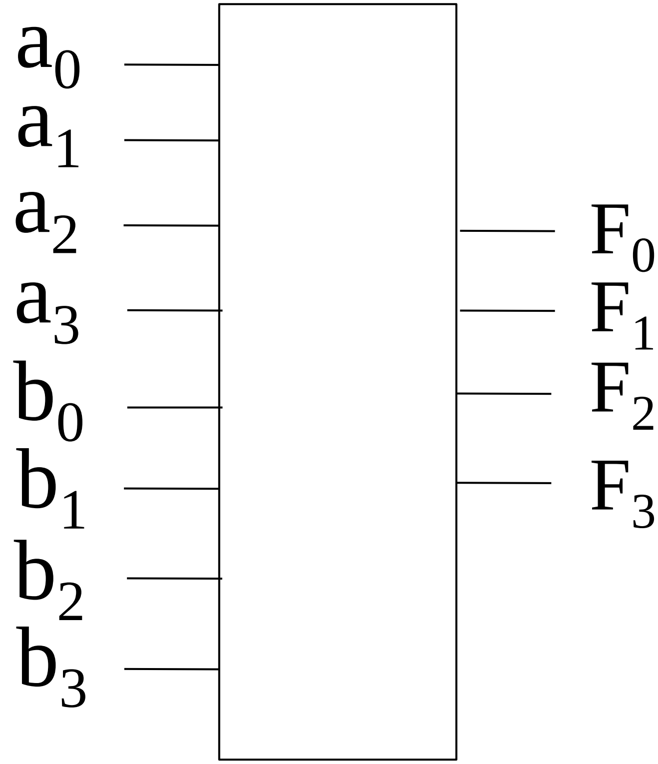
- Combinational Logic
  - Outputs completely determined by inputs
- Combinational logic design
  - Given: AND, OR, NOT gates
  - Paradigm
    - Determine input and output variables
    - Build truth table
    - Outputs: Boolean functions of input variables
      - † VLSI CAD tools

Imagine ALU design

- 4-bit adder
  - 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

$$
\begin{array}{lcr}
9_{10} & = & 1\;0\;0\;1 \\
4_{10} & = & 0\;1\;0\;0 \\
\hline
 & & 1\;1\;0\;1 \quad = \; 13_{10}
\end{array}
$$

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



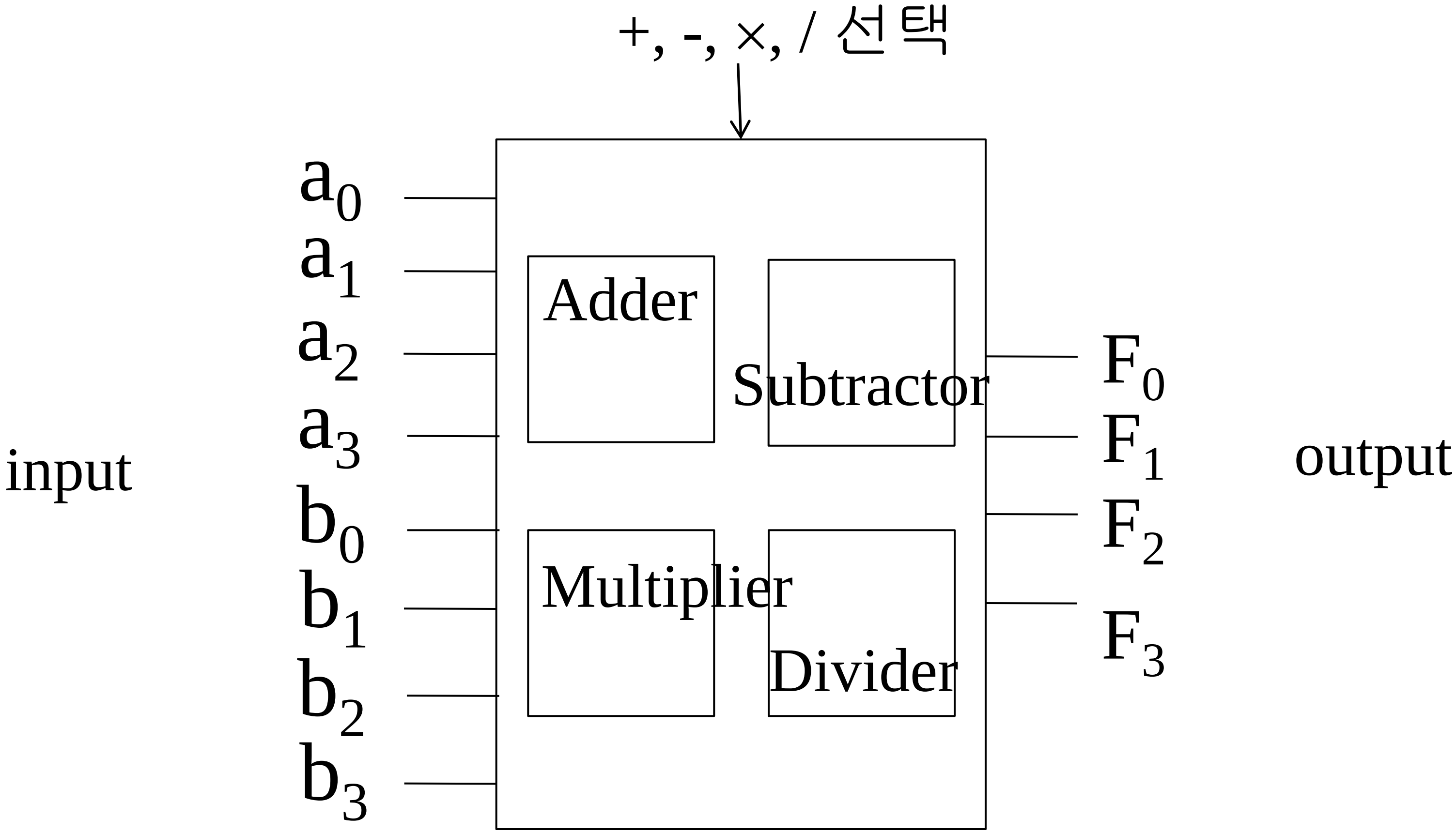
4-bit Adder

- Generate (large) truth table (with 2^8 rows)
- Find minimum Boolean expression
  - 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

복잡도를 줄여나가는 도구

(Fundamental Engineering Concept)

How to deal with complexity

Engineering: Building Abstractions

- Programmer
  - Use machine instructions for programming
    - "Interface" (사용법) 먼저 고려한다.
  - Not know computer design/organization/operation
    - "Implementation" (설계/구조/동작)
- Computer (CPU, SW) 를 포함한 모든 공학 도구/물건
  - Implementation 몰라도 interface 알면 사용 가능
    - † Fundamental concept of abstraction
- Complex engineering product (예: SW, 자동차, 건물)
  - 작은 부품들 사서 복잡한 모듈 만듬, 모듈들로 더 복잡한 ...
    - † Recursive abstraction building

사용법 external service interface
- - - - - - - - - - - - - - - - - -
물건 implementation internal
ex) IS,assemble
- - - - - - - - - - - -
processor

c language
- - - - - - - - - - -
c compile

계층적, hierarchy
ex) cpu,i/o -> mother board -> pc
더 큰 abstraction으로 확장되어간다.

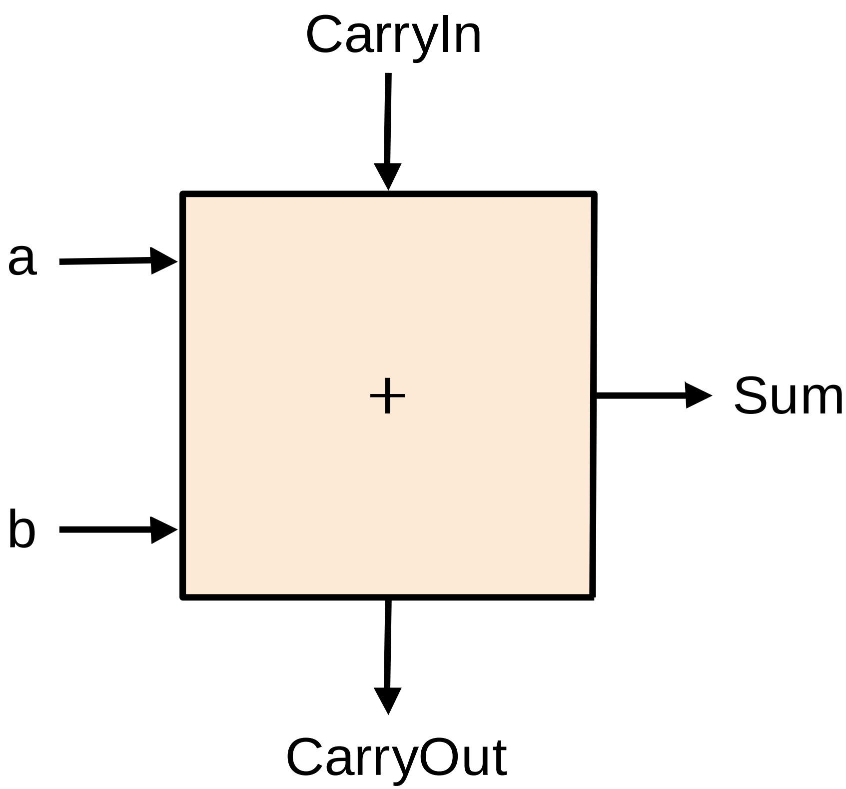
Add Binary Numbers

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

$$\begin{array}{rcccccccccl}
 & & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & \text{carry} \\
9_{10} & = & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \\
12_{10} & = & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \\
\hline
 & & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & = 21_{10}
\end{array}$$

1-bit Full Adder Design

(Source: Computer Organization and Design, Hennessy and Patterson)



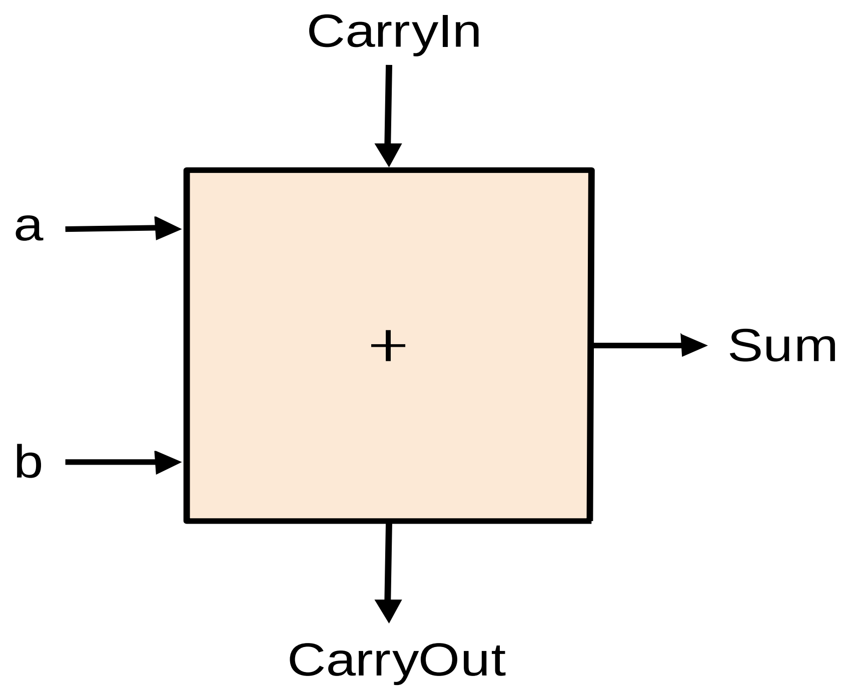
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

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

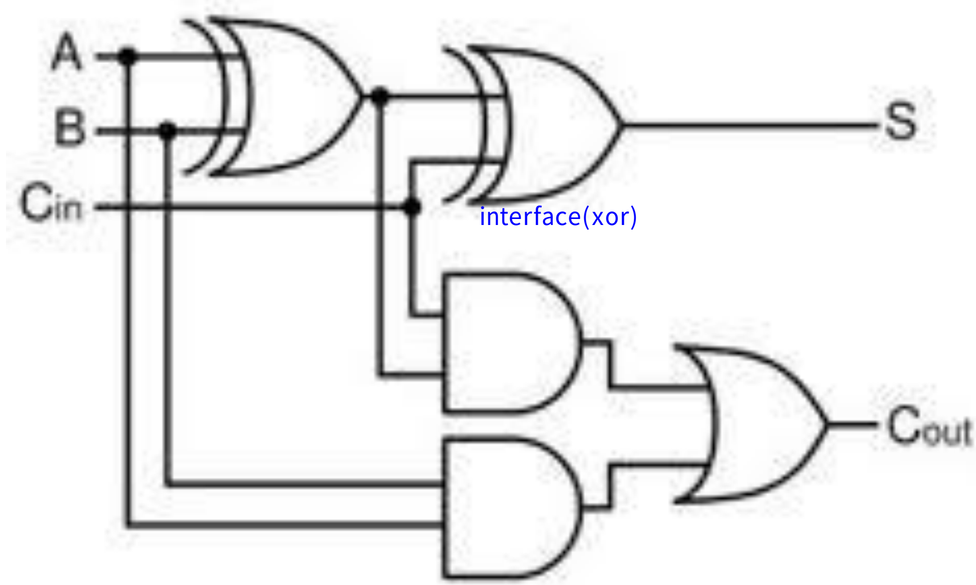
$sum = a \text{ xor } b \text{ xor } c_{in}$

1-bit Full Adder Design

(Source: Computer Organization and Design, Hennessy and Patterson)



interface



implementation

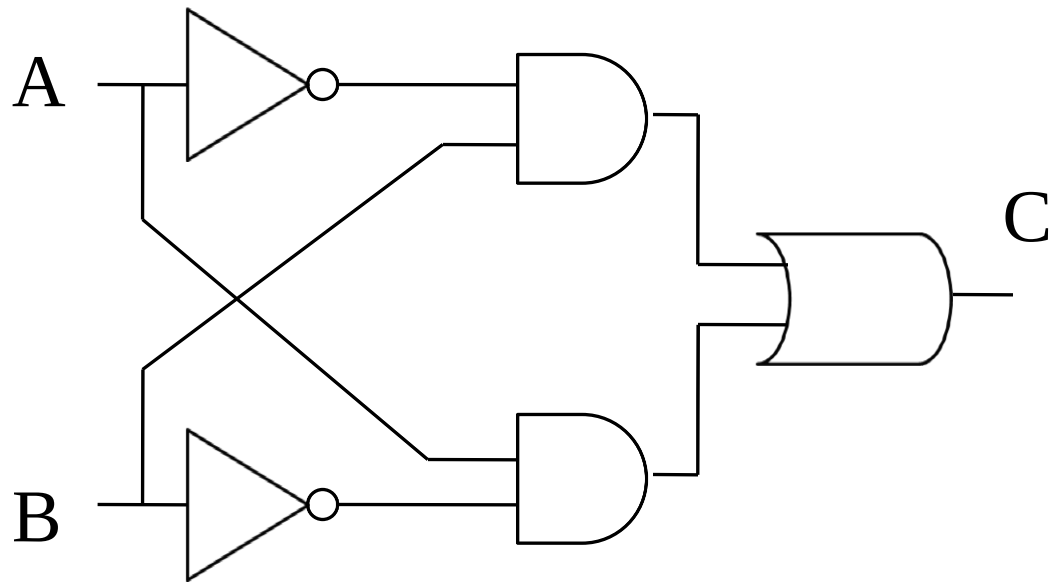
$$c_{out} = a \cdot b + a \cdot c_{in} + b \cdot c_{in}$$

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

XOR (Exclusive-OR) Gate

- C = A XOR B = A ⊕ B

p	q	p ⊕ q
1	1	0
1	0	1
0	1	1
0	0	0

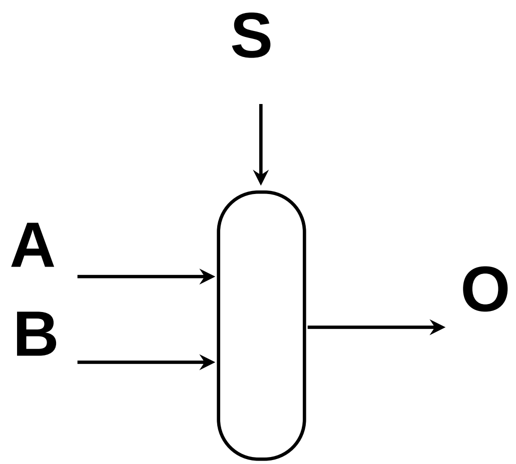


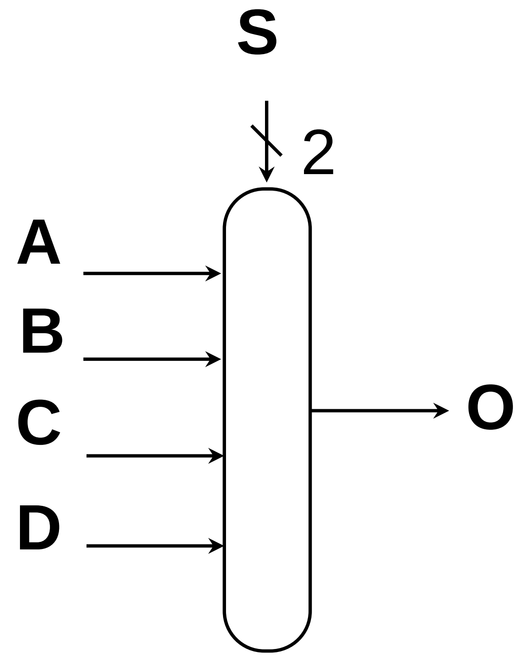
$$C = A \cdot \underline{B} + \underline{A} \cdot B$$

Multiplexor (Data Selector)

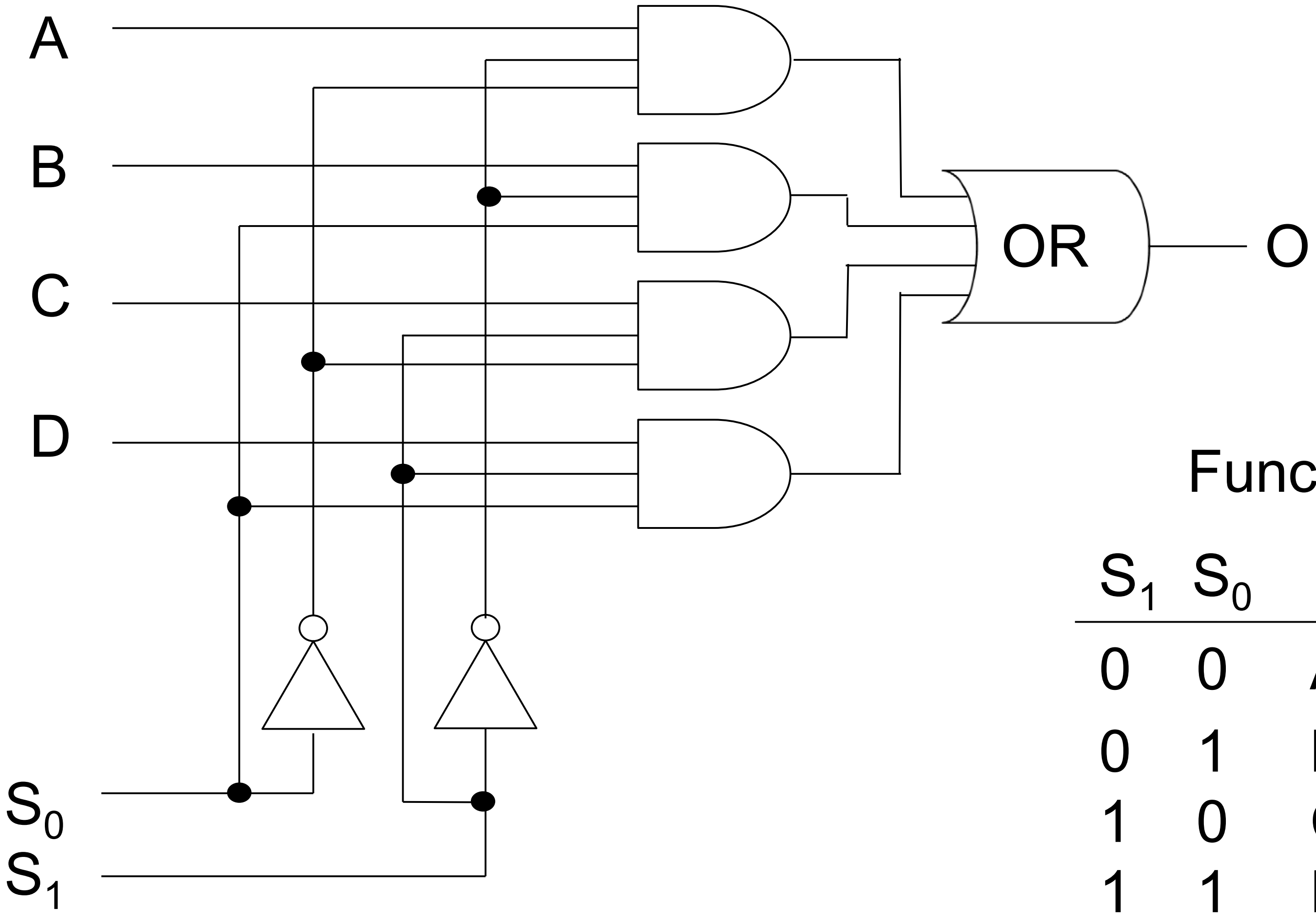
(Source: Computer Organization and Design, Hennessy and Patterson)

- 2-to-1 MUX, 4-to-1 MUX (c.f., Demultiplexer)





4-to-1 Multiplexor (복습)



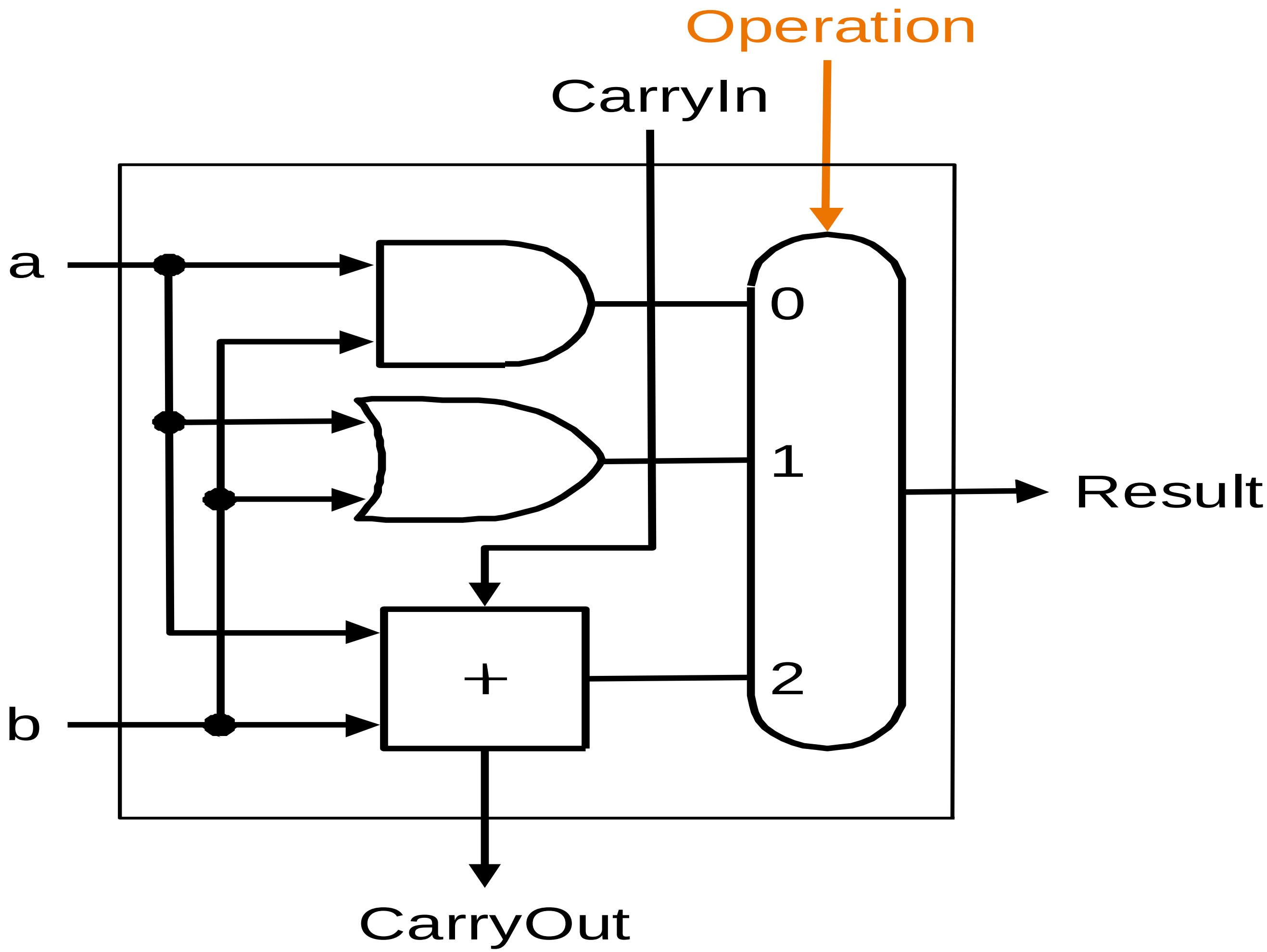
Function Table

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

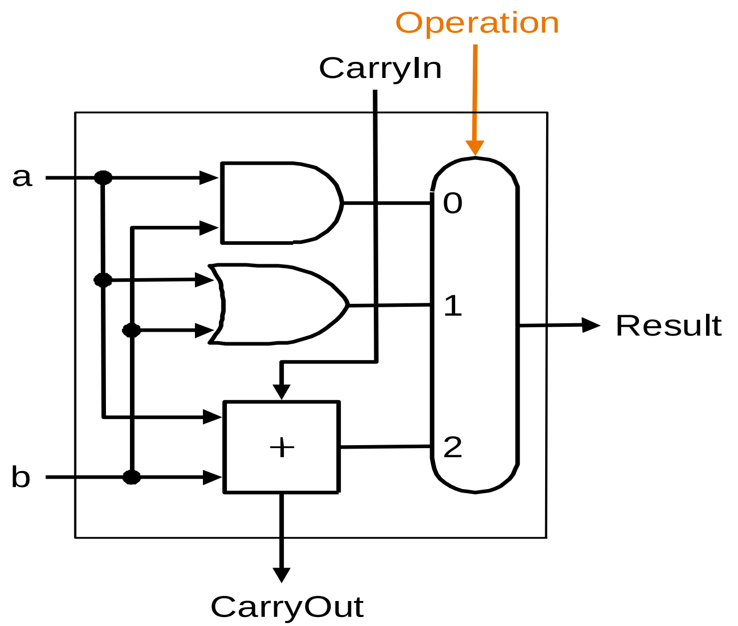
1-bit ALU Design

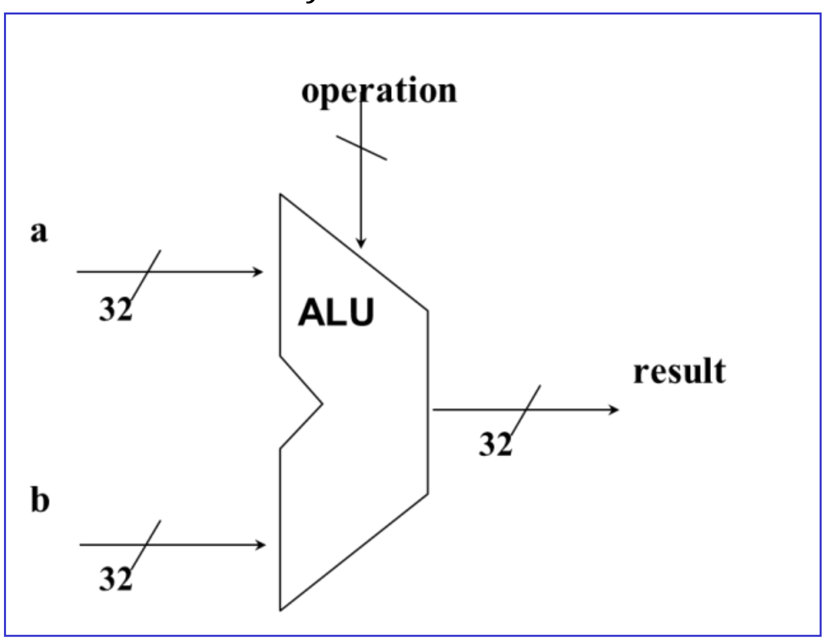
(Source: Computer Organization and Design, Hennessy and Patterson)

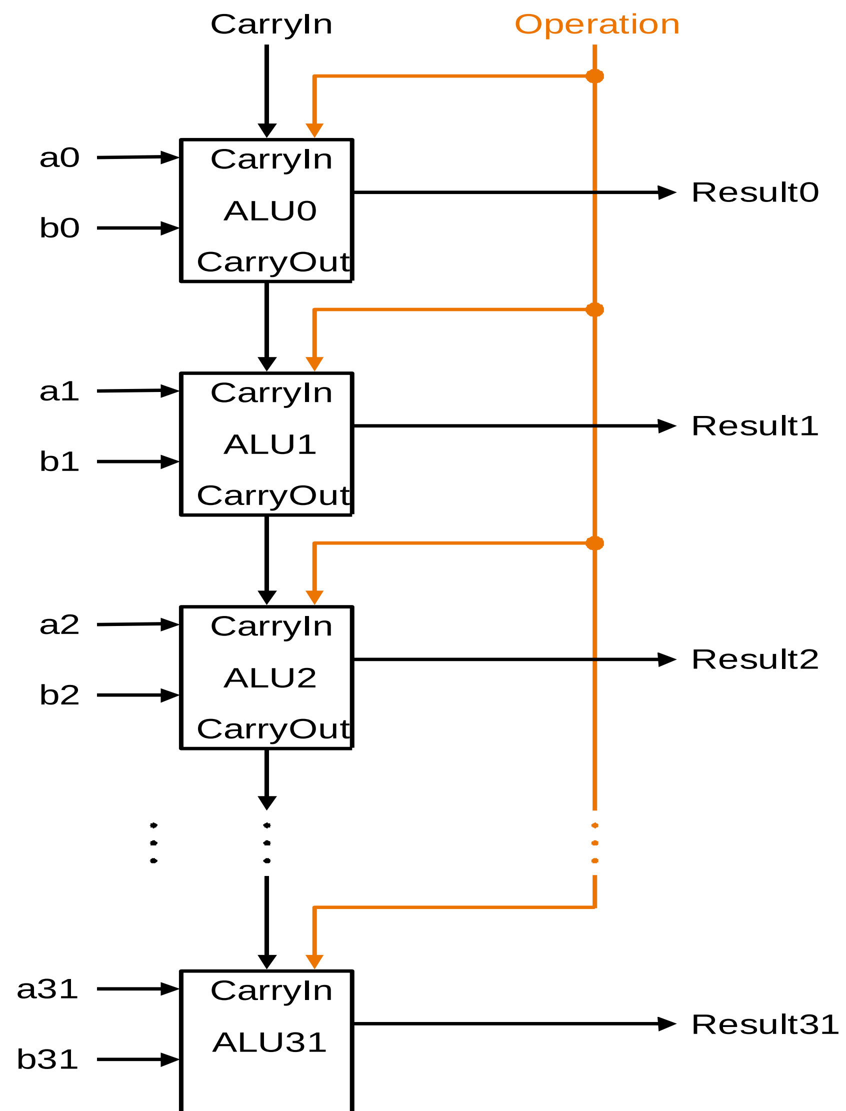
- 1-bit ADD, AND, OR



32-bit ALU Design (Source: Computer Organization and Design, Hennessy and Patterson)







Primitive-Composition-Abstraction

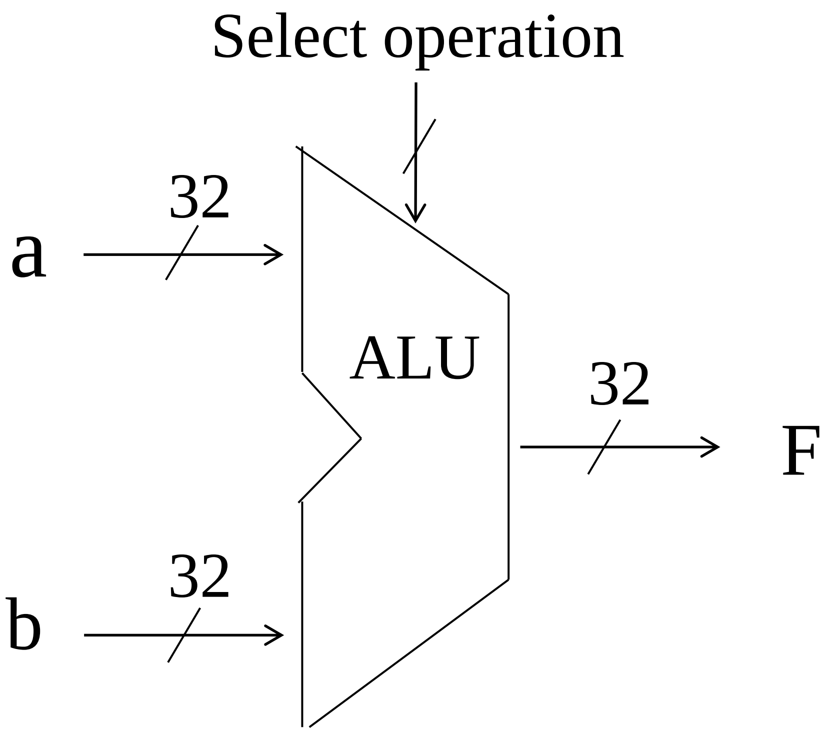
- Fundamental paradigm in engineering
  - Primitives: AND, OR, NOT
  - Composition: build function unit (FU) using gates
  - Abstraction
    - Given its interface, can use FU
    - Functional unit (FU) become primitive

elegant
논리적,체계적
으로 설계

- What is hardware design
  - Hierarchically build (more and more complex) abstraction
    - True in all engineering

32-bit ALU

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



Logical Operations

- Bitwise AND, OR, NOT

AND

Output	0	1	0	0	1	0	0	0
Input 1	0	1	1	0	1	0	1	0
Input 2	0	1	0	1	1	0	0	0

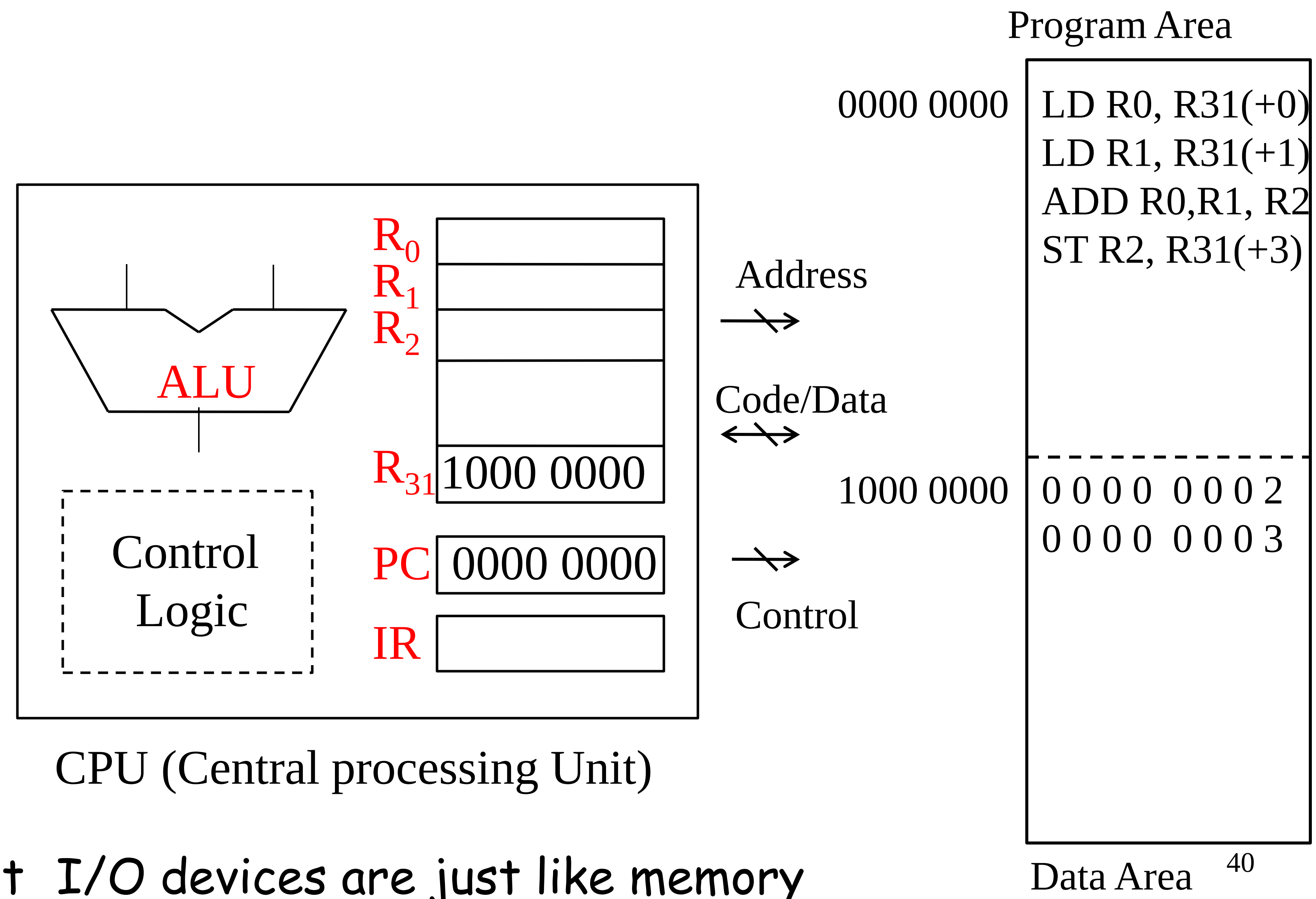
OR

Output	0	1	1	1	1	0	1	0
Input 1	0	1	1	0	1	0	1	0
Input 2	0	1	0	1	1	0	0	0

NOT

Output	0	1	0	1	1	0	0	0
Input	1	0	1	0	0	1	1	1

Machine Called Computer



CPU (Central processing Unit)

- I/O devices are just like memory

Inside CPU

- ALU (arithmetic and logic unit)
  - Add, subtract, multiply, divide, AND, OR, NOT
  - Input: registers, output: register
- 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”

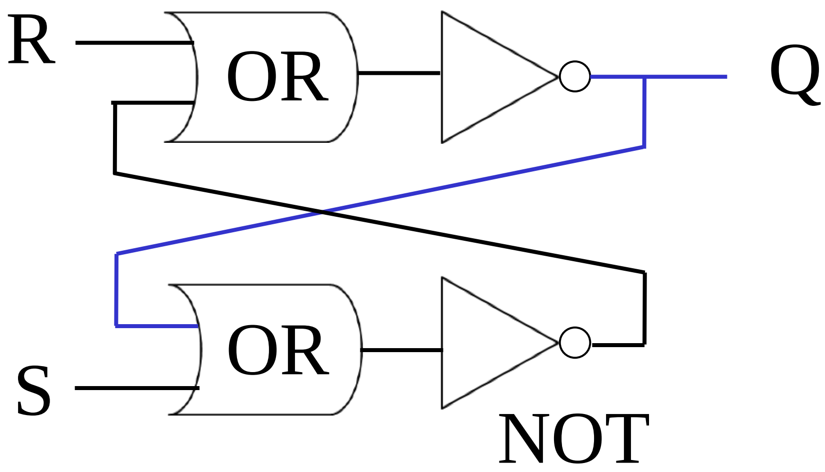
Sequential Logic Design

- Storage (Registers and Memory)
- Notion of "Address"

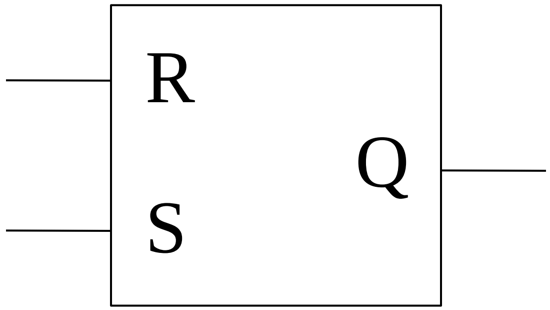
(메모리: AND, OR, NOT 기반의 자동장치)

SR Flip-Flop (구조나 동작을 암기할 필요 없음)

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

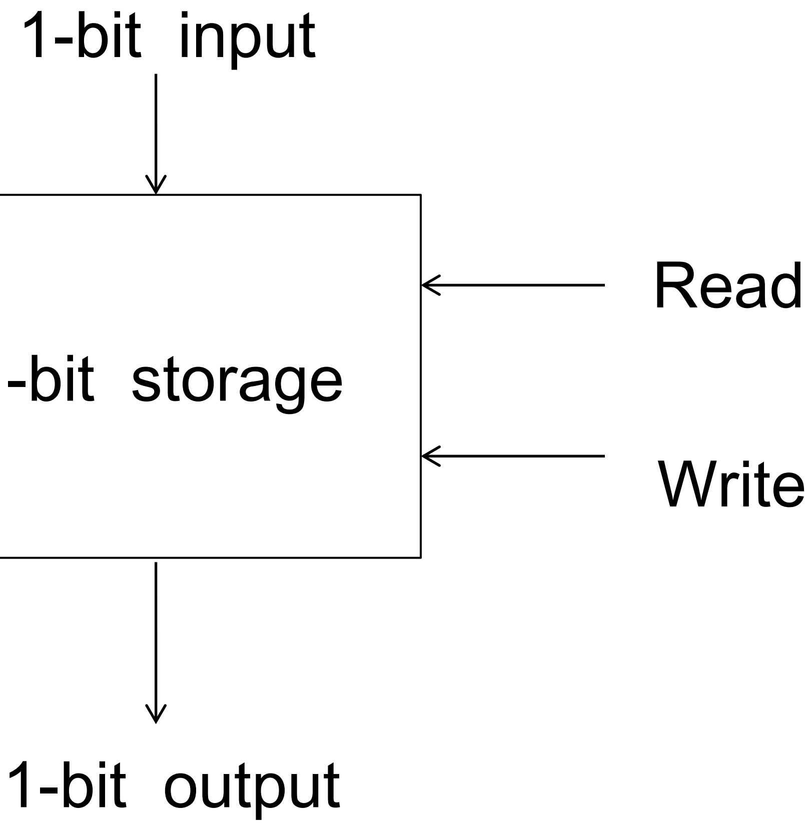


S	R	Q	action
0	0	Q	hold
1	0	1	set
0	1	0	reset
1	1	not allowed	



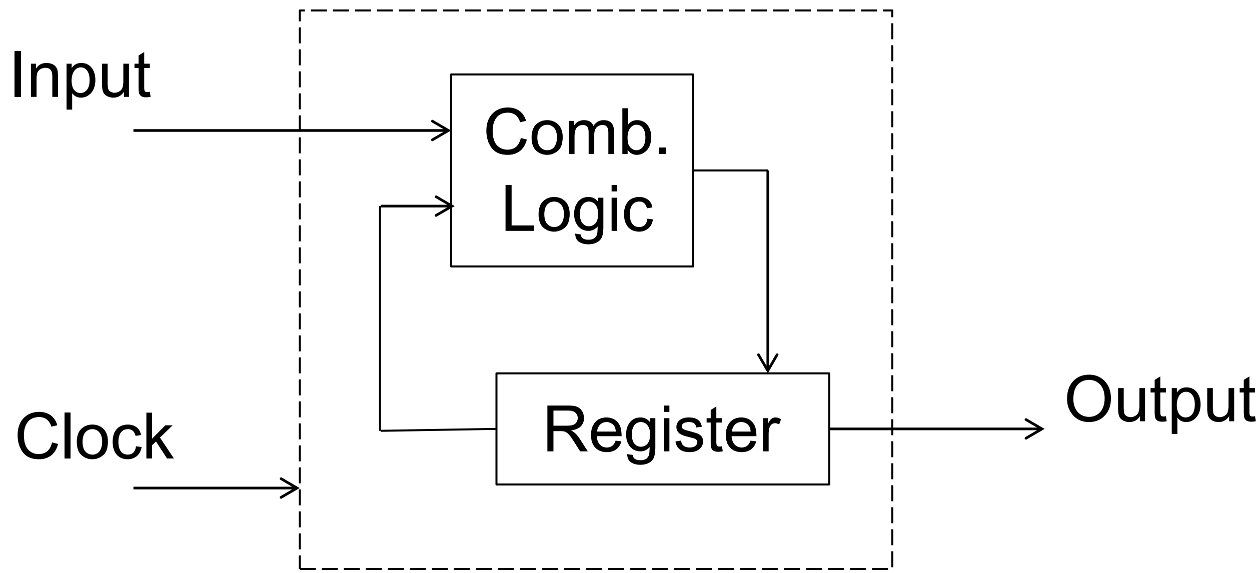
- Read Q with S=0, R=0
- Write to Q
  - To store 1: S=1, R=0
  - To store 0: S=0, R=1

1-bit Storage



Sequential Logic Design (복습)

- Registers (simplest ones)
- Binary counters

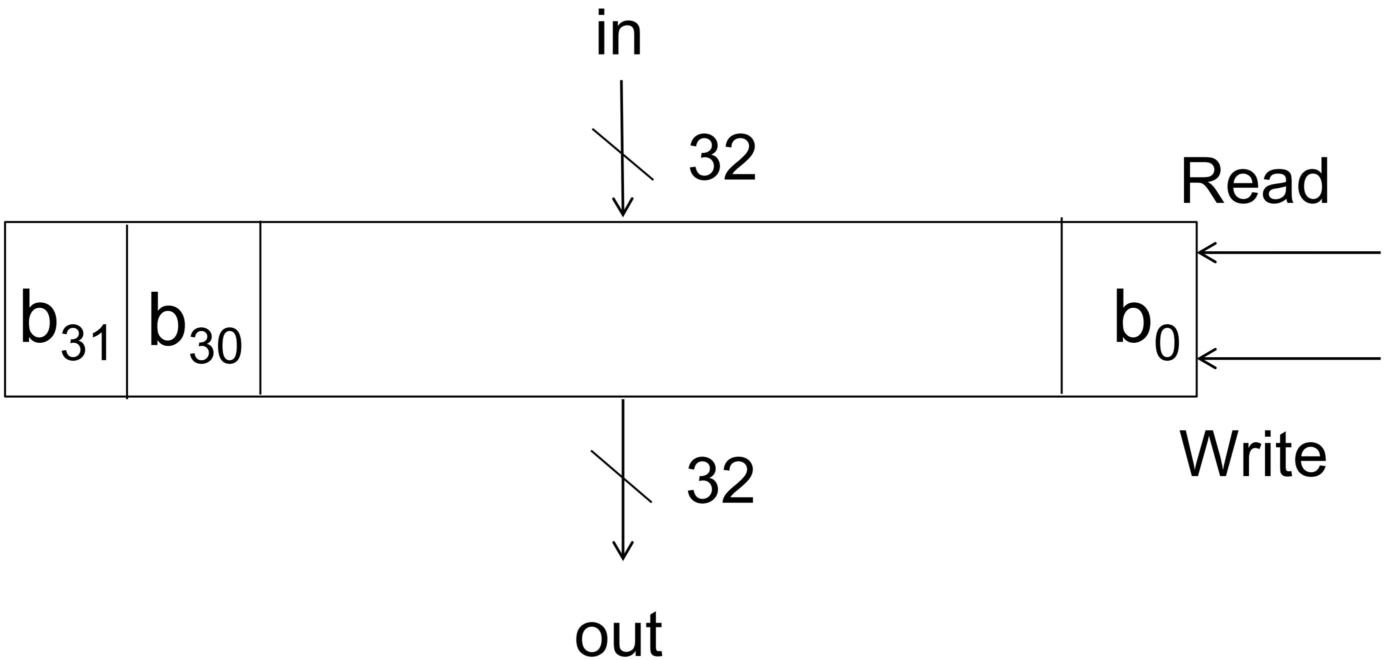


† Sequential logic design more complex than comb. Logic

- Truth table vs. state diagram

32-bit Storage

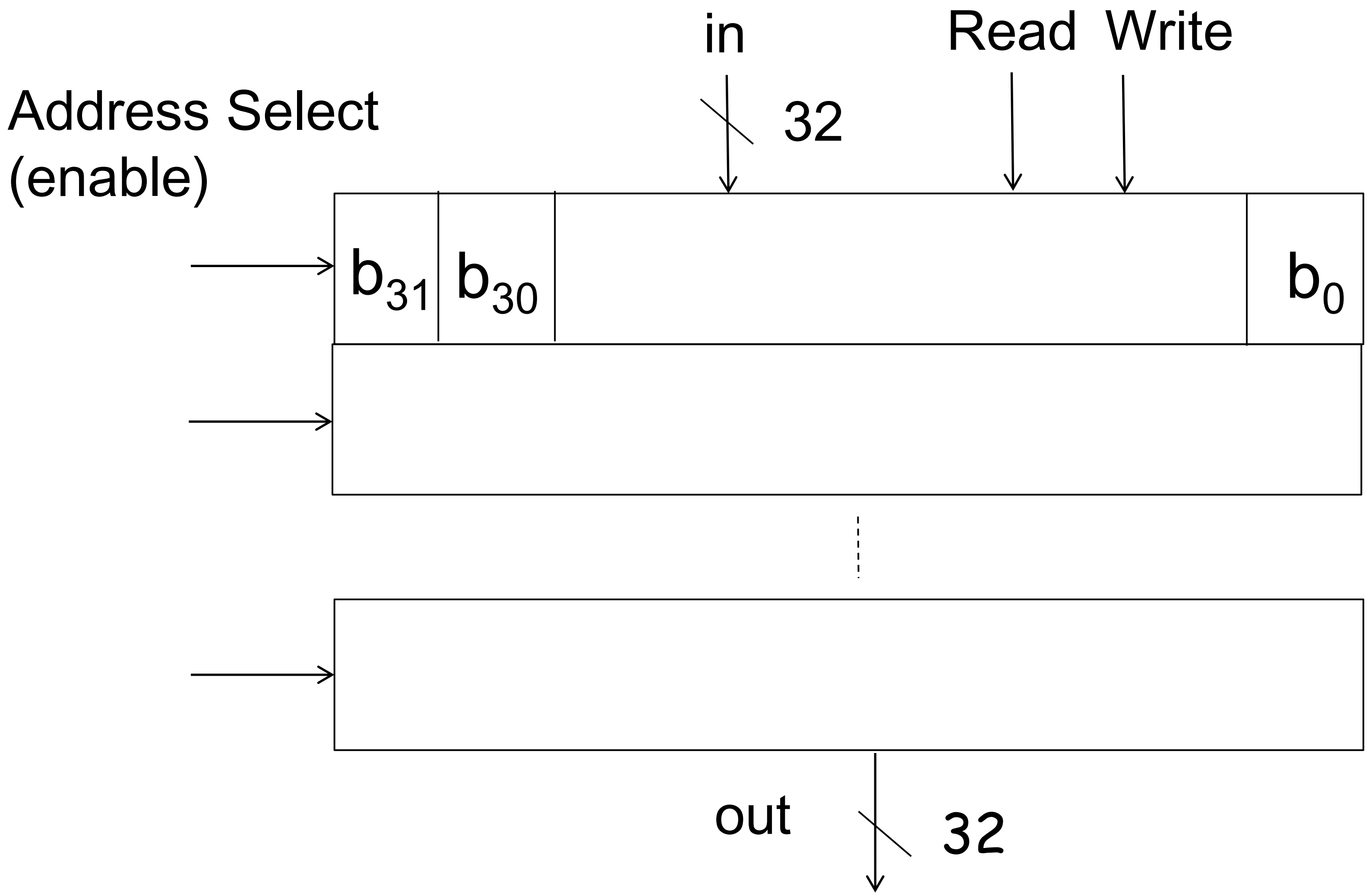
- Use 32 of 1-bit storages in parallel (share "address")



- Register: 32-bit storage in processor

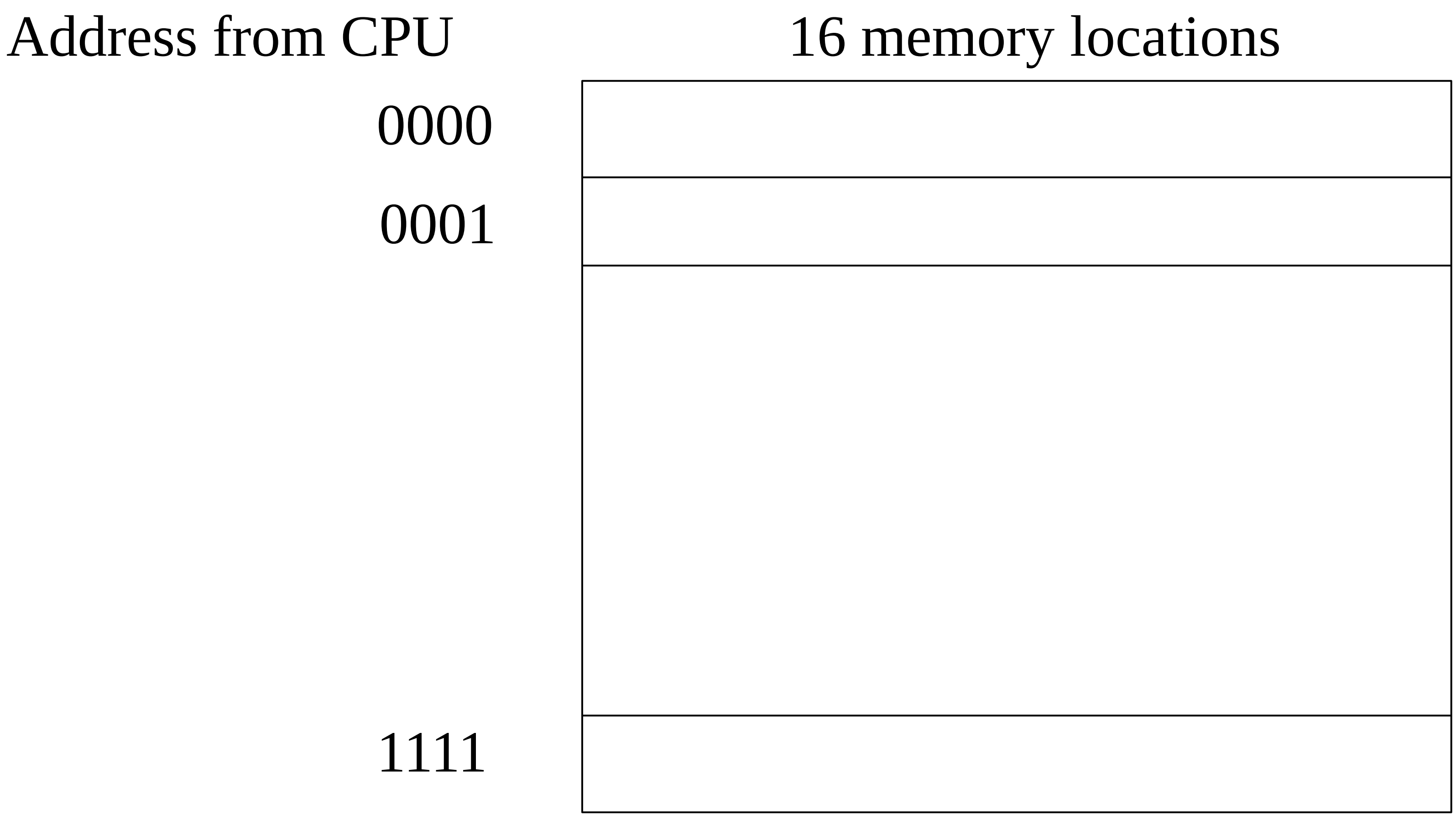
Main Memory in 32-bit Computer

- Many locations – each has distinct address



Meaning of Address

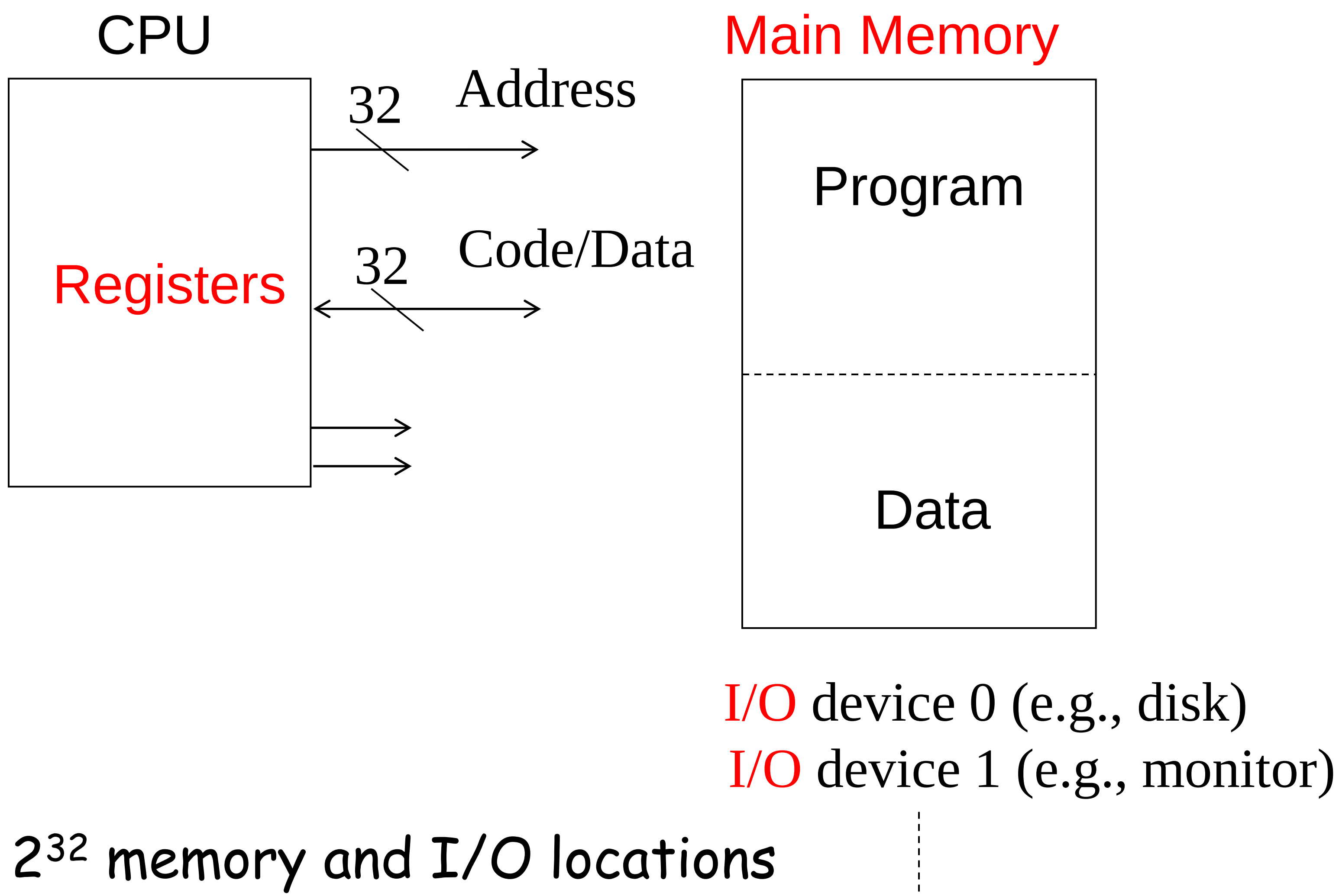
- Unique identifier for locations



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

32-bit Computer



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

Primitive-Composition-Abstraction

- Fundamental paradigm in engineering
  - 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 and more complex) abstraction
    - True in all engineering

Levels of Abstraction (참고자료)
(GEB by Hofstadter, AI by Winston)

Machine Instruction

Processor (Complex Functional Unit)

(hierarchical abstraction process)

Simple Functional Unit

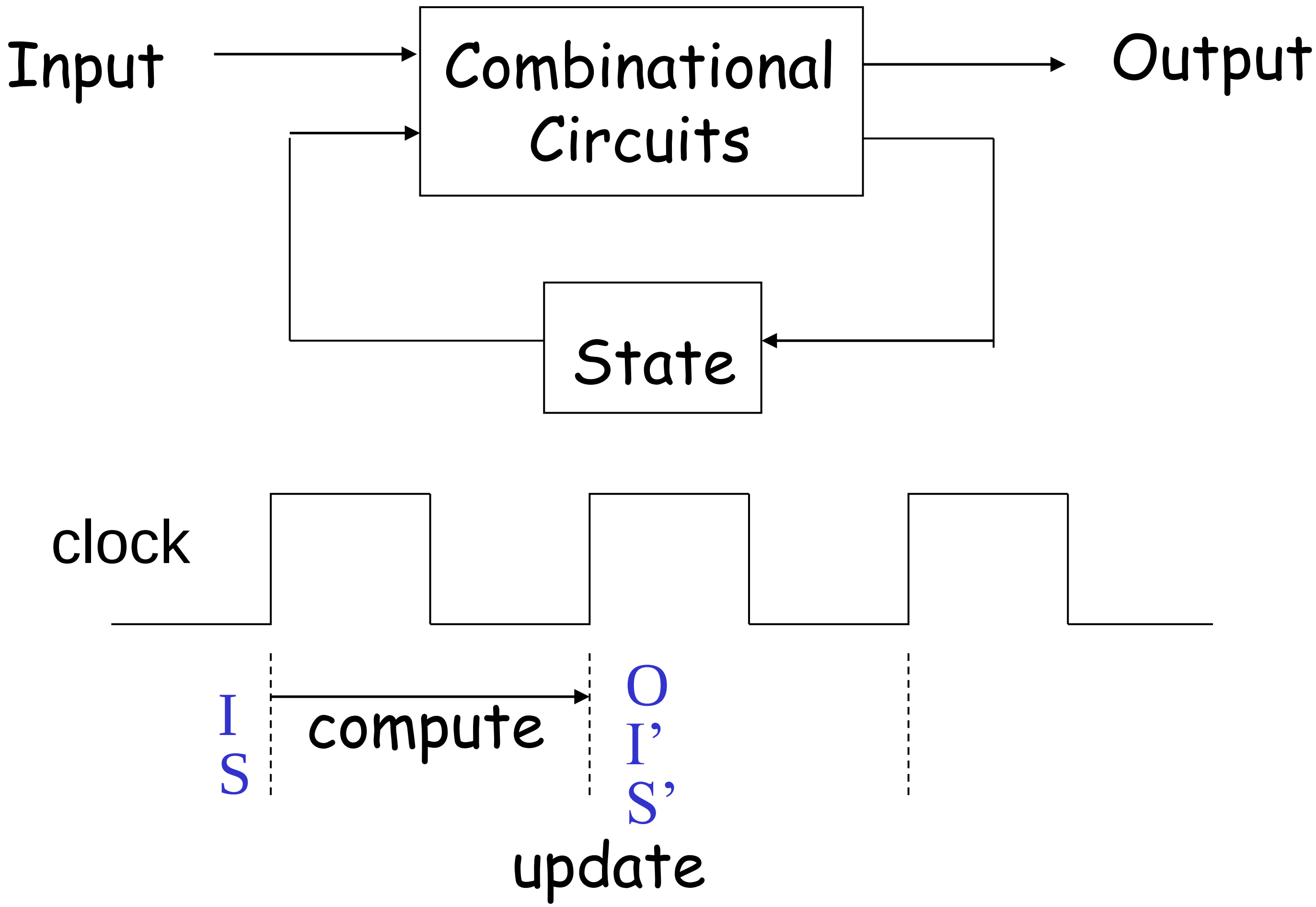
Gates

Transistors

Digital Logic Design
(Combinational/Sequential Logic)

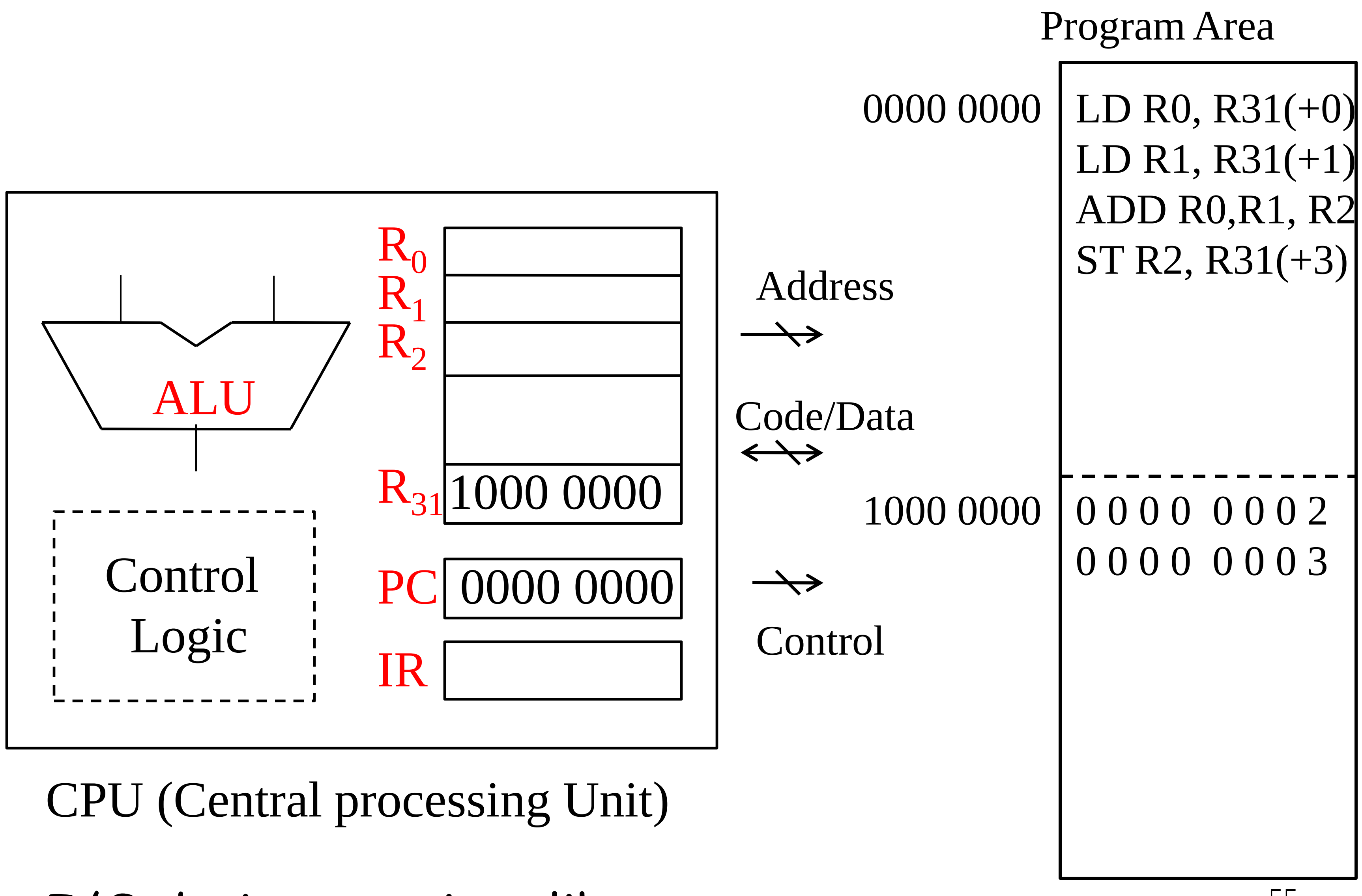
CPU, Memory
(AND, OR, NOT 기반의 자동장치)
(IF 개념은 곧 다시 나옴)

Combinational and Sequential Circuits
(참고자료)



† Synchronous logic circuits

Machine Called Computer



- I/O devices are just like memory