

Parallel and Reconfigurable VLSI Computing (2)

# Hardware Preliminary

Hiroki Nakahara

Tokyo Institute of Technology

# Outline

- Boolean Logic
- Boolean Arithmetic
- Sequential Circuit
- Computer Architecture

# Boolean Logic

# Boolean Logic

- Represent number by "bool" variable
  - true/false, 1/0, yes/no, on/off
- (Boolean) Logic function
  - Both In/Out are bool variables

# Truth Table

- A kind of representation for logic function
- $2^{2^n}$  functions exist for  $n$  input

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

# Boolean Expression

- Boolean operator: OR(+), AND( $\cdot$ ) and not( $\bar{\phantom{x}}$ )

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



$$f(x,y,z) = (x+y) \cdot \bar{z}$$

For all combinations of input,  
evaluate a given expression (Verification)

# Canonical Representation

- Any Boolean function can be represented by at least a canonical representation
- Conversion a truth table to a canonical representation
  - For each "1" output line, concatenate input literals by AND operation
  - Concatenate these terms by OR operation  
→ AND-OR standard representation
  - c.f. OR-AND standard representation
- Arbitrary Boolean function can be represented by operator set {AND,OR,NOT} (**Completeness**)

# 2-input Boolean Function

- 16 functions

x,y	0,0	0,1	1,0	1,1
Constant 0	0	0	0	0
AND	0	0	0	1
x AND Not y	0	0	1	0
x	0	0	1	1
Not x AND y	0	1	0	0
y	0	1	0	1
EXOR	0	1	1	0
OR	0	1	1	1
NOR	1	0	0	0
EXNOR	1	0	0	1
Not y	1	0	1	0
If y then x	1	0	1	1
Not x	1	1	0	0
If x then y	1	1	0	1
NAND	1	1	1	0
Constant 1	1	1	1	1

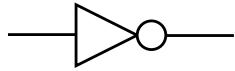


# Boolean (Logic) Gate

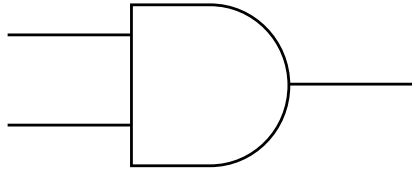
- A physical device which realizes a Boolean function
- Transistor: Made by connecting a switch by specified wires
- Almost all digital computer operates use electricity to represent and operate binary data
- Other elements can be used:
  - Silicon device (Major)
  - Magnet, light, bio, hydraulic, and pneumatic
- Boolean algebra: A concept abstraction
- Composite gate: Consists of primitive logic gates

# Primitive Gate

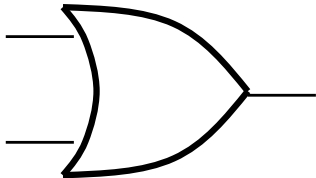
- NOT



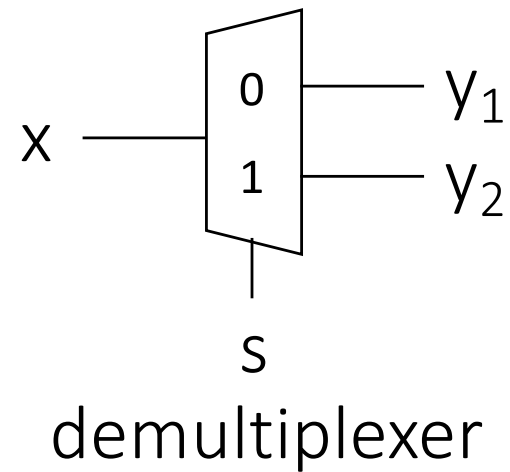
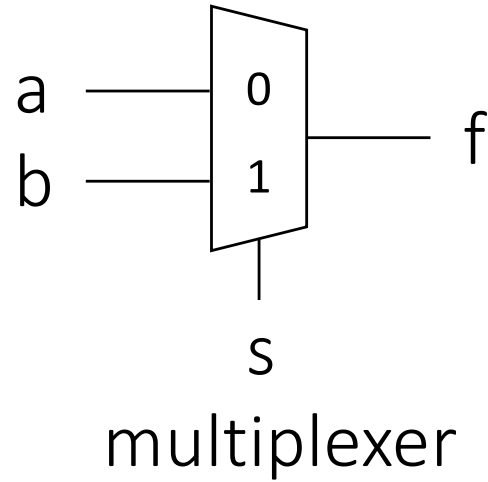
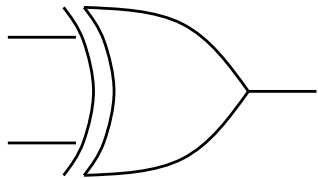
- AND



- OR

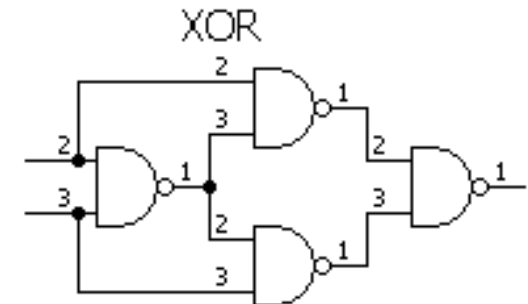
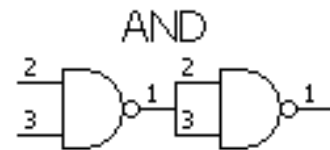
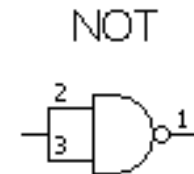
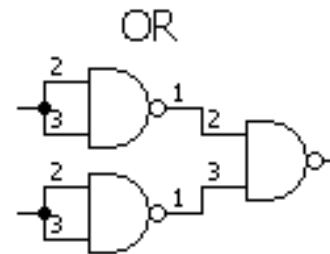
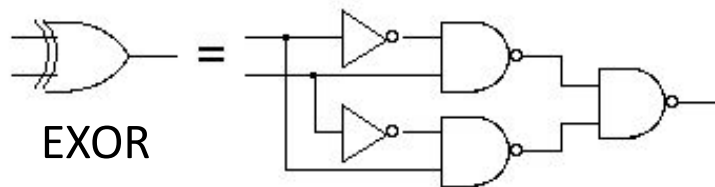
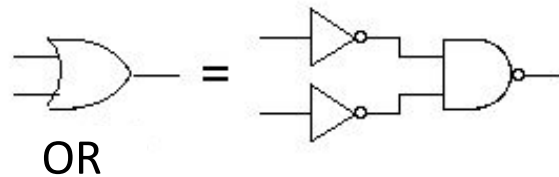
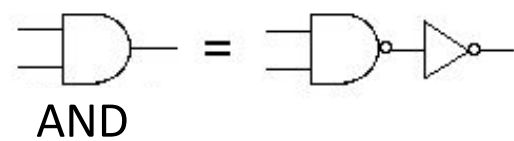
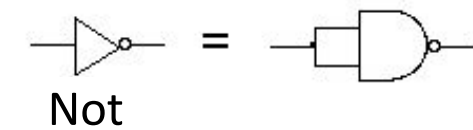


- EXOR



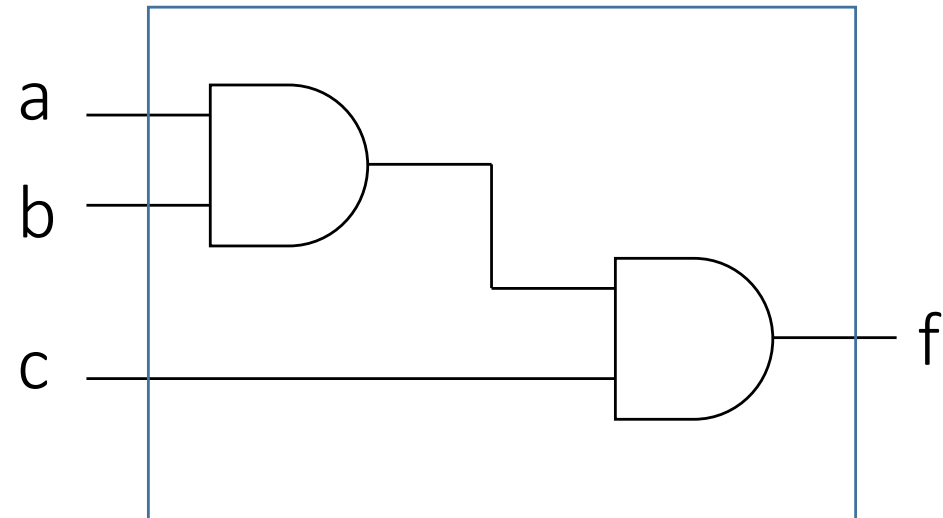
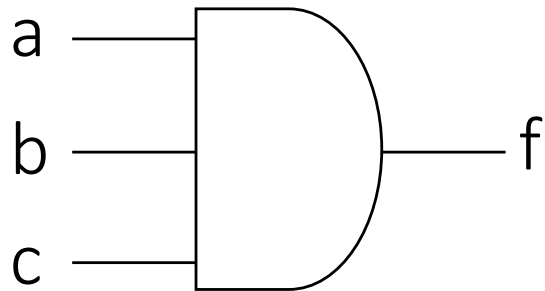
# Example of Completeness

- NAND (NOR):



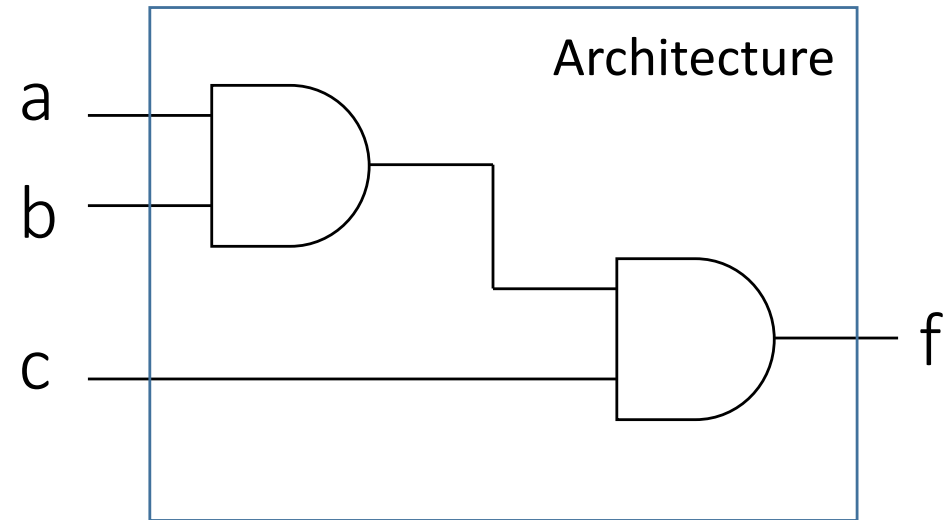
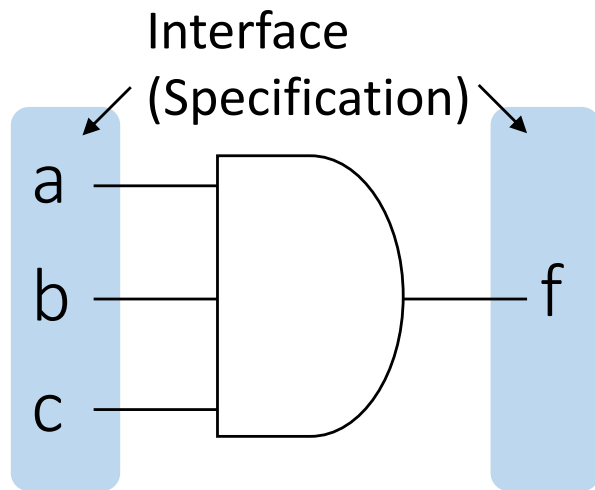
# Composite Gate

- $a, b, c, f \in \{0, 1\}$
- $\text{AND}(x, y, z)$  for Boolean expression:  $a \cdot b \cdot c = (a \cdot b) \cdot c$



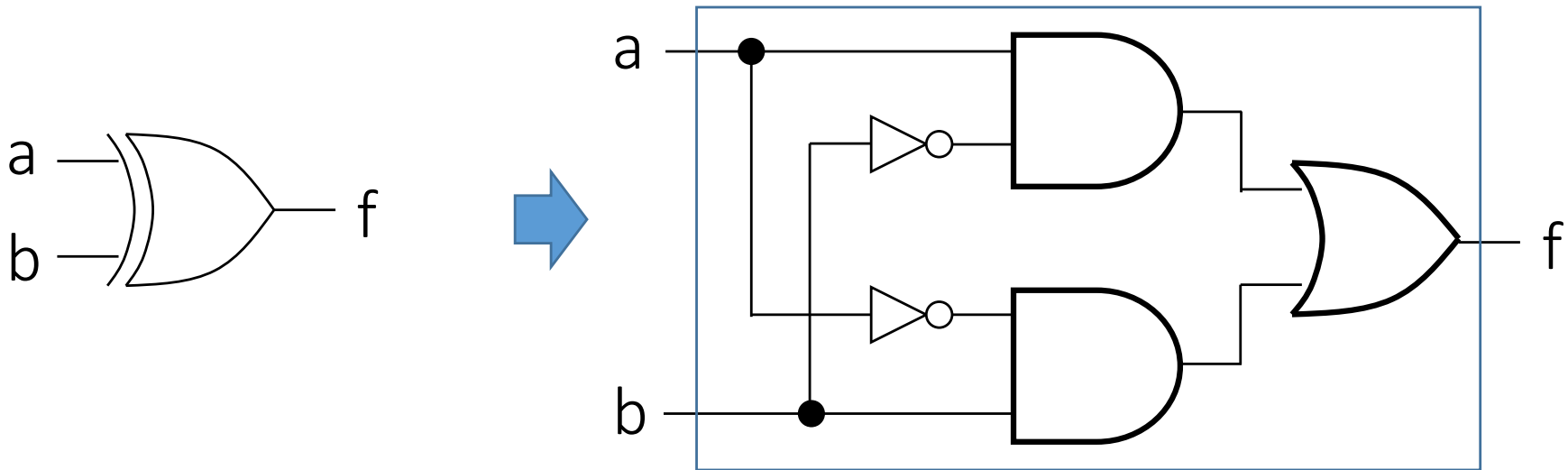
# Logic Design

- Design method for connecting the gate
  - Composite gate for complicated function is designed using primitive gate
- Different point of view
  - Left: Interface outside the gate → Designers treat it as a black box
  - Right: Implementation method inside gate (architecture)



# Example: Logic design for EXOR

- Gate interface (Specification) is an unique
- Several realizations exist
  - Area, speed, power, simplicity, cost, and/or reliability  
→ Design method based on a cost function

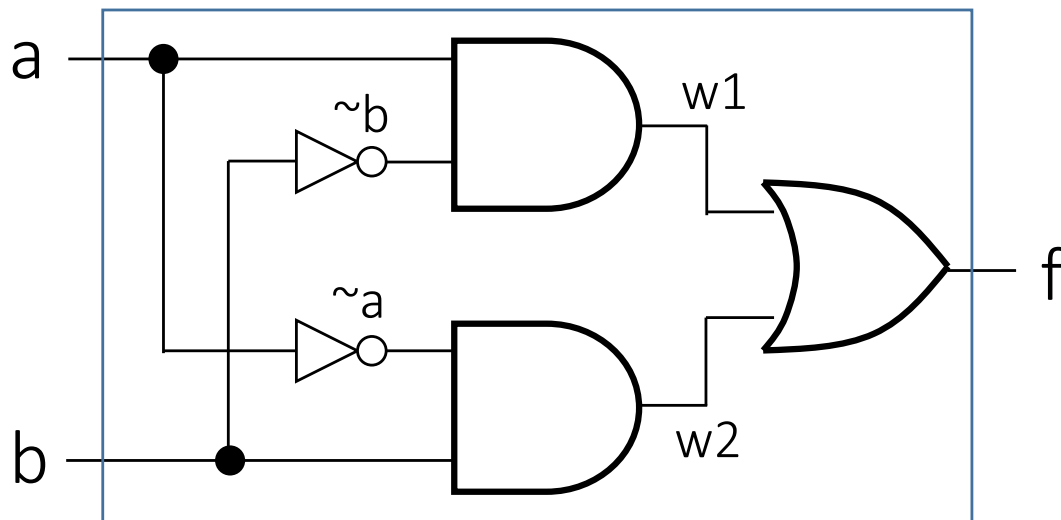


# Hardware Description Language (HDL)

- Gate level design wastes time
- Circuit assembly mistakes in wiring process
- Design and verification of the architecture on a virtual circuit (computer) with software
  - Logic Synthesis
  - Logic Simulation
- Architecture can be represented by HDL program
- Design and verification can be done with no money
- After HDL design, it prints on real silicon

# Example of Verilog-HDL

- HDL specification
- HDL simulation



```
module exor ( a, b, f)
  input a, b;
  output f;
```

```
  assign w1 = a & ~b;
  assign w2 = b & ~a;
  assign f = w1 | w2;
endmodule
```

```
module
  reg a_t, b_t;
  wire f_t;

  #0
    a_t = 1'b0; b_t = 1'b1;
  #1
    a_t = 1'b1; b_t = 1'b1;
  $display(a_t,b_t);
endmodule
```

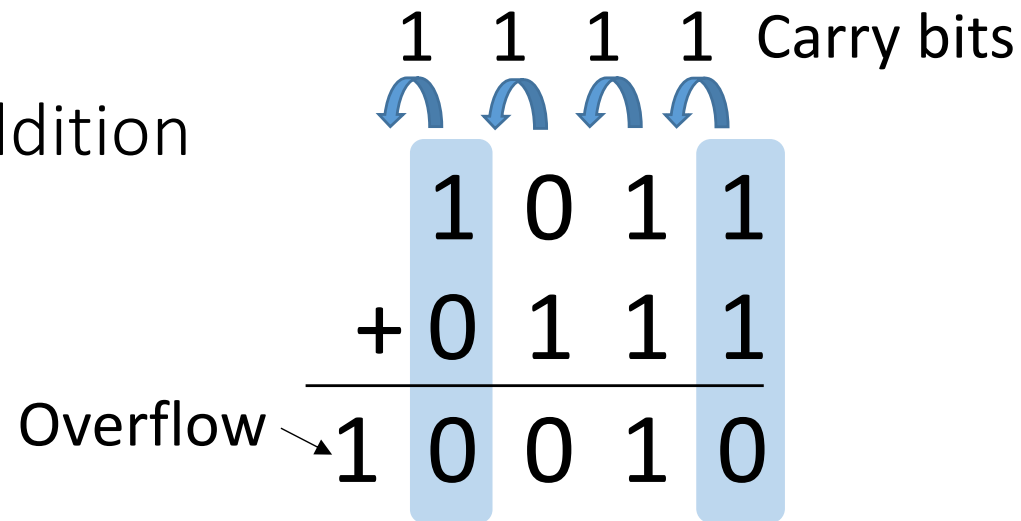


# Boolean Arithmetic

# Binary Number

- Representation of numbers based on two
- $(10011)_b = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 19_d$

- Binary addition



MSB: Most Significant Bit

LSB: Least Significant Bit

# Specifications for Adders

- Half adder: Addition for  $x$  and  $y$ , then output carry( $c$ ) and sum ( $s$ )
- Full adder: Addition for  $x$ ,  $y$  and  $c$ , then output  $c$  and  $s$
- (Multibit) Adder: Addition for  $n$ -bit of  $x$  and  $y$
- Incrementor: Add  $+1$  for a given  $x$

# 2's Complement (Radix Complement)

- MSB represents sign (plus (0) or minus (1))
- 2's complement  $\bar{x}$  for a given  $x$  (n bit):

$$\bar{x} = \begin{cases} 2^n - x & (x \neq 0) \\ 0 & (Otherwise) \end{cases}$$

- Example:
  - Five bit for  $-2_d$  is represented by  $2^5_d - (00010)_b = 32_d - 2_d = 30_d = (11110)_b$ , since  $(00010)_b + (11110)_b = (00000)_b$
- Known technique:  $\bar{x}$  is obtained by  $\sim x + 1$

# 2's Complement for Four bit

- Range: from  $2^{n-1}-1$  to  $-2^{n-1}$

Plus number		Minus number	
0	0000		
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010
7	0111	-7	1001
		-8	1000

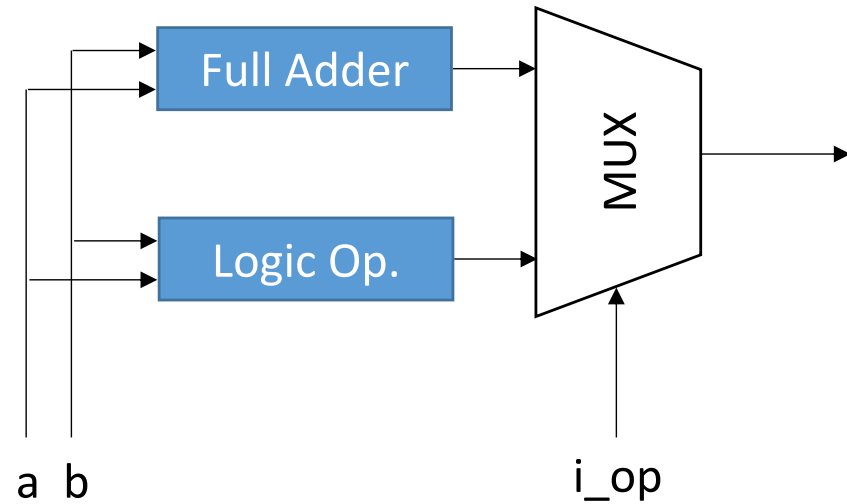
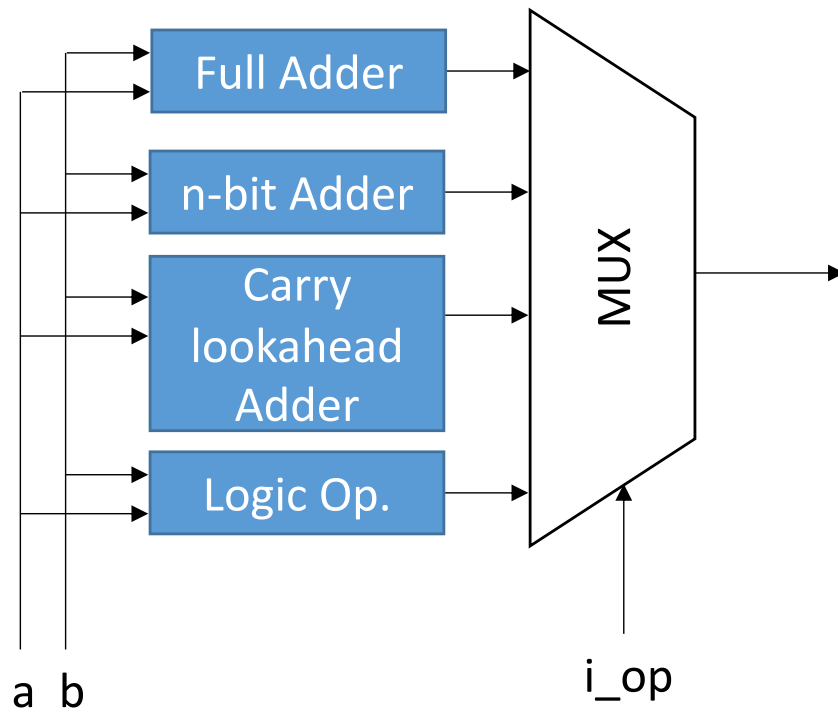
# Trade-off

- Sum of signed 2's complement number can be calculated in the same procedure as the sum of positive numbers
- Increment operation can be done in the same procedure as the sum of constant 1 and positive number
- The sum of  $n$ -bit positive numbers can be realized by repeating the full adder  $n$ -times
- Fast addition → Carry look ahead
- We should think cost-performance issues
  - Area, performance, power, and cost

# Extension to Arithmetic Logic Unit (ALU)

- Primitive arithmetic and logic operations
- Operations to be provided are considered in cost-performance
- Hardware and software functions are provided as a pair of ALU and operating system (OS)
  - Multiplication, division, floating point operation, e.t.c.

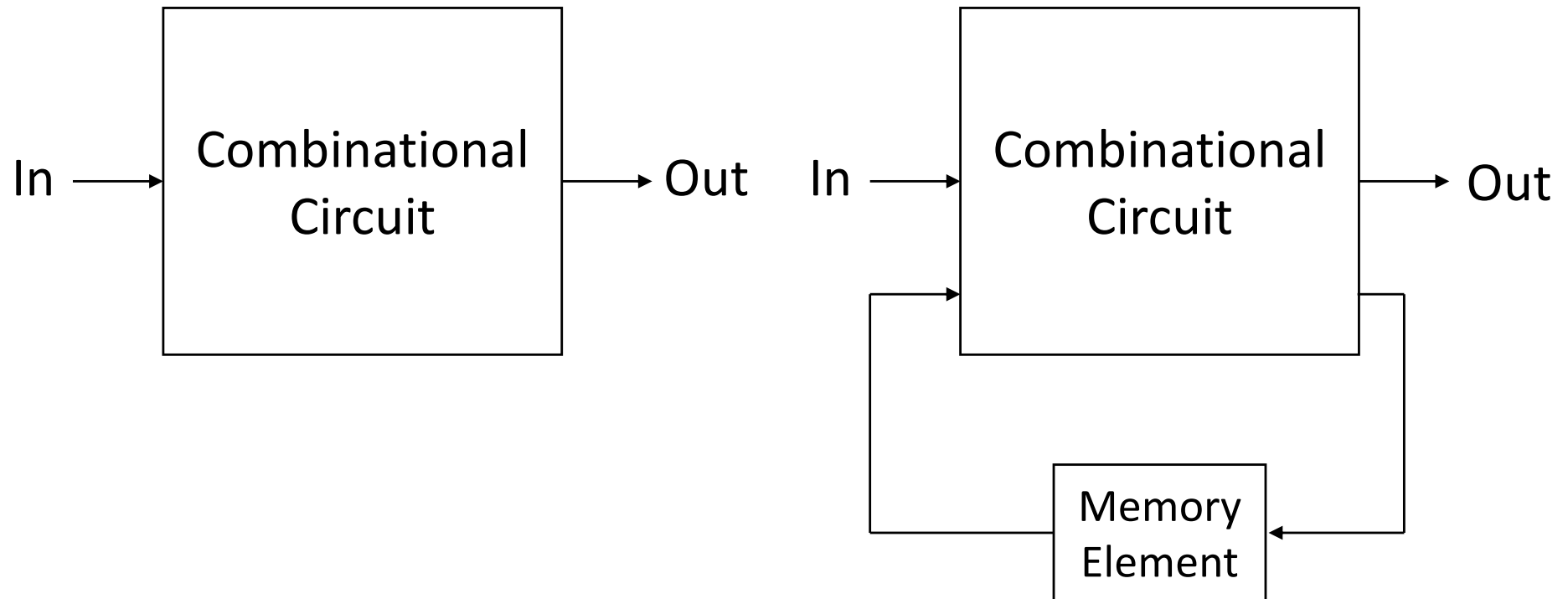
# Example of ALU



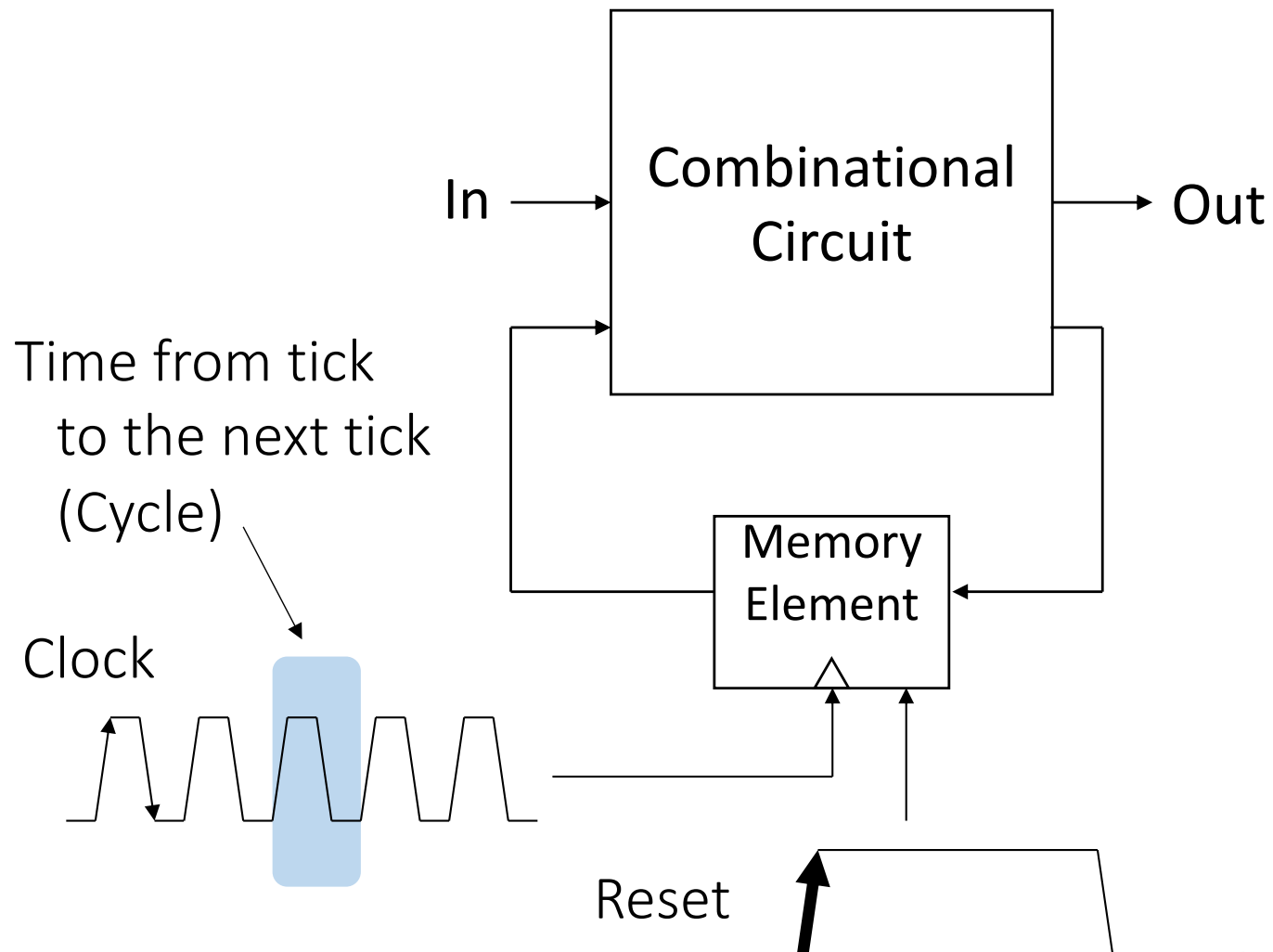


# Sequential Circuit

# Combinational and Sequential

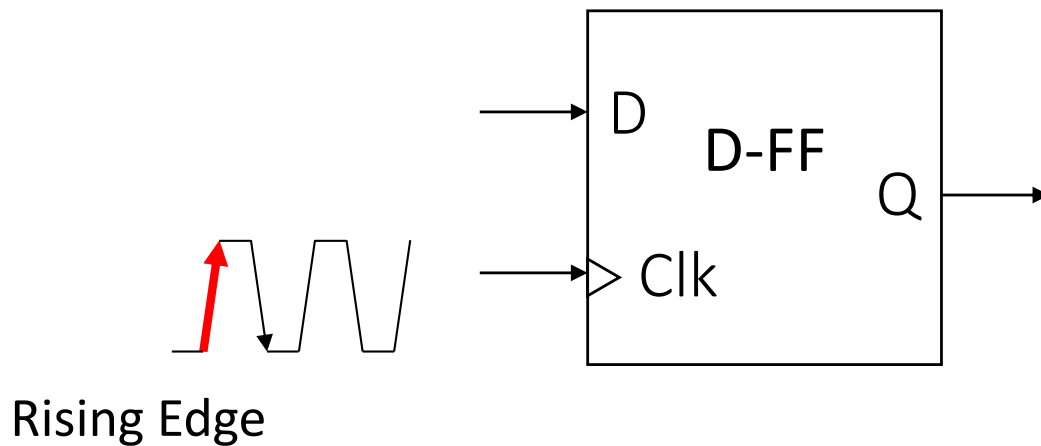


# Synchronous Sequential Circuit



# D-Flip Flop

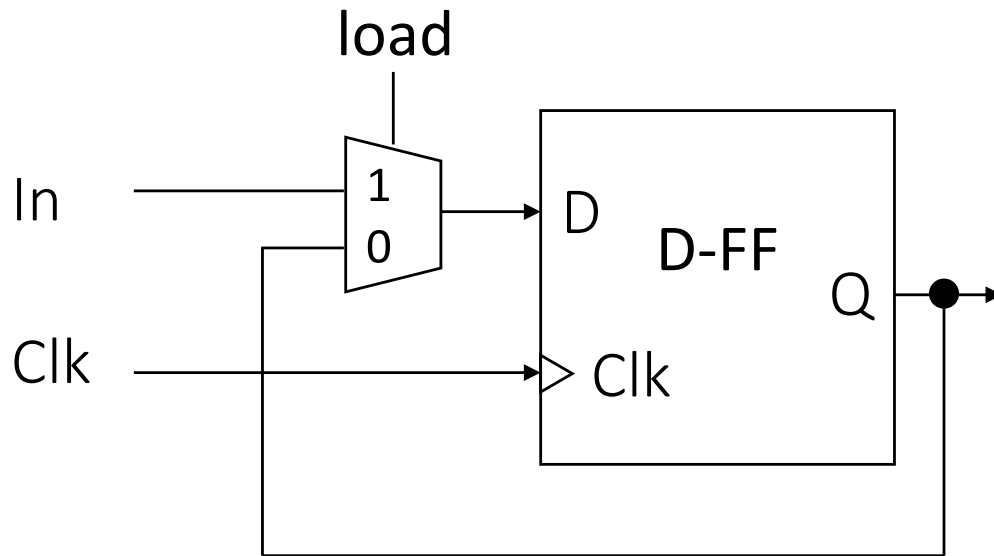
- Keep the past internal value
- Input, output, clock, (with reset)



Clk	D	Q(t)
↑	0	0
↑	1	1
Otherwise	---	Q(t-1)

# D-Flip Flop (Cont'd)

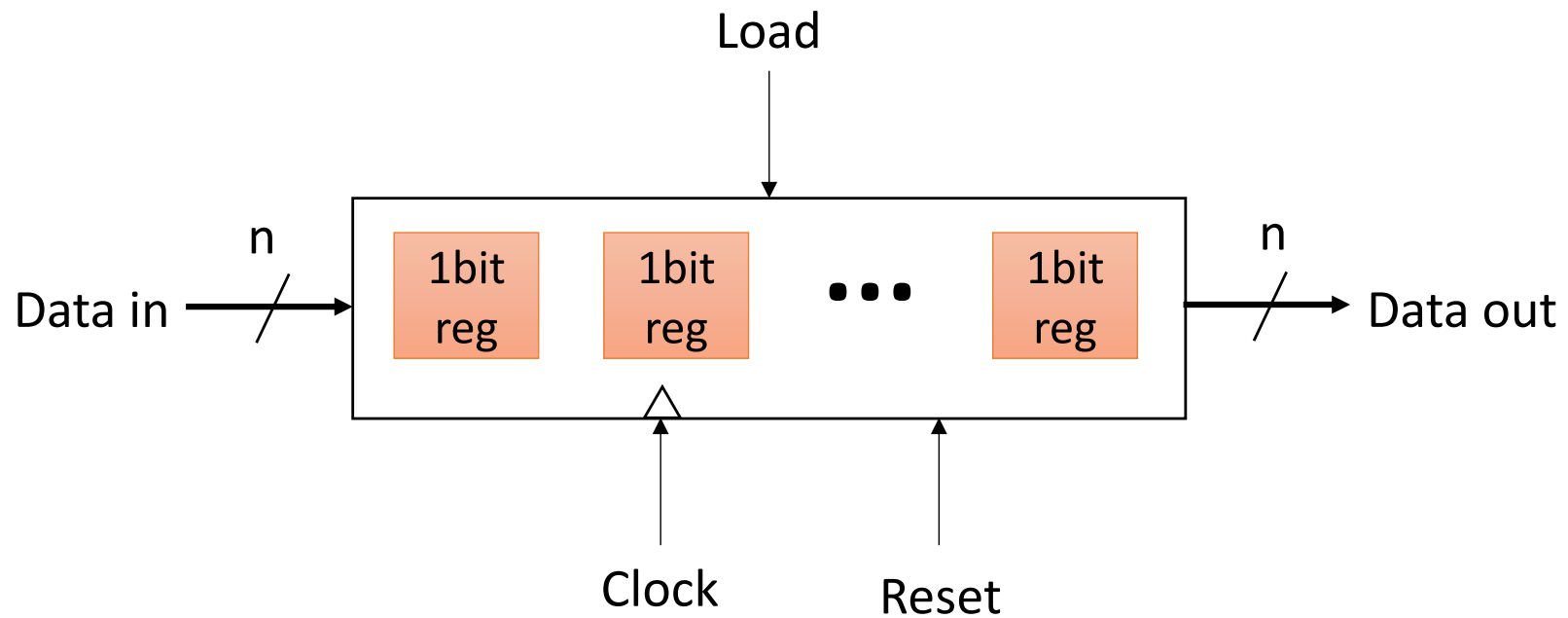
- In practice, data load signal is used with a multiplexer
- In other words, **1-bit register**



Clk	load	Q(t)
↑	0	Q(t-1)
↑	1	In
Otherwise	---	Q(t-1)

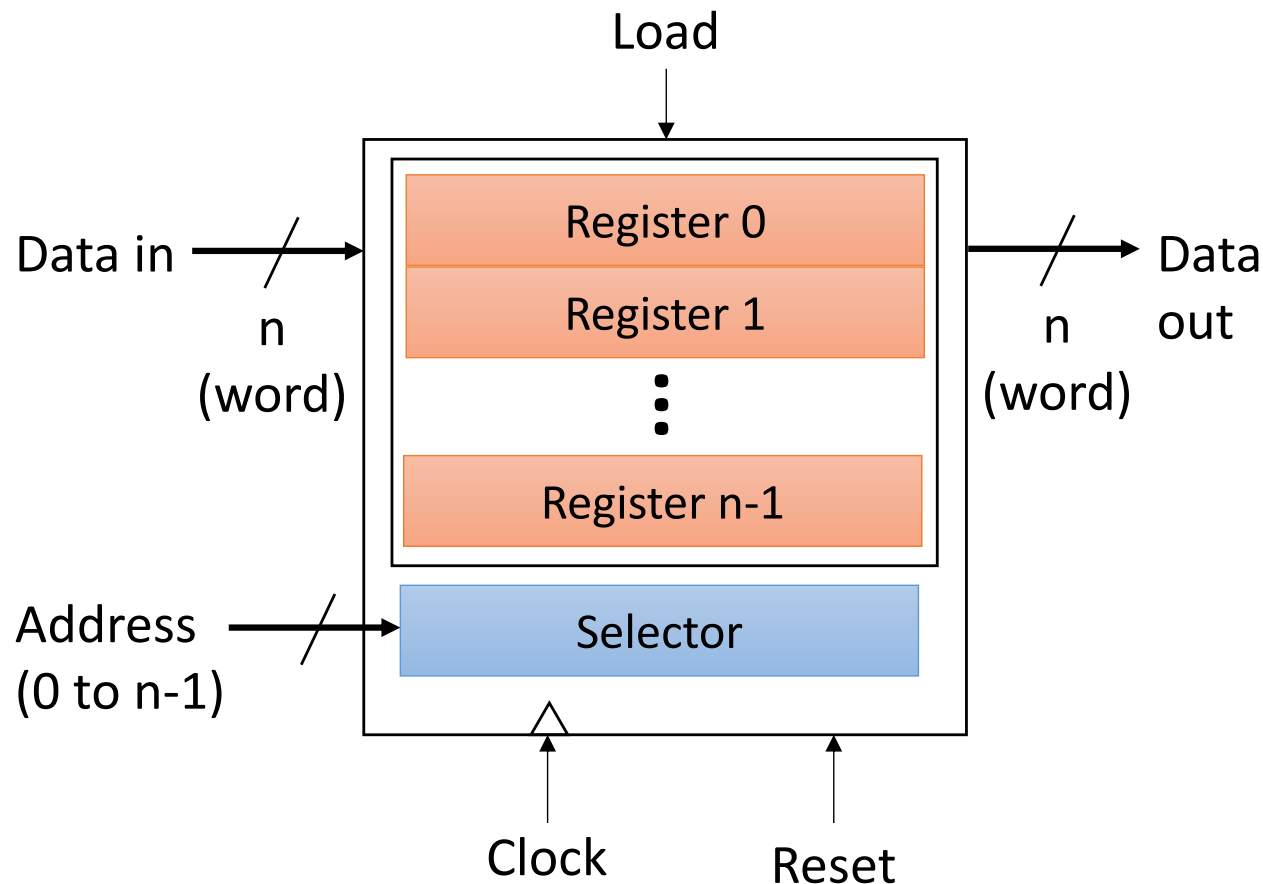
# Register

- Consists of n-copies of D-FFs
- # of D-FFs: 16, 32, 64  $\rightarrow$  1 [word]



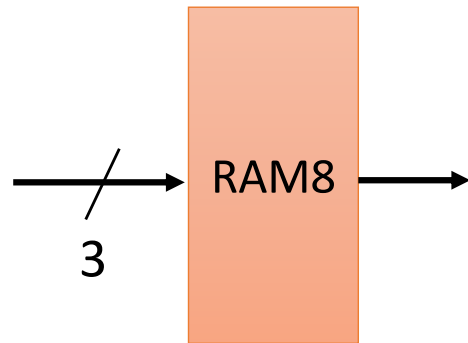
# Random Access Memory (RAM)

- RAM → Accessible for arbitrary word

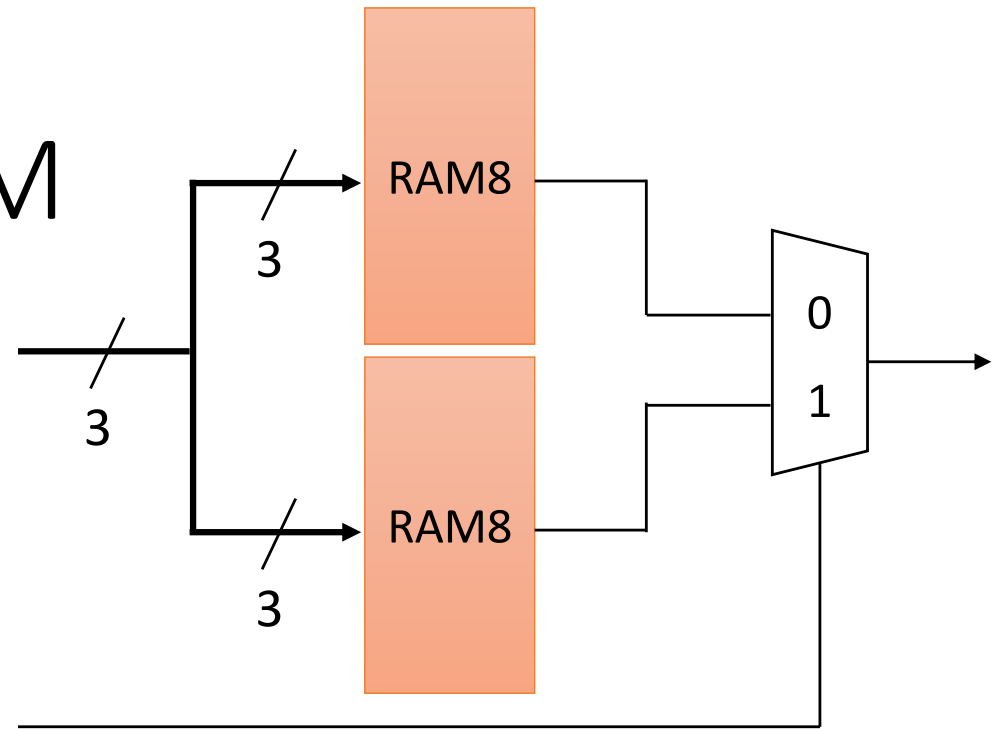


c.f. Sequential access  
memory

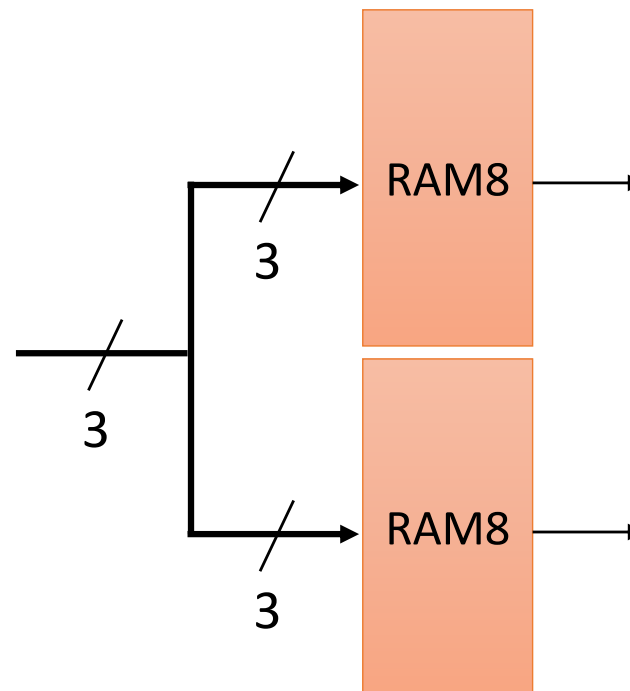
# Composite RAM



3-input single output RAM



4-input  
single output RAM



3-input  
2-output RAM



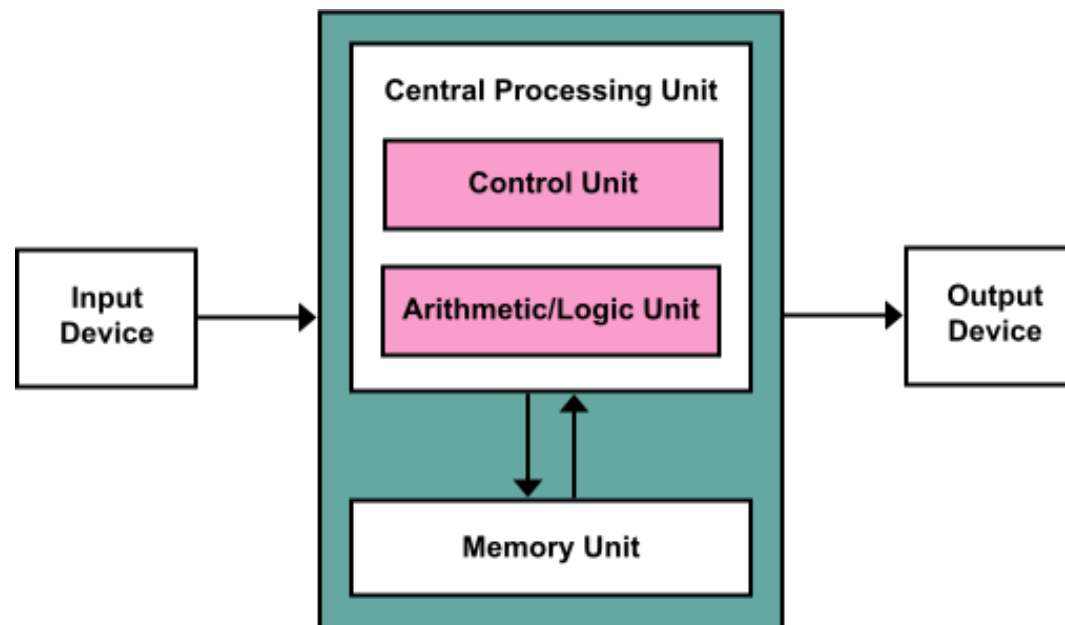
# Computer Architecture

# Stored Program Computer

- Operate according to "program" stored in memory
  - Run various applications on the same hardware
- Its idea can be traced back to the 1936 theoretical concept of a universal Turing machine
- Von Neumann was aware of the paper, and he impressed it on his collaborators as well

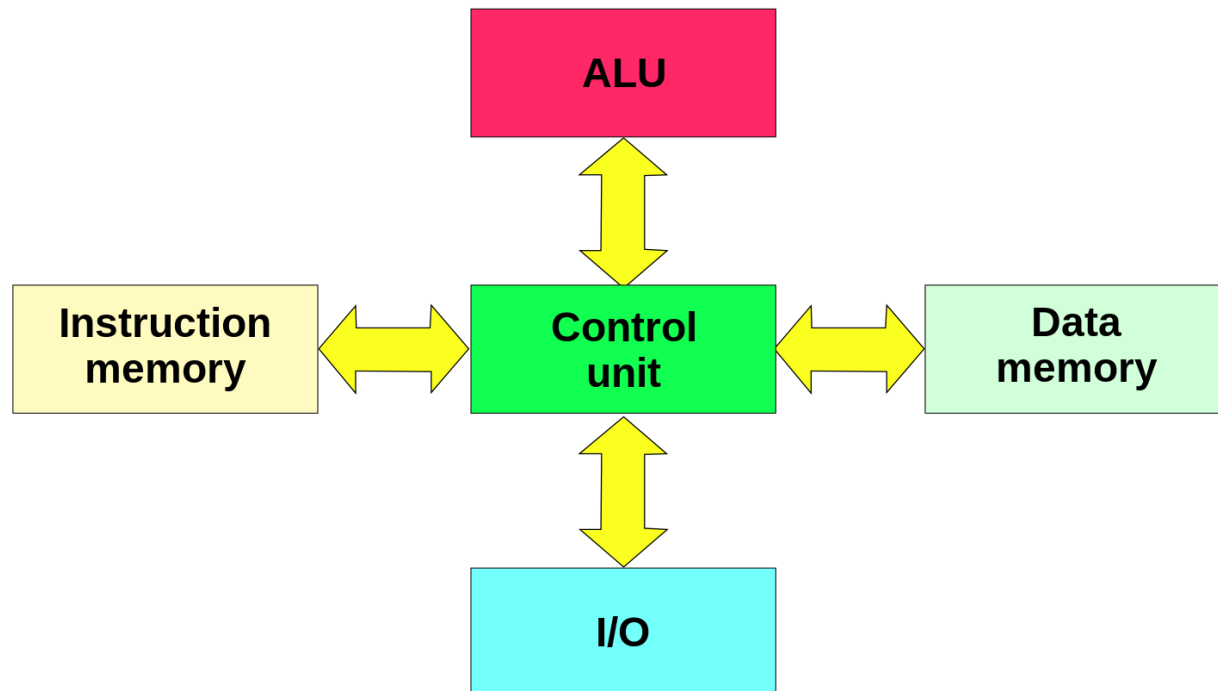
# von Neumann Architecture

- It also known as the von Neumann model and Princeton architecture
- Based on the 1945 description by the mathematician and physicist John von Neumann and others in the First Draft of a Report on the EDVAC



# Harvard Architecture

- physically separate storage and signal pathways for instructions and data
- From the Harvard Mark I relay-based computer, which stored instructions on punched tape (24 bits wide)

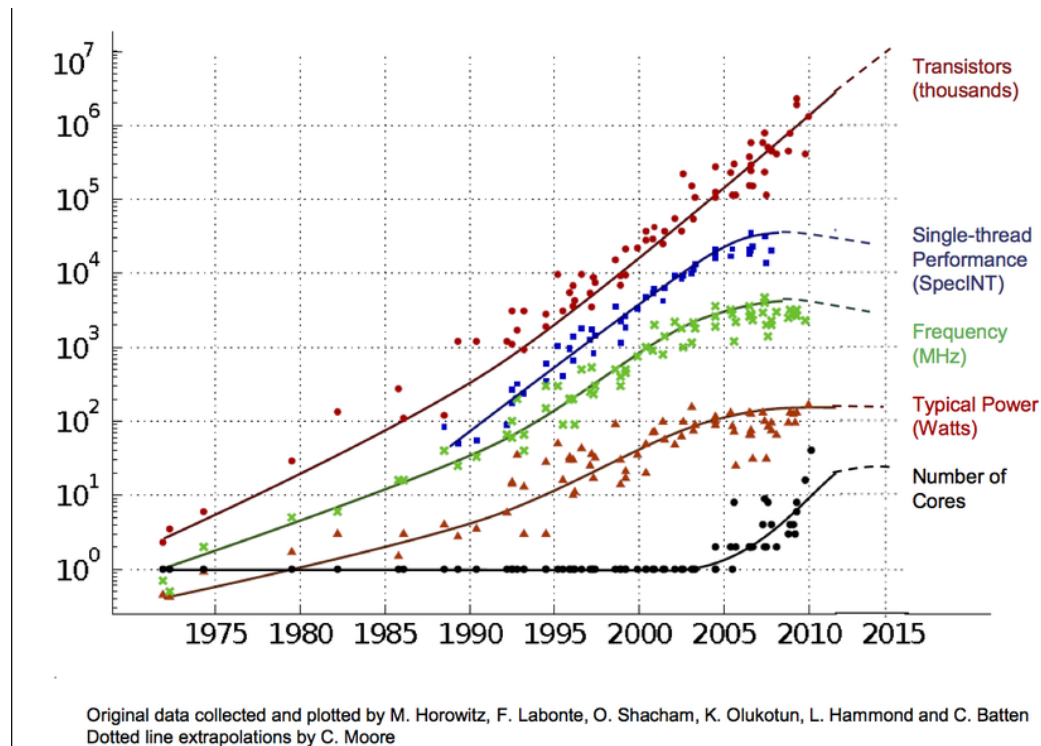


# Modern Computer Architecture

- Cache, and its prediction
- Out-of-order
- Hyper pipeline
- SIMD
- Super Scaler
- RISC vs. CISC
- Hyper threading
- Multi core/many core

# General-purpose v.s. Specified

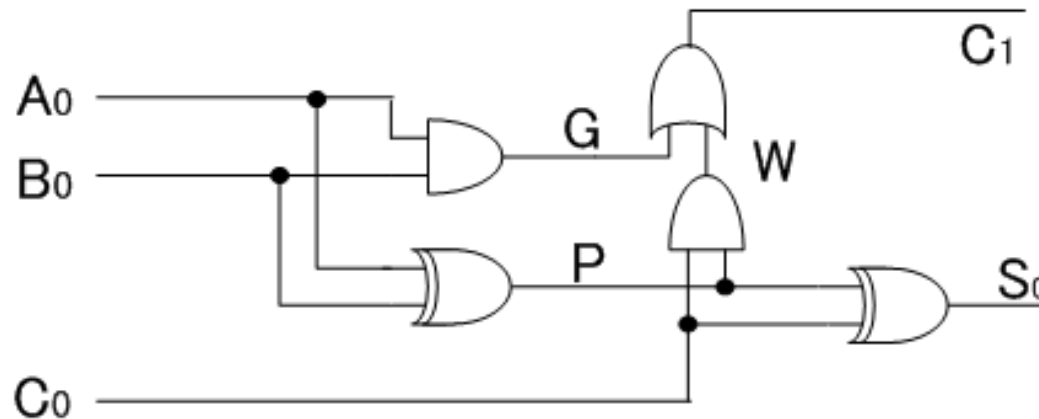
- Power-consumption wall
- Specified computer, however, dedicated application
- Special hardware on the same device? → **FPGA**



# Summary

- Boolean Logic
- Boolean Arithmetic
- Sequential Circuit
- Computer Architecture
  - Power wall
  - General-purpose
    - Special HW on the same device (FPGA)
    - Reconfigurability? Architecture? Design? Application?

# Exercise 2



1. (Mandatory) Show the truth table of full adder  $f=(x,y,z)$ , then convert an AND-OR canonical representation by using Karnaugh map
2. (Mandatory) Design a AND-OR canonical representation for above circuit
3. (Mandatory) Perform formal verification between  $f$  and above circuit, and these are the functionally same circuit? or not?

Send to OCW-I by a PDF file

Deadline is 21<sup>st</sup>, June, 2019 JST PM 13:20 (At the beginning of the next lecture)