Course Info

- Lab 5 will be released, get yourself prepared before going to lab sessions! (FSM preview, FSM background knowledge not a must, just use sum of minterm/Karnaugh map)
- Project 1.1! Start early! Do not waste your slip days (CS110P)! DDL March 13th.
- Project 1.2 will be available this week, and will be marked in lab sessions. Deadline March 31th.
- HW3 ddl March 18th.
- Next week discussion on synchronized digital system (SDS)

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

CS 110
Computer Architecture
Combinational & Sequential Circuits

Instructors:

Siting Liu & Chundong Wang

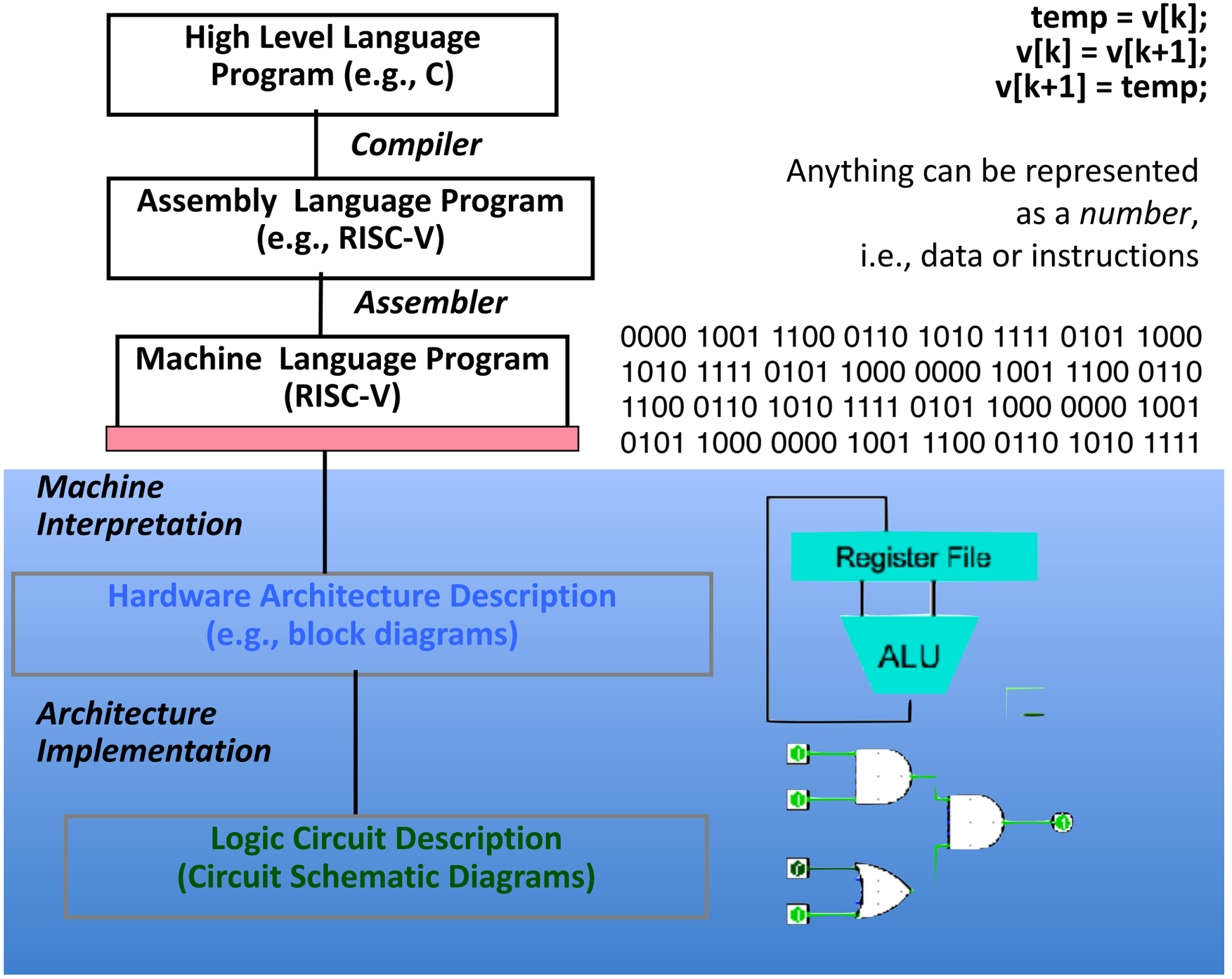
Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2023/index.html

School of Information Science and Technology (SIST)

ShanghaiTech University

2023/3/5

Where are we?

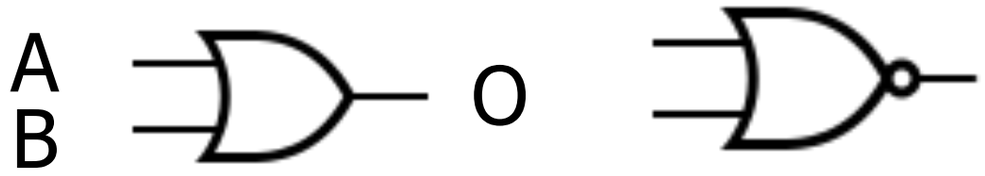

High Level Language Program (e.g., C)
Compiler
Assembly Language Program (e.g., RISC-V)
Assembler
Machine Language Program (RISC-V)
Machine Interpretation
Hardware Architecture Description (e.g., block diagrams)
Architecture Implementation
Logic Circuit Description (Circuit Schematic Diagrams)
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
Anything can be represented as a *number*, i.e., data or instructions
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
Register File
ALU

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!

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

$\bar{A}\bar{B}$	m_0
$\bar{A}B$	m_1
$A\bar{B}$	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

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 Simplification Method —Karnauph Map

Gray coded

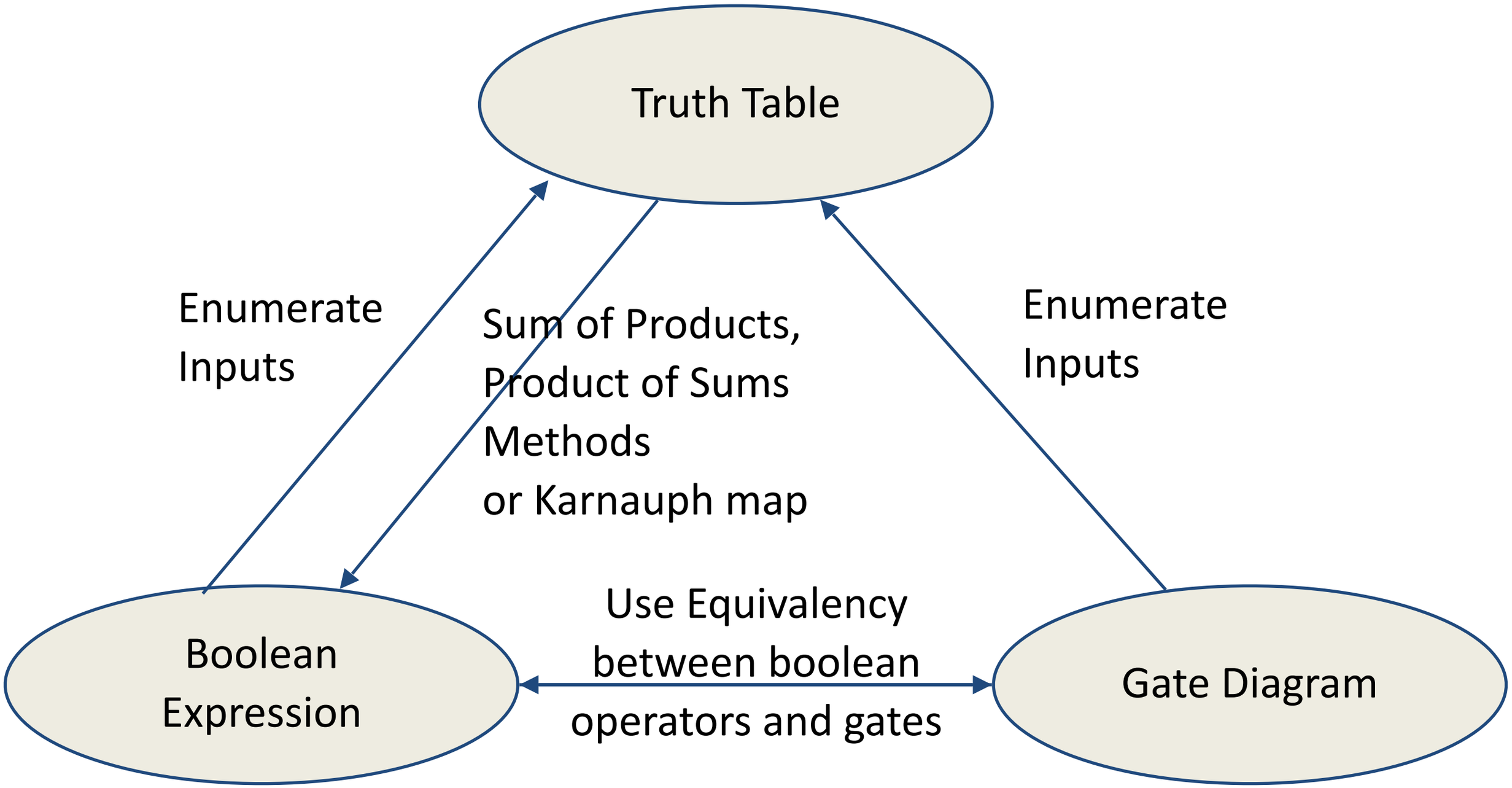
CD \ AB	00	01	11	10
00				
01			1	
11		1	1	1
10			1	1

Gray coded

Each cell corresponds to a minterm

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

Representations of Combinational Logic

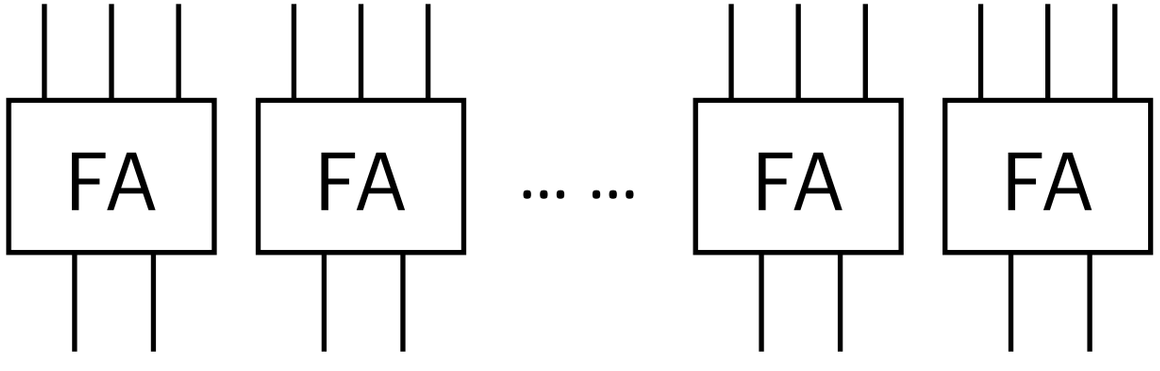


Build Larger Blocks—like LEGO®

```
  01010101
+ 01110011
```

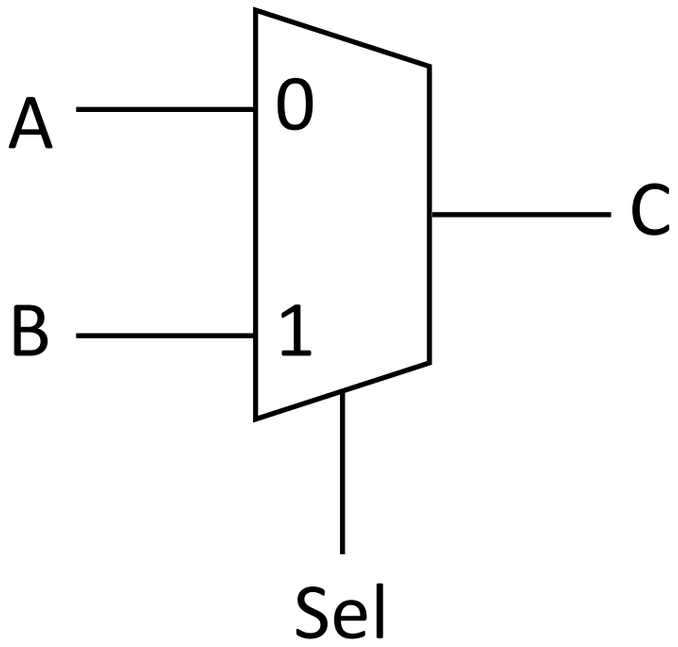
- Build a full adder (FA): truth table

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

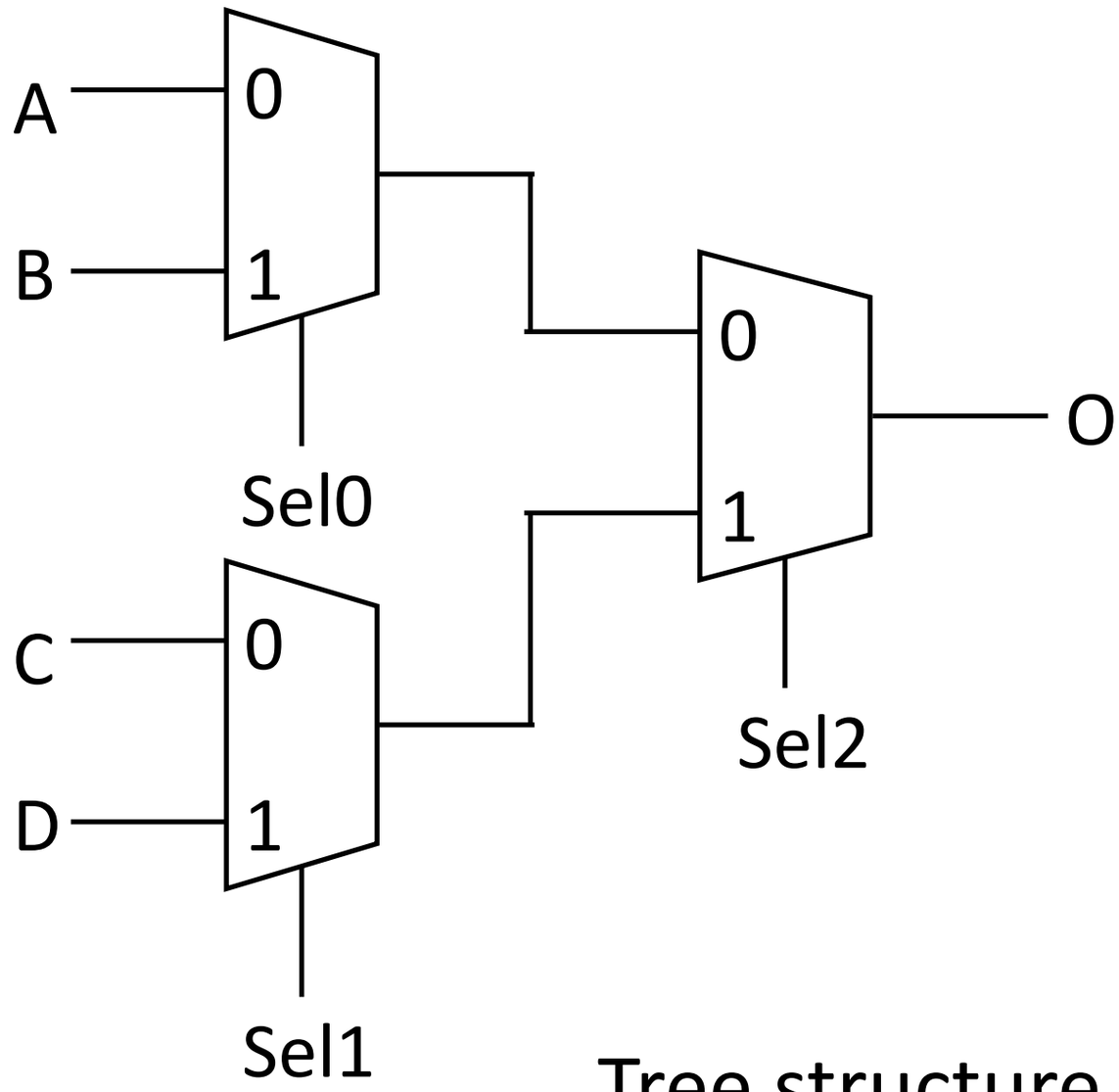


Other Useful Combinational Circuits

- Multiplexer (2-to-1)



- Multiplexer (2^n-to-1)

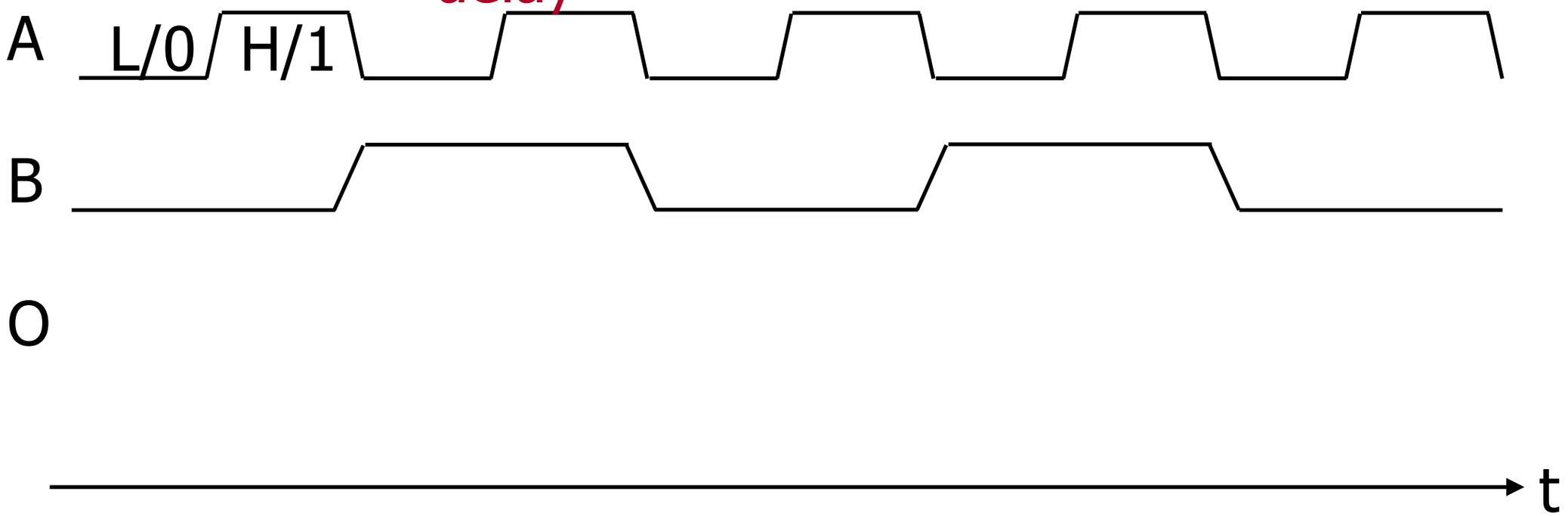


Tree structure

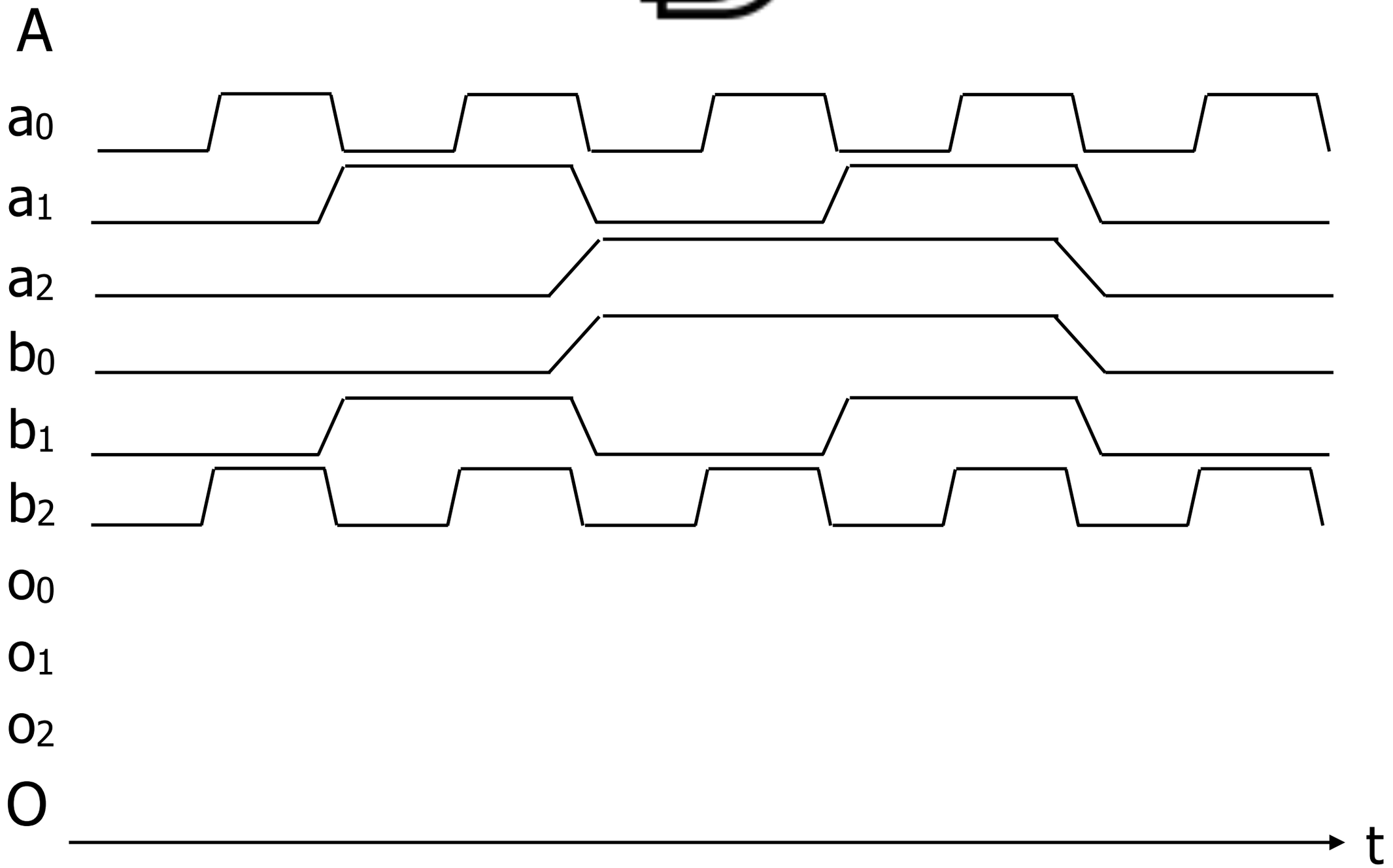
Timing Diagram—Signal & Waveform

Transistors: non-ideal switches

Propagation delay



Timing Diagram—Signal Grouping



Build an ALU

Instructions are Abstract of Hardware

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI

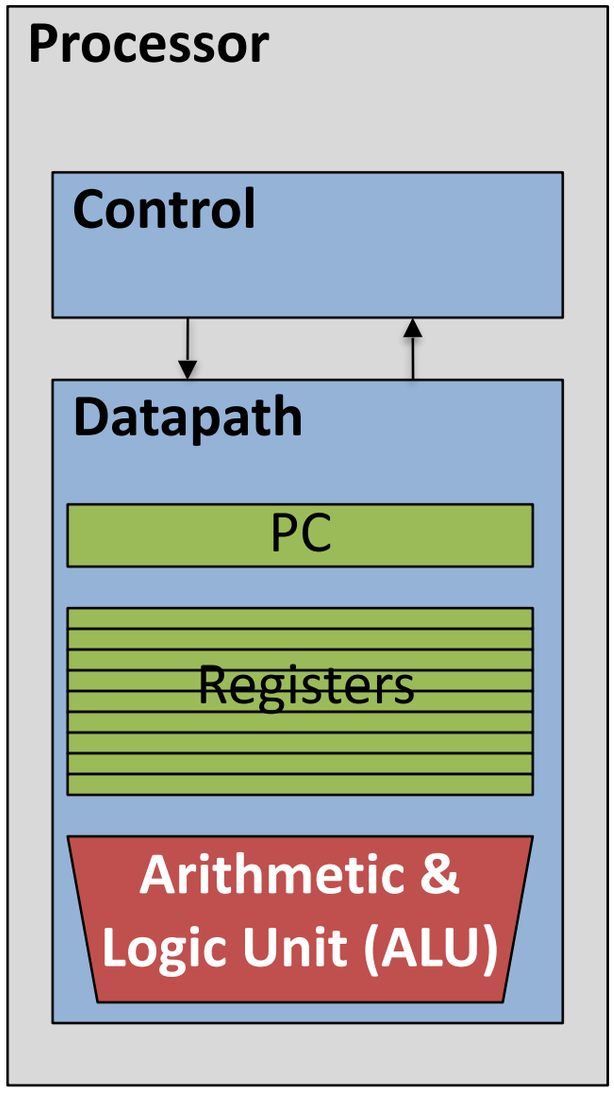
Instructions are Abstract of Hardware

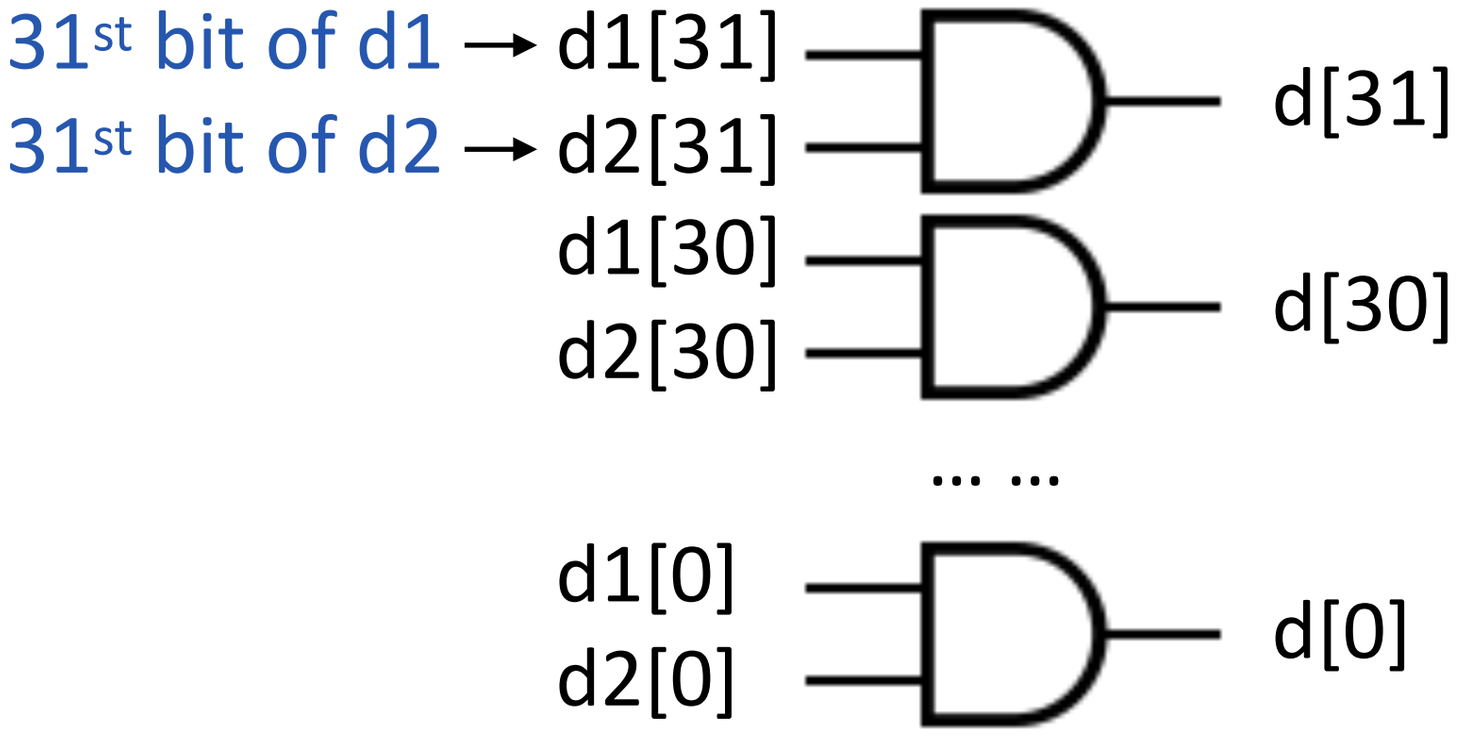
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000	pred	succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK
csr			rs		d	1110011	CSRRW
csr			rs		d	1110011	CSRRS
csr			rs		d	1110011	CSRRC
csr			zimm	101	rd	1110011	CSRRWI
csr			zimm	110	rd	1110011	CSRRSI
csr			zimm	111	rd	1110011	CSRRCI

Not in CS110

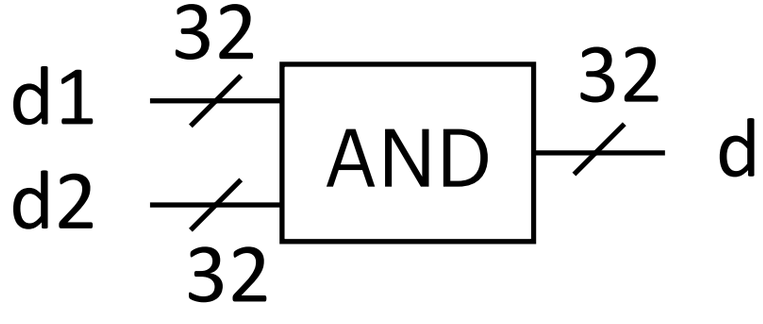
An Arithmetic & Logic Unit (ALU)

- Arithmetic: Add/Sub/Addi
- Logic: **And**/Or/Xor(i) (bit-wise)



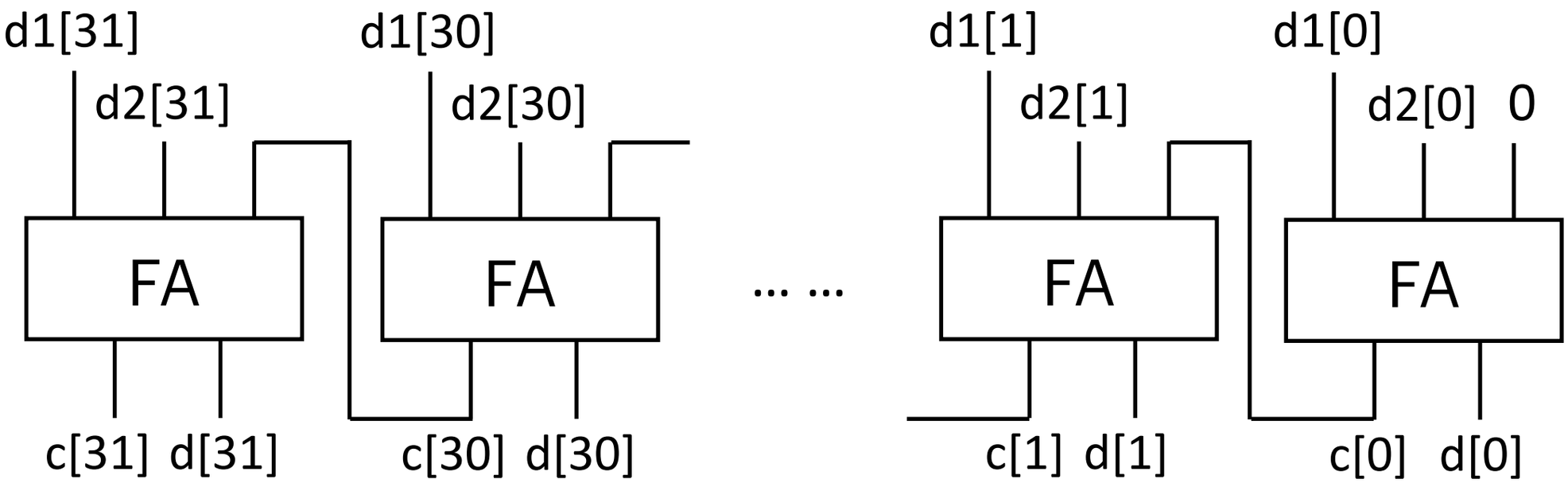


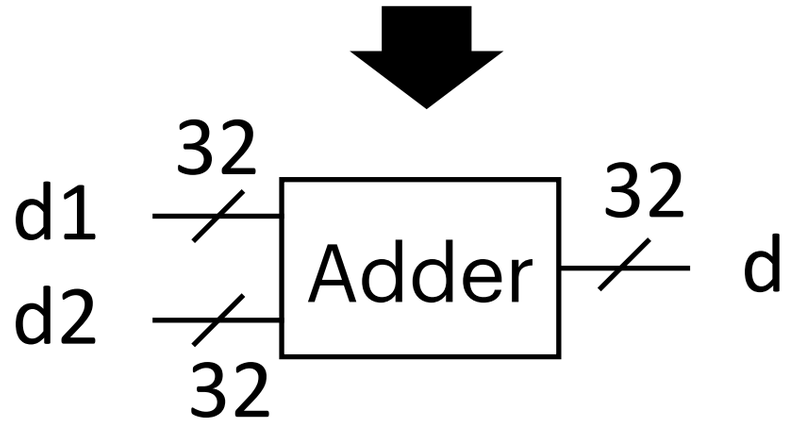
Or/Xor similar



An Arithmetic & Logic Unit (ALU)

- Arithmetic: **Add**/Sub/**Addi**
- Logic: And/Or/Xor(i) (bit-wise)

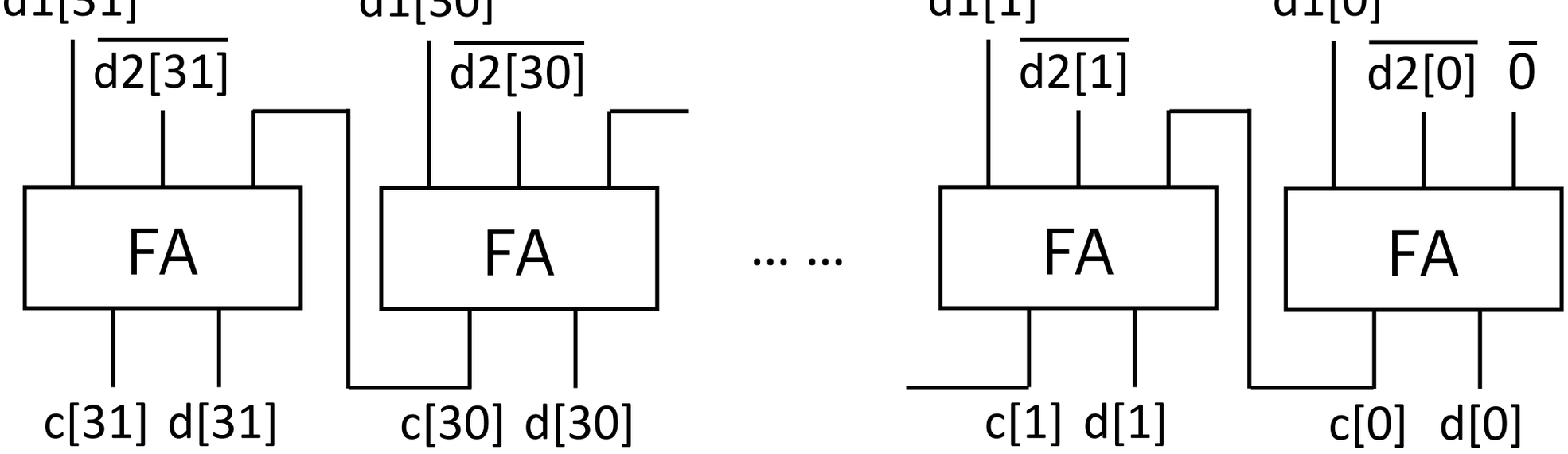




An Arithmetic & Logic Unit (ALU)

- Arithmetic: Add/**Sub**/Addi
- Logic: And/Or/Xor(i) (bit-wise)

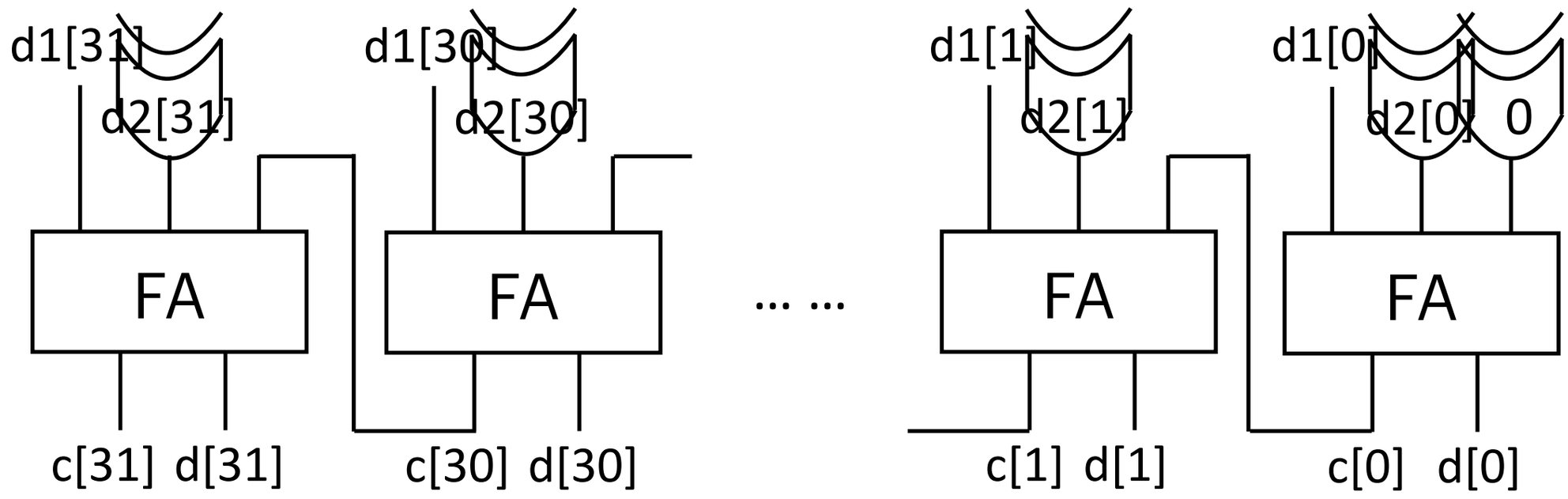
$$A - B = A + (-B) = A + \overline{B} + 1 \pmod{2^{N-1}}$$



An Arithmetic & Logic Unit (ALU)

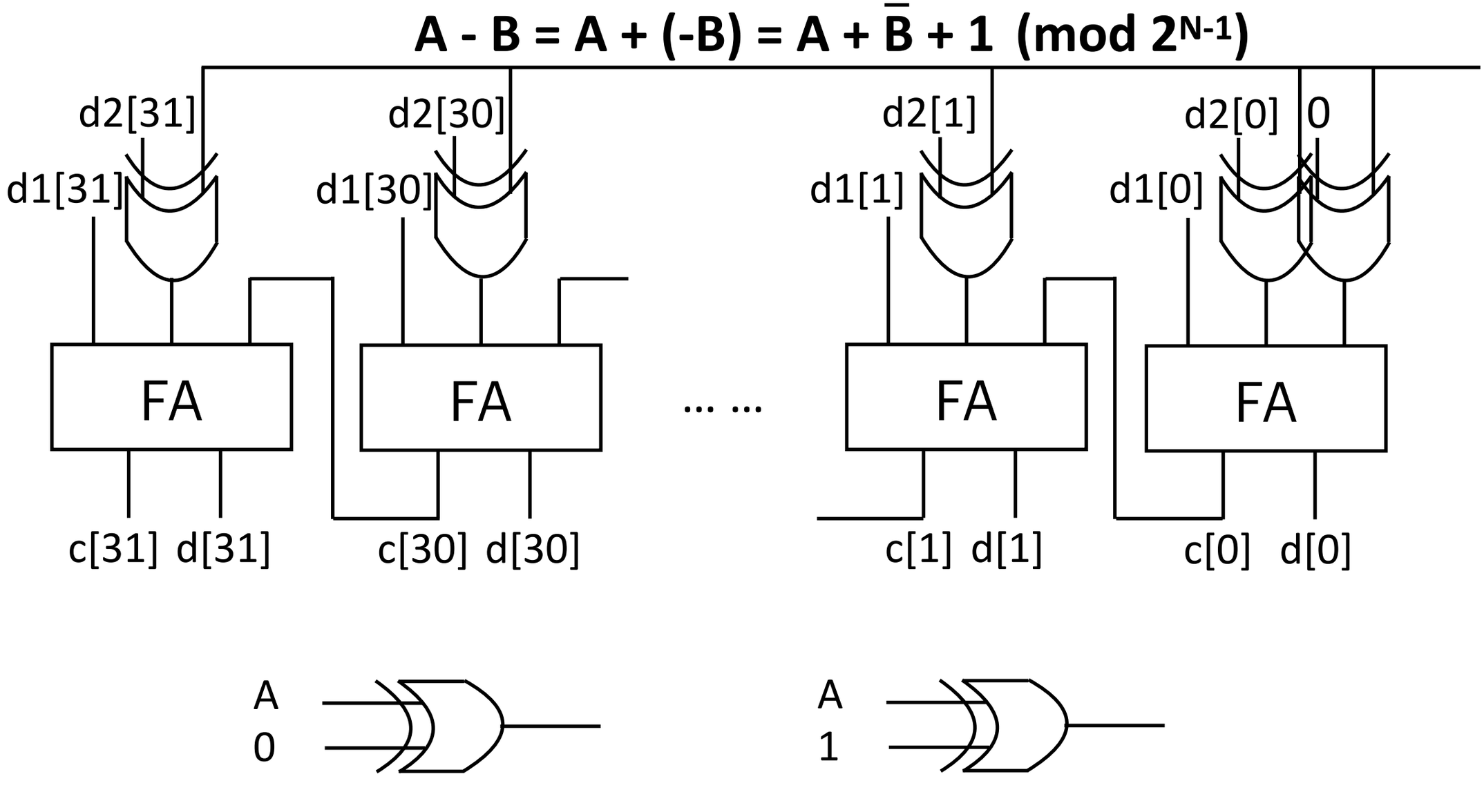
- Arithmetic: Add/**Sub**/Addi
- Logic: And/Or/Xor(i) (bit-wise)

$$A - B = A + (-B) = A + \bar{B} + 1 \pmod{2^{N-1}}$$



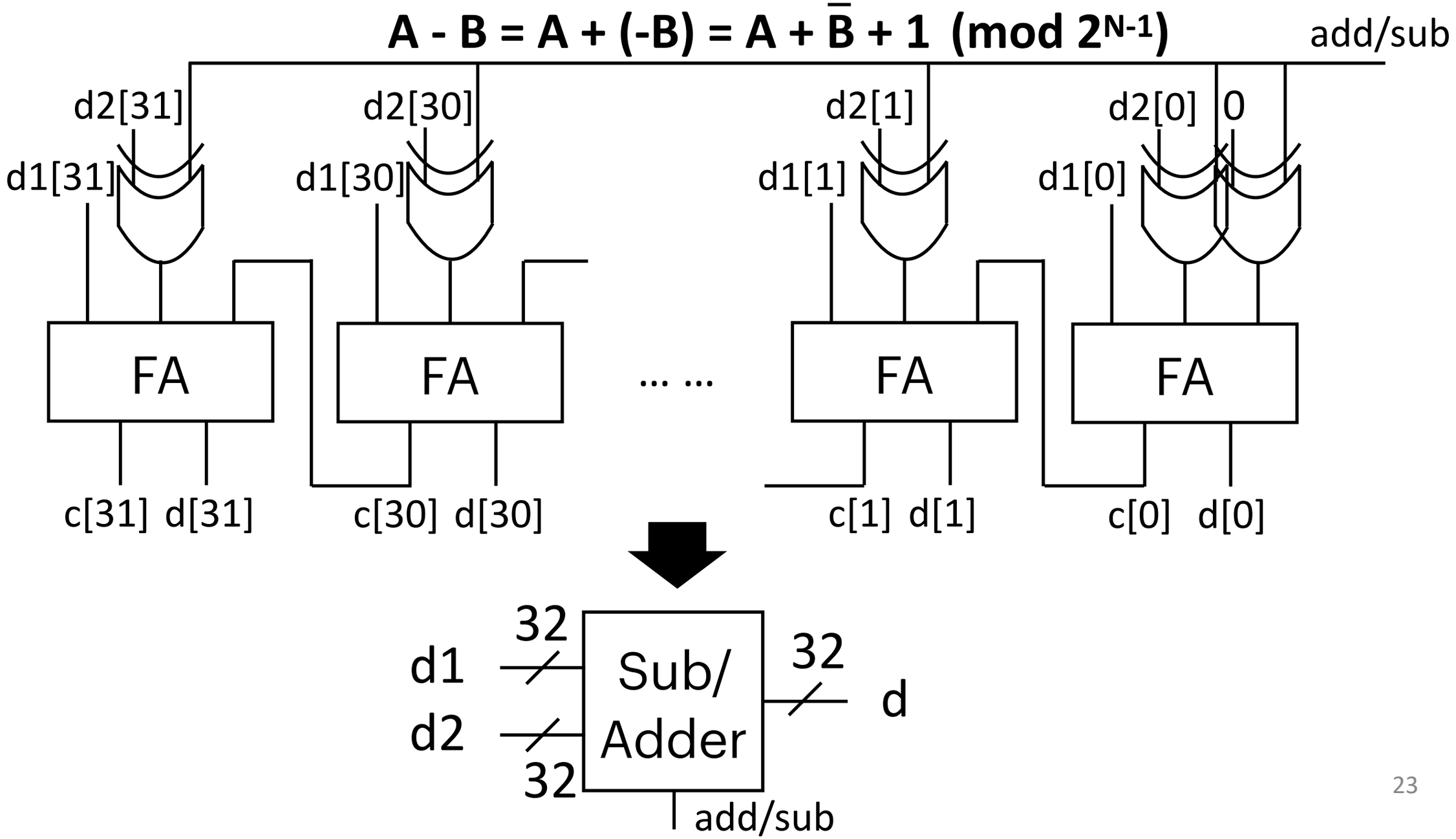
An Arithmetic & Logic Unit (ALU)

- Arithmetic: Add/**Sub**/Addi
- Logic: And/Or/Xor(i) (bit-wise)



An Arithmetic & Logic Unit (ALU)

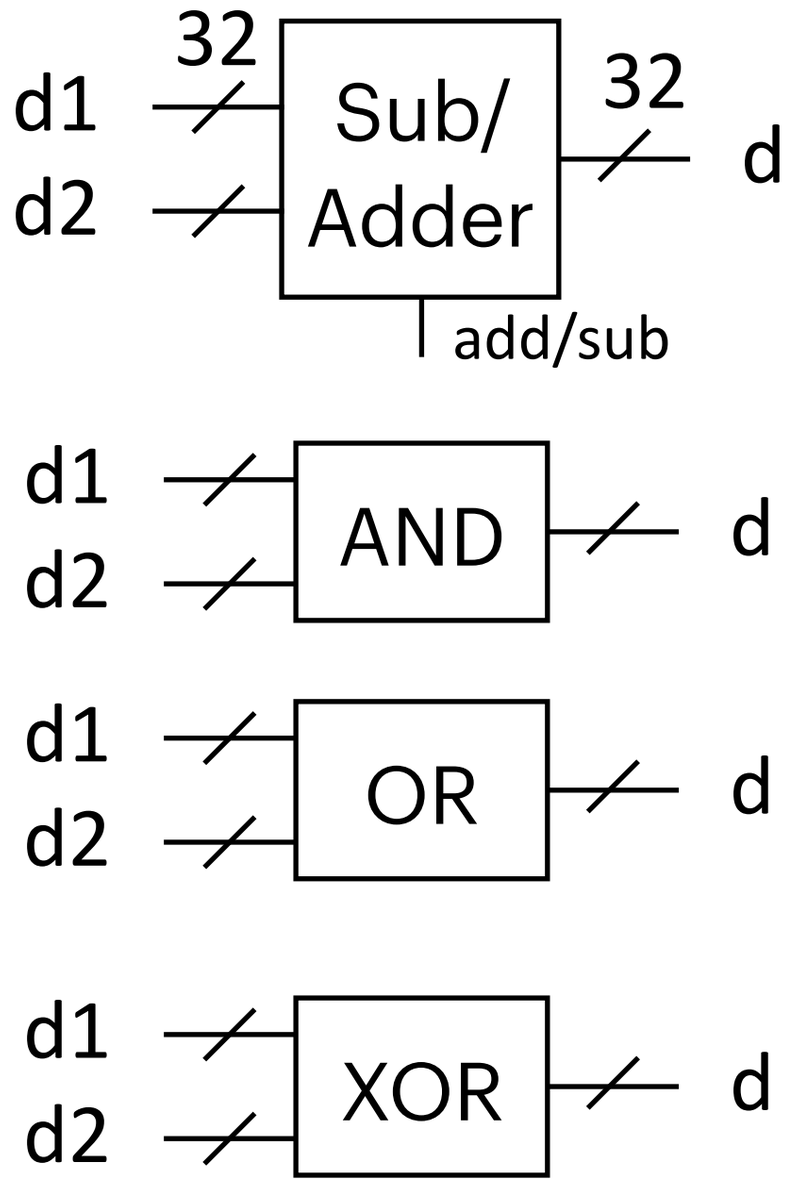
- Arithmetic: Add/**Sub**/Addi
- Logic: And/Or/Xor(i) (bit-wise)



An Arithmetic & Logic Unit (ALU)

- Arithmetic: Add/Sub/Addi
- Logic: And/Or/Xor(i) (bit-wise)

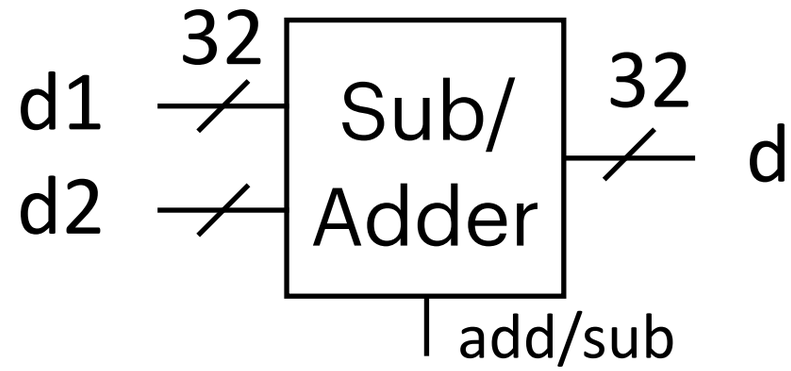
From `rs1`/`rs2`/`imm`

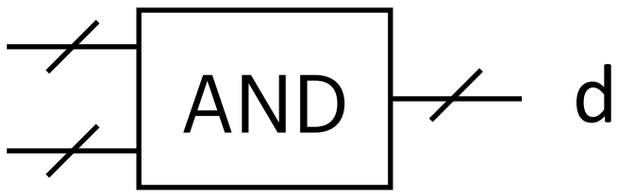


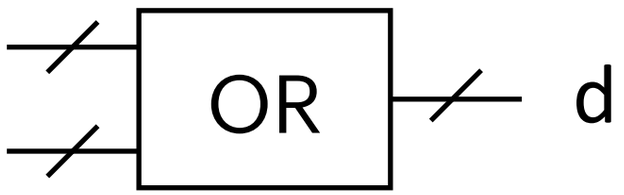
An Arithmetic & Logic Unit (ALU)

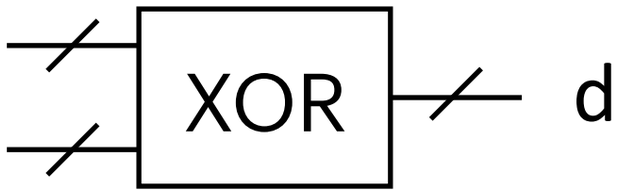
- Arithmetic: Add/Sub/Addi
- Logic: And/Or/Xor(i) (bit-wise)

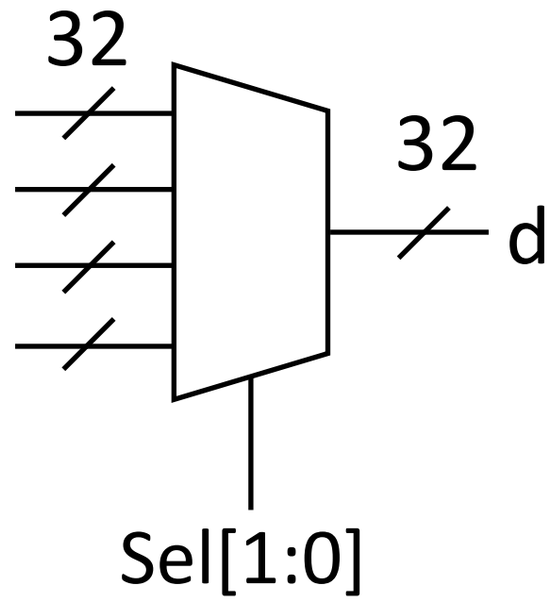
From `rs1`/`rs2`/`imm`











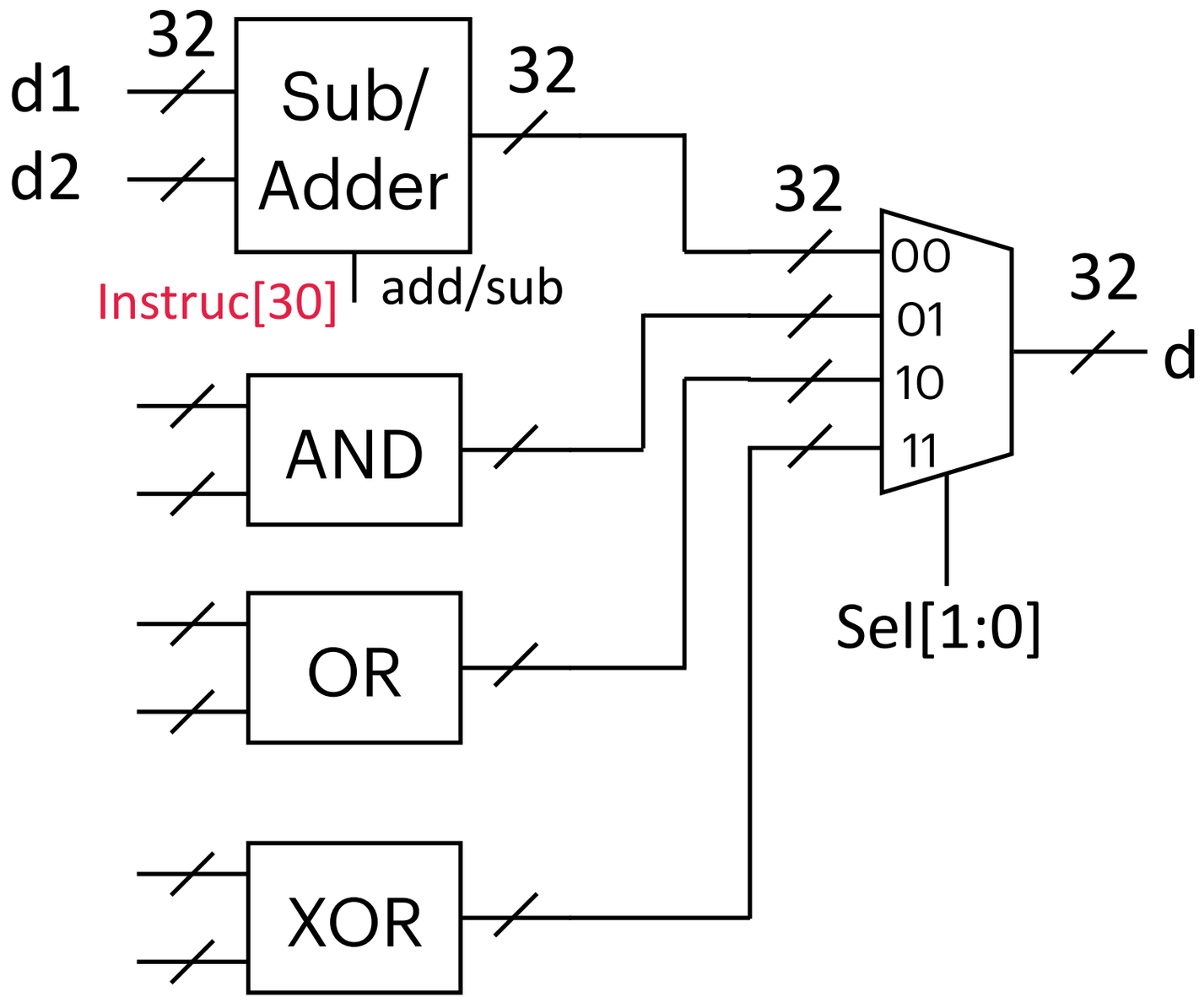
Recall: RISC-V Instruction Format

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

imm	rs1	000	rd	0010011	ADDI
imm	rs1	100	rd	0010011	XORI
imm	rs1	110	rd	0010011	ORI
imm	rs1	111	rd	0010011	ANDI

Recall: RISC-V Instruction Format

funct3		sel
000	ADD(I)	00
000	SUB	00
100	XOR(I)	11
110	OR(I)	10
111	AND(I)	01

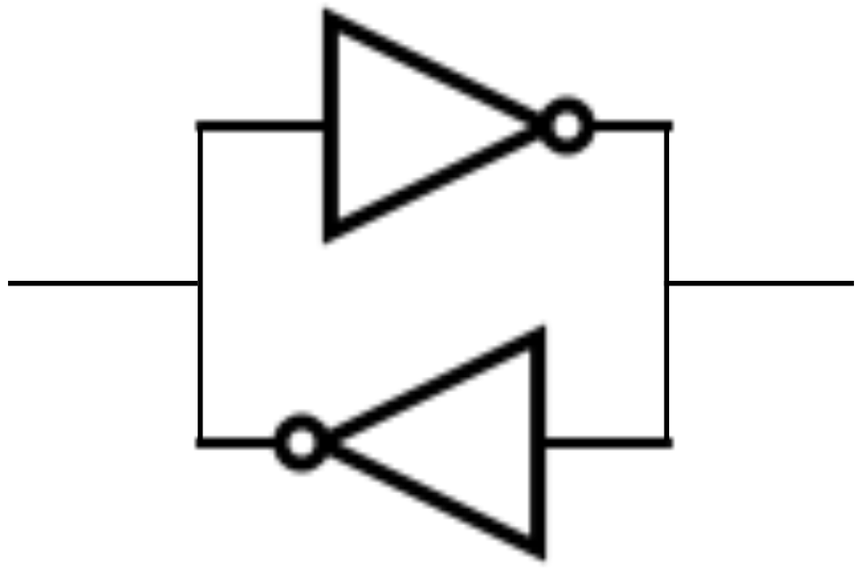


Sequential Elements

What about the registers?

Combinational Circuits with Feedbacks

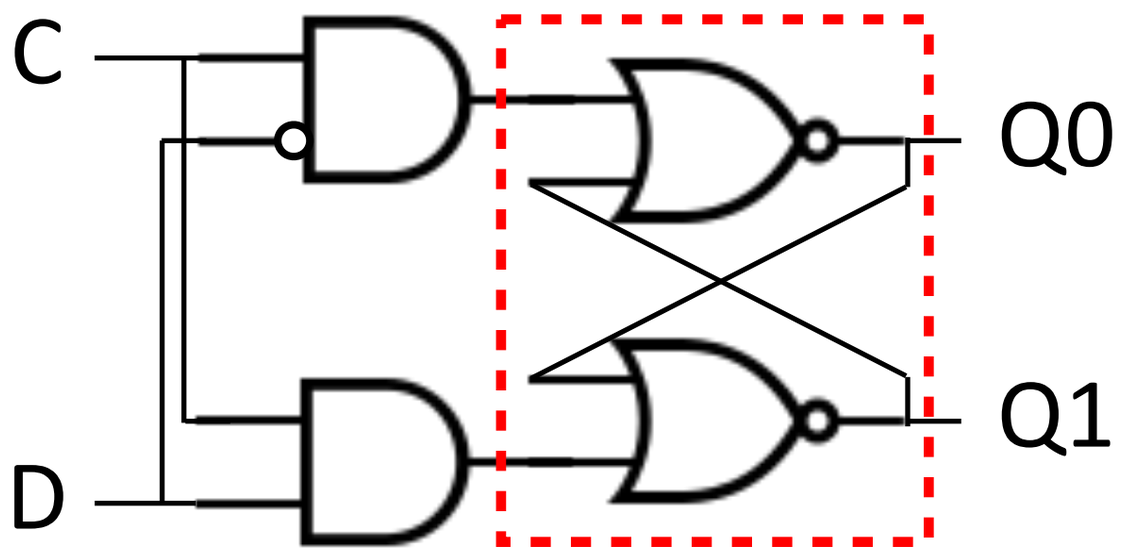
- Circuits that can remember or store information (steady state only)
- Output depends on not only input but also current state



This structure can implement SRAM/ latches/FFs.

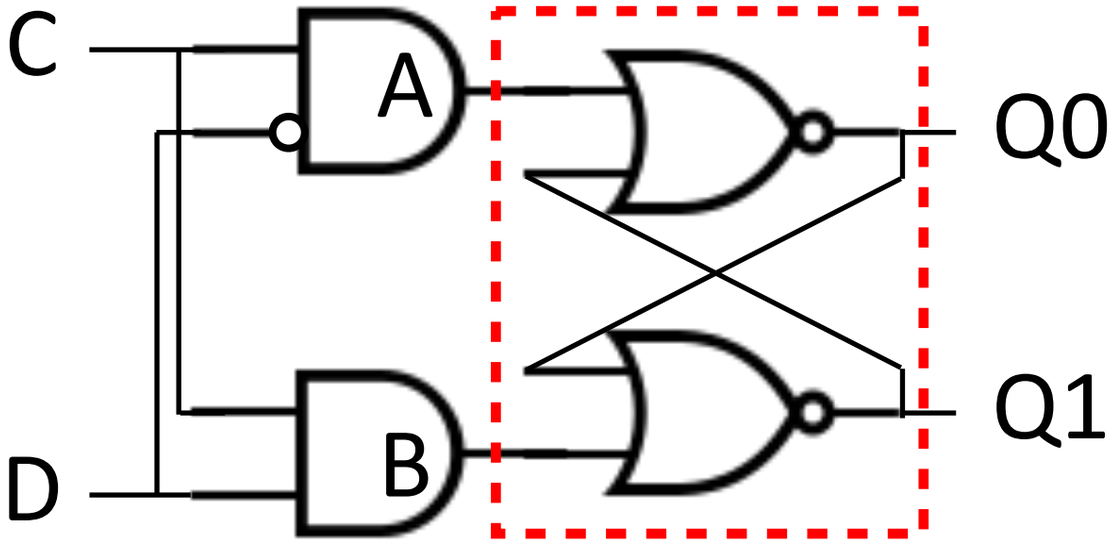
D Latches & D Flip-Flops

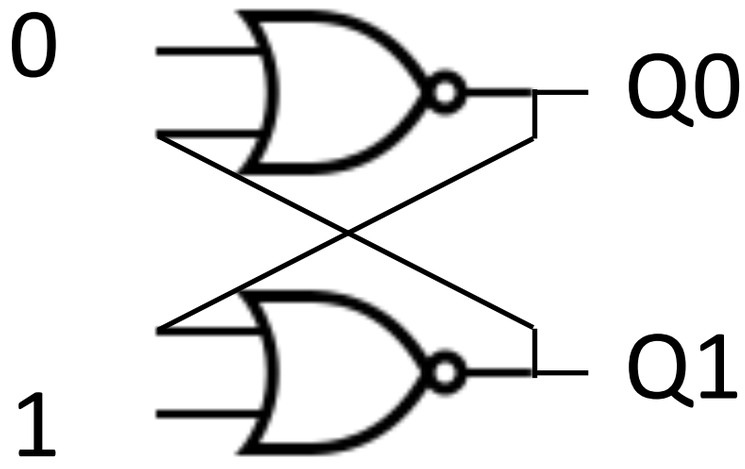
- Latches & Flip-Flops are basic sequential circuits
  - D/R-S/J-K/T



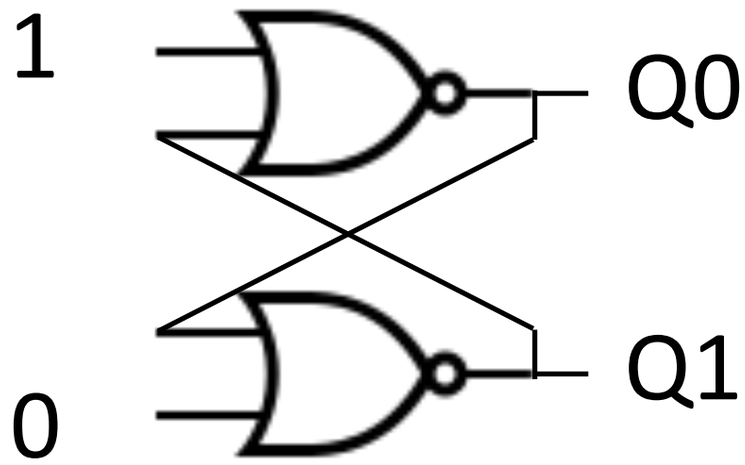
One possible implementation of a D latch.

D Latches

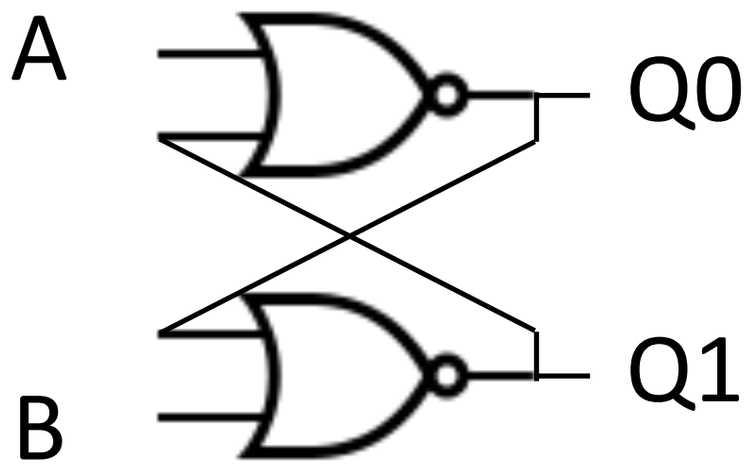


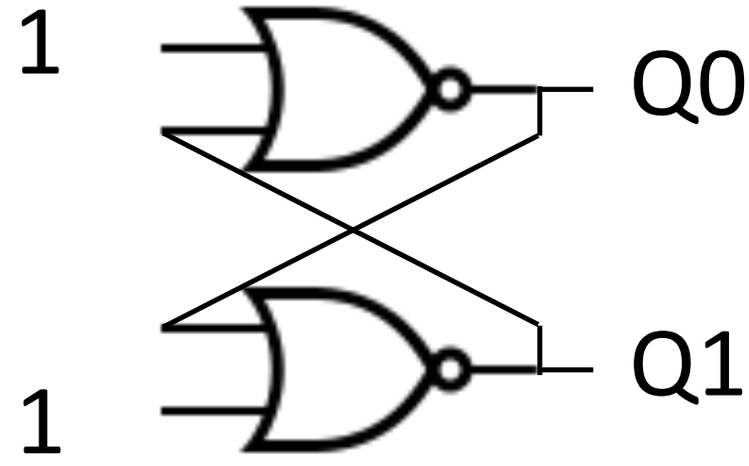


A	B	Q0	Q1
0	0		
0	1		
1	0		
1	1		

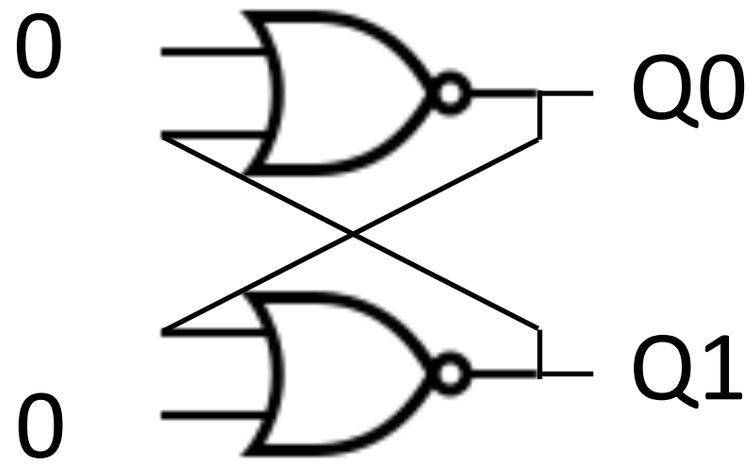


D Latches

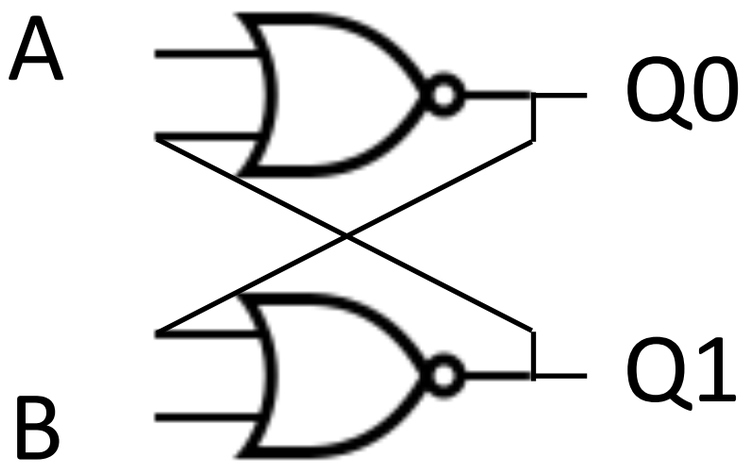


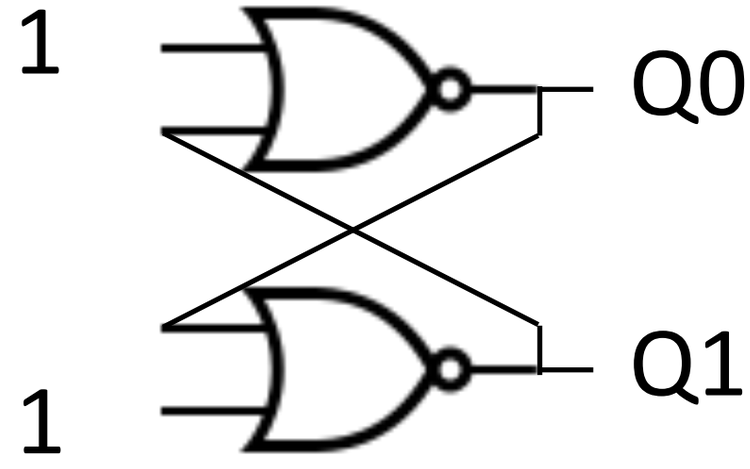


A	B	Q0	Q1
0	0		
0	1		
1	0		
1	1		

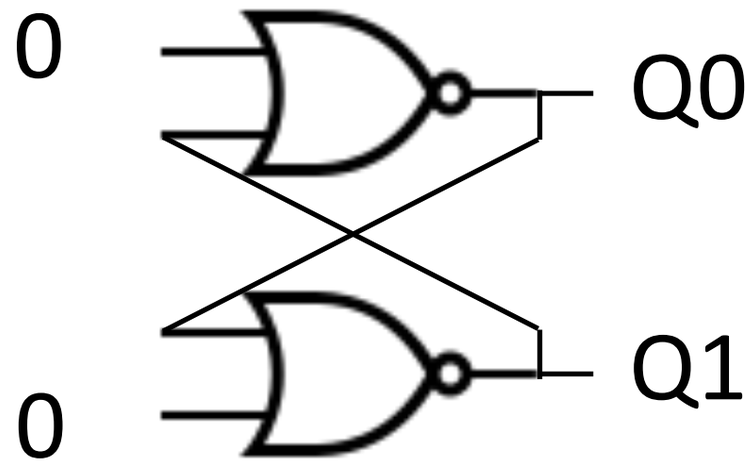


D Latches

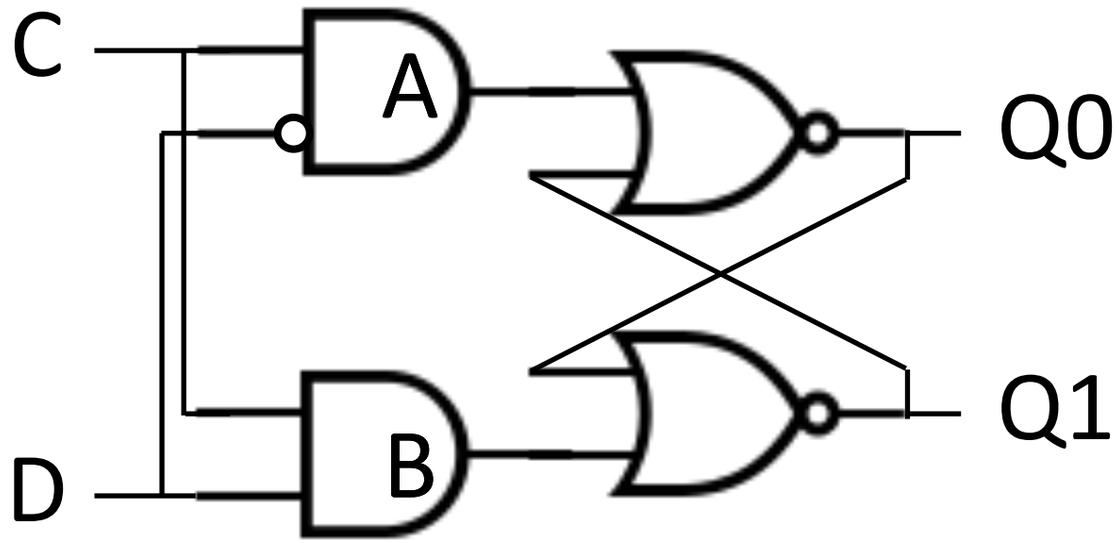




A	B	$Q_0^{(n+1)}$	$Q_1^{(n+1)}$
0	0		
0	1		
1	0		
1	1		

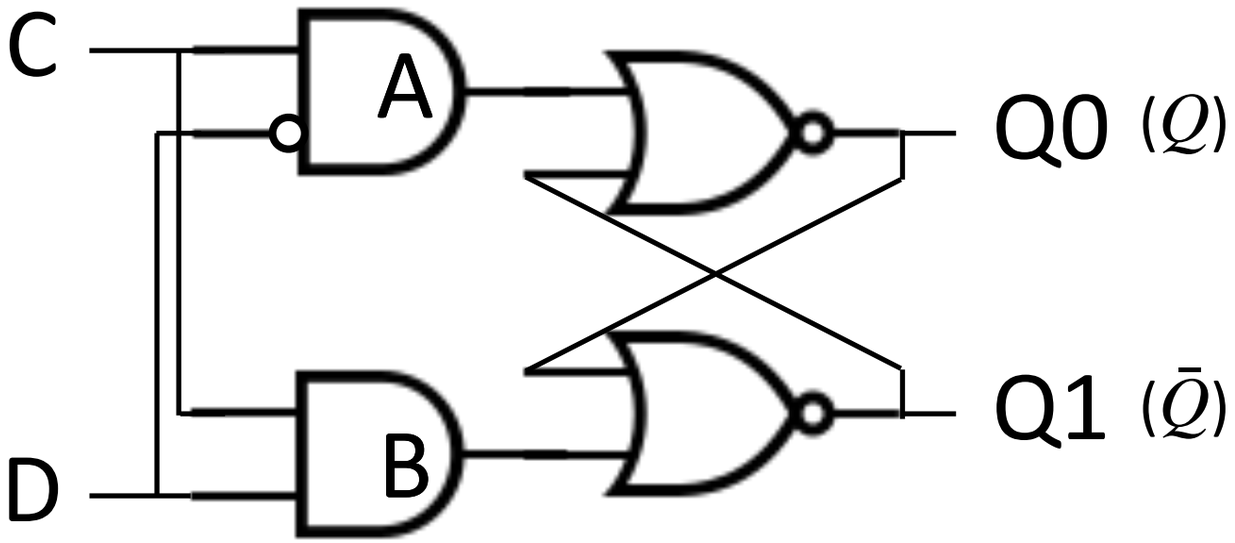


D Latches



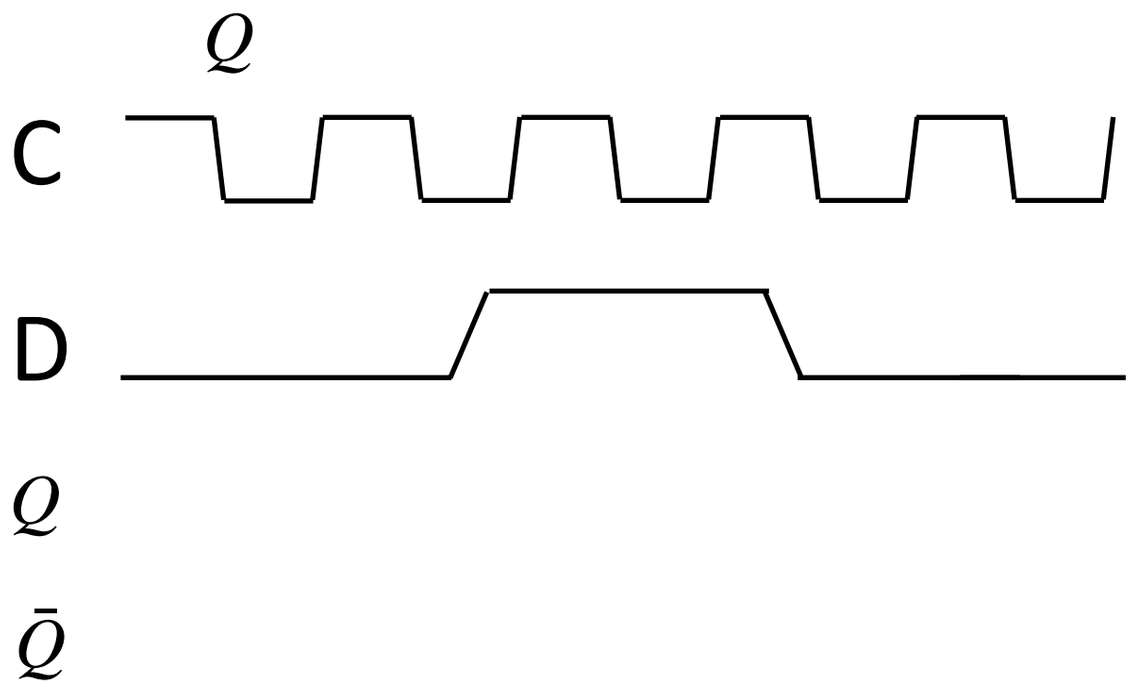
C	D	A	B	$Q_0^{(n+1)}$	$Q_1^{(n+1)}$
0	0				
0	1				
1	0				
1	1				

D Latches—Timing Diagram



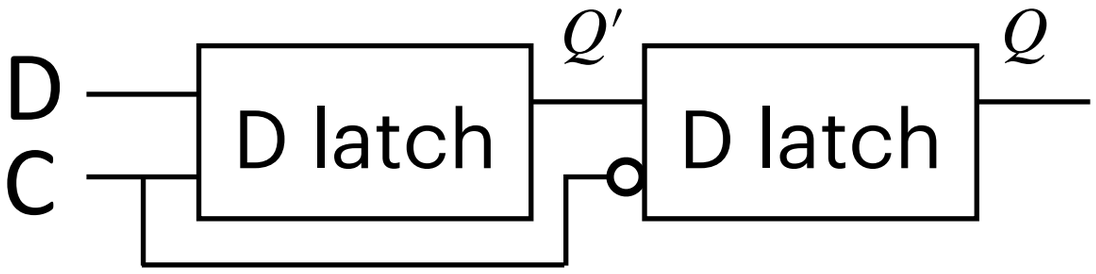
A symbol.

C	D	A	B	$Q_0^{(n+1)}$	$Q_1^{(n+1)}$
0	0				
0	1				
1	0				
1	1				



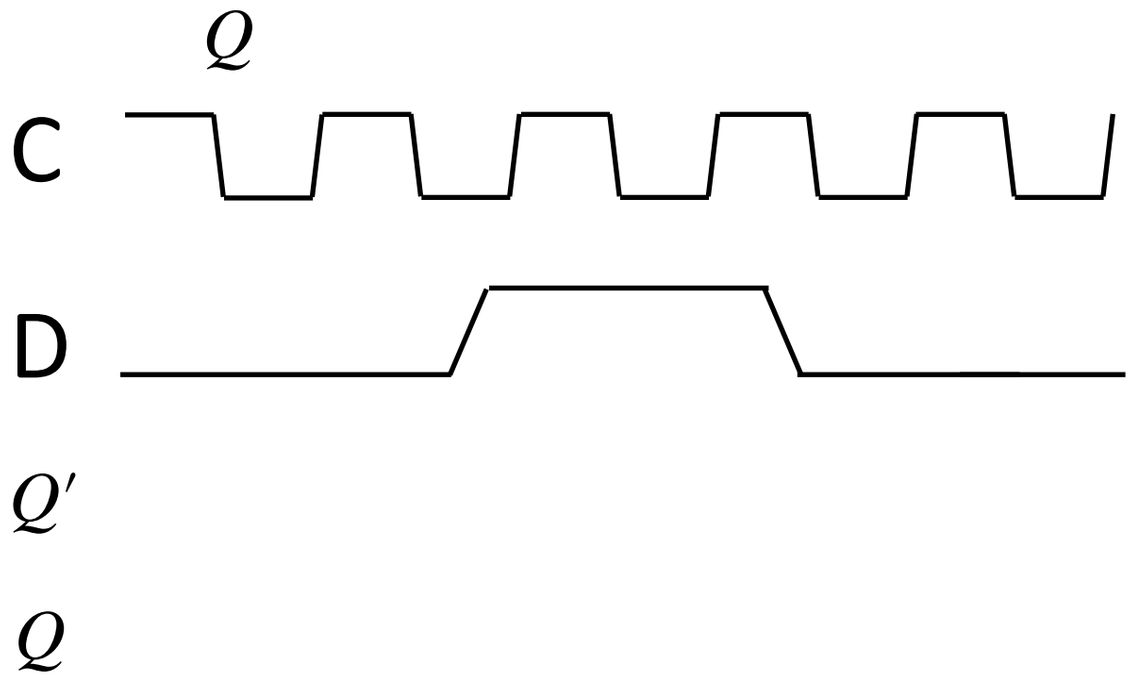
Level-triggered

D Flip-Flops



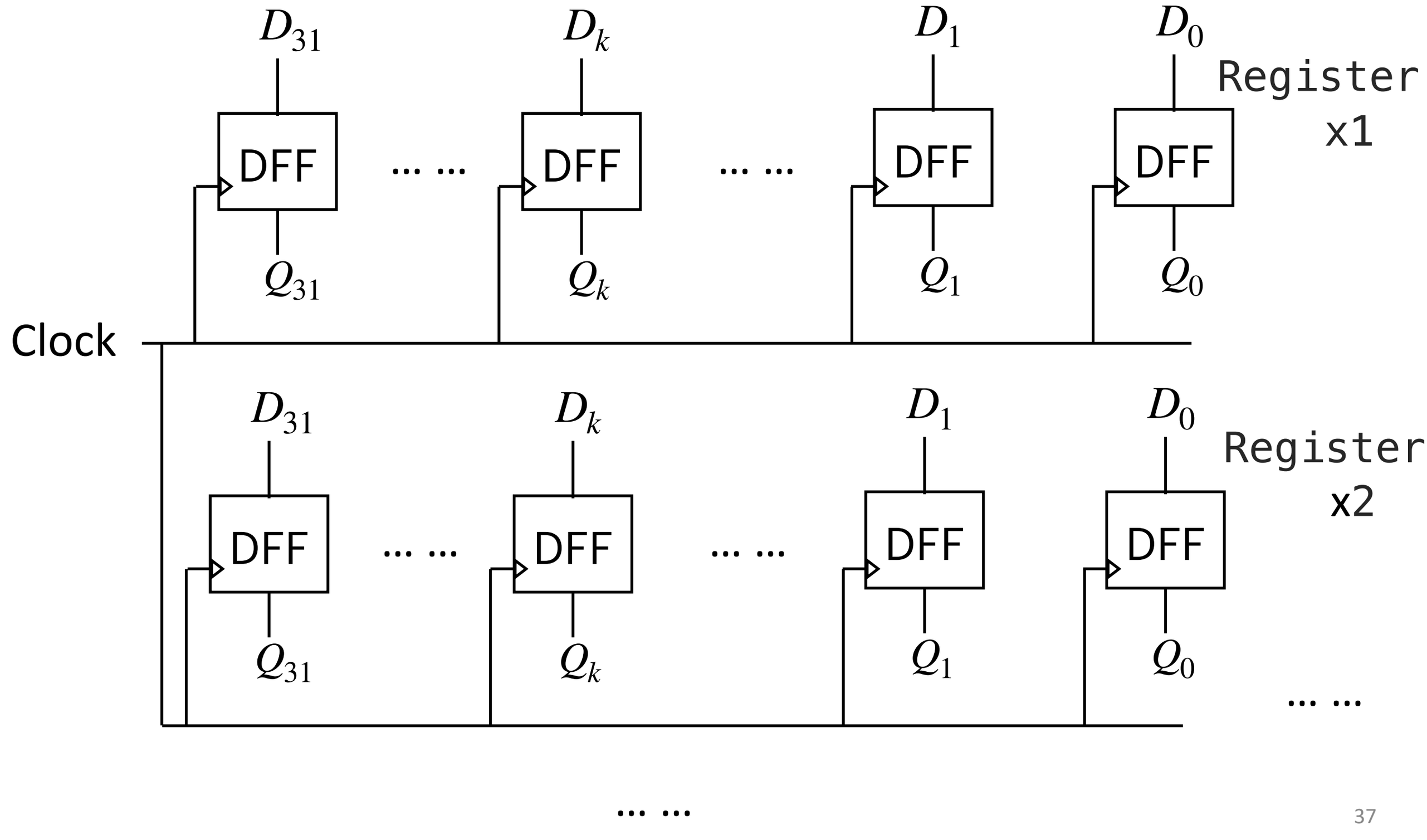
A symbol.

C	D	$Q'^{(n+1)}$	$Q^{(n+1)}$
0	0		
0	1		
1	0		
1	1		

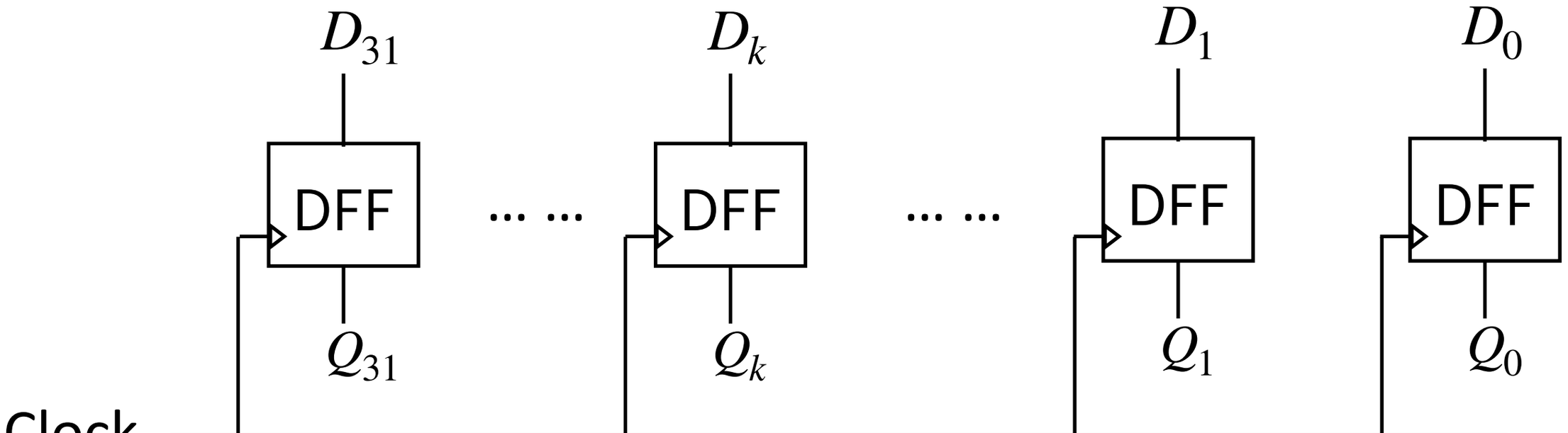


Edge-triggered

Registers & Synchronized Circuits

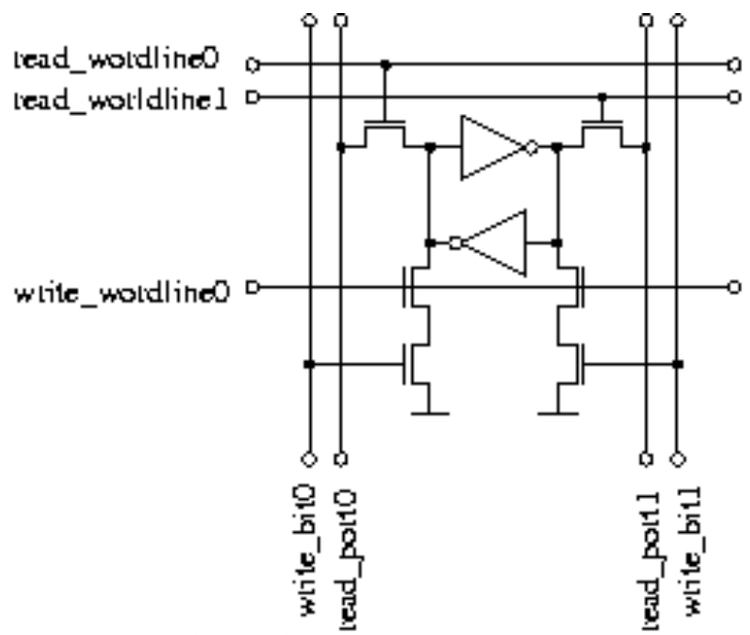


Registers & Synchronized Circuits



Clock: operations coordinated by it; Help control flow of combinational blocks; “Heartbeat” of the system

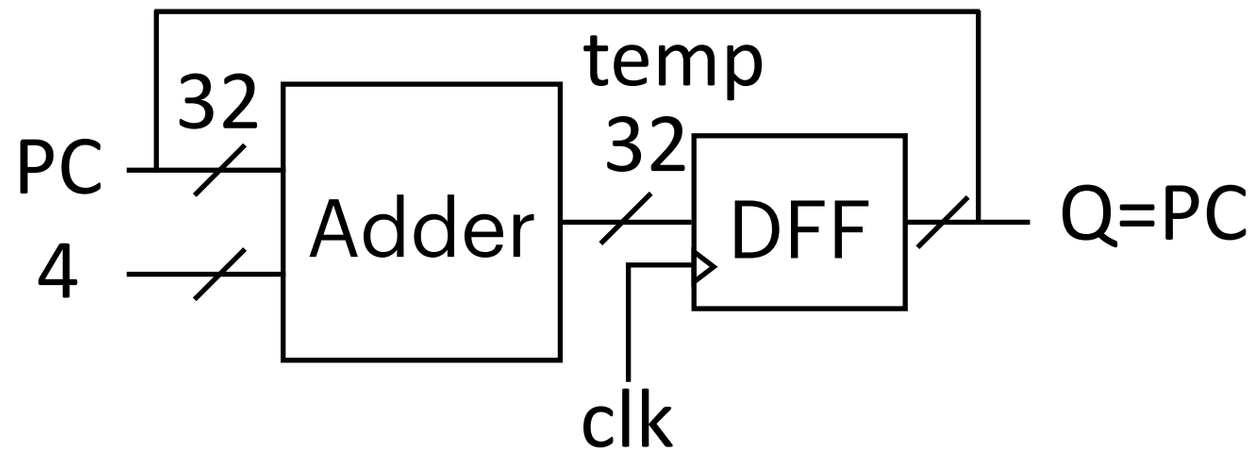
Modern CPUs use fast SRAMs with multiple (dedicated read and write) ports as register files.

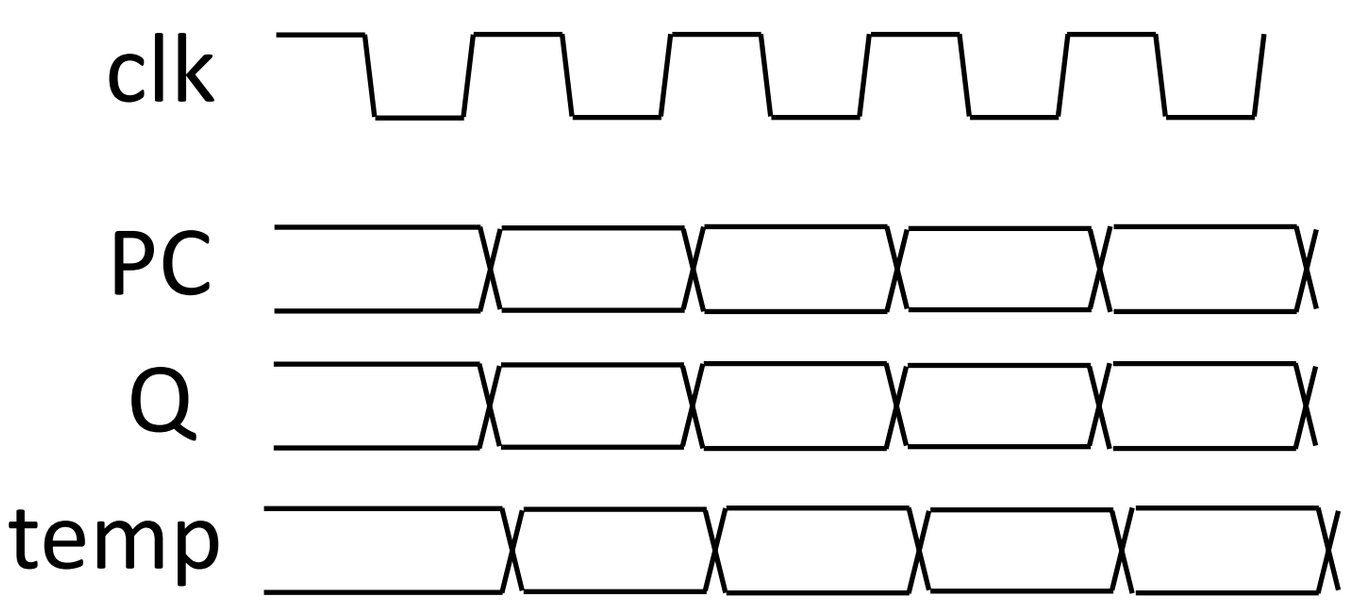


From Wikipedia.

Other Usage of Synchronized Circuits

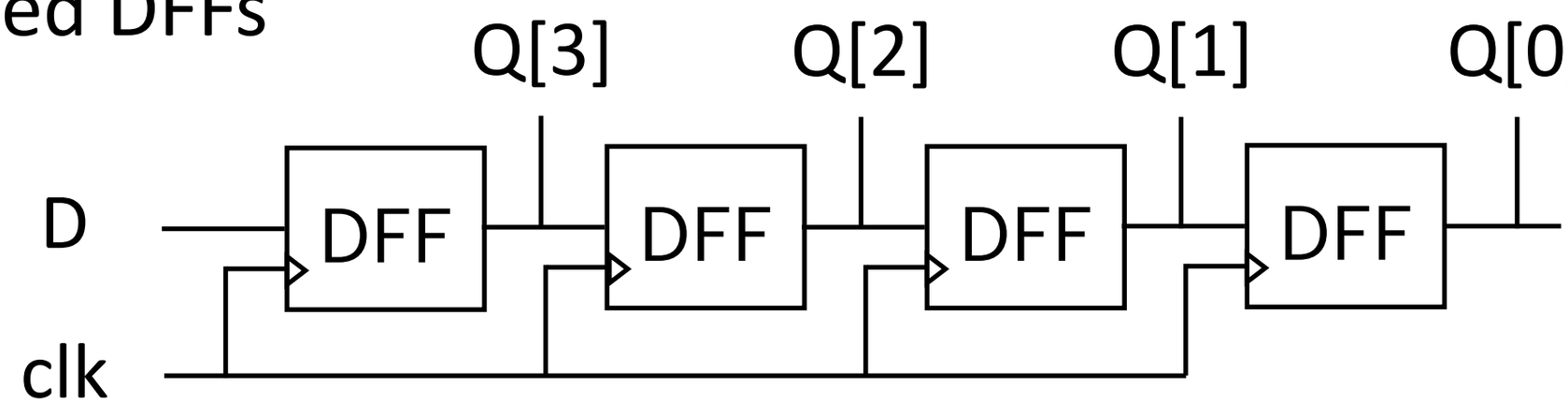
- PC counter: PC = PC + 4 (w/o considering branch/jump)

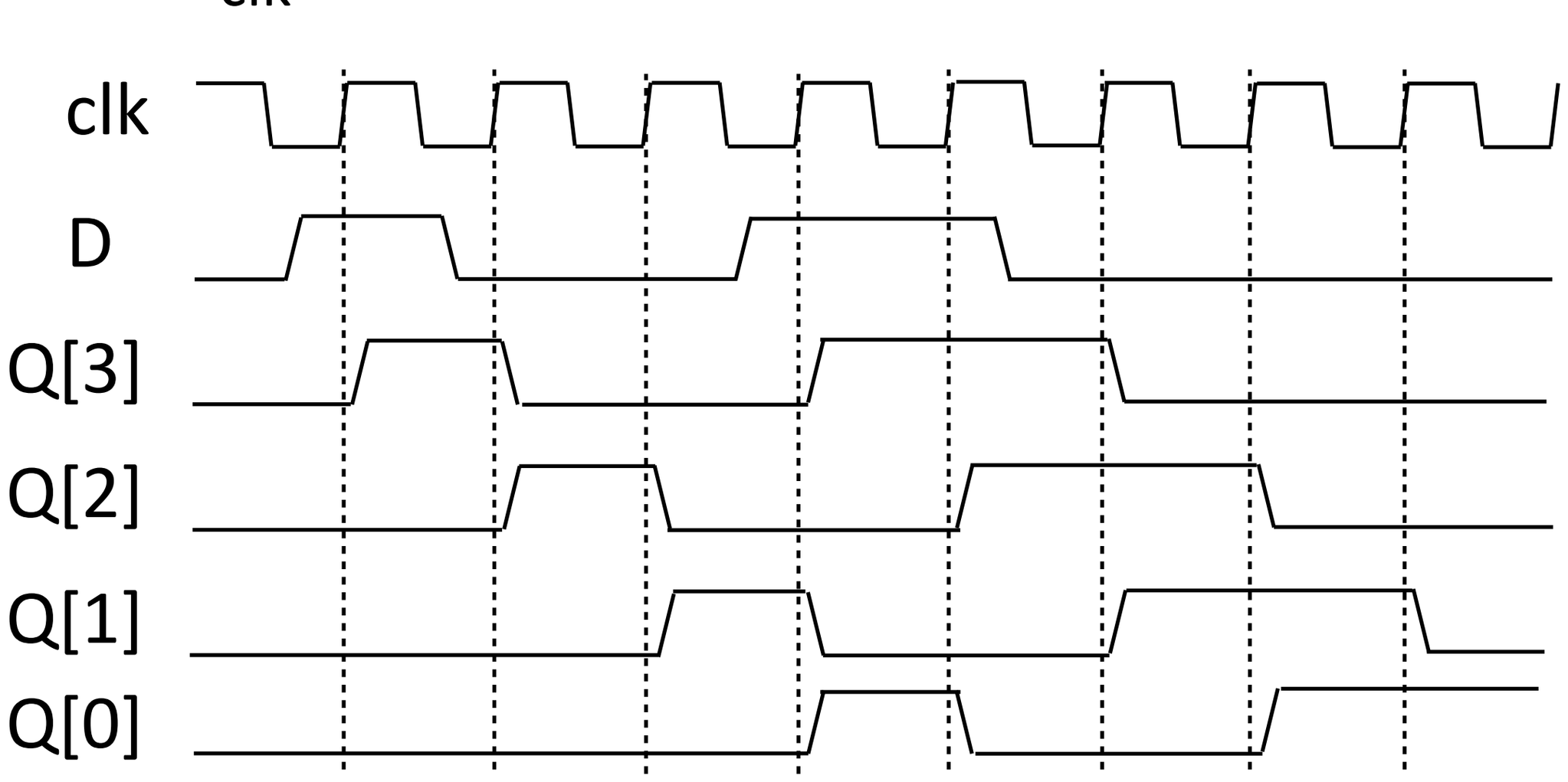




Other Components: Shift Register

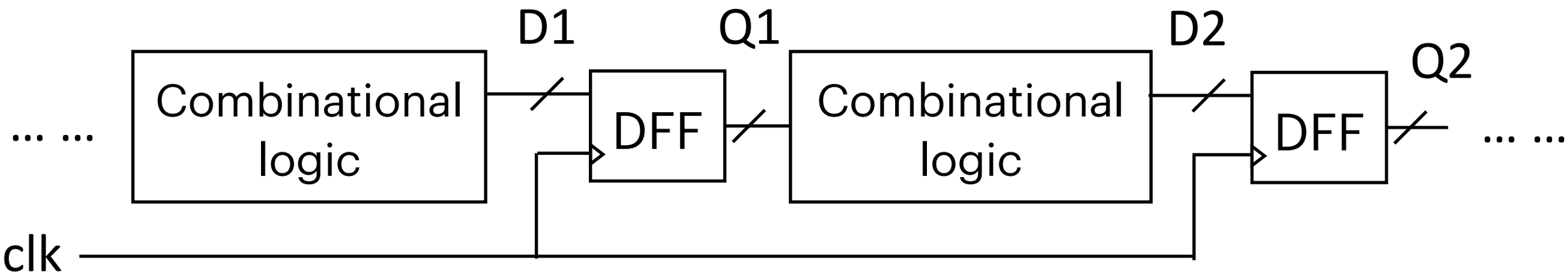
- Cascaded DFFs





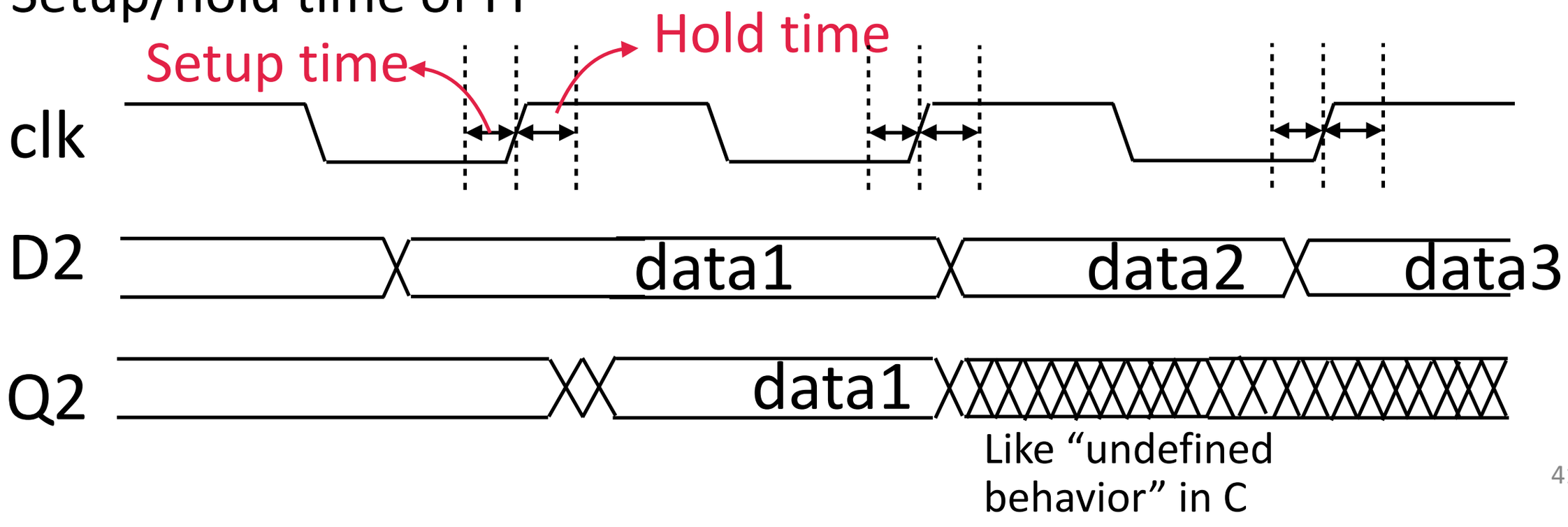
Timing Issues

- Why clk frequency cannot goes to infinity?



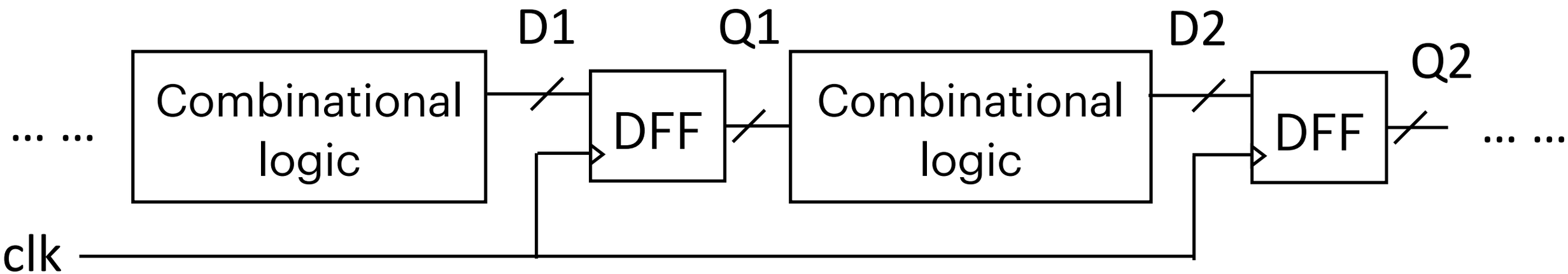
Typical digital system: a mix of combinational & sequential circuits

1. Setup/hold time of FF



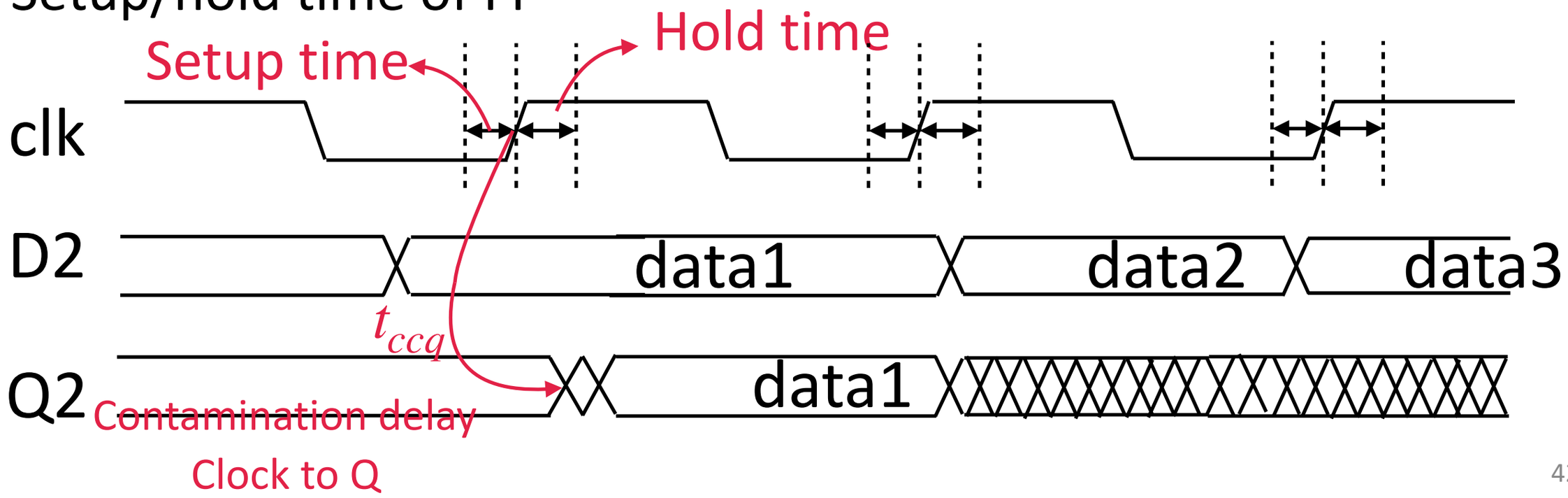
Timing Issues

- Why clk frequency cannot goes to infinity?



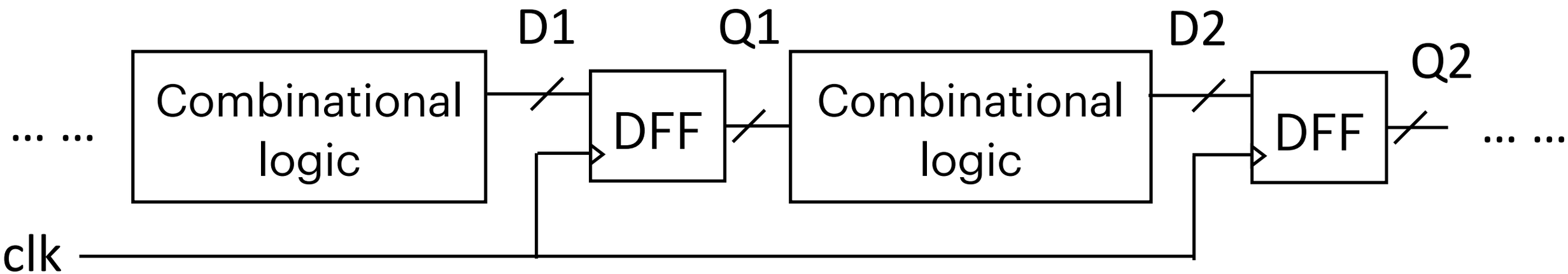
Typical digital system: a mix of combinational & sequential circuits

1. Setup/hold time of FF



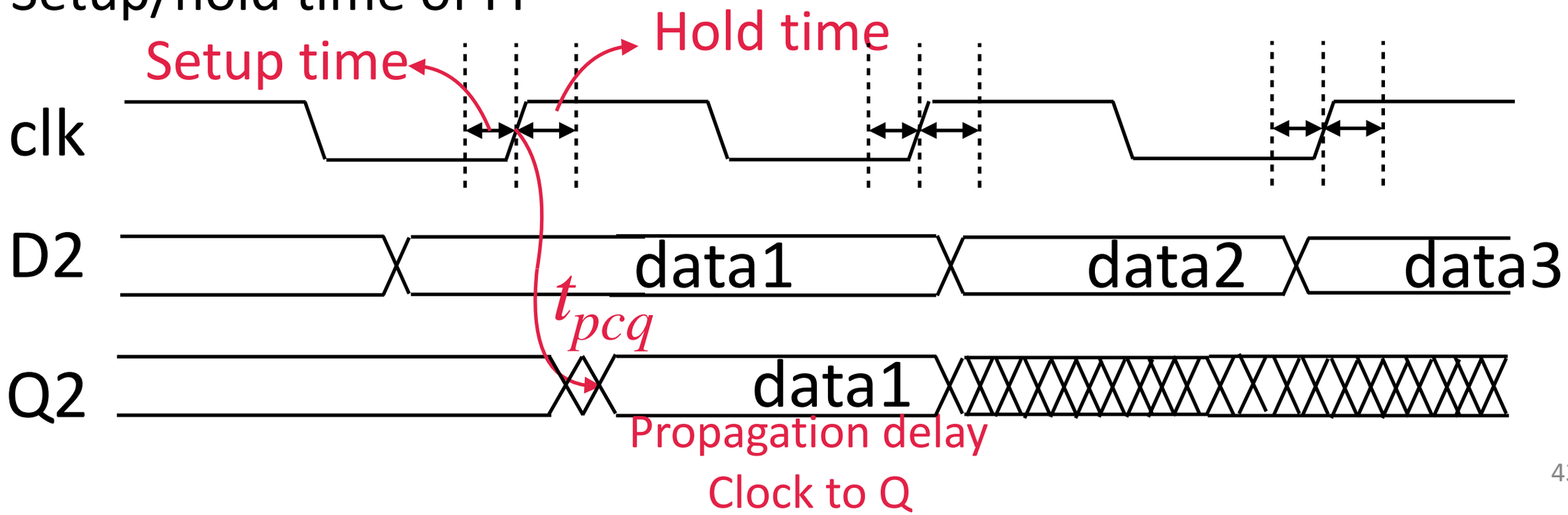
Timing Issues

- Why clk frequency cannot goes to infinity?



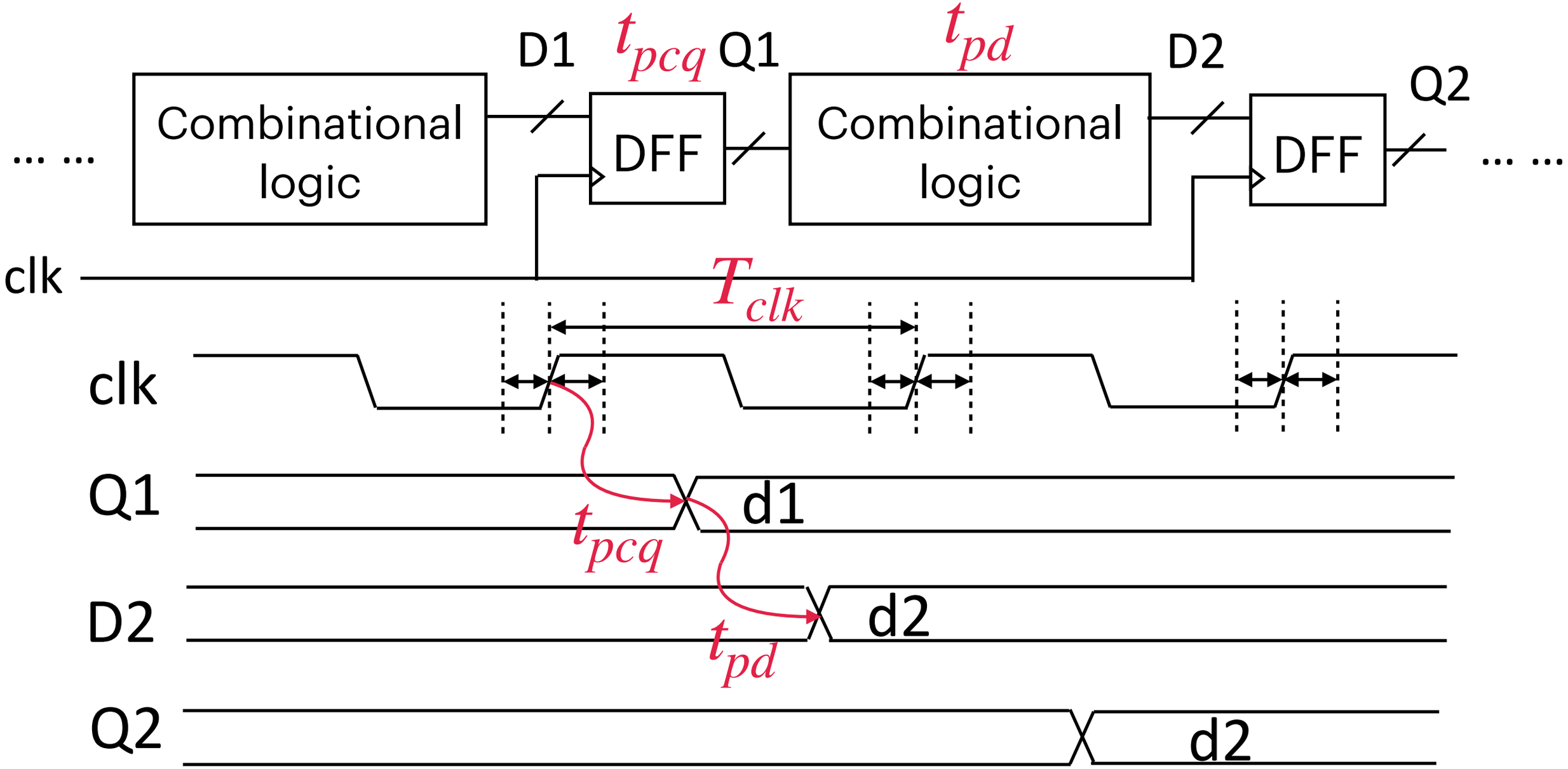
Typical digital system: a mix of combinational & sequential circuits

1. Setup/hold time of FF



Max-Delay Constraints

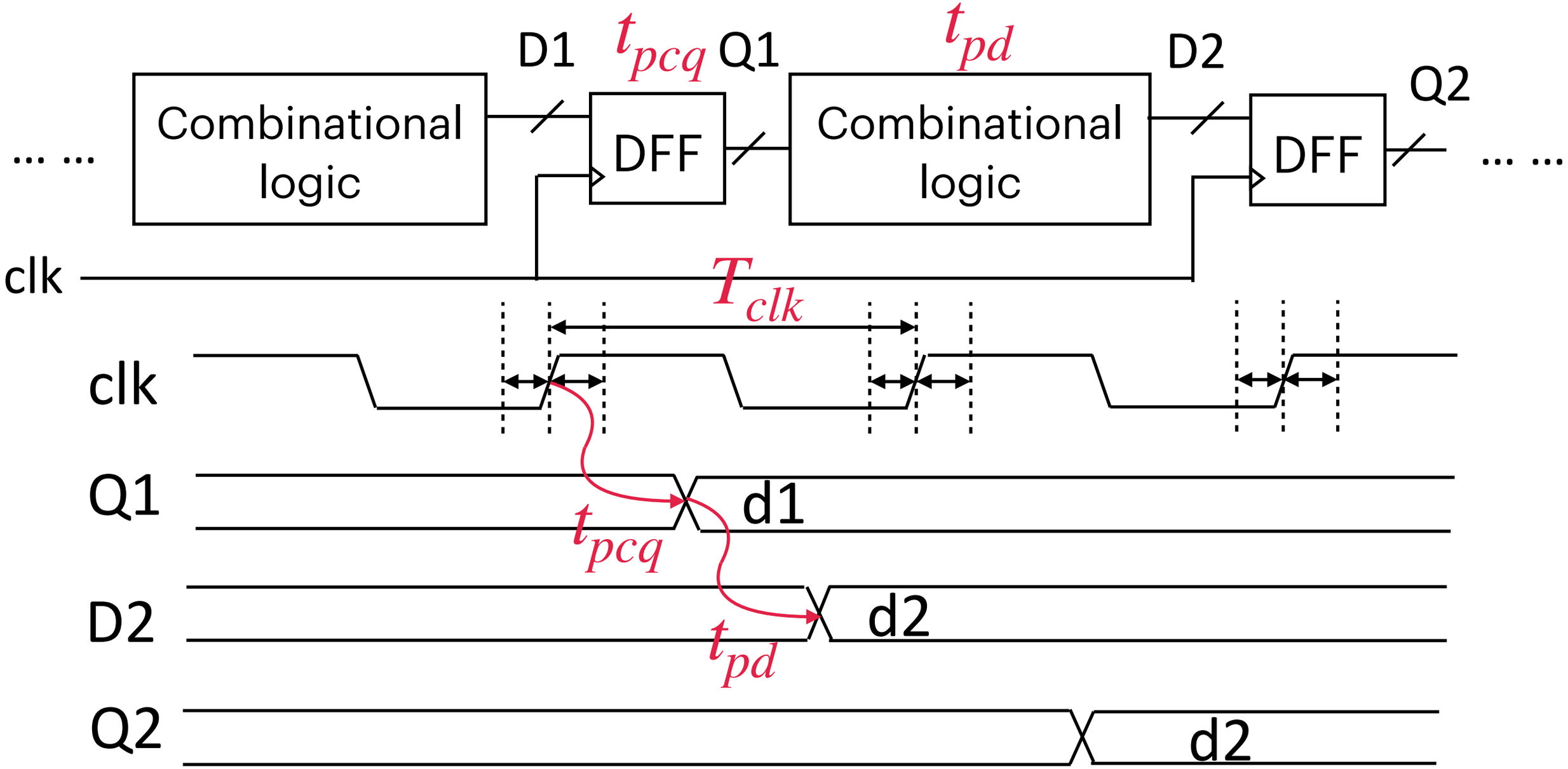
- Why clk frequency cannot goes to infinity?



$$T_{clk} \geq t_{pcq} + t_{pd} + \textbf{setup time}$$

Max-Delay Constraints

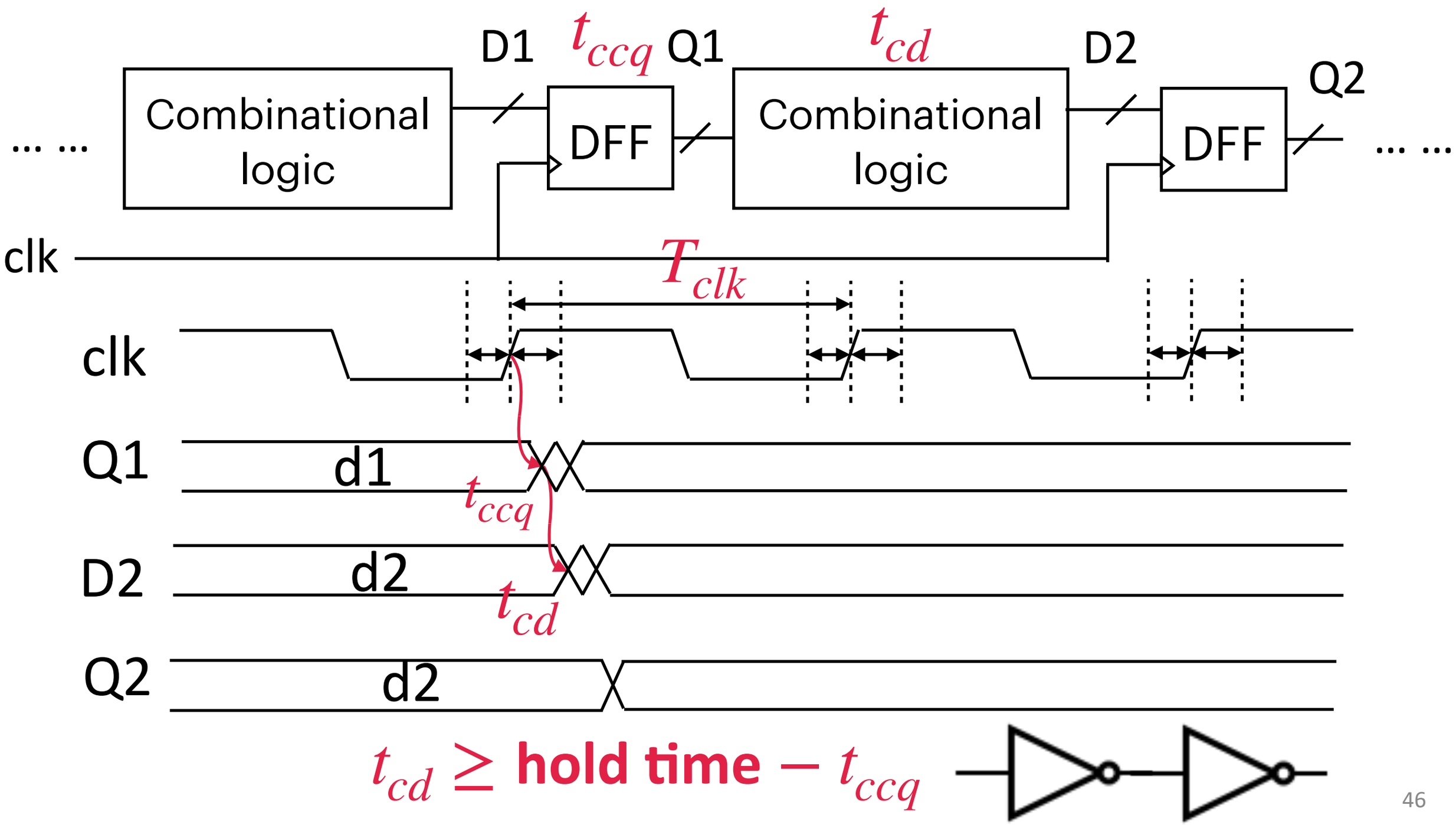
- Why clk frequency cannot goes to infinity?



$T_{clk} \geq t_{pcq} + t_{pd} +$ **setup time** *Critical path*

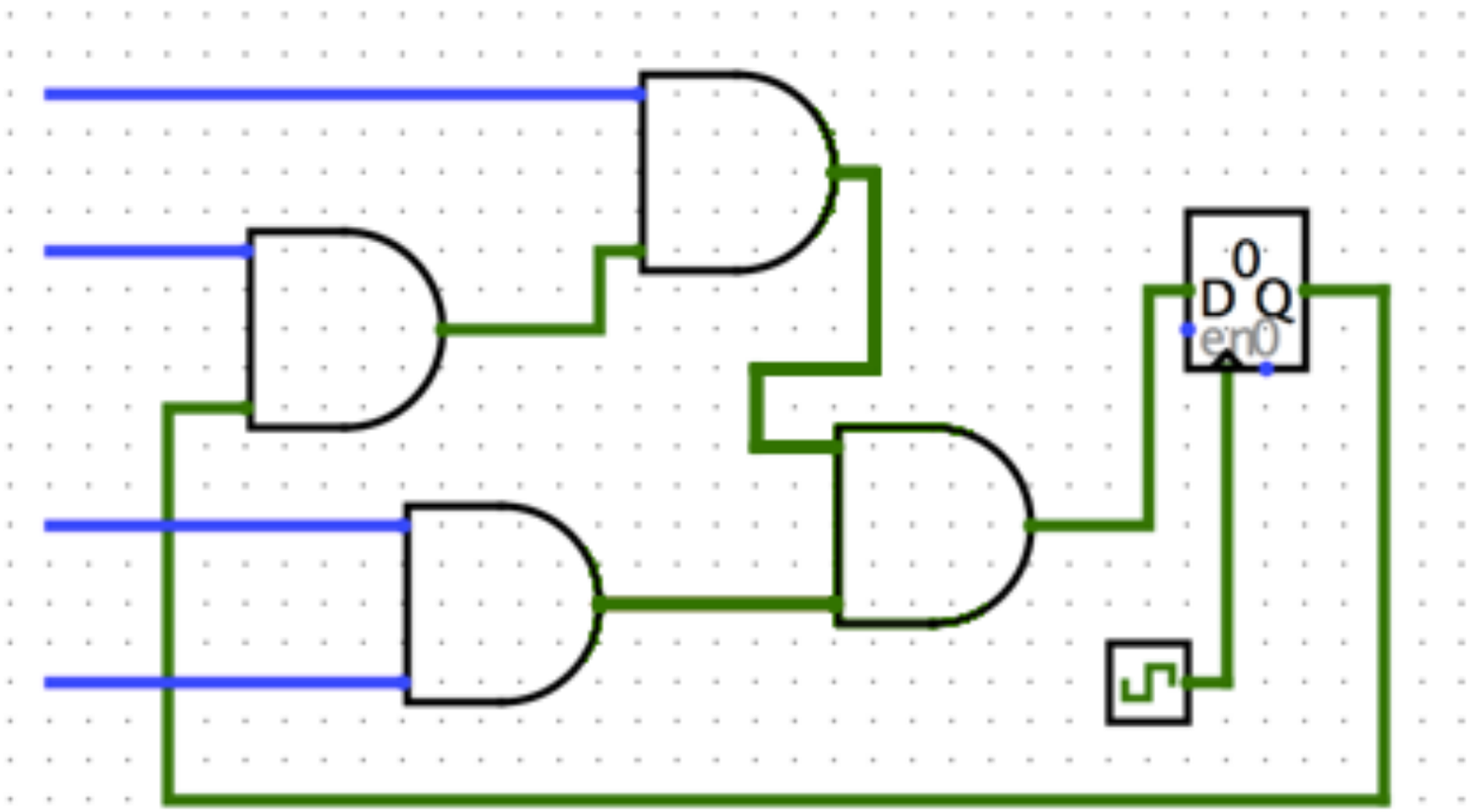
Min-Delay Constraints

- Avoid hold time violation



$t_{cd} \geq$ **hold time** $- t_{ccq}$

Question



Clock->Q (P) 1ns
Setup 1ns
Hold 1ns
AND delay 1ns

What is maximum clock frequency?

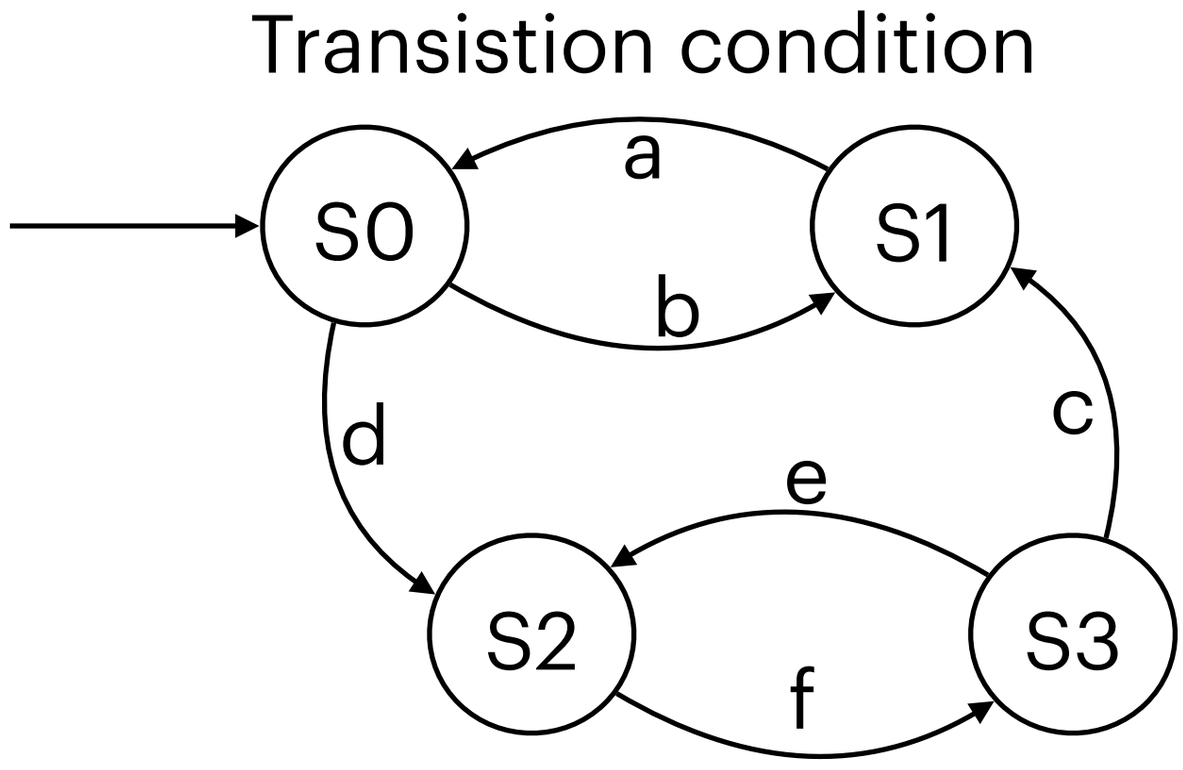
- A: 5 GHz
- B: 500 MHz
- C: 200 MHz
- D: 250 MHz
- E: 1/6 GHz

Finite-State Machine (FSM)

“Give me the place to stand, and I shall move the earth.”—Archimedes

FSM

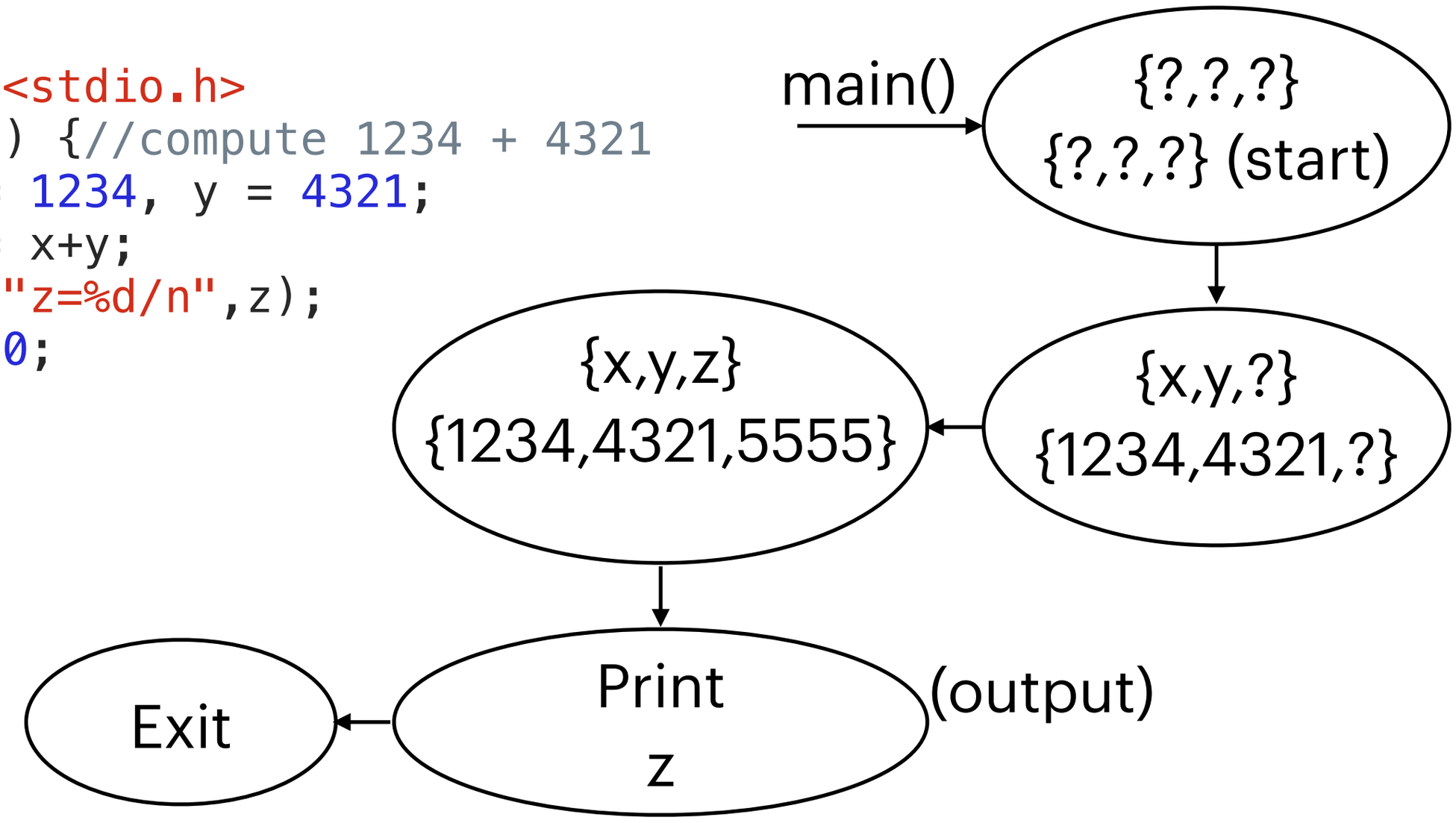
- FSMs consists of states, transitions, an entrance (initial state) and input/output (optional)



C Program as an FSM

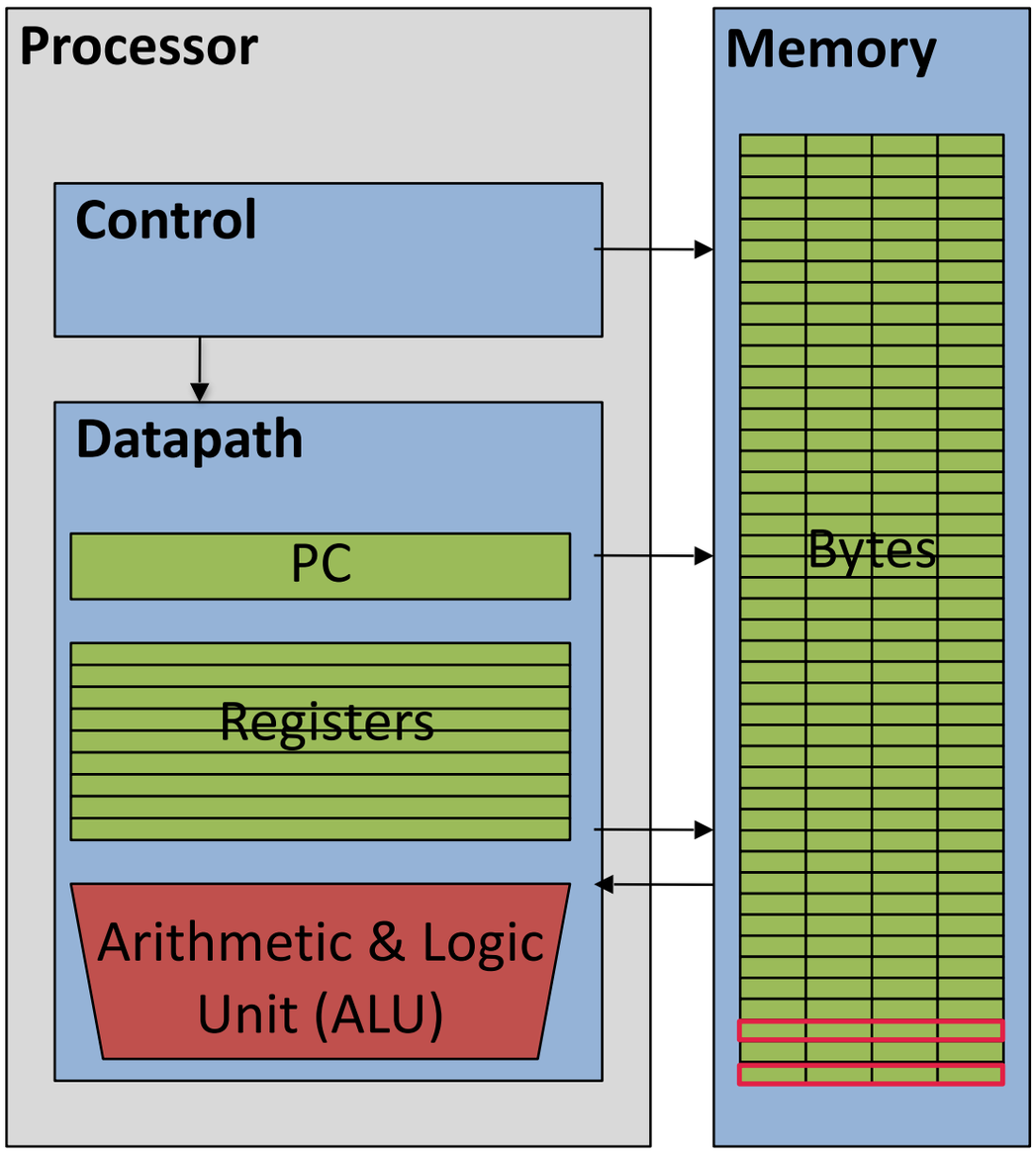
- FSMs consists of states, transitions, an entrance (initial state) and input/output (optional)

```c
#include <stdio.h>
int main() {//compute 1234 + 4321
   int x = 1234, y = 4321;
   int z = x+y;
   printf("z=%d/n",z);
   return 0;
}
```



ISA as an FSM

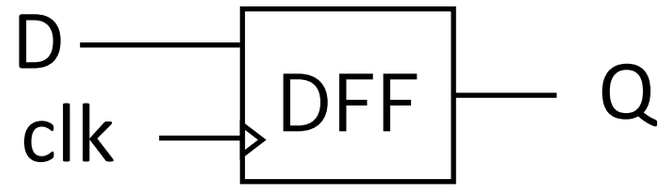
- FSMs consists of states, transitions, an entrance (initial state) and input/output (optional)

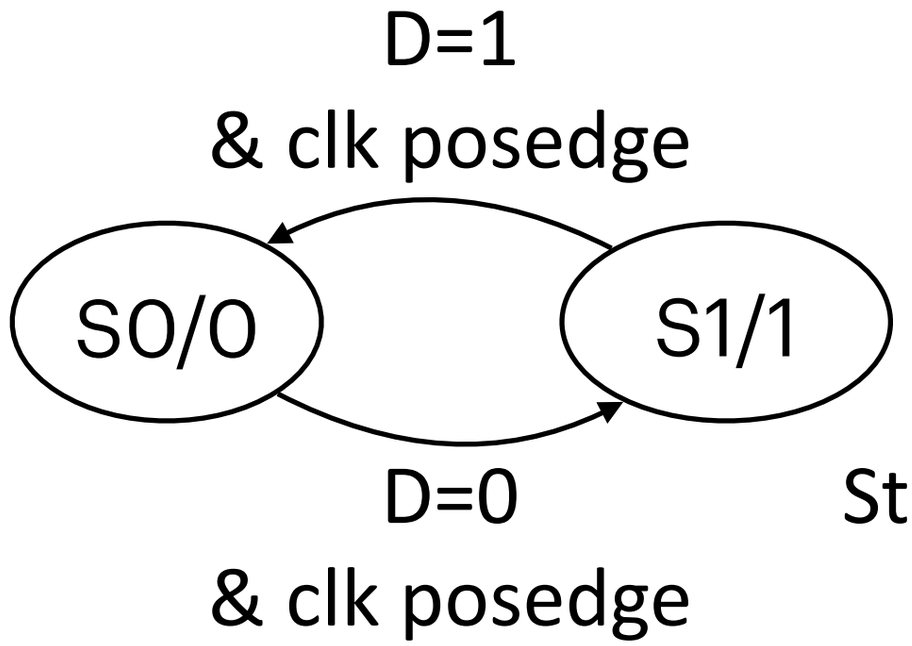


- States: all registers & memory
- Transition: register/memory value change
- Entrance: power on
- Input: instructions, can change registers/memory value
- Output: states of each registers/ memory

Digital Systems as FSMs

- FSMs consists of states, transitions, an entrance (initial state) and input/output (optional)

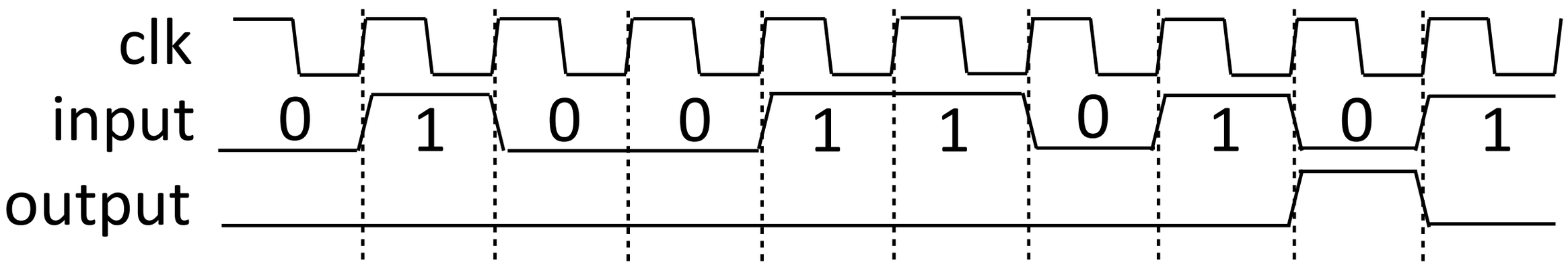




State transition diagram

A Classic Problem: Build Digital Circuit for Sequence Detection

- Build a digital circuit, detecting the occurrence of {101} in the input 0/1 sequence (non-overlapping)



input: 0 or 1 in a sequence, one bit at a clock cycle

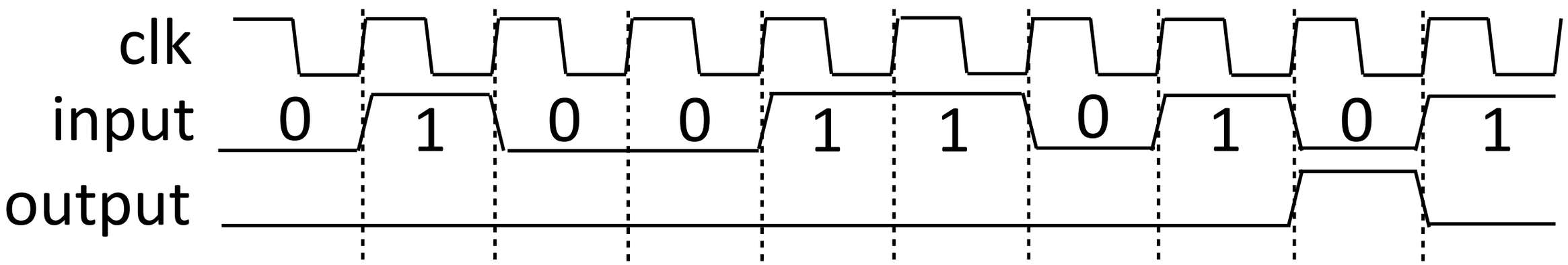
Output: 1 after {101} detected, otherwise 0;

States: ?

Transition: ?

A Classic Problem: Build Digital Circuit for Sequence Detection

- Build a digital circuit, detecting the occurrence of {101} in the input 0/1 sequence (non-overlapping)



input: 0 or 1 in a sequence, one bit at a clock cycle

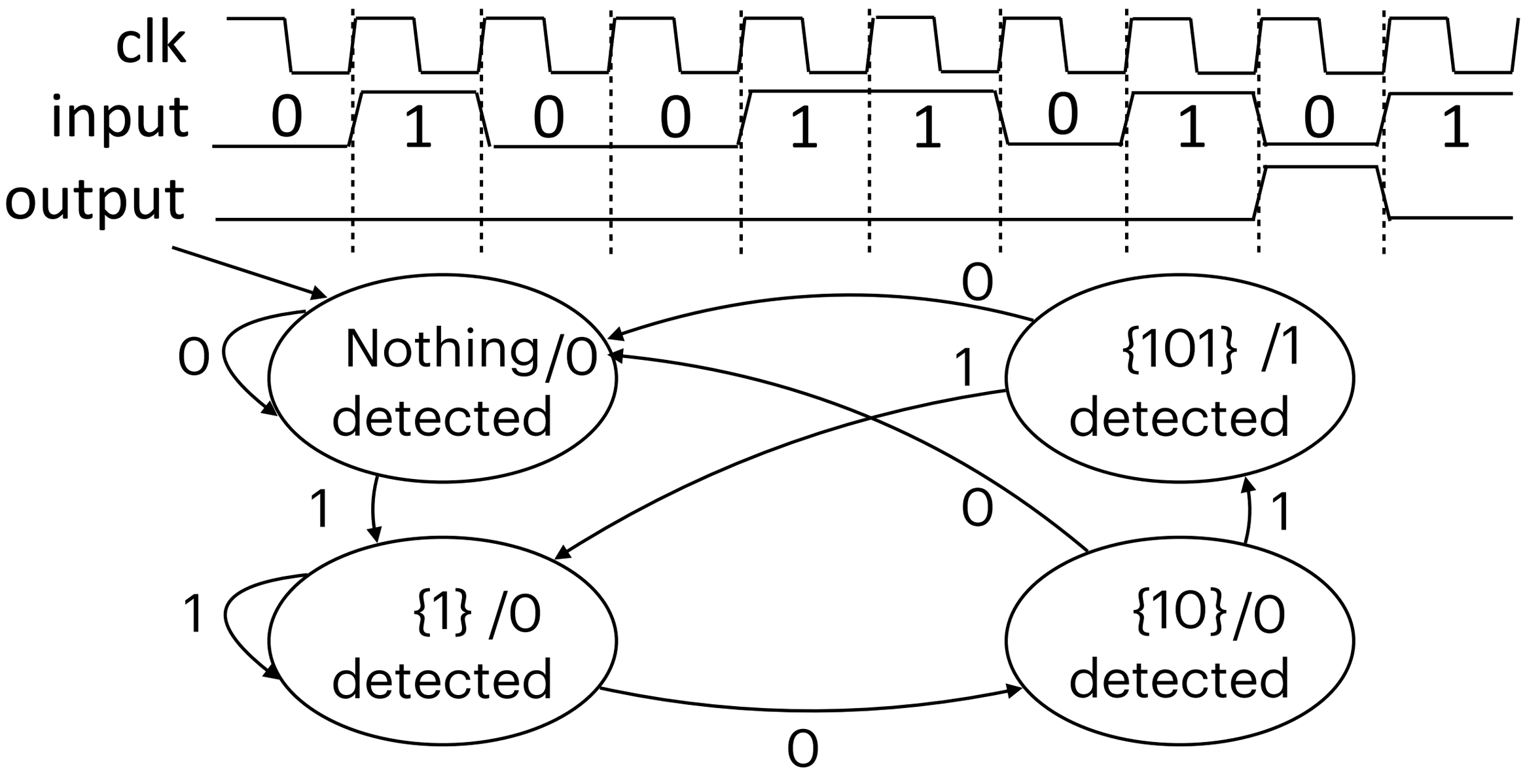
Output: 1 after {101} detected, otherwise 0;

States:

Transition:

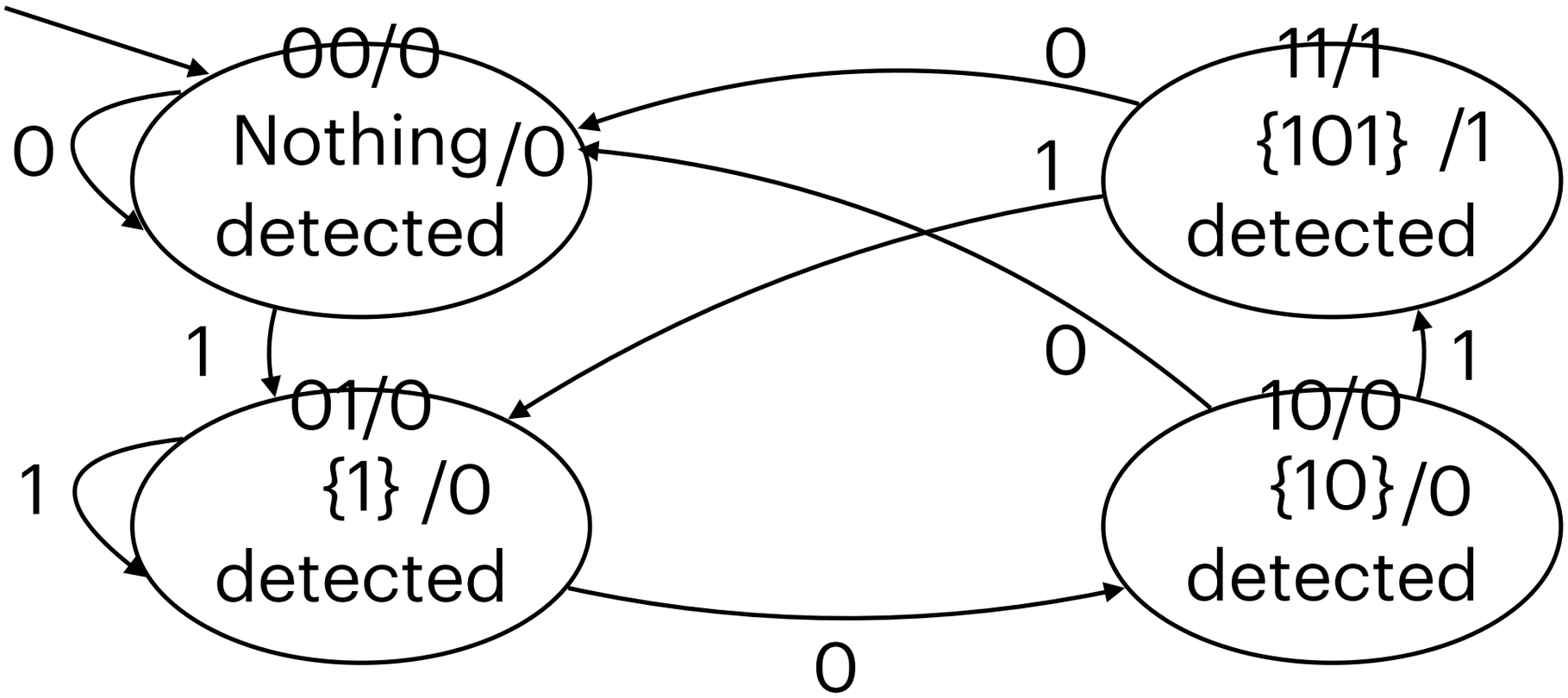
A Classic Problem: Build Digital Circuit for Sequence Detection

- Build a digital circuit, detecting the occurrence of {101} in the input 0/1 sequence (non-overlapping)



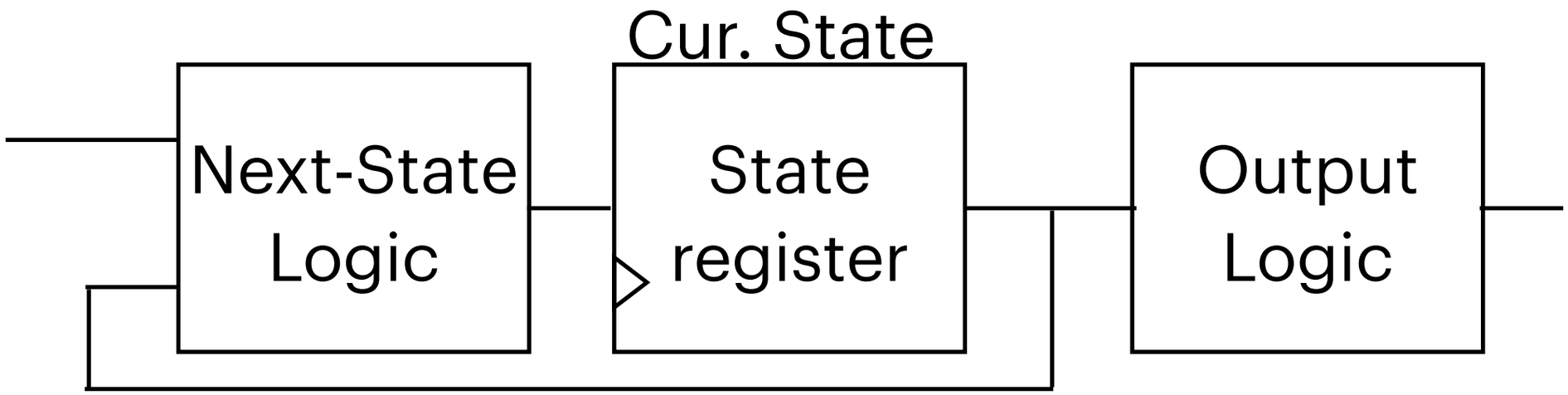
A Classic Problem: Build Digital Circuit for Sequence Detection

- Everything is a number. Use binary numbers to encode states: {00, 01, 10, 11}



A Classic Problem: Build Digital Circuit for Sequence Detection

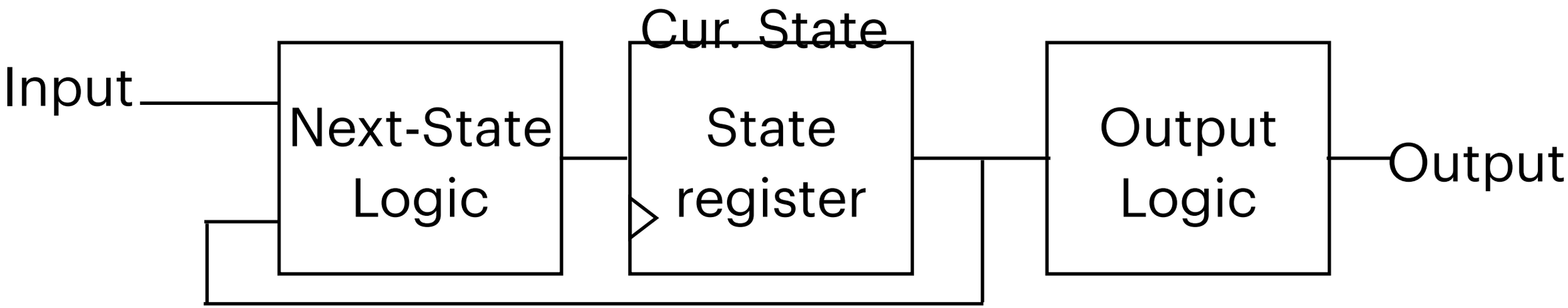
- A digital circuit model (TaoLu) for FSM (Moore machine)



- Timing diagram

A Classic Problem: Build Digital Circuit for Sequence Detection

- A digital circuit model for FSM (Moore machine)

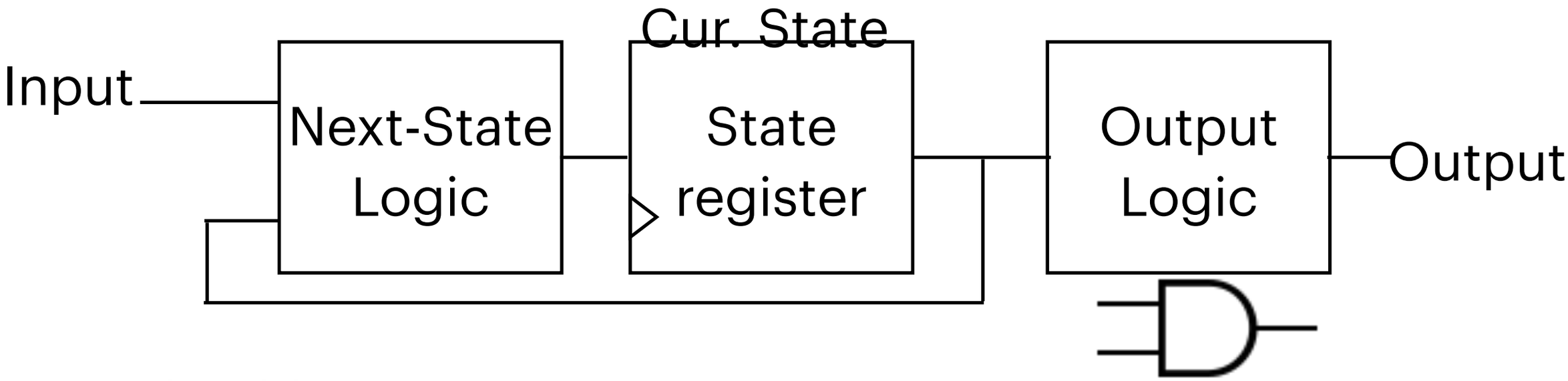


- Truth table

Cur.State	Input	Next State	Output
00	0	00	0
00	1	01	0
01	0	10	0
01	1	01	0
10	0	00	0
10	1	11	0
11	0	00	1
11	1	01	1

A Classic Problem: Build Digital Circuit for Sequence Detection

- A digital circuit model for FSM (Moore machine)



- Truth table

Cur.State	Input	Next State	Output
00	0	00	0
00	1	01	0
01	0	10	0
01	1	01	0
10	0	00	0
10	1	11	0
11	0	00	1
11	1	01	1