



## การจัดองค์การคอมพิวเตอร์

# Sequential Logic

31110321 Computer Organization

สำหรับนักศึกษาชั้นปีที่ 3 สาขาวิชาวิศวกรรมคอมพิวเตอร์

กรุงฤทธิ์ กิตติครีวรพันธุ์

[songrit@npu.ac.th](mailto:songrit@npu.ac.th)

สาขาวิชาวิศวกรรมคอมพิวเตอร์  
มหาวิทยาลัยบูรพา

# Outline

---

- 3.1 วงจรเชิงลำดับ (Sequential Circuit)
- 3.2 พลิปฟล็อป (Flip-flops)
- 3.3 អនុយគម្រោះ
- 3.4 วงຈនាប
- 3.5 ការងារ 3

# วงจรไม่ขึ้นกับเวลา (combinational)

---

- วงจรที่ทำงานเกี่ยวข้องกับ Time
- ที่ผ่านมาไม่ได้กล่าวถึงความสัมพันธ์ของการทำงานกับเวลา
  - $f(t) = x(t)+y(t)$
- ที่ผ่านมา output ขึ้นอยู่กับ input
  - input เปลี่ยน ทำให้ output ตอบสนองกันที
  - ลำดับการเปลี่ยนแปลง input ไม่ส่งผลต่อการทำงาน
    - (เริ่มอินพุตไหนก็เหมือนกัน)
- รูปแบบที่ผ่านมาเรียกว่า
  - time-independent
  - combinational logic

# Combinational vs Sequential

---

- วงศ์ Combinational

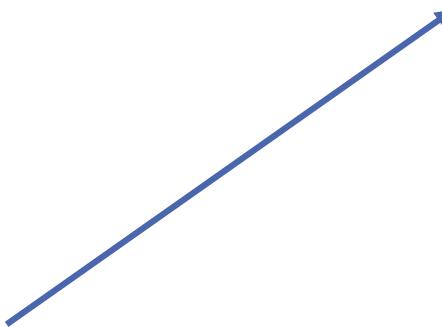
```
For i = 1.. 100  
    a[i] = b[i] + c[i]
```

- วงศ์สามารถจดจำสถานะตัวเอง (state)
  - Memory, Counter

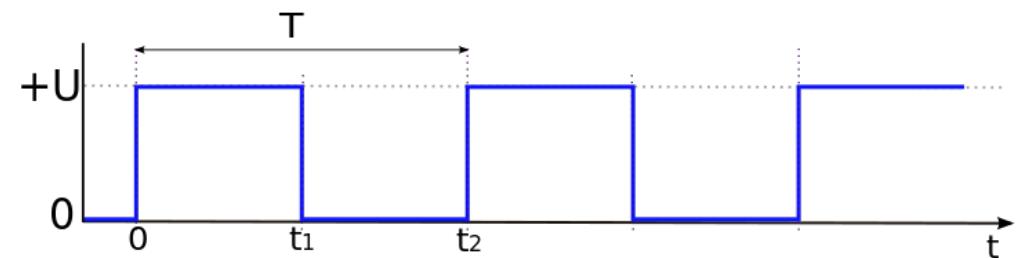
```
For i = 1.. 100  
    sum[i] = sum+ i
```

# เวลา (times)

- ธรรมชาติของ เวลา มีการเพิ่มค่าโดยไม่ย้อนกลับ
- การเพิ่มของเวลาเป็นแบบต่อเนื่อง (Continuous)
  - เป็น real number
- ระบบดิจิทัล ทำให้ times กว้างขึ้นด้วยการใช้ สัญญาณ นาฬิกา



เวลา

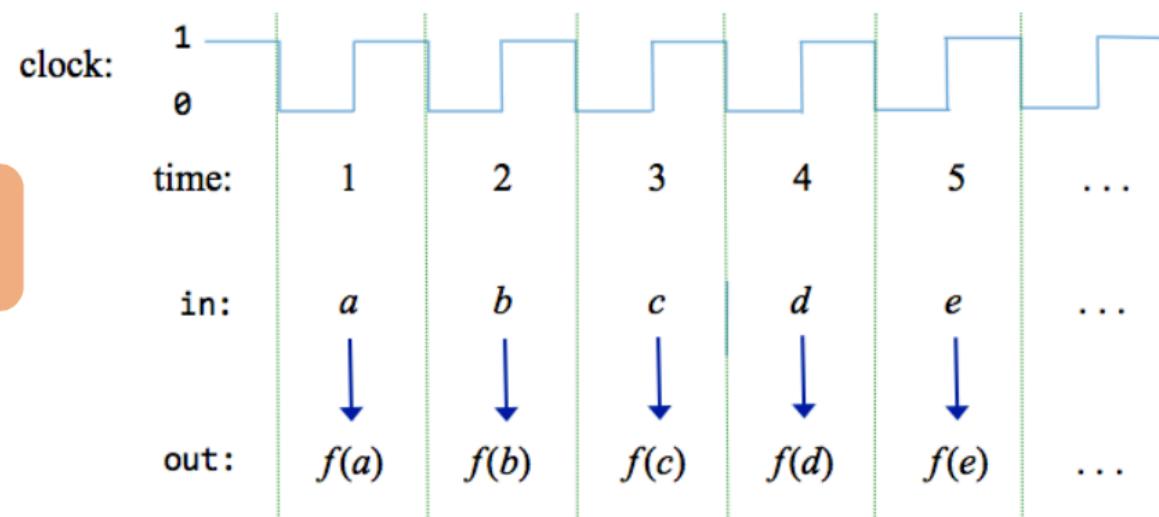


$F(t) = \{0, 1\}$   
สัญญาณนาฬิกา

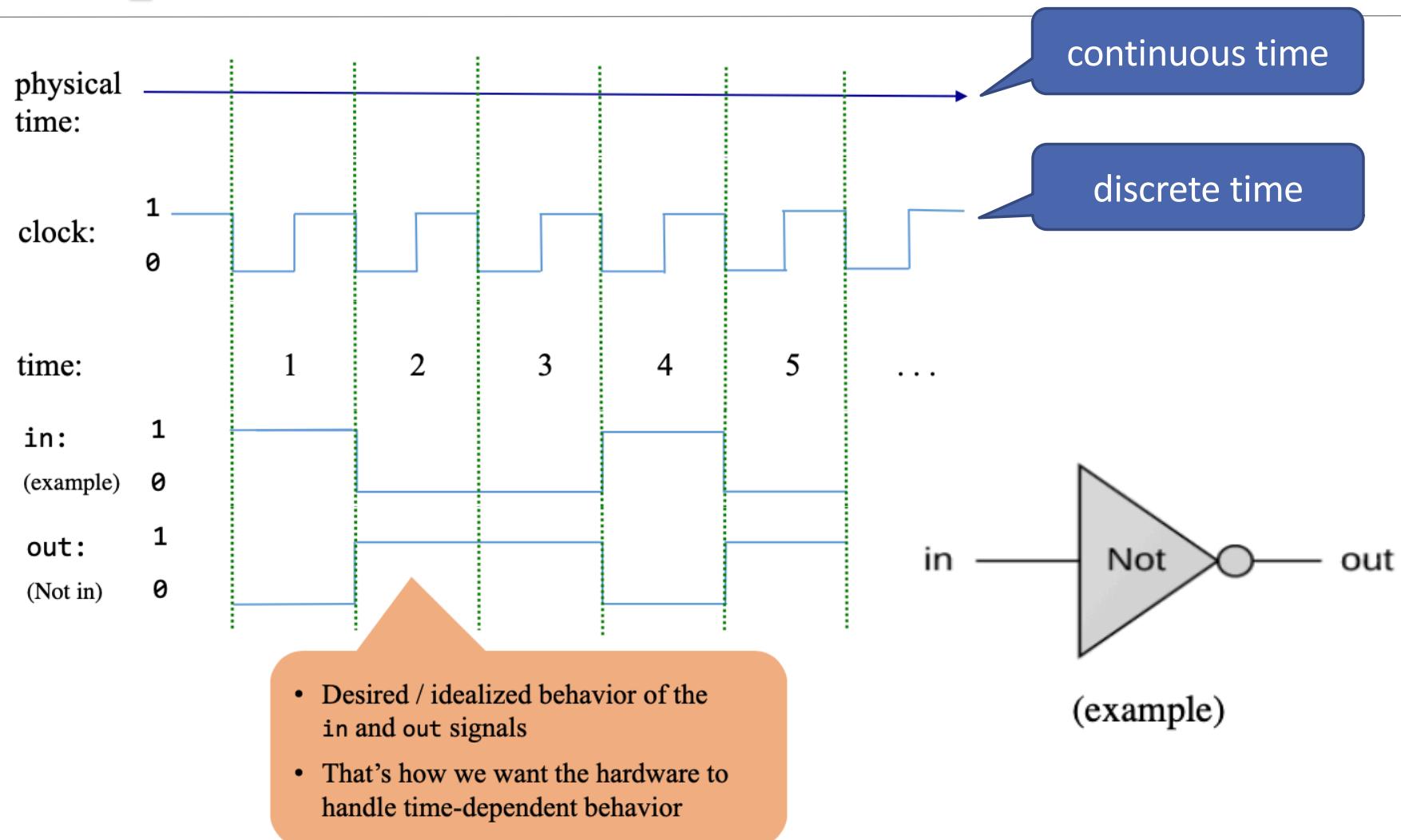
# ເວລາ ແລະ ສັນຍາລາංພາ

## Combinational logic:

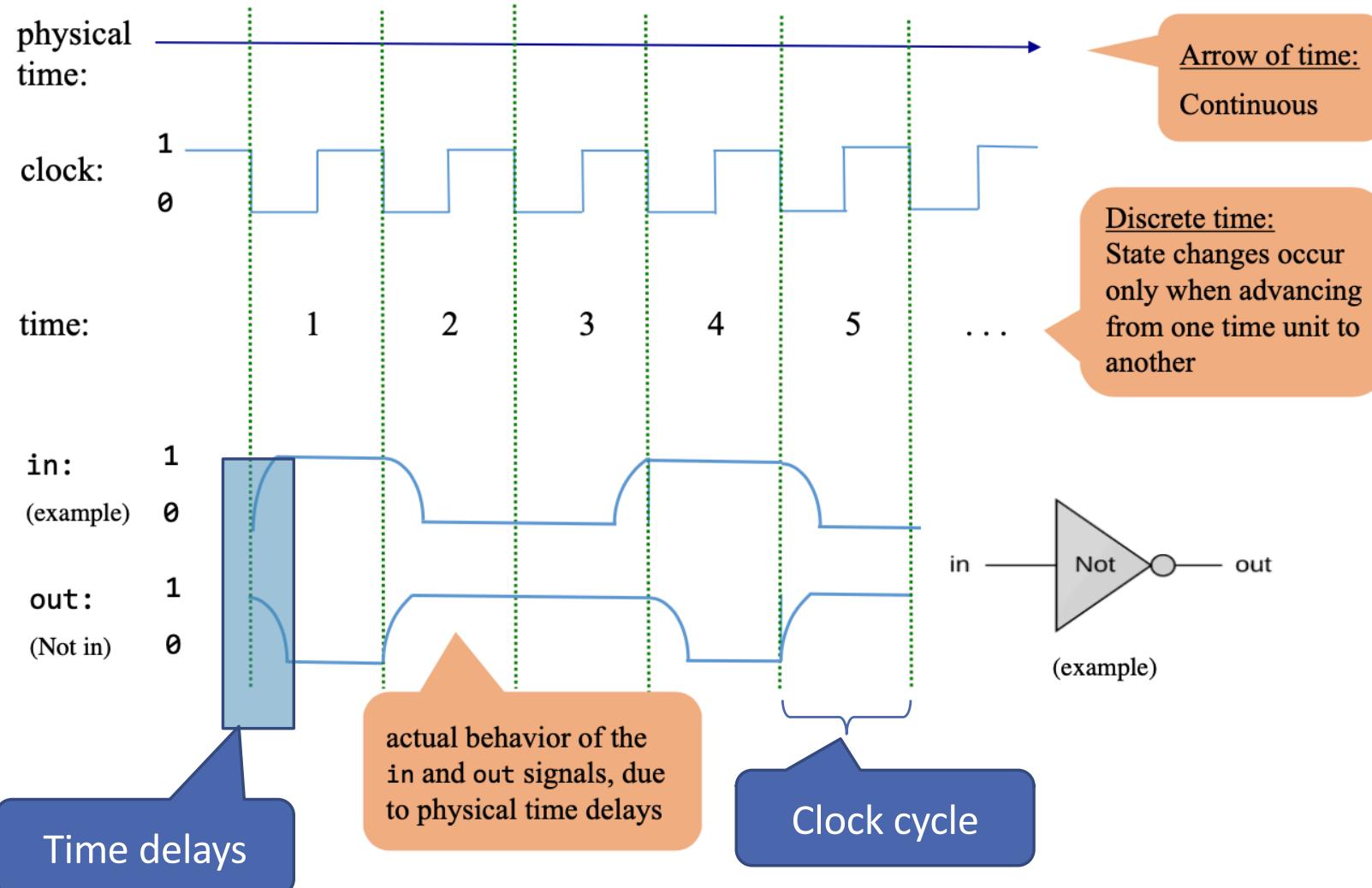
The output is a pure function  
of the present input only



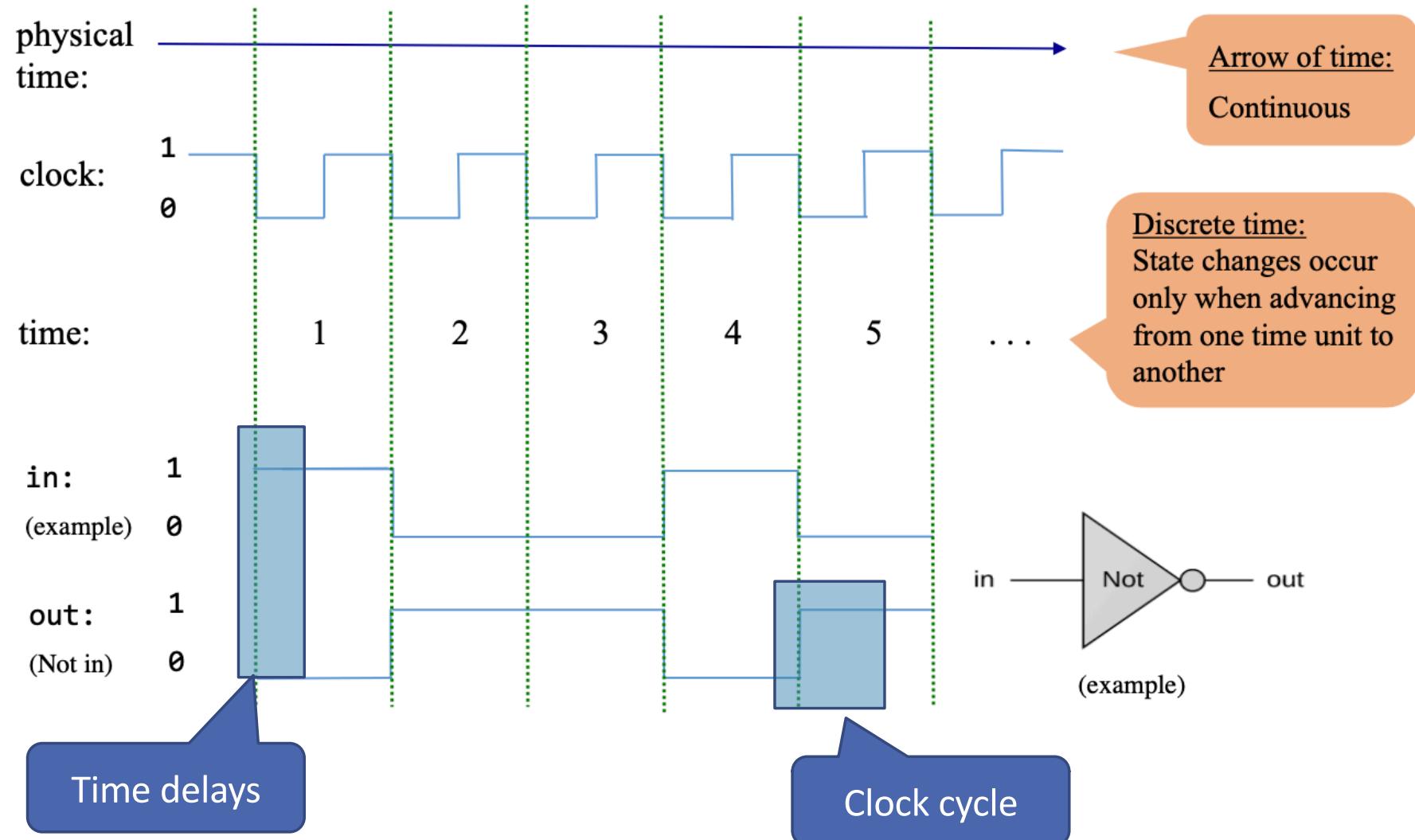
# chip เมื่อเวลาเปลี่ยน



# ความจริงของสัญญาณ



# การใช้ 0, 1 กำหนดที่ยังตรงขึ้น



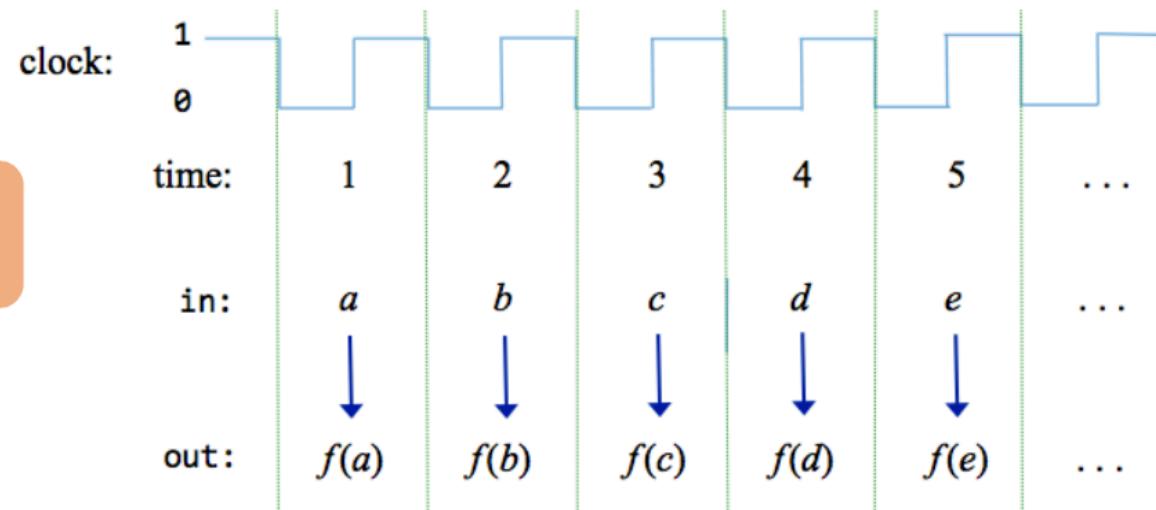
# ວັງຈາຣອົງກ ກັບ ວັງຈາຣເຊີງລຳດັບ

- Combinatorial vs Sequential Logic

- Combinatorial :  $out[t] = \text{function}(in[t])$
- Sequential :  $out[t] = \text{function}(in[t-1])$

Combinational logic:

The output is a pure function  
of the present input only

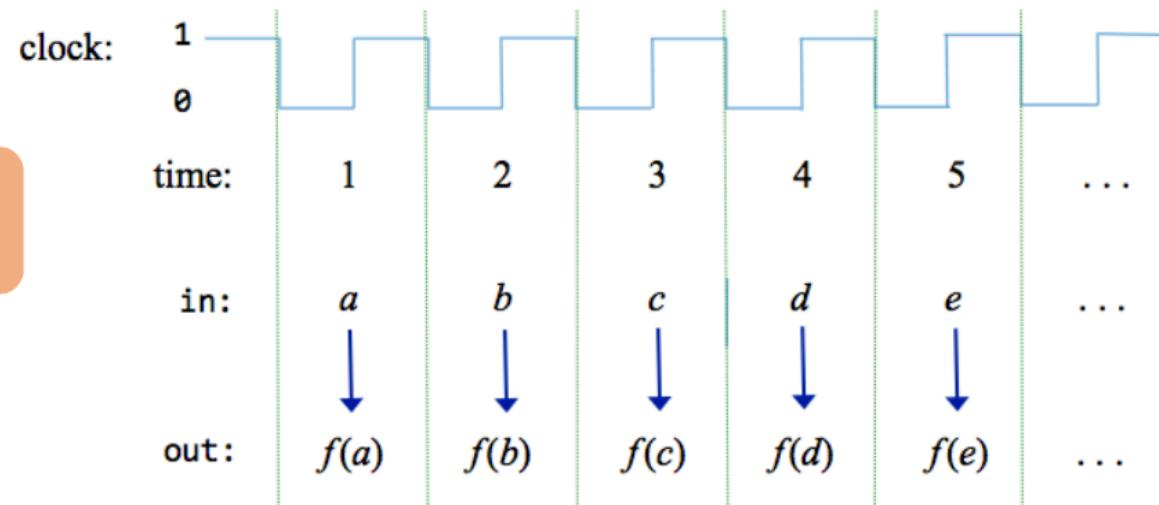


# Combinational logic

- Combinatorial :  $\text{out}[t] = \text{function}(\text{in}[t])$

Combinational logic:

The output is a pure function  
of the present input only



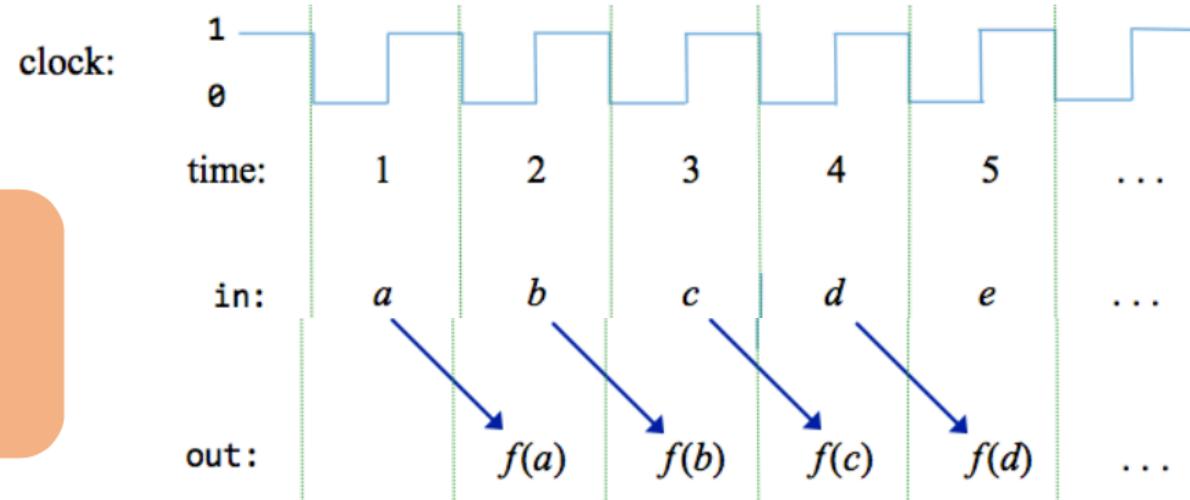
# Sequential logic

- Sequential :  $\text{out}[t] = \text{function}(\text{in}[t-1])$

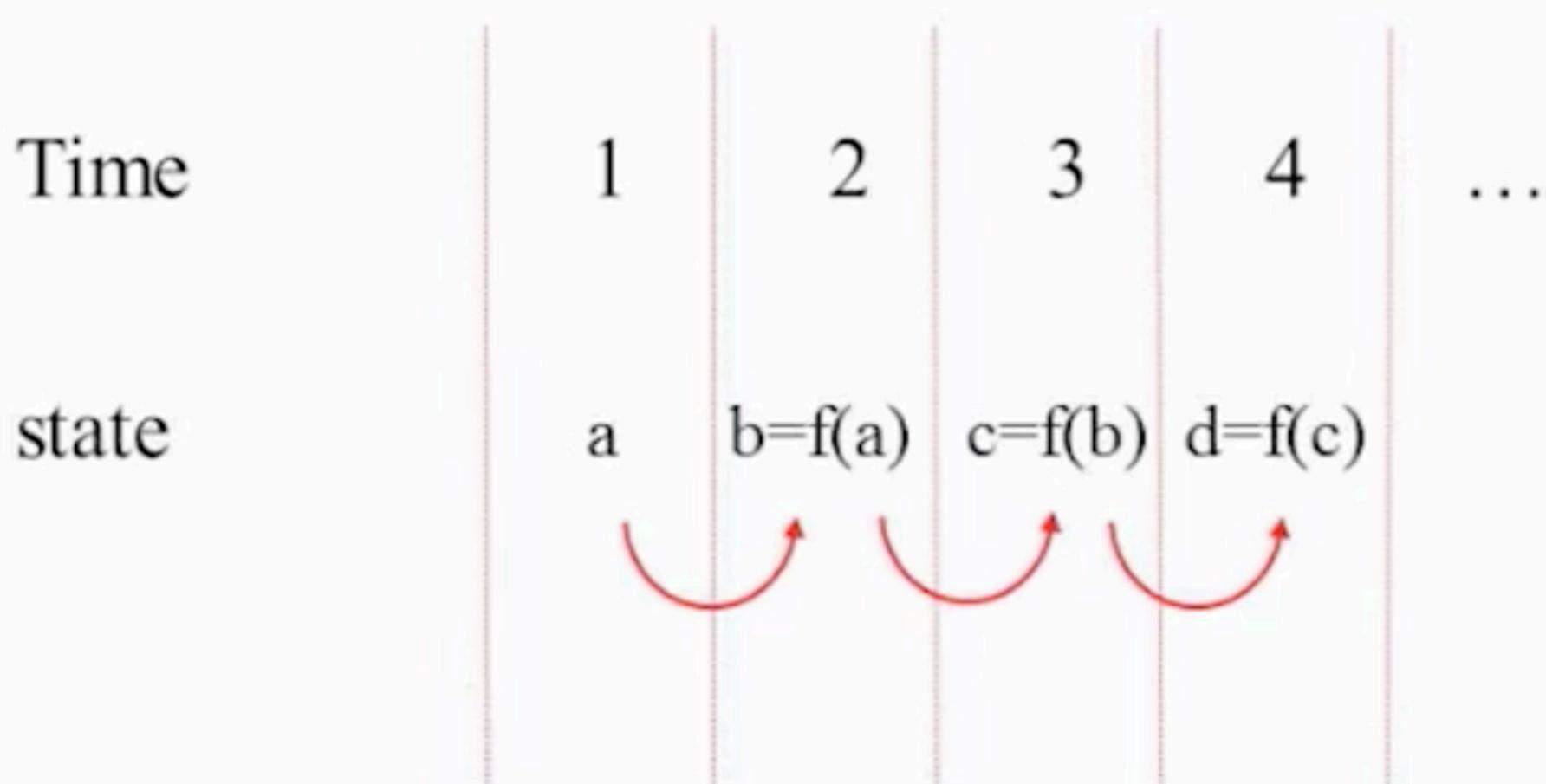
## Sequential logic:

The output depends on:

- the present input (optionally)
- the history of the input  
(creates a memory effect).

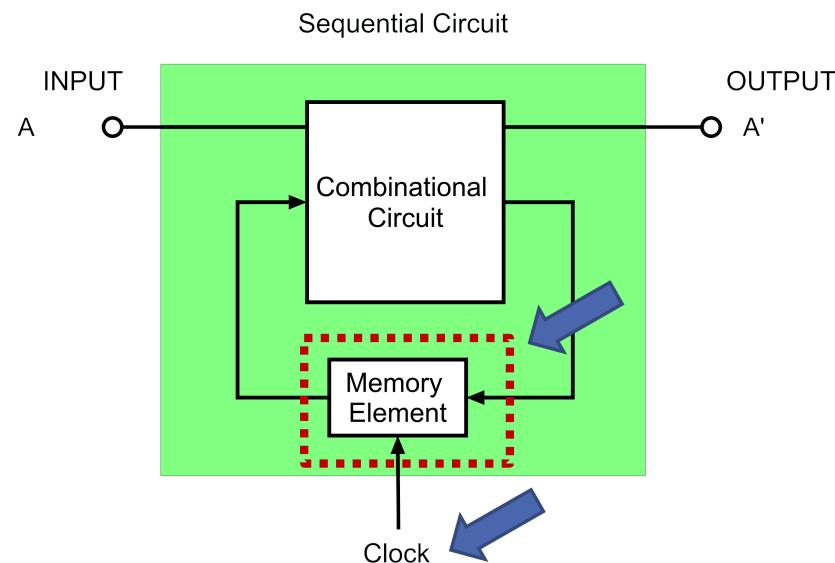


# Sequential Logic



# Sequential Circuit

- วงจร Combinational คือวงจรที่ใช้สถานะปัจจุบันร่วม
  - ผลลัพธ์เปลี่ยนตาม **INPUT** และ **สถานะ** ในเวลาใด ๆ
  - INPUT เดิมแต่ให้ OUTPUT เปลี่ยน เมื่อสถานะไม่คงเดิม
  - สถานะ เรียกว่า **State**
- มีอุปกรณ์จำสถานะ → หน่วยความจำ (Memory)

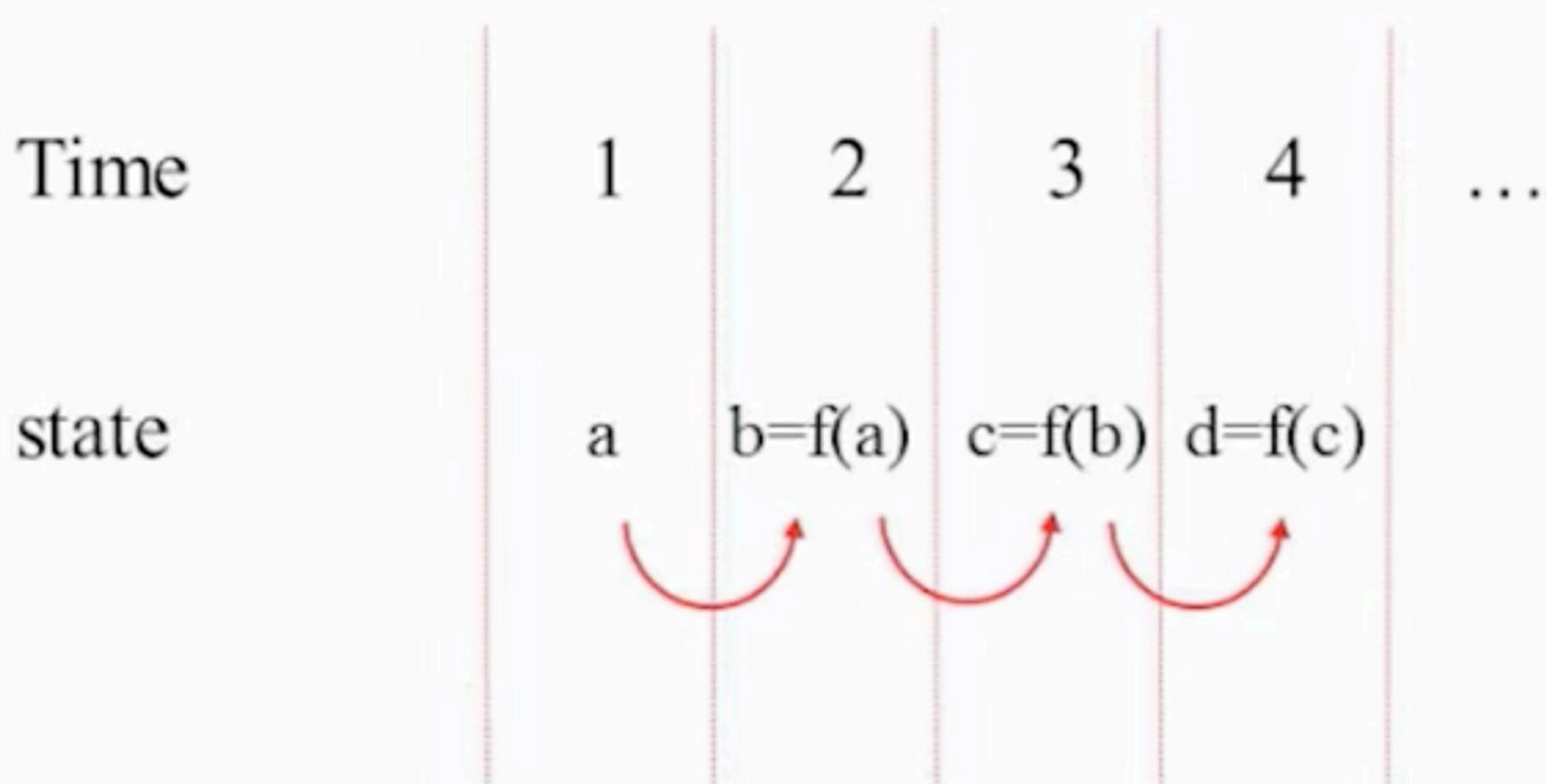


# Outline

---

- 3.1 วงจรเชิงลำดับ (Sequential Circuit)
- **3.2 พลิปฟล็อป (Flip-flops)**
- 3.3 หน่วยความจำ
- 3.4 วงจรนับ
- 3.5 โปรเจ็ค 3

# Sequential Logic



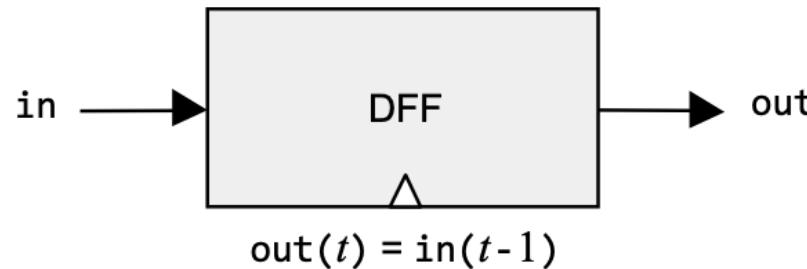
# จดจำเวลาทั้งพุตในอดีตได้

---

- มีตัวช่วยจดจำเวลาทั้งพุตที่เวลา  $t-1$
- เวลาทั้งพุตในช่วงเวลา  $t$  ใดๆ เรียกว่า state
- เมื่อเวลา  $t$  ผ่านไป 1 หน่วยเวลา เวลาหนึ่งจะกลายเป็น  $t-1$
- เวลาปัจจุบันคือ  $t$  เวลาในอดีตจึงเป็น  $t-1$
- Element พื้นที่สามารถจดจำสถานะ  $t-1$  ได้เรียกว่า
  - พลิปฟล็อป (flip-flop)
  - พลิปฟล็อป สลับอันพุกจาก
    - $0 \rightarrow 1$  ที่เวลา  $t$
    - เปลี่ยนจาก  $1 \rightarrow 0$  ที่  $t+1$

# Flip-Flop

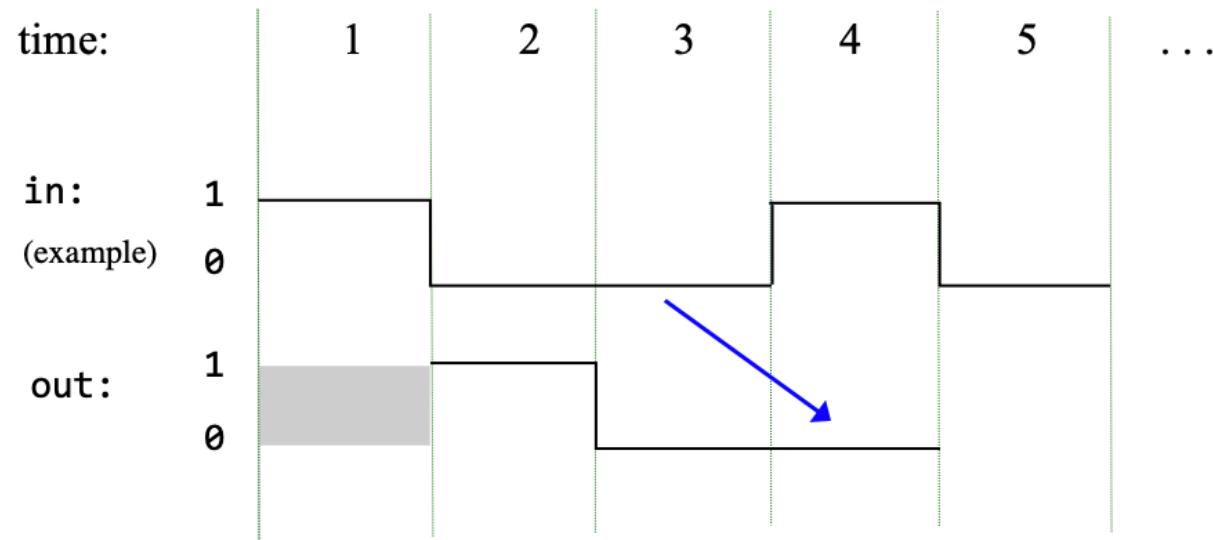
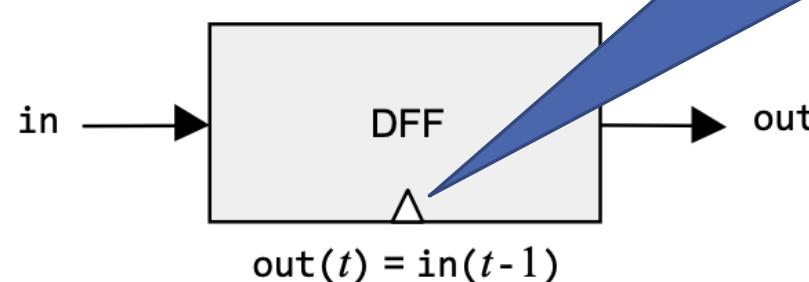
---



- สเปค
  - 1 อินพุต , 1 เอาท์พุต
- การส่งค่าออก  $out(t) = in(t-1)$
- ถ้าค่าเริ่มต้น  $in=1$ ,  $out=0$ 
  - เมื่อป้อน  $in=0$ , ทำให้  $out = 1$  // อดีต  $t-1 = 1$
  - เมื่อ  $in=0$  ทำให้  $out = 0$  // จาก  $t-1 = 0$

# Flip Flop

รูปสามเหลี่ยมใช้แทน gate  
• ต่อ clock  
• ใช้กำหนดให้ทุก gate  
ทำงานพร้อมกัน

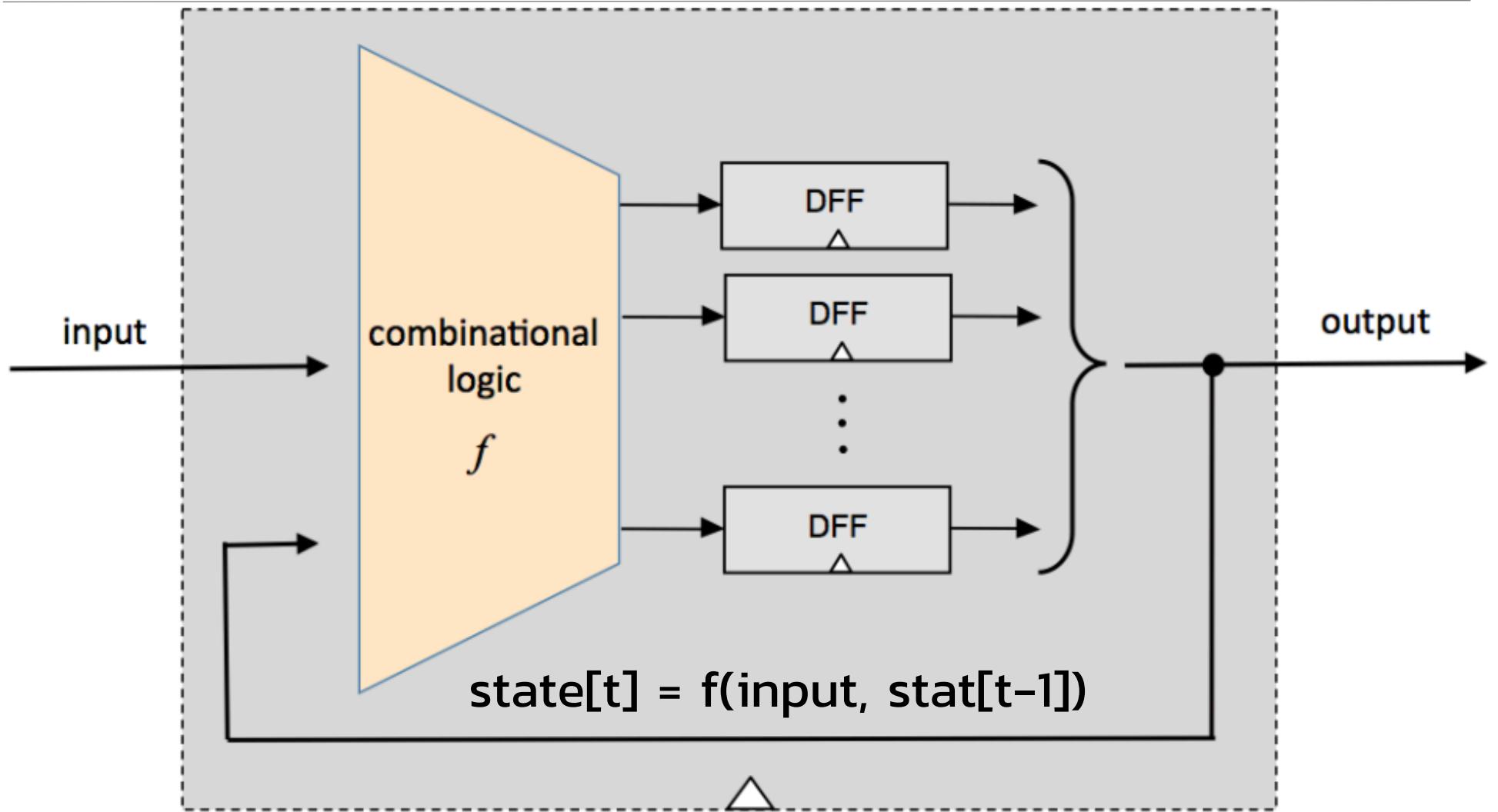


# DFF (Data Flip Flop)

---

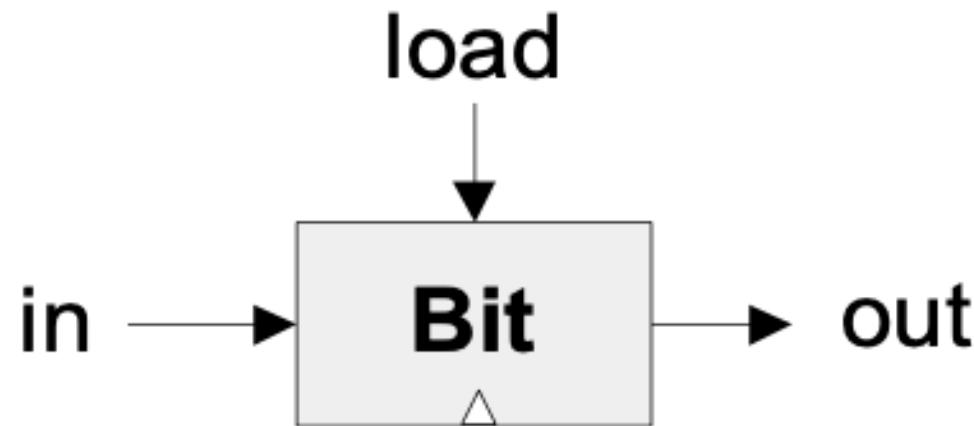
- DFF เป็นการทำงาน เพียง 2 สถานะ (bi-state)
- สามารถสร้างได้จาก Nand gate 2 เกต
  - Step 1: สร้าง input-output loop,
  - ได้ composite chip เป็น flip-flop

# Concept Implementation



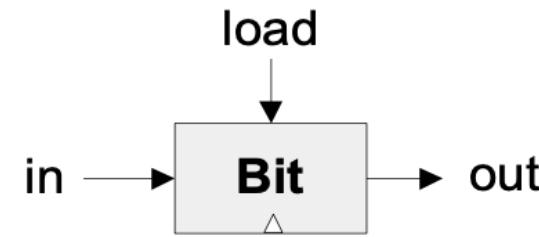
# การทำ chip ให้จำ bit ได้

---



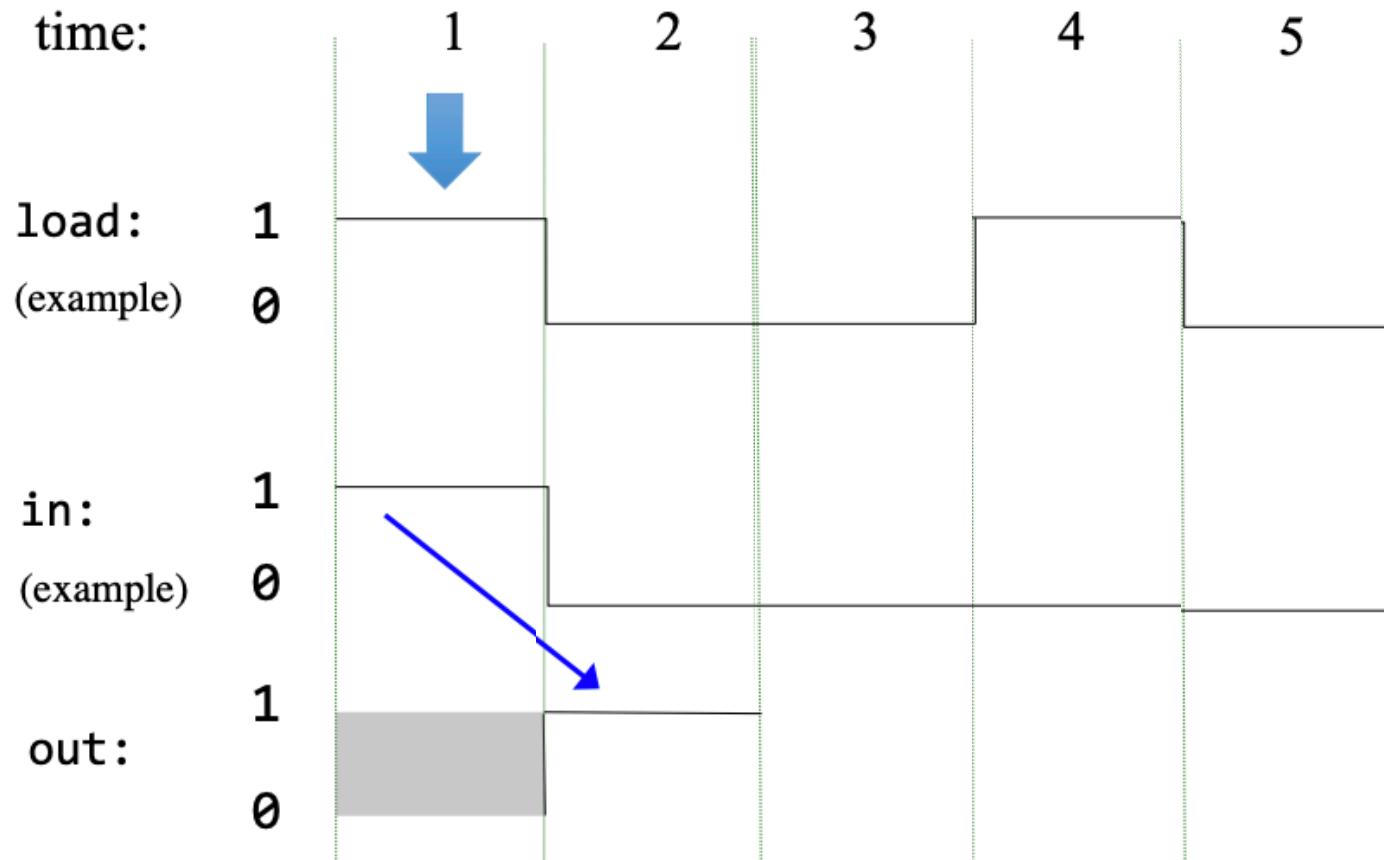
if  $\text{load}(t)$  then  $\text{out}(t+1) = \text{in}(t)$   
else  $\text{out}(t+1) = \text{out}(t)$

# ຈັດຈຳ 1 bit

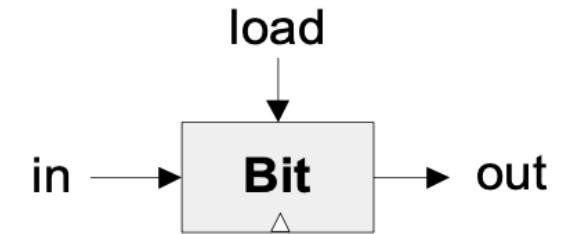


if  $\text{load}(t)$  then  $\text{out}(t+1) = \text{in}(t)$   
else  $\text{out}(t+1) = \text{out}(t)$

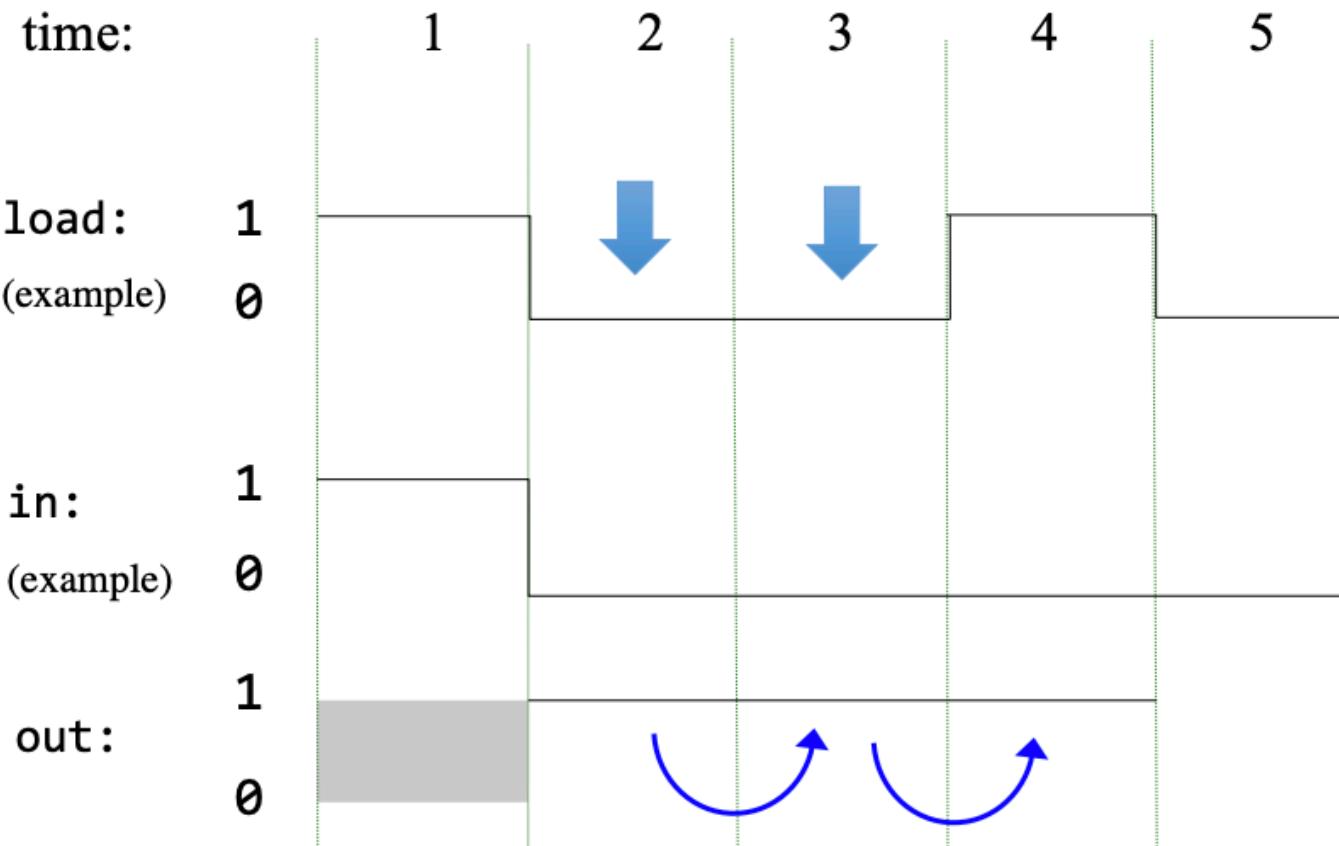
time:



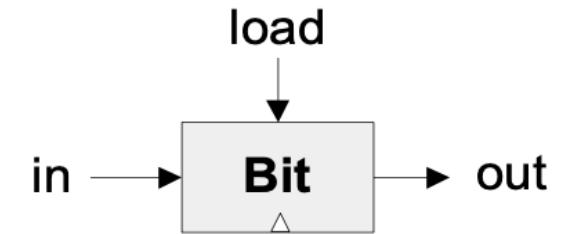
# 1-bit register



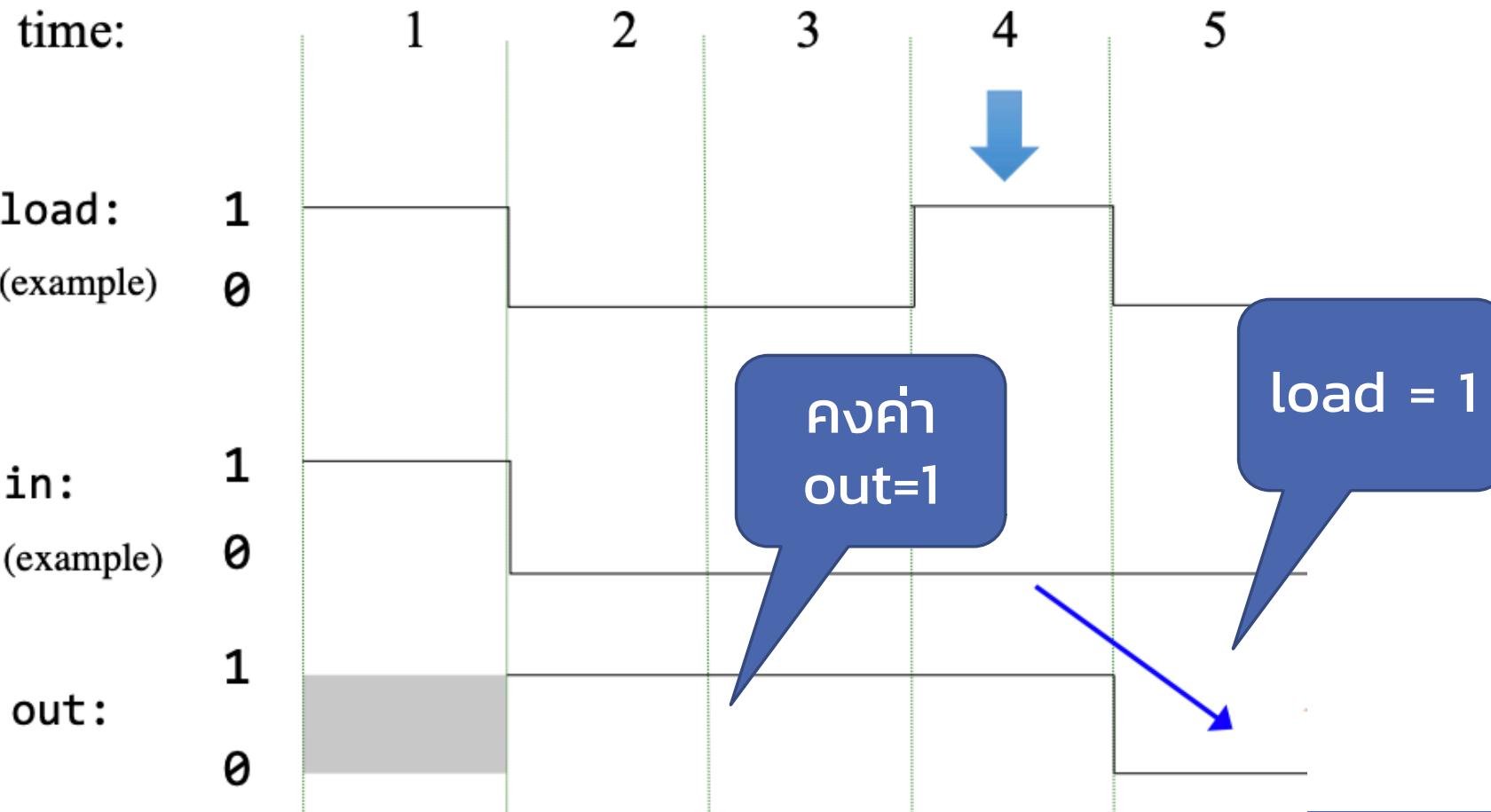
if  $\text{load}(t)$  then  $\text{out}(t+1) = \text{in}(t)$   
else  $\text{out}(t+1) = \text{out}(t)$

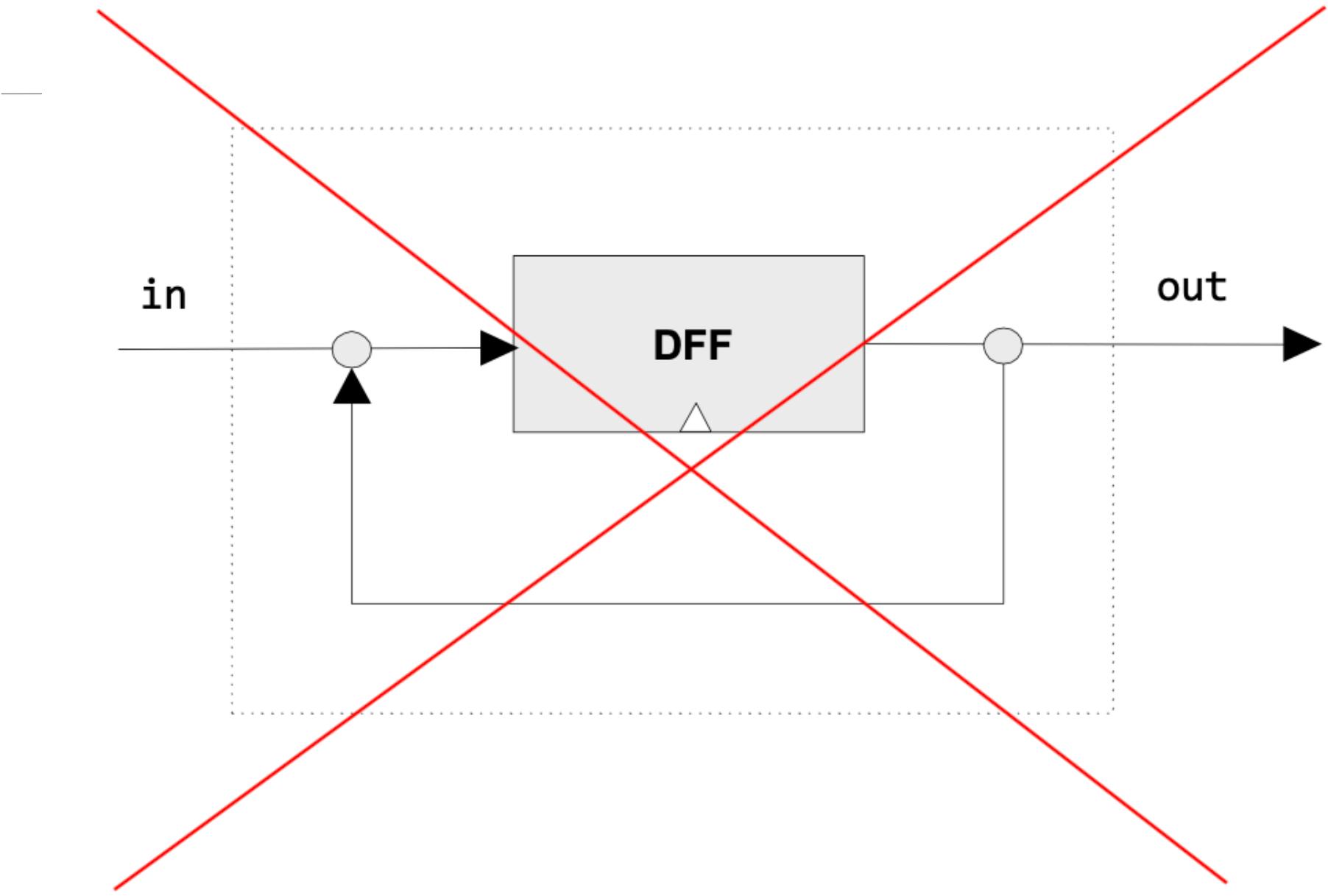


# 1-bit register

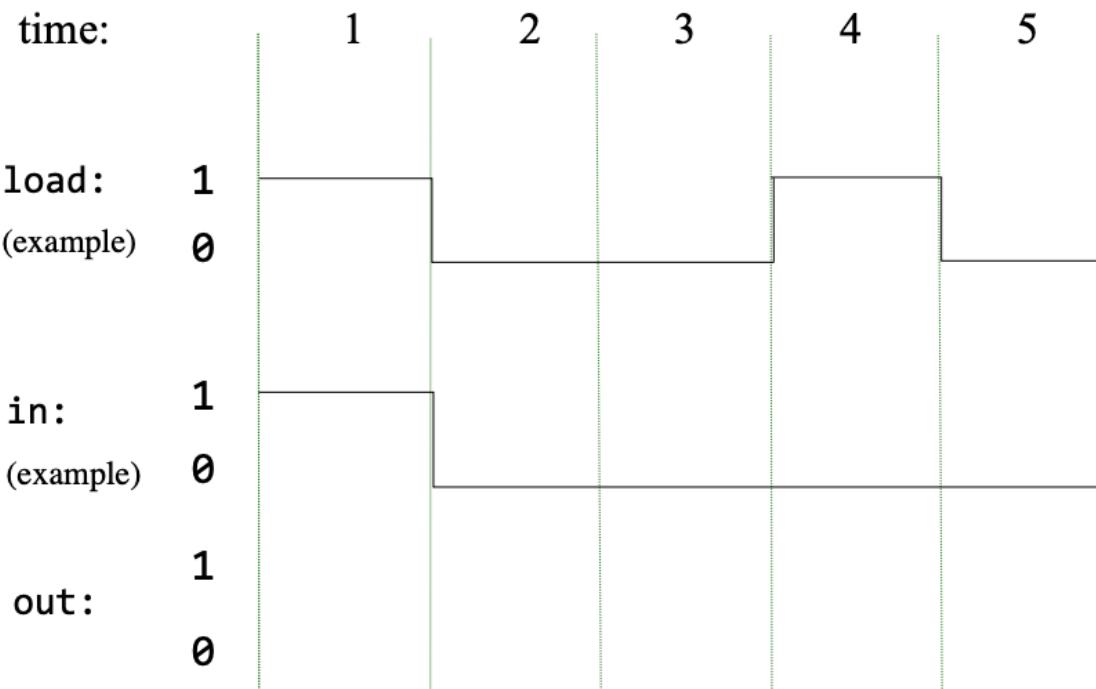
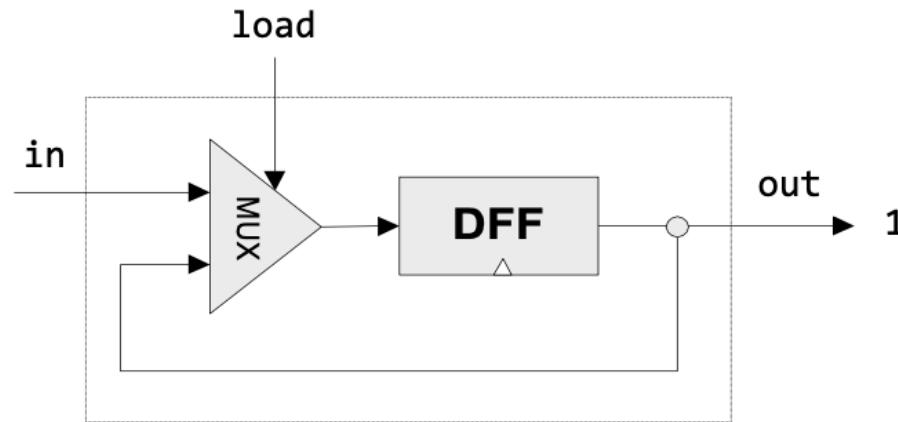


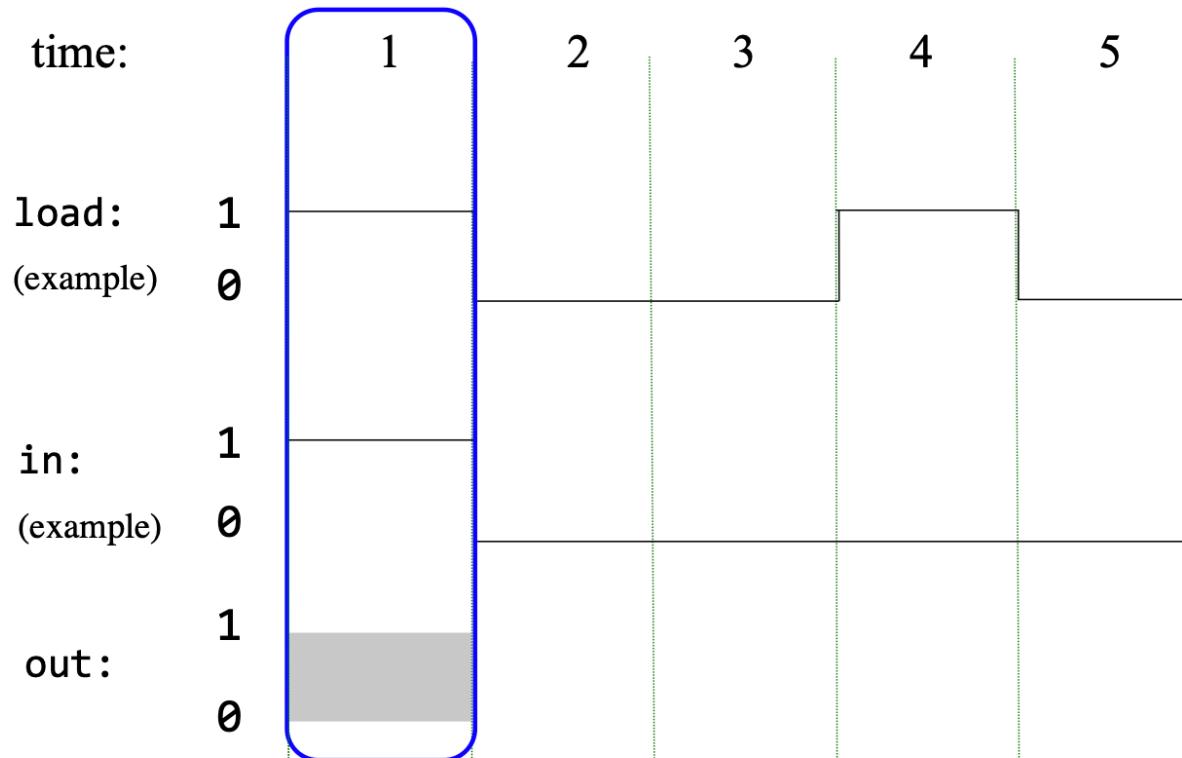
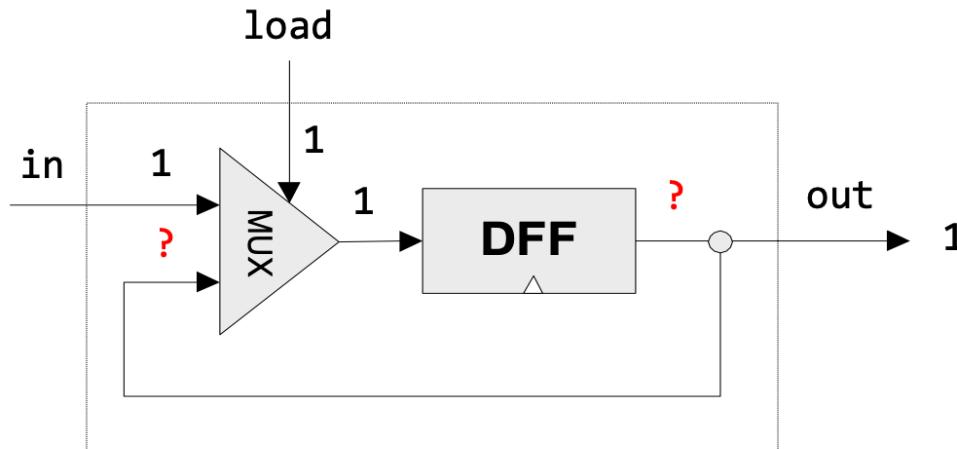
if  $\text{load}(t)$  then  $\text{out}(t+1) = \text{in}(t)$   
else  $\text{out}(t+1) = \text{out}(t)$

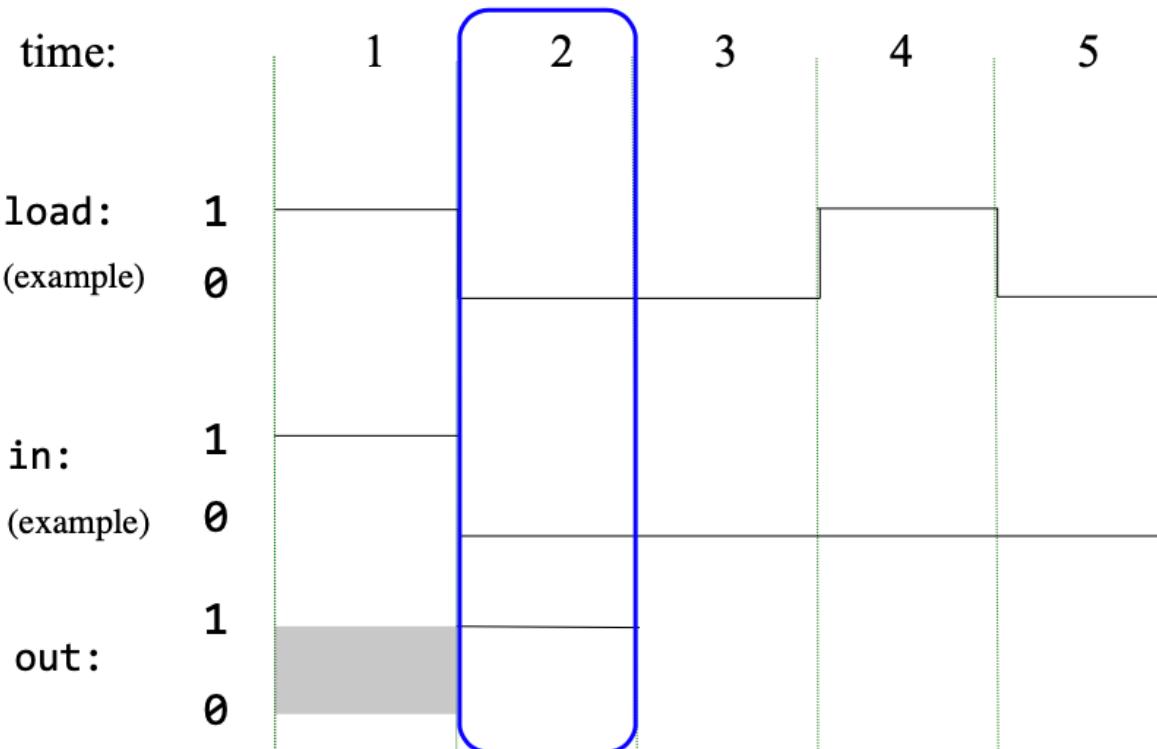
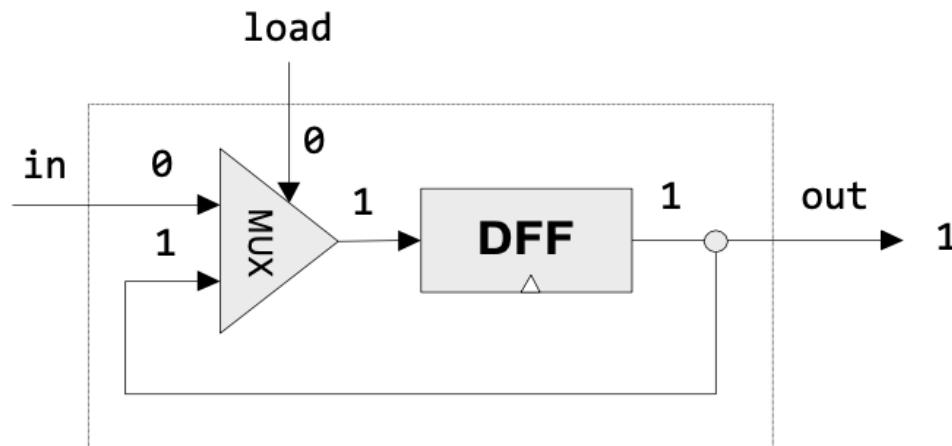


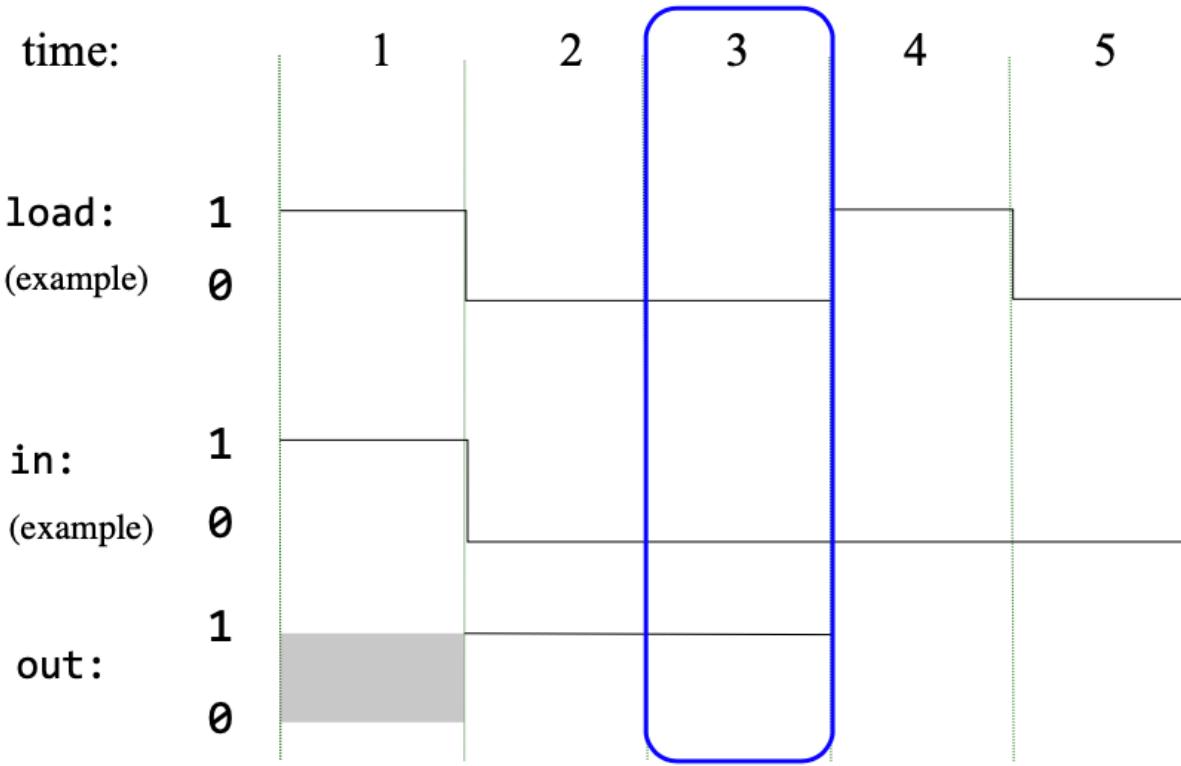
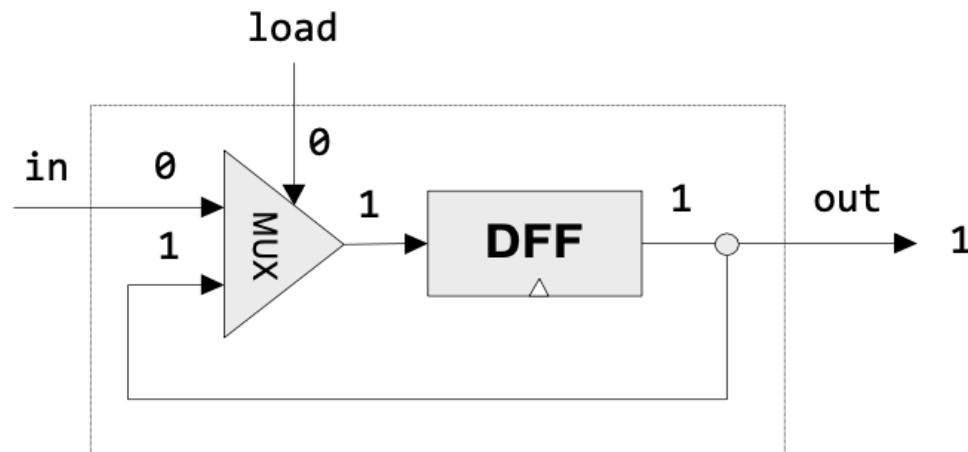


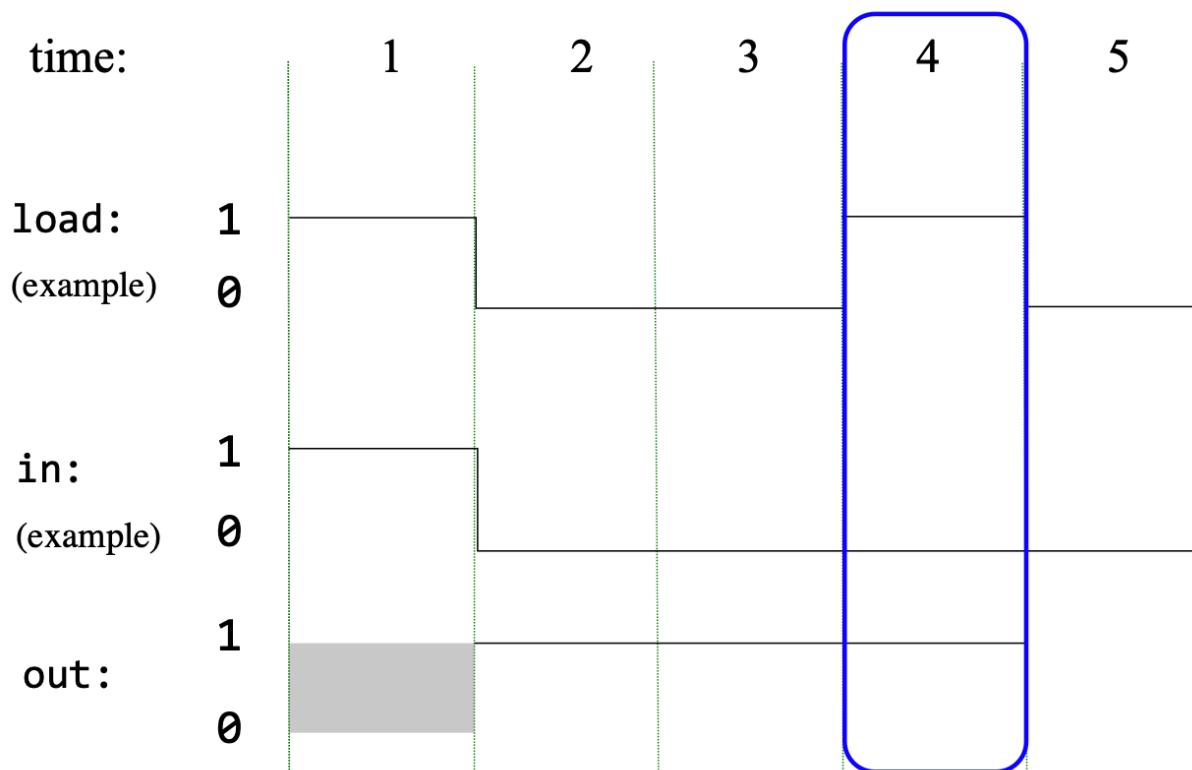
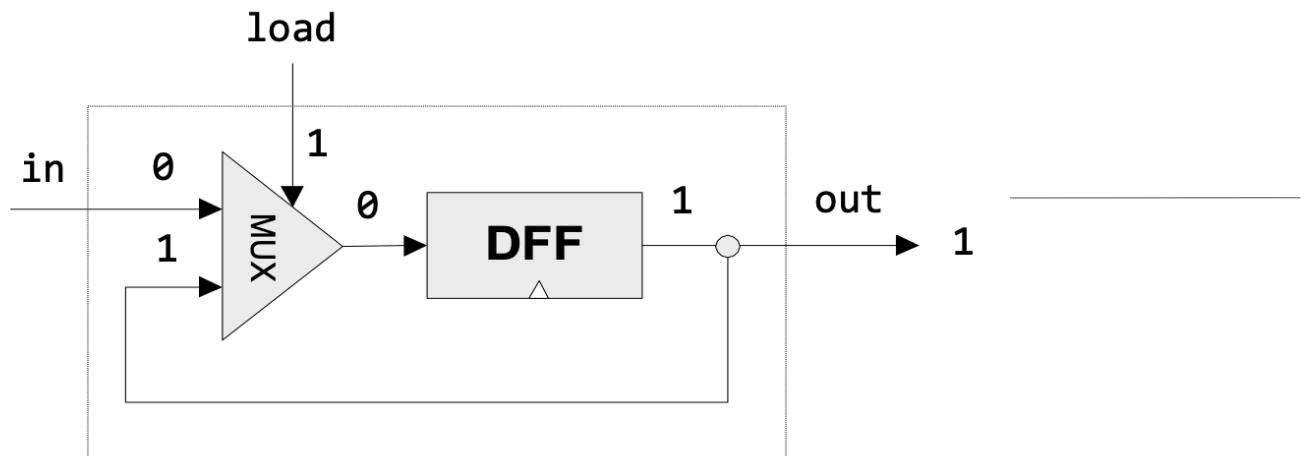
# 1-bit register implementation

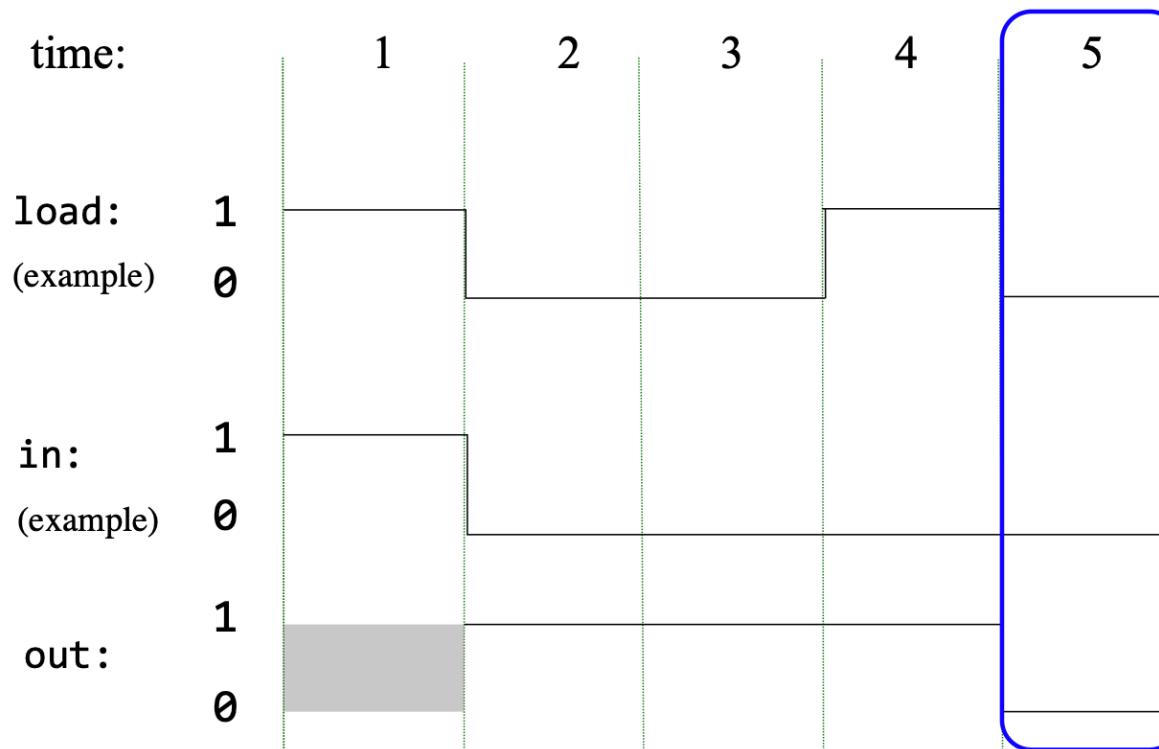
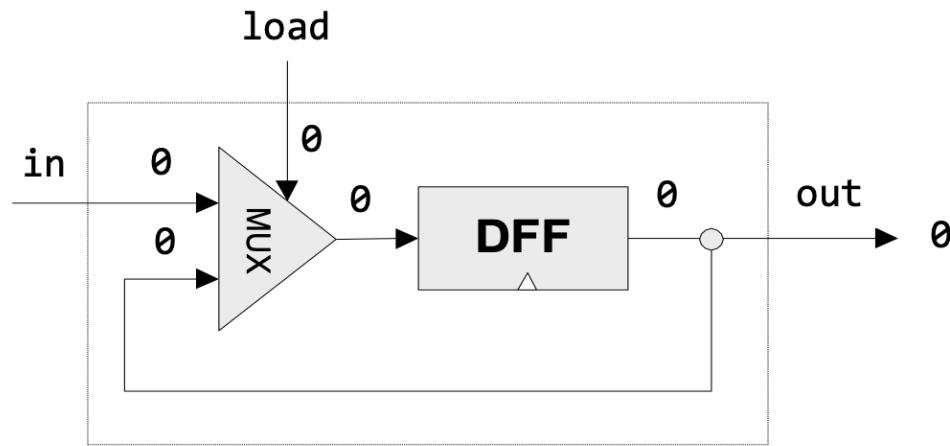


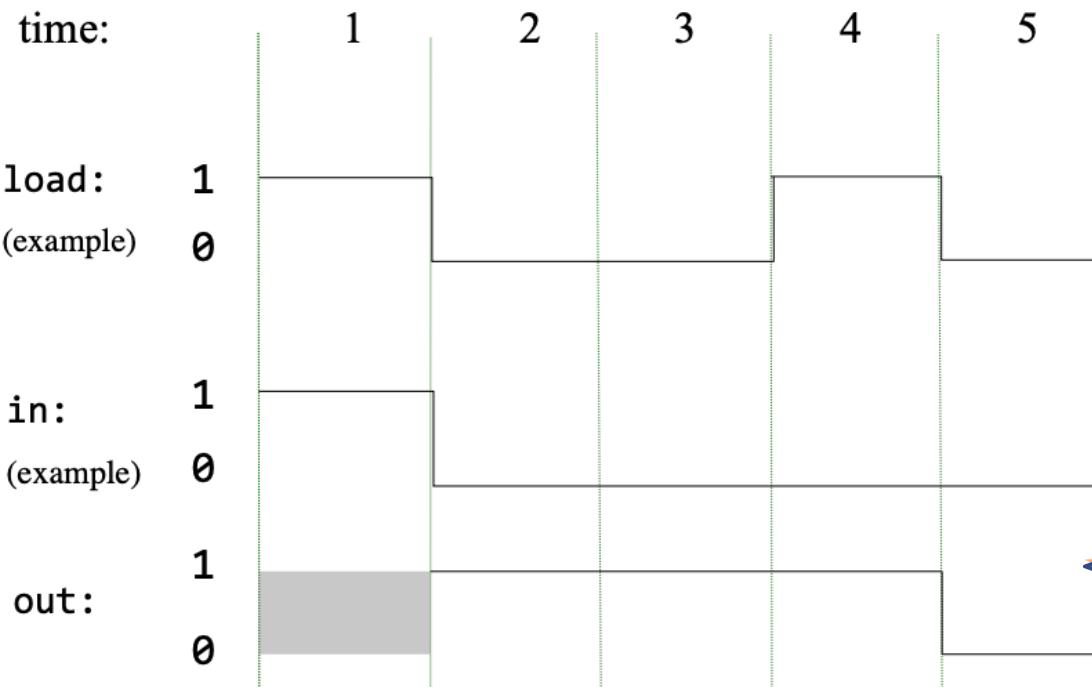
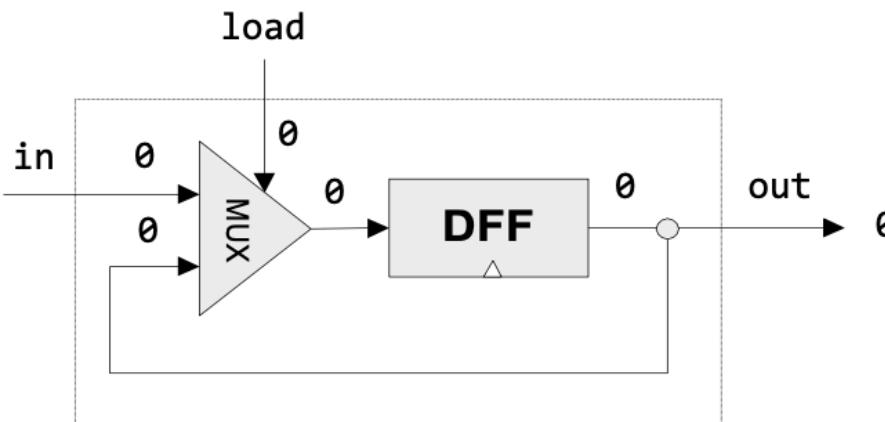












รอจนกว่า  
load เปลี่ยน

# Outline

---

- 3.1 วงจรเชิงลำดับ (Sequential Circuit)
- 3.2 พลิปฟล็อป (Flip-flops)
- **3.3 หน่วยความจำ**
- 3.4 วงจรบันทึก
- 3.5 โปรเจ็ค 3-1

# អង់គេយកគាមតា

---

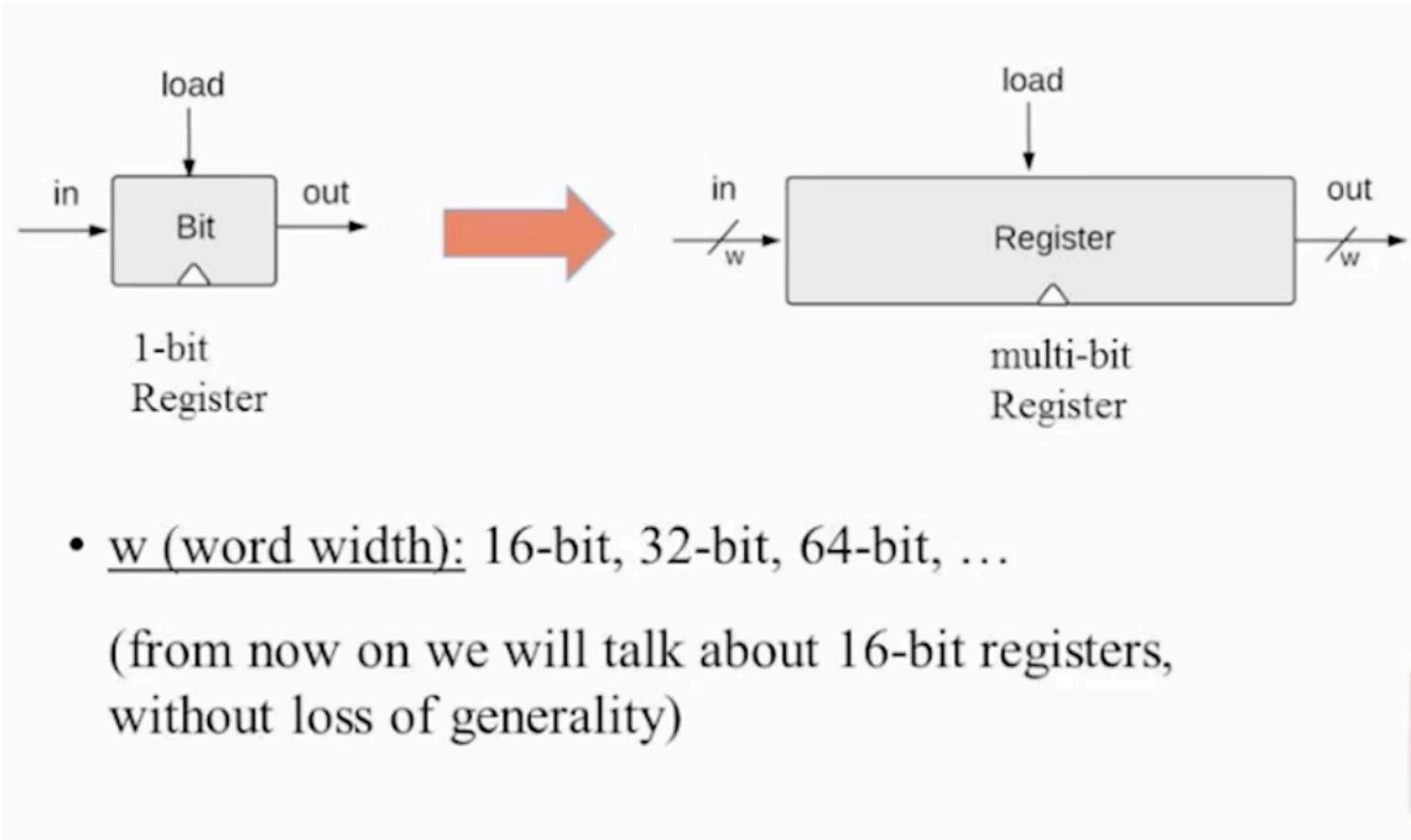
- អង់គេយកគាមតា(Memory , RAM) ធនប  
◦ អារ៉ី សរោះពីកម្លែង DFF
- ឈានតាំងដែលមែនីយ៉ាងមីផែន្ទា → None-volatile
- ឈានតាំងបាតកដែលមែនមីផែន្ទា → volatile memory

# អង់យកវាមាំ

---

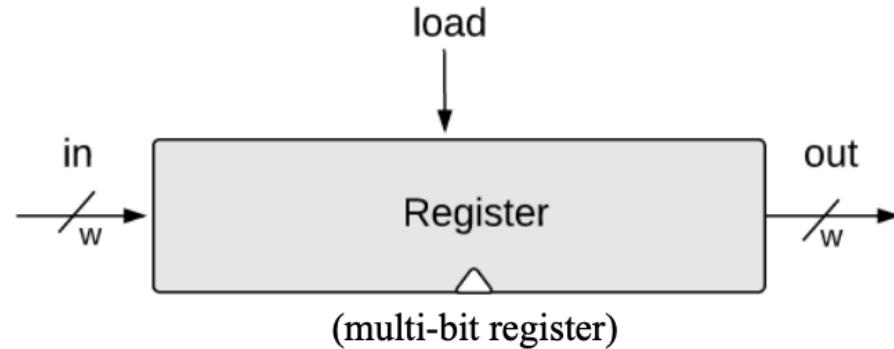
- កំពុងមានការបង្កើតការងារដែលបានរាយការណ៍ជាបន្ទាល់ 1-bit ដែល
  - 1-bit register
  - ត្រូវបានរាយការណ៍ជាបន្ទាល់ 1-bit register មានចំណាំថាអាយុបាន
  - **Multi-bit register**

# The most basic memory element: Register



# Register: abstraction

---



- ต้องการอ่าน (read)
  - probe out
- ต้องการบันทึกลง register
  - set in=v
  - set load=1

# Register chip in action

Run the clock

Chip Nam... DRegister (Clocked) Time : 0

Input pins

Name	Value
in[16]	17
load	0

Set in to 17

Output pins

Name	Value
out[16]	0

Inspect the register's output

HDL

```
* This built-in chip implements  
* providing a GUI representation  
* called "D register" (typically)  
*/  
  
CHIP DRegister {  
  
    IN  in[16], load;  
    OUT out[16];  
  
    BUILTIN DRegister;  
    CLOCKED in, load;  
}
```

Inspect the register's contents

D: 0

For the demo, we use a built-in 16-bit register from the Hack chipset, named **Dregister**, or simply **D**.

# Register chip in action

The screenshot shows a digital design tool interface with the following components:

- Toolbar:** Includes icons for file operations, project navigation, and simulation controls. A red circle highlights the clock icon (an alarm clock).
- Chip Name:** DRegister (Clocked)
- Time:** Set to 12
- Input pins:** in[16] (Value: 17), load
- Output pins:** out[16] (Value: 0)
- HDL:** Chip definition code (shown below)
- Output Inspect:** Shows the register's contents (D) as 0.

Annotations with orange callouts:

- A callout points to the clock icon in the toolbar with the text "Run the clock".
- A callout points to the output pin "out[16]" with the text "Inspect the register's output".
- A callout points to the "D" input field with the text "Inspect the register's contents".

```
* This built-in chip implementation provides a GUI representation of a D register (typically implemented by a flip-flop). */
CHIP DRegister {
    IN in[16], load;
    OUT out[16];
    BUILTIN DRegister;
    CLOCKED in, load;
}
```

# Register chip in action

The screenshot shows the NANOZLETRIS software interface with a focus on a DRegister chip. The top menu bar includes icons for file operations, a toolbar with symbols like >>> and <<<, and a status bar with 'Format: D...', 'View: Scr...', and clock speed settings 'Slow' and 'Fast'. A red circle highlights the clock icon in the toolbar, with a callout 'Run the clock' pointing to it.

The main window displays the 'DRegister (Clocked)' chip configuration. It has two sections: 'Input pins' and 'Output pins'.

- Input pins:** Shows 'in[16]' set to 17 and 'load' set to 1. A callout 'Set in to 17' points to the value 17, and another callout 'Set load to 1' points to the value 1.
- Output pins:** Shows 'out[16]' set to 0. A callout 'Inspect the register's output' points to this value.

A large orange arrow points from the 'Output pins' section towards a status bar at the bottom right. The status bar shows 'D:' followed by a text input field containing the value '17'. A blue arrow points upwards from the status bar towards the 'Output pins' section.

**HDL** code for the DRegister chip is shown in the bottom left:

```
* This built-in chip implements a D register
* providing a GUI representation
* called "D register" (typically used for memory)
*/
CHIP DRegister {
    IN in[16], load;
    OUT out[16];

    BUILTIN DRegister;
    CLOCKED in, load;
}
```

At the bottom of the screen, a footer bar reads: 'NANOZLETRIS / www.nanozletris.org / Chapter 5 / Copyright © Noam Nissim and Shimon Schocken'.

# Register chip in action

Run the clock

Time : 13

Inspect the register's output

Inspect the register's contents

Input pins		Output pins	
Name	Value	Name	Value
in[16]	17	out[16]	17
load	1		

**HDL**

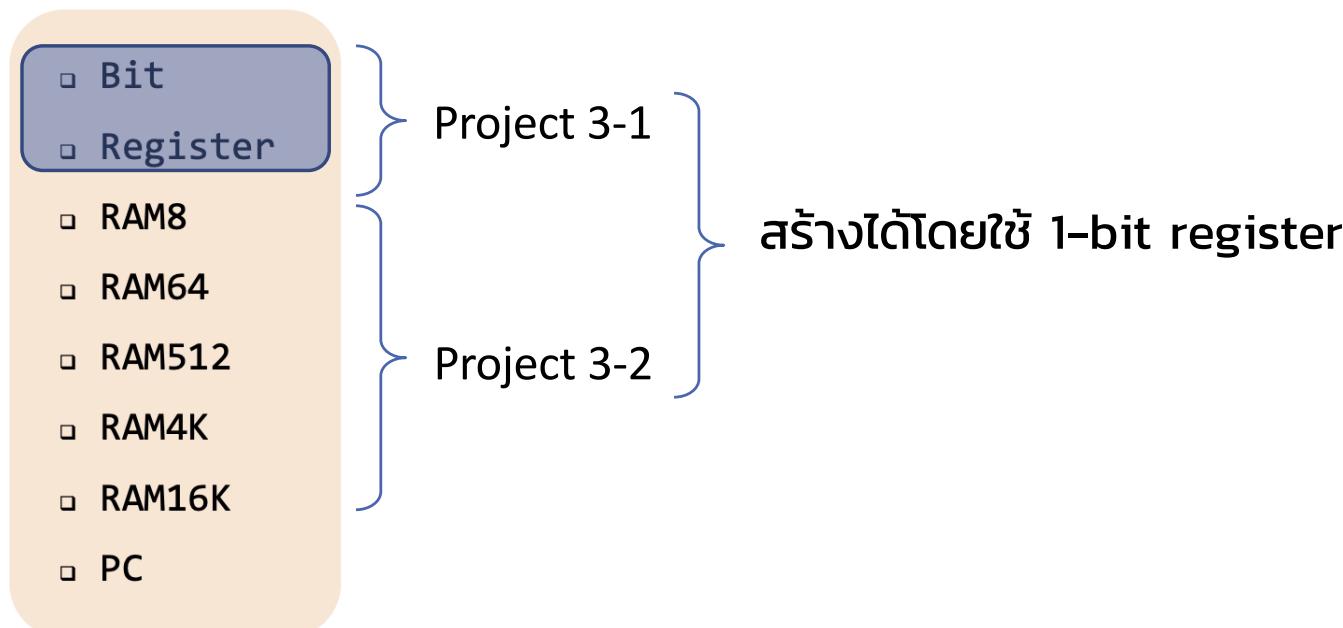
```
* This built-in chip implements
* providing a GUI representation
* called "D register" (typically
*/
CHIP DRegister {
    IN in[16], load;
    OUT out[16];

    BUILTIN DRegister;
    CLOCKED in, load;
}
```

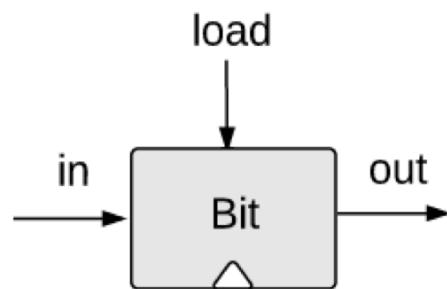
# Project 3-1

- Given:
  - All the chips built in Project 1 and 2
  - Flip-Flop (built-in DFF gate)

## • ผลิตซึป



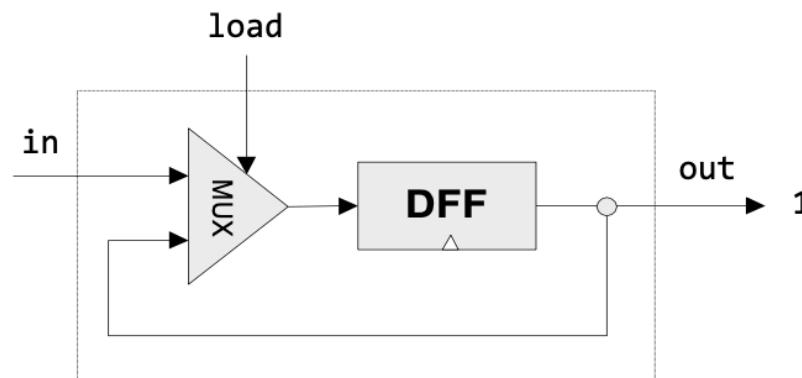
# 1-bit register



Bit.hdl

```
/**  
 * 1-bit register:  
 * If load(t) then out(t+1) = in(t)  
 * else          out(t+1) = out(t)  
 */  
  
CHIP Bit {  
    IN in, load;  
    OUT out;  
  
    PARTS:  
        // Put your code here:  
}
```

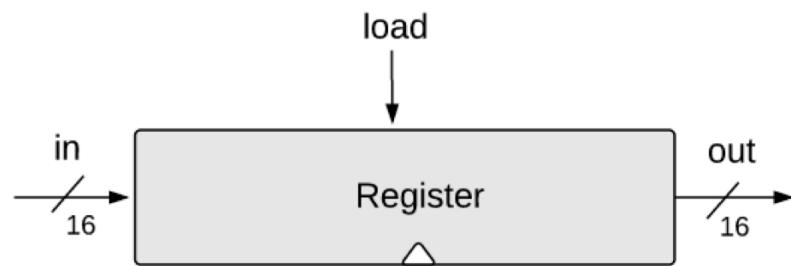
- สร้าง 1-bit register โดยใช้ DFF



```
CHIP Bit {  
    IN in, load;  
    OUT out;  
  
    PARTS:
```

```
        Mux(a=rout1,b=in,sel=load,out=a);  
        DFF(in=a,out=out,out=rout1);  
    }
```

# 16-bit register



Register.hdl

```
/**  
 * 16-bit register:  
 * If load(t) then out(t+1) = in(t)  
 * else out(t+1) = out(t))  
 */  
  
CHIP Register {  
    IN in[16], load;  
    OUT out[16];  
  
    PARTS:  
        // Put your code here:  
}
```

- สร้าง 16-bit register โดยใช้
  - 1-bit register

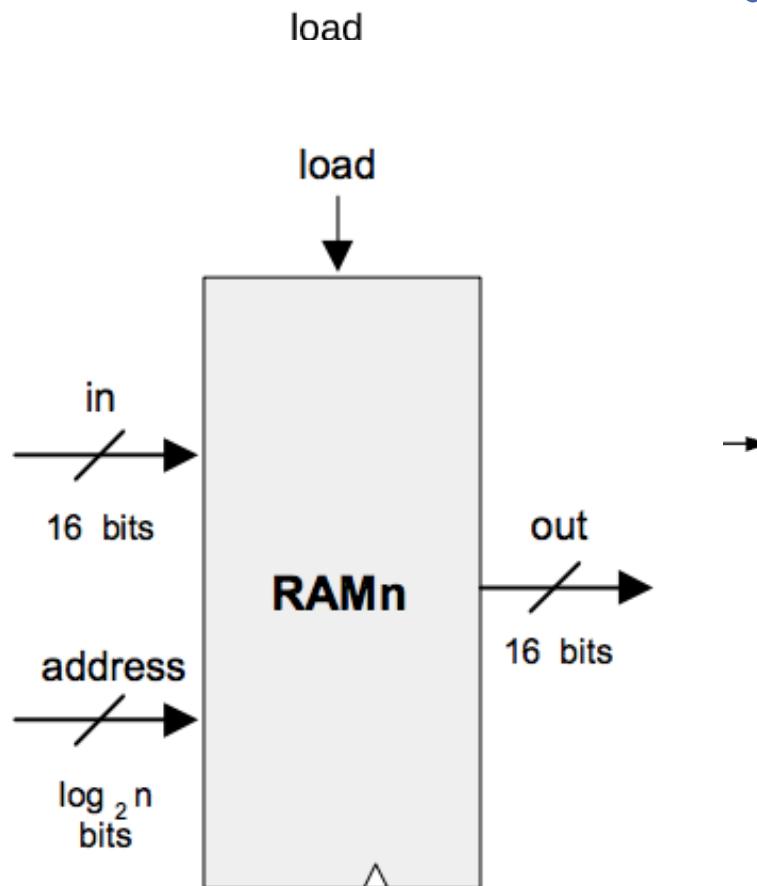
```
CHIP Register {  
    IN in[16], load;  
    OUT out[16];
```

PARTS:

```
    Bit(in=in[0],load=load,out=out[0]);  
    Bit(in=in[1],load=load,out=out[1]);  
    Bit(in=in[2],load=load,out=out[2]);  
    Bit(in=in[3],load=load,out=out[3]);  
    Bit(in=in[4],load=load,out=out[4]);  
    Bit(in=in[5],load=load,out=out[5]);  
    Bit(in=in[6],load=load,out=out[6]);  
    Bit(in=in[7],load=load,out=out[7]);  
    Bit(in=in[8],load=load,out=out[8]);  
    Bit(in=in[9],load=load,out=out[9]);  
    Bit(in=in[10],load=load,out=out[10]);  
    Bit(in=in[11],load=load,out=out[11]);  
    Bit(in=in[12],load=load,out=out[12]);  
    Bit(in=in[13],load=load,out=out[13]);  
    Bit(in=in[14],load=load,out=out[14]);  
    Bit(in=in[15],load=load,out=out[15]);
```

}

# 8-Register RAM



- สร้าง 8-Register โดยใช้
  - 16-bit input /output
  - 3-bit addressing

RAM8.hdl

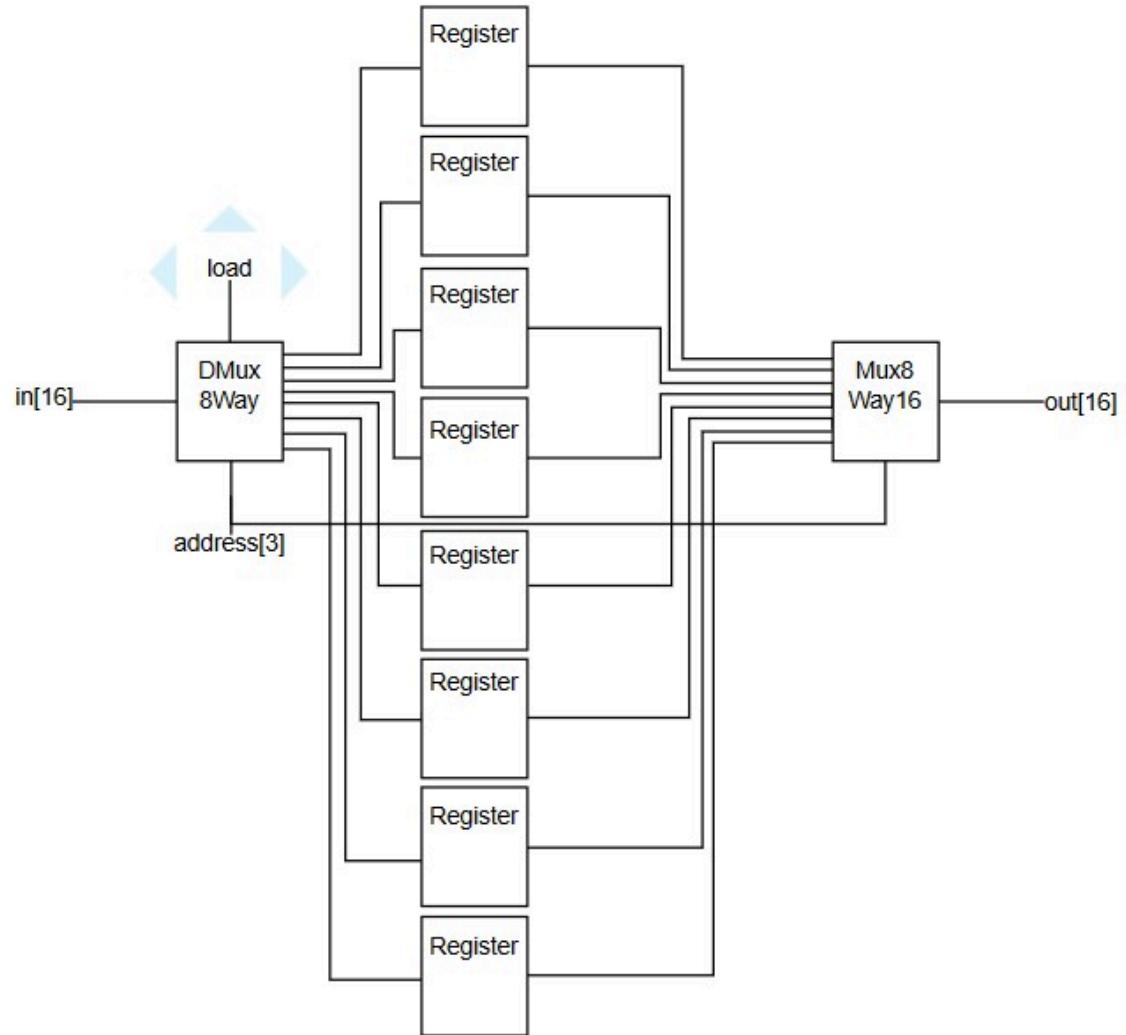
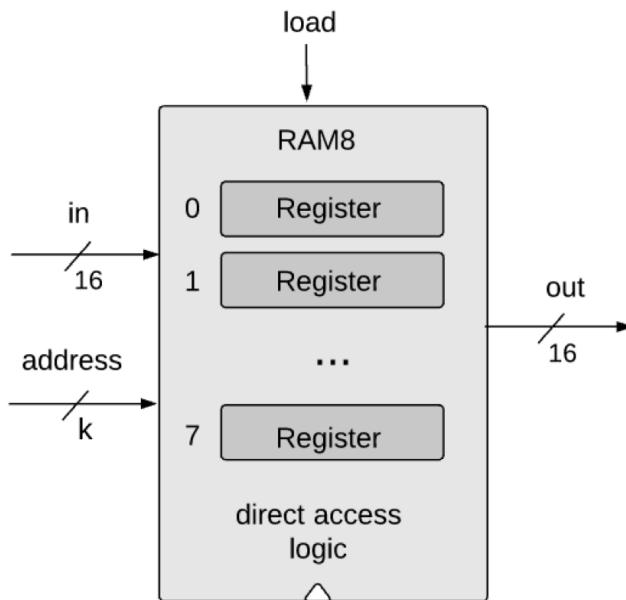
```
/*
 * Let M stand for the state of the
 * register selected by address.
 * if load(t) then {M=in(t), out(t+1)=M}
 * else
 *          out(t+1)=M
 */

CHIP RAM8 {
    IN in[16], load, address[3];
    OUT out[16];

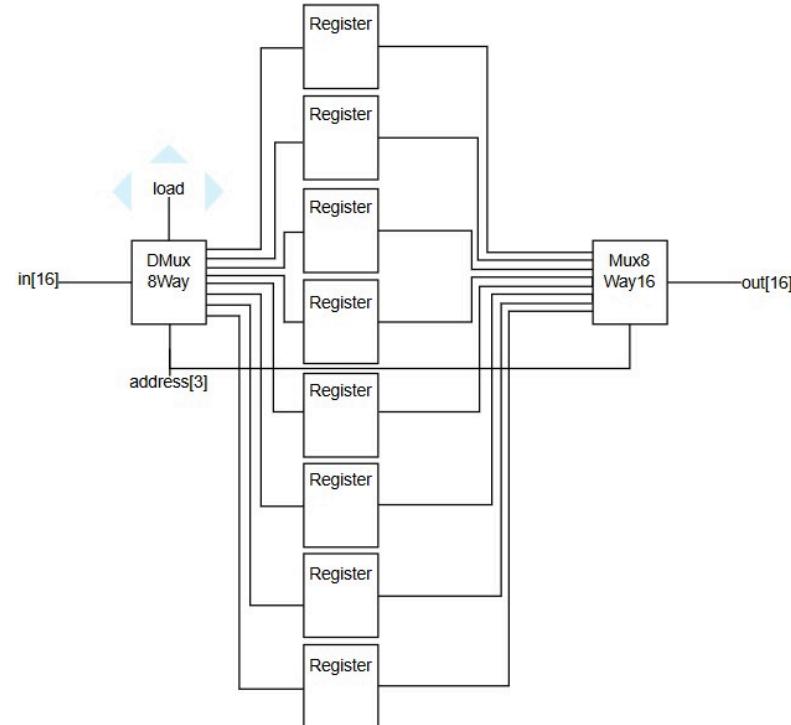
    PARTS:
        // Put your code here:
}
```

# 8-Register RAM

- RAM8.hdl



# RAM8.hdl

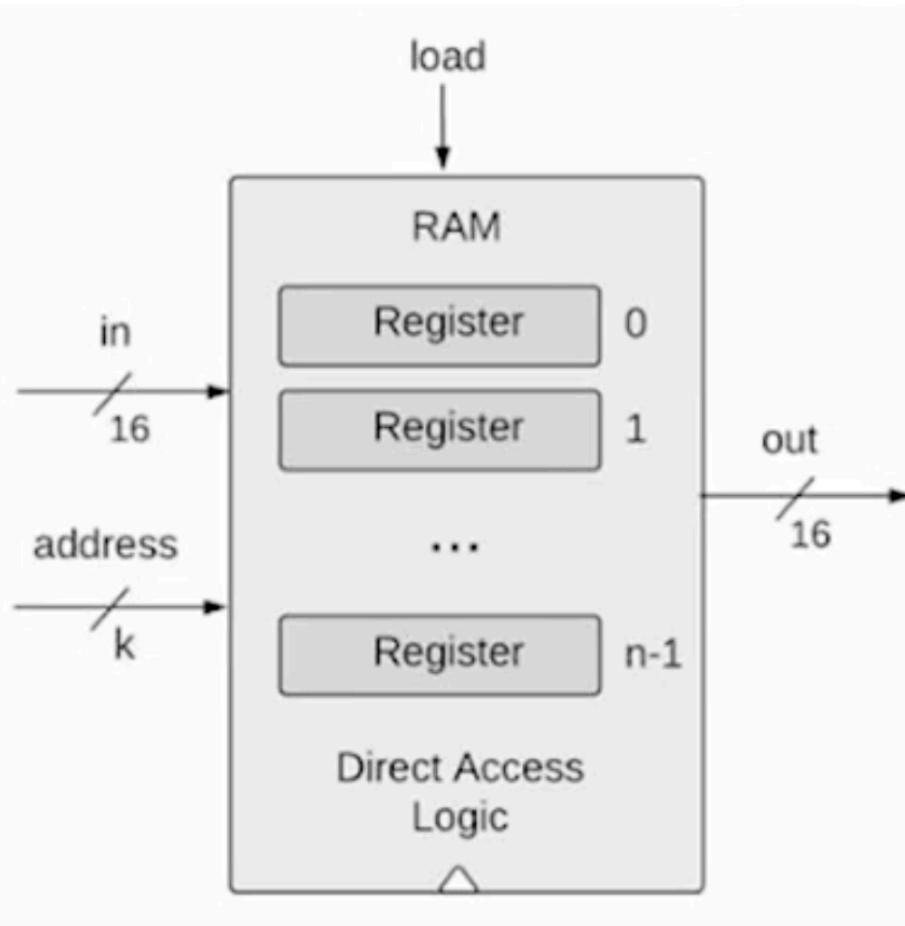


```
CHIP RAM8 {
    IN in[16], load, address[3];
    OUT out[16];

    PARTS:
        DMux8Way(in=load,sel=address[0..2],a=loadA,b=loadB,c=loadC,d=loadD,e=loadE,f=loadF,g=loadG,h=loadH);
        Register(in=in, load=loadA, out=outA);
        Register(in=in, load=loadB, out=outB);
        Register(in=in, load=loadC, out=outC);
        Register(in=in, load=loadD, out=outD);
        Register(in=in, load=loadE, out=outE);
        Register(in=in, load=loadF, out=outF);
        Register(in=in, load=loadG, out=outG);
        Register(in=in, load=loadH, out=outH);

        Mux8Way16(a=outA, b=outB, c=outC, d=outD, e=outE, f=outF, g=outG, h=outH, sel=address[0..2], out=out)
}
```

# A family of 16-bit RAM chips



chip name	n	k
RAM8	8	3
RAM64	64	6
RAM512	512	9
RAM4K	4096	12
RAM16K	16384	14

# Counters

---

- สมมติ ร้านอาหาร Fast food มีระบบแจกบัตรคิว
  - 1. เดินเข้าร้านอาหาร
  - 2. รับบัตรคิว อาจเป็นเลข 0 หรือเลขใด ๆ ( $n$ )
  - 3. พนักงานรับ Order และ กดปุ่มเลื่อนเลข
  - 4. คนต่อไปจะได้รับบัตรคิว  $0+1$  หรือ  $n+1$  เป็นอย่างนี้เพิ่มขึ้นเรื่อยๆ
  - 5. พนักงานกดปุ่มเดิม แต่เลขเพิ่มทีละ 1