

# State & Finite State Machines

**Hakim Weatherspoon**

**CS 3410, Spring 2011**

Computer Science

Cornell University

See P&H Appendix C.7, C.8, C.10, C.11

# Announcements

---

Make sure you are

Registered for class

Can access CMS

Have a Section you can go to

Have a project partner

**Sections are on this week**

HW 1 out later today

Due in one week, start early

Work **alone**

Use your resources

- Class notes, book, Sections, office hours, newsgroup, CSUGLab<sub>2</sub>

# Announcements

---

Check online syllabus/schedule

Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims: Evening of Thursday, March 10 and April 28<sup>th</sup>

Schedule is subject to change

## HW1 Correction:

Hint 1: Your ALU should use your adder and left shifter as components. But, as in class, your ALU should only use a single adder component to implement both addition and subtraction. Similarly, your ALU should use only a single left shifter component to implement all of the shift

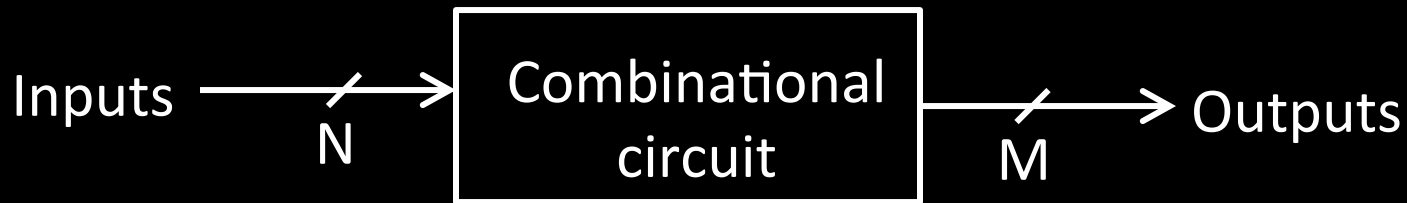
operations. For instance, **left right** shifting can be accomplished by transforming the inputs and outputs to your left shifter. You will be penalized if your final ALU circuit uses more than one adder or left shifter. Of course, always strive to make your implementation clear, but do not duplicate components in an effort to do so.

# Goals for Today: Stateful Components

---

Until now is combinatorial logic

- Output is computed when inputs are present
- System has no internal state
- Nothing computed in the present can depend on what happened in the past!



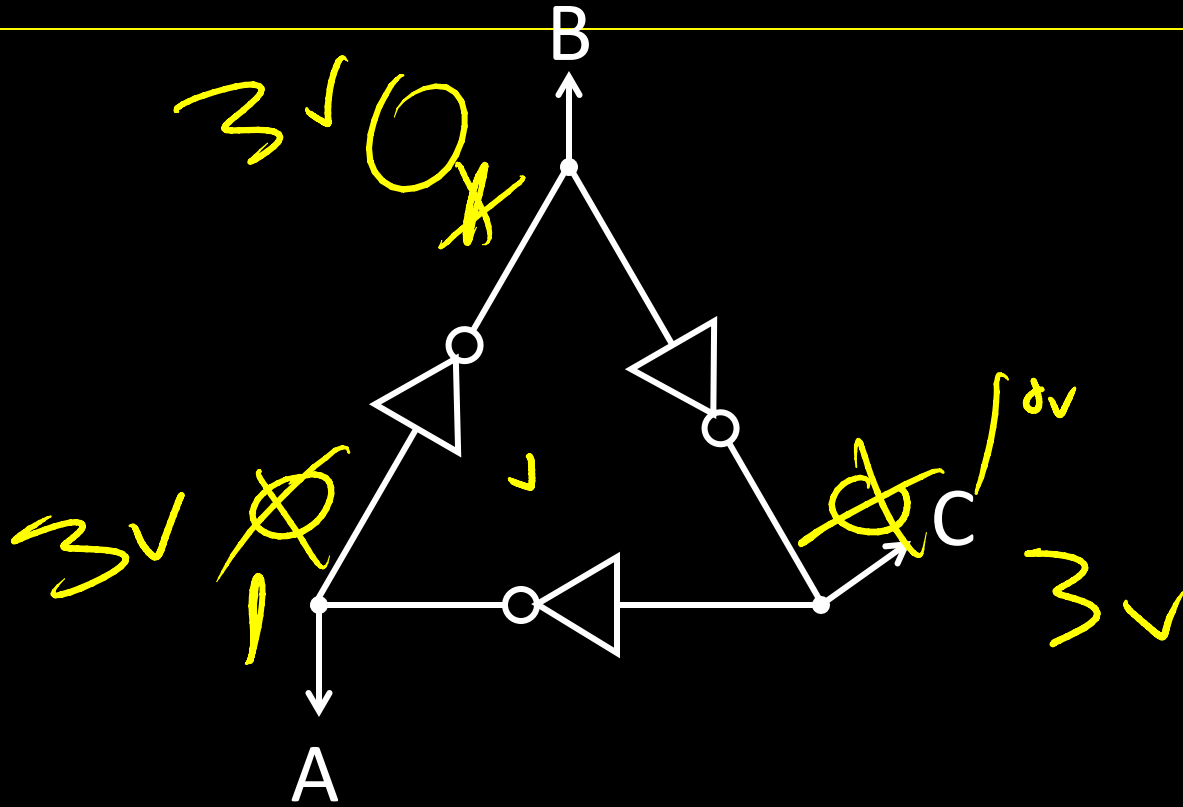
Need a way to record data

Need a way to build **stateful** circuits

Need a state-holding device

## Finite State Machines

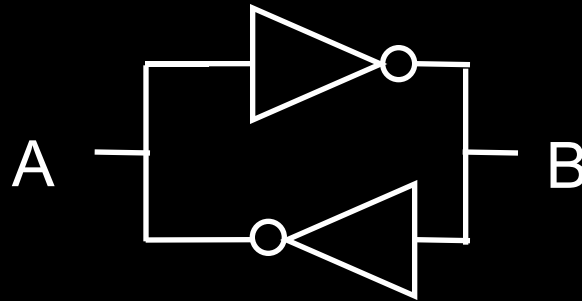
# Unstable Devices



# Bistable Devices

---

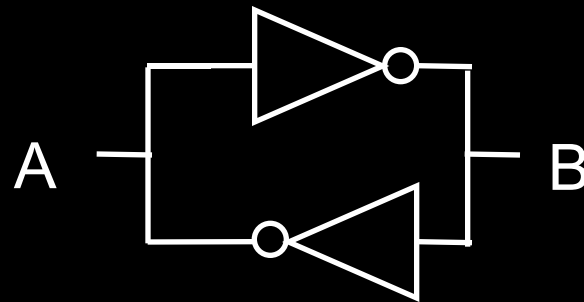
- Stable and unstable equilibria?



A Simple Device

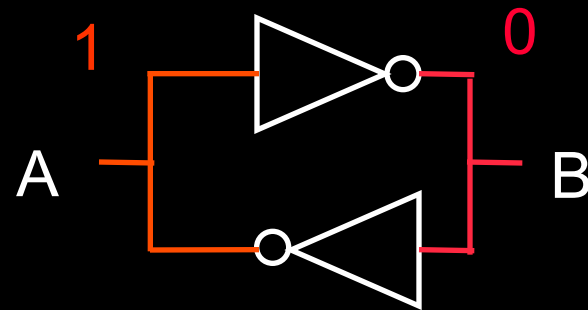
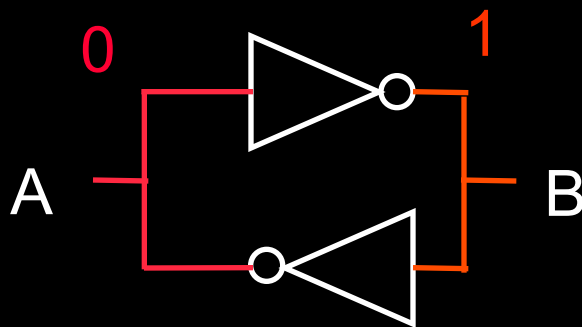
# Bistable Devices

- Stable and unstable equilibria?



A Simple Device

- In stable state,  $\bar{A} = B$



- How do we change the state?

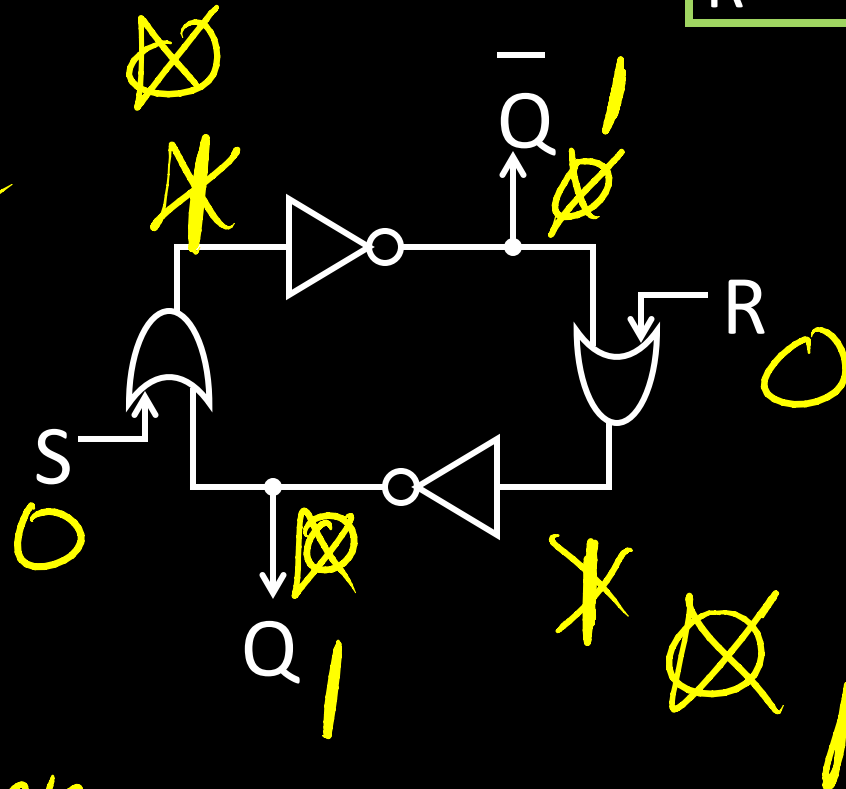
# SR Latch

## Set-Reset (SR) Latch

Stores a value  $Q$  and its complement  $\bar{Q}$



S	R	Q	$\bar{Q}$
0	0	hold state	
0	1	0	1
1	0	1	0
<del>1</del>	<del>1</del>	<del>forbidden</del>	<del>forbidden</del>

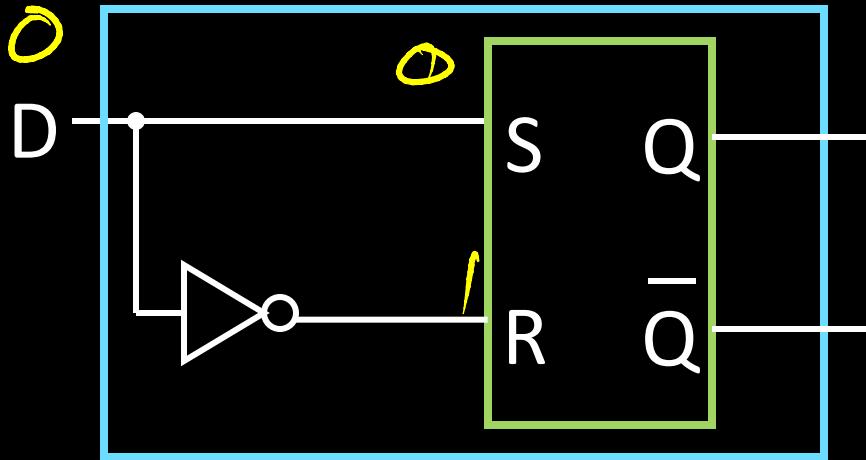


forbidden



# Unclocked D Latch

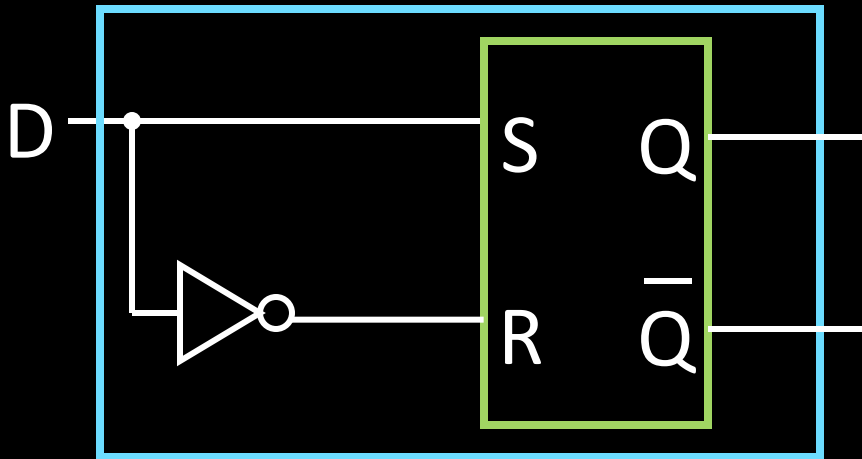
Data (D) Latch



D	Q	$\bar{Q}$
0	0	1
1	1	0

# Unclocked D Latch

## Data (D) Latch



D	Q	$\bar{Q}$
0	0	1
1	1	0

## Data Latch

- Easier to use than an SR latch
- No possibility of entering an undefined state

When D changes, Q changes

– ... immediately (after a delay of 2 Ors and 2 NOTs)

Need to control when the output changes

# D Latch with Clock

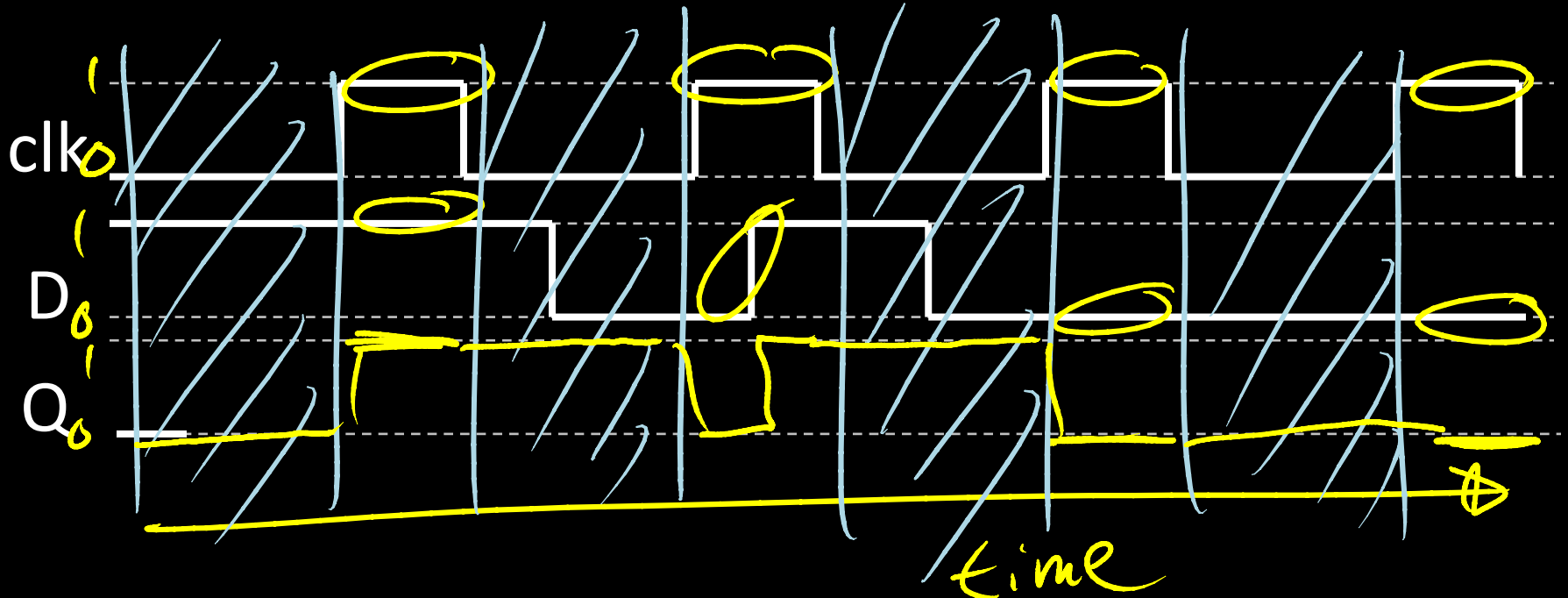
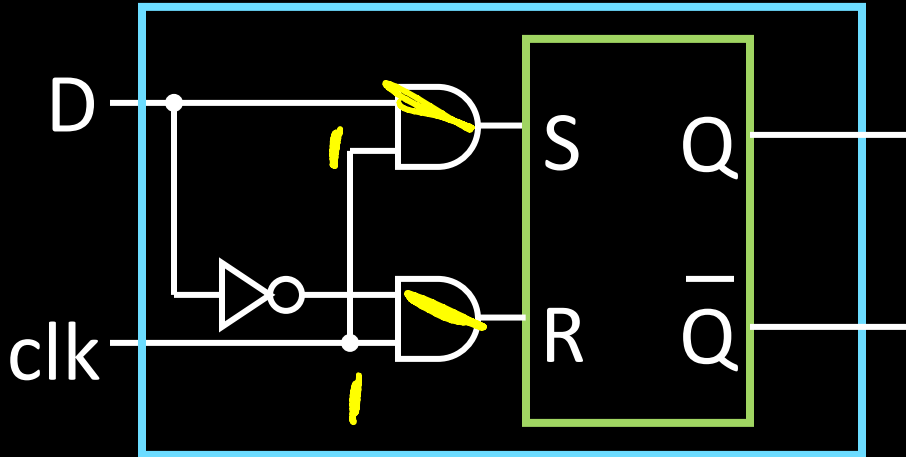
## Level Sensitive D Latch

Clock high:

set/reset (according to D)

Clock low:

keep state (ignore D)

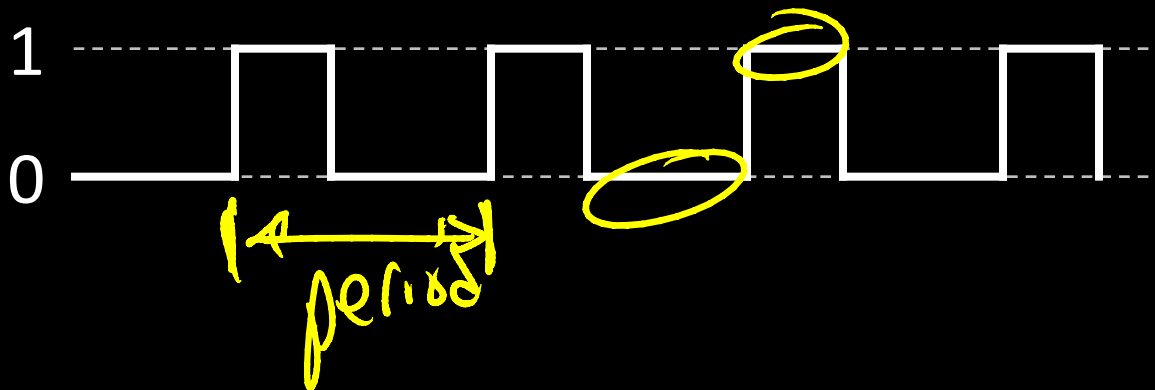


# Clocks

---



**Clock** helps coordinate state changes

- Usually generated by an oscillating crystal
- Fixed period; frequency =  $1/\text{period}$



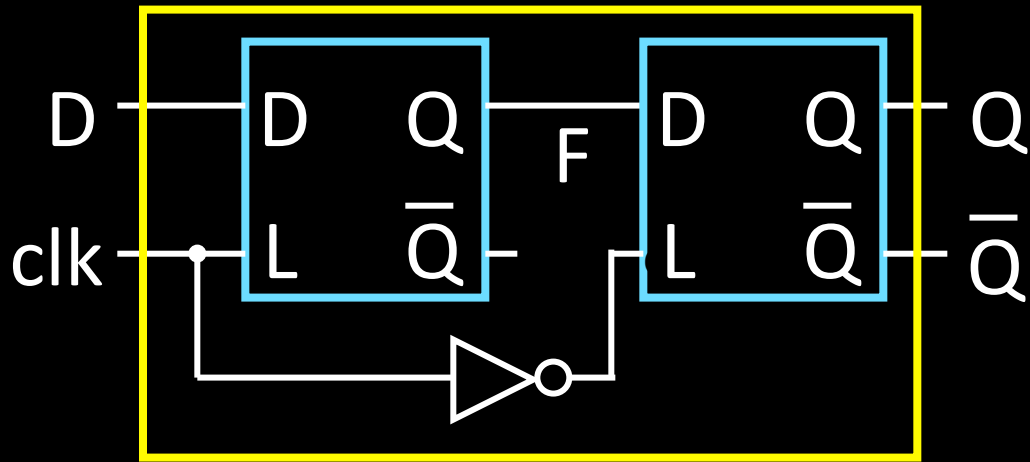
# Edge-triggering

---

- Can design circuits to change on the rising or falling edge
- Trigger on rising edge = positive edge-triggered 
- Trigger on falling edge = negative edge-triggered 
- Inputs must be stable just before the triggering edge

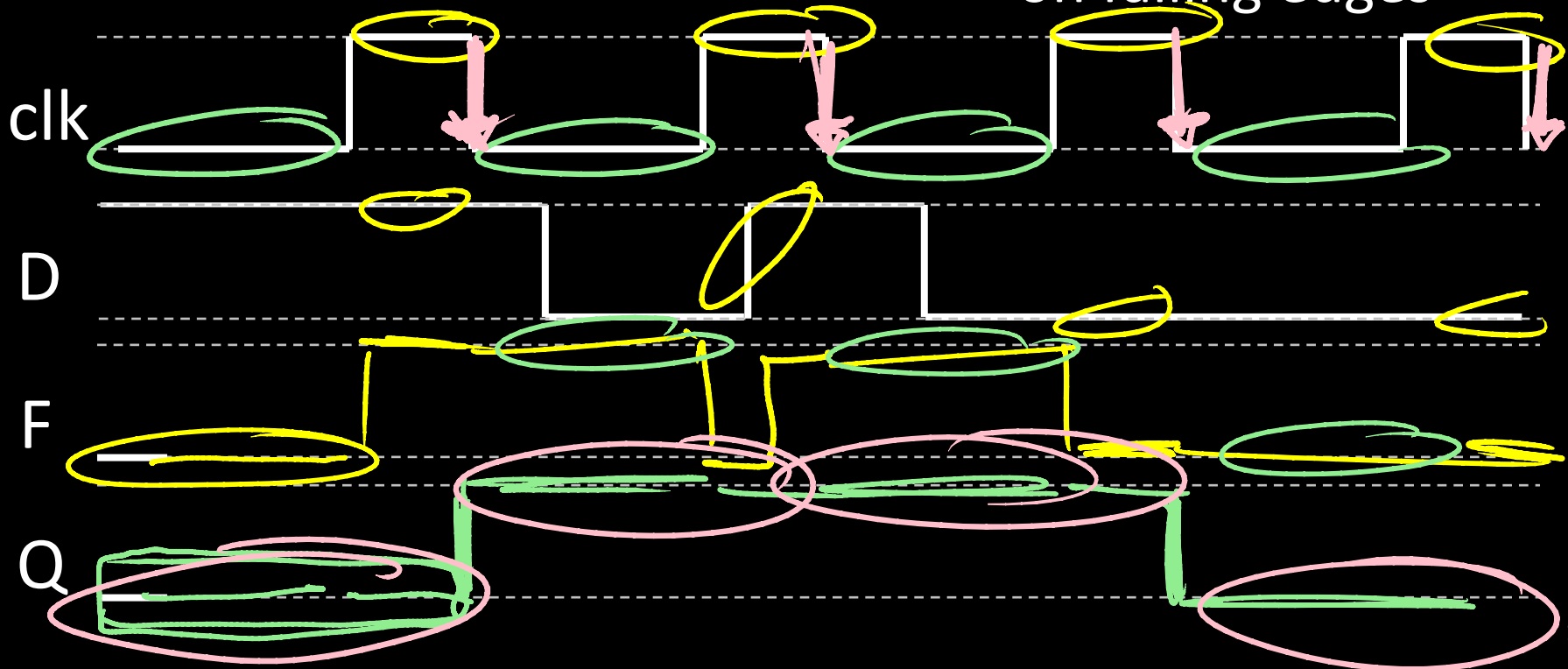


# Edge-Triggered D Flip-Flop



## D Flip-Flop

- Edge-Triggered
- Data is captured when clock is high
- Outputs change only on falling edges



# Clock Disciplines

---

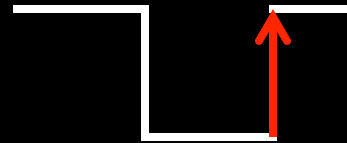
## Level sensitive

- State changes when clock is high (or low)

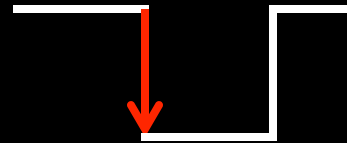
## Edge triggered

- State changes at clock edge

positive edge-triggered



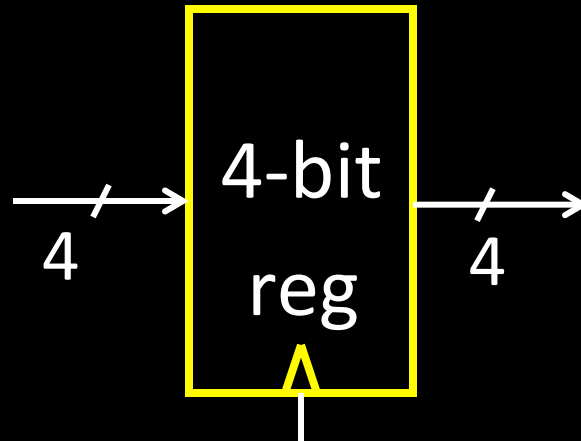
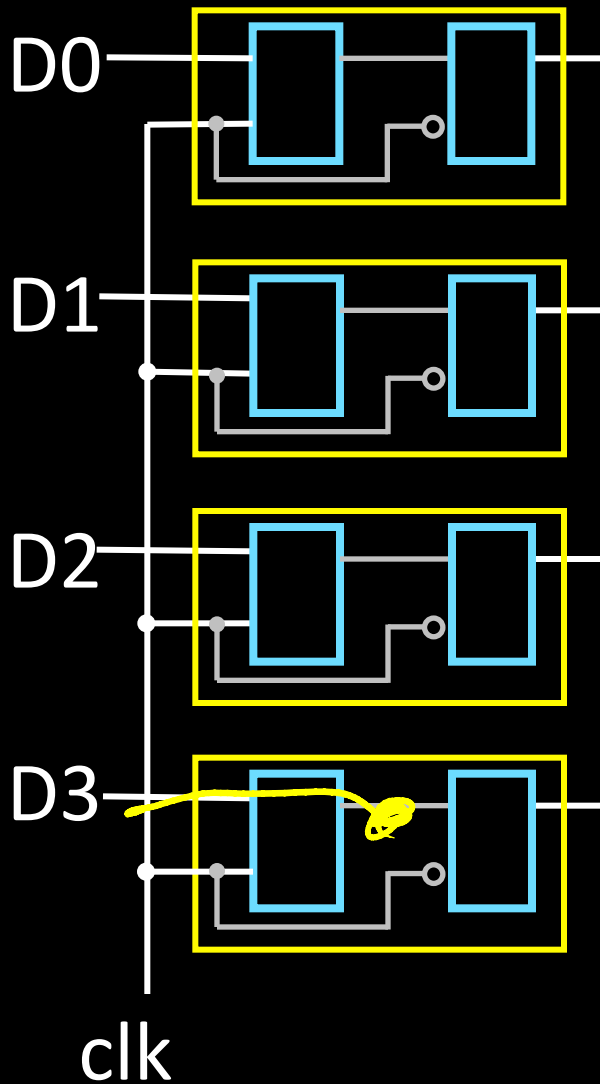
negative edge-triggered



# Registers

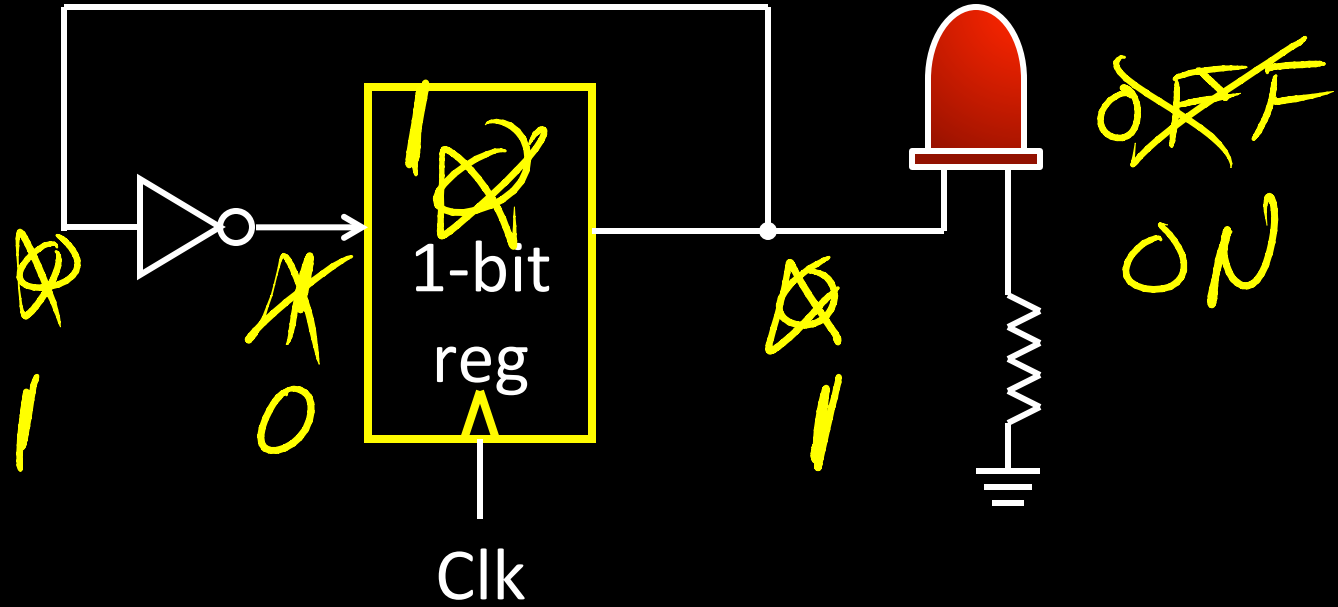
## Register

- D flip-flops in parallel
- shared clock
- extra clocked inputs: write\_enable, reset, ...





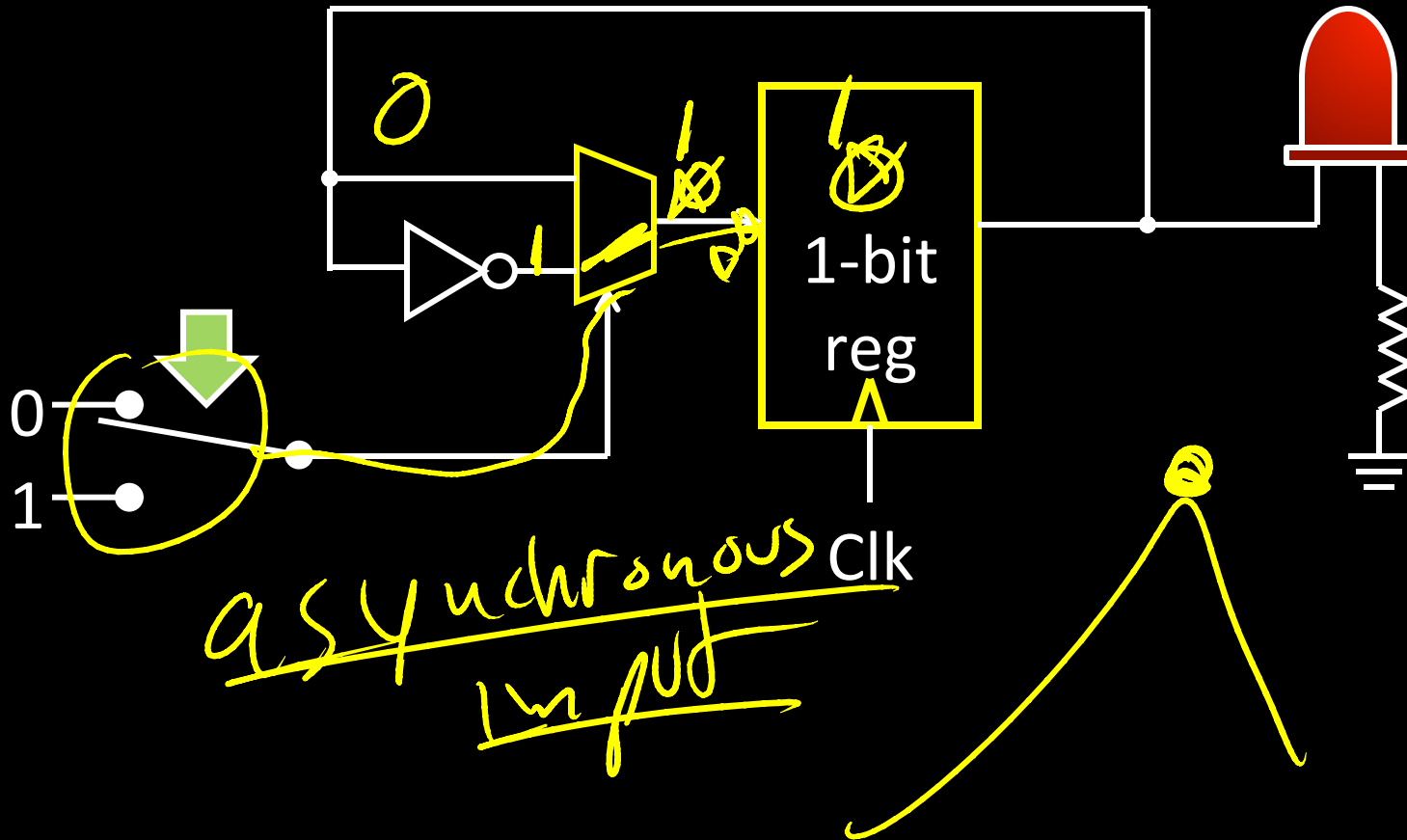
# Metastability and Asynchronous Inputs



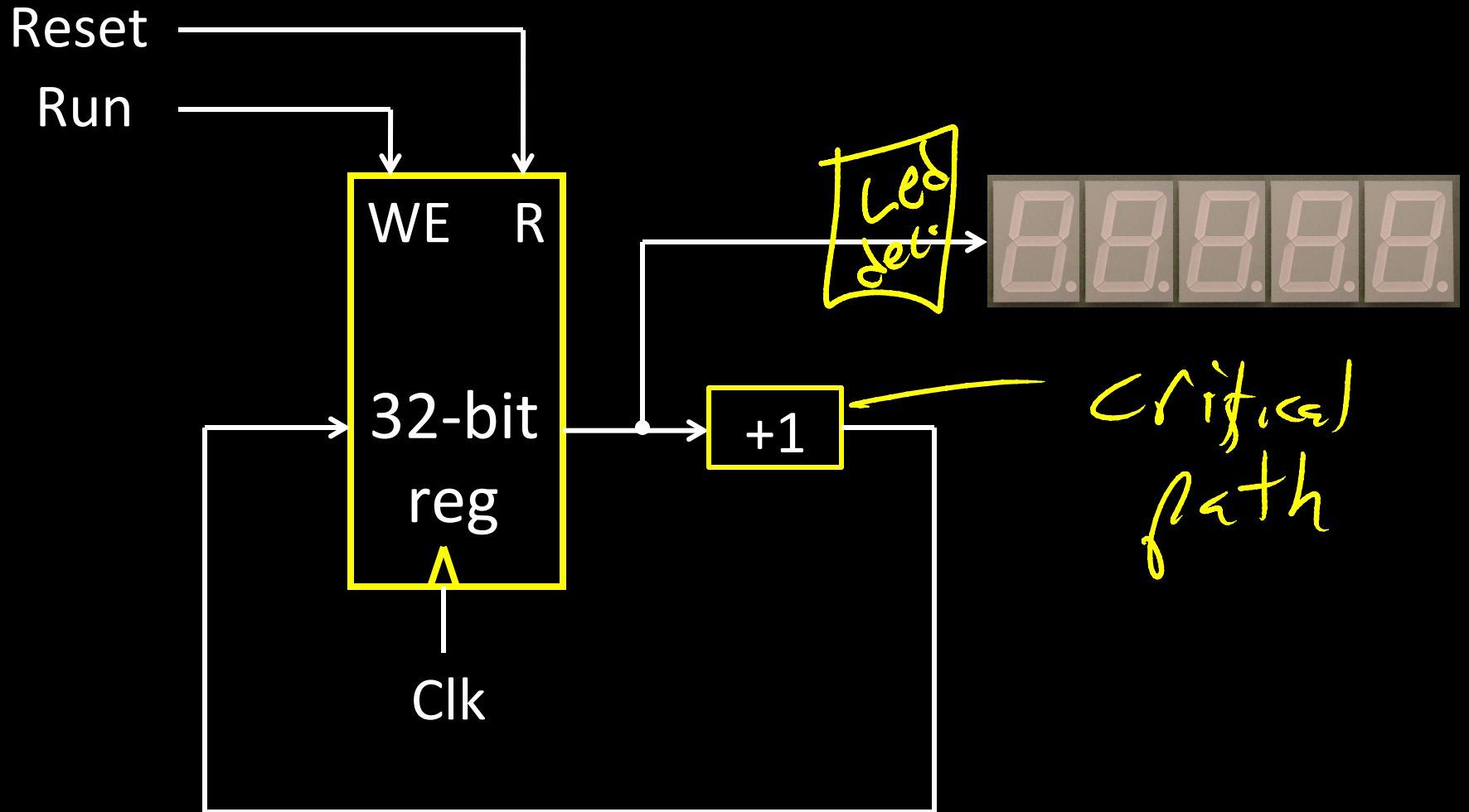
# Metastability and Asynchronous Inputs

Q: What happens if input ~~changes~~ changes near clock edge?

A: Google "Buridan's Principle" by Leslie Lamport



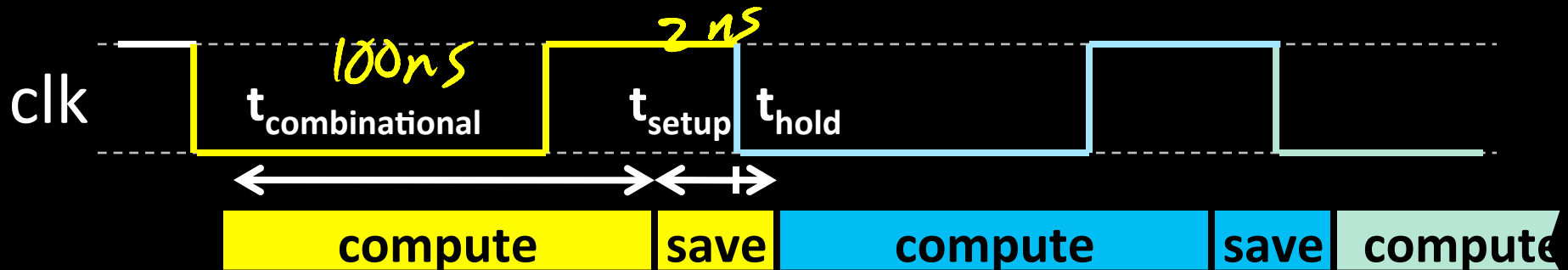
# An Example



# Clock Methodology

## Clock Methodology

- Negative edge, synchronous



– Signals must be stable near falling clock edge

- Positive edge synchronous
- Asynchronous, multiple clocks, . . .

# Finite State Machines

# Finite State Machines

---

An electronic machine which has

- external inputs
- externally visible outputs
- internal state

Output and next state depend on

- inputs
- current state

# Abstract Model of FSM

---

Machine is

$$M = ( S, I, O, \delta )$$

$S$ : Finite set of states

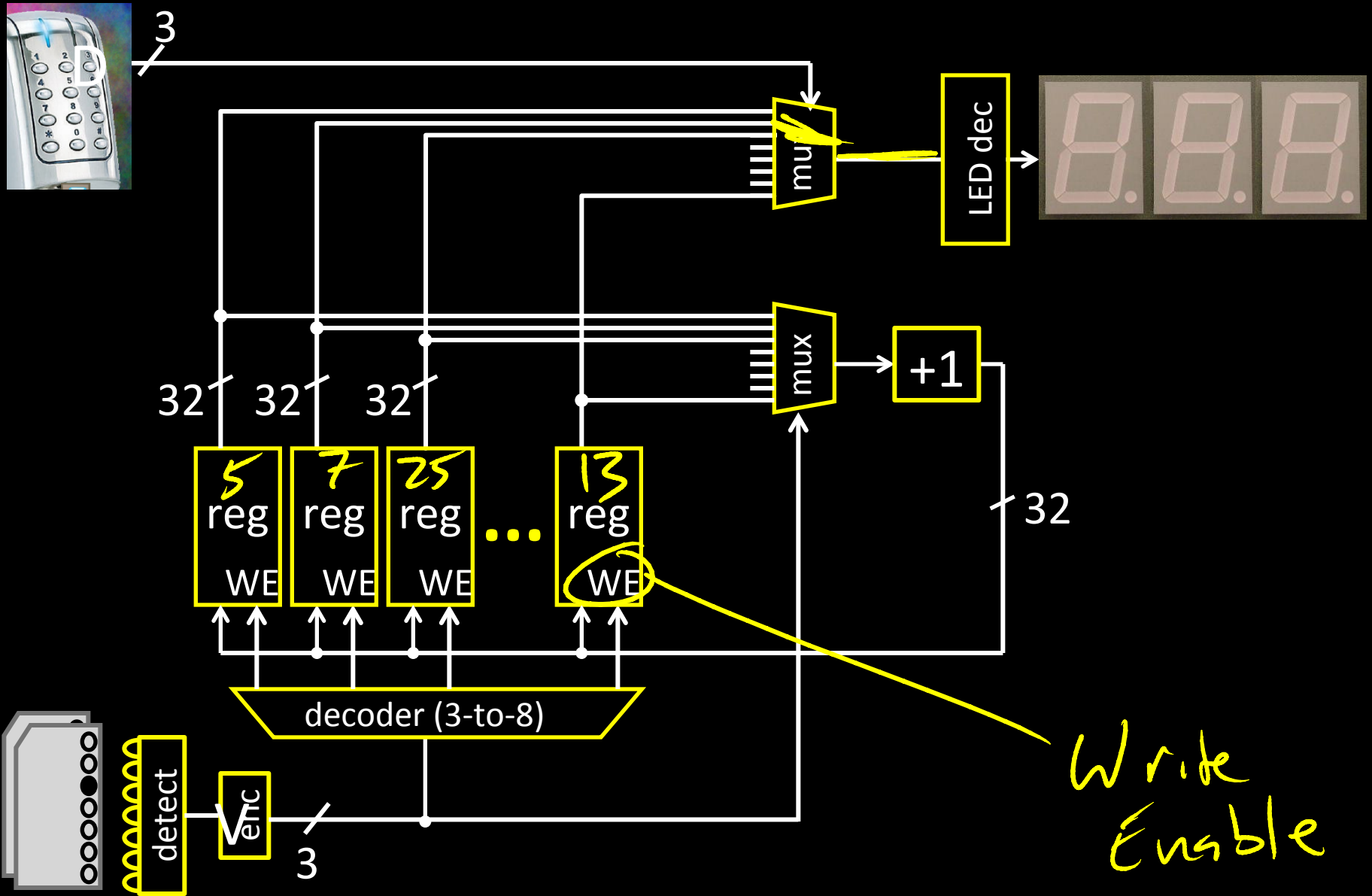
$I$ : Finite set of inputs

$O$ : Finite set of outputs

$\delta$ : State transition function

**Next state** depends on **present input** *and*  
**present state**

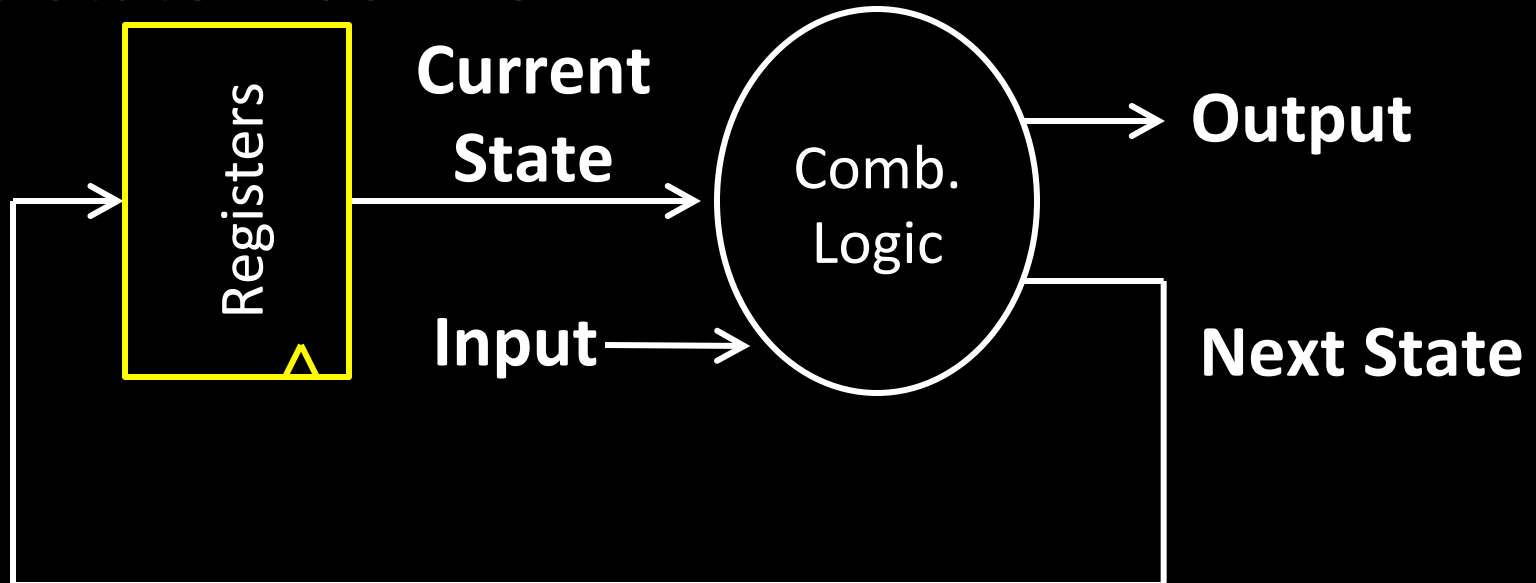
# Voting Machine





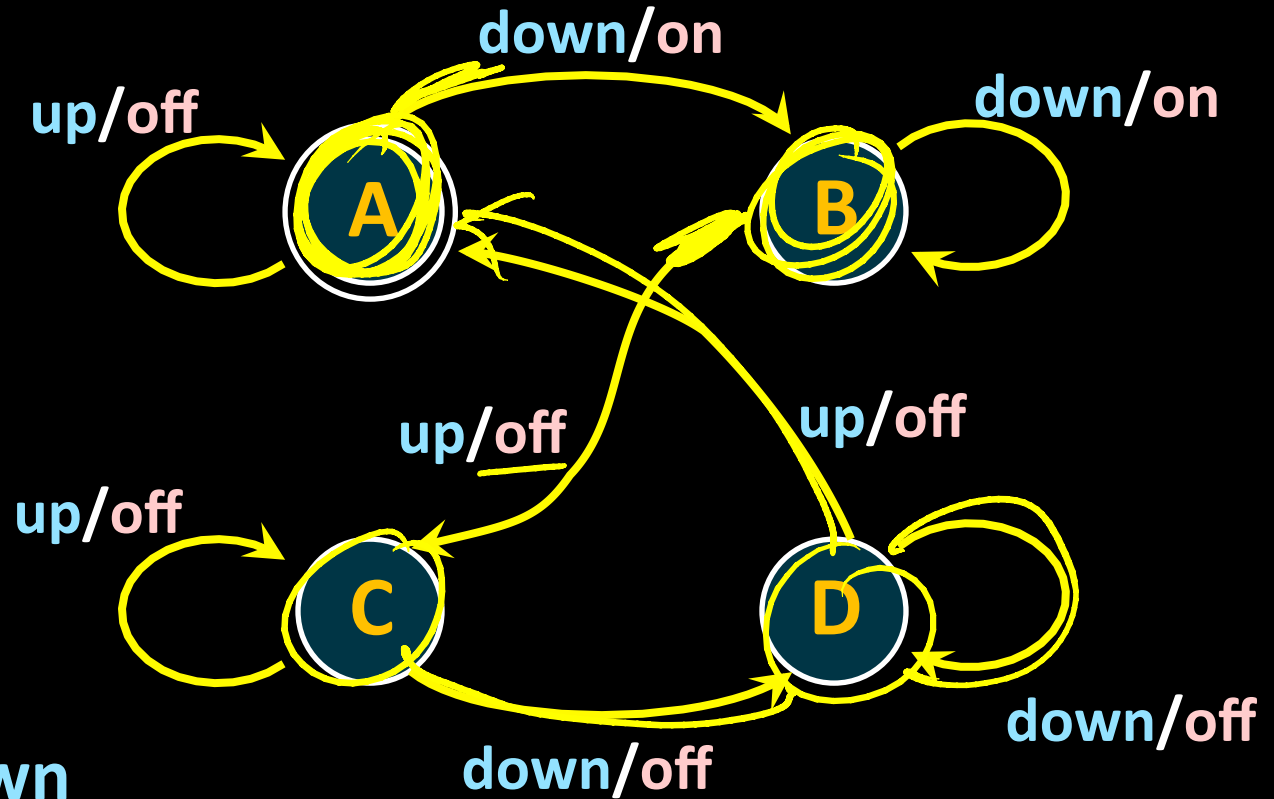
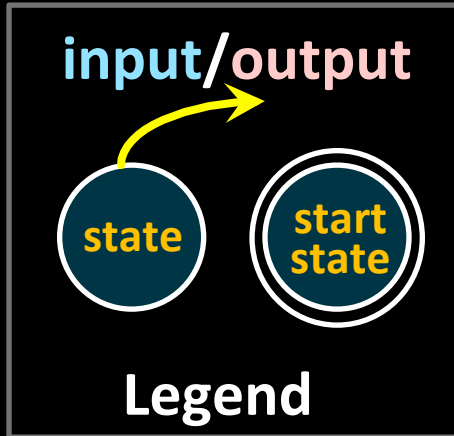
# Automata Model

## Finite State Machine



- inputs from external world
- outputs to external world
- internal state
- combinational logic

# FSM Example



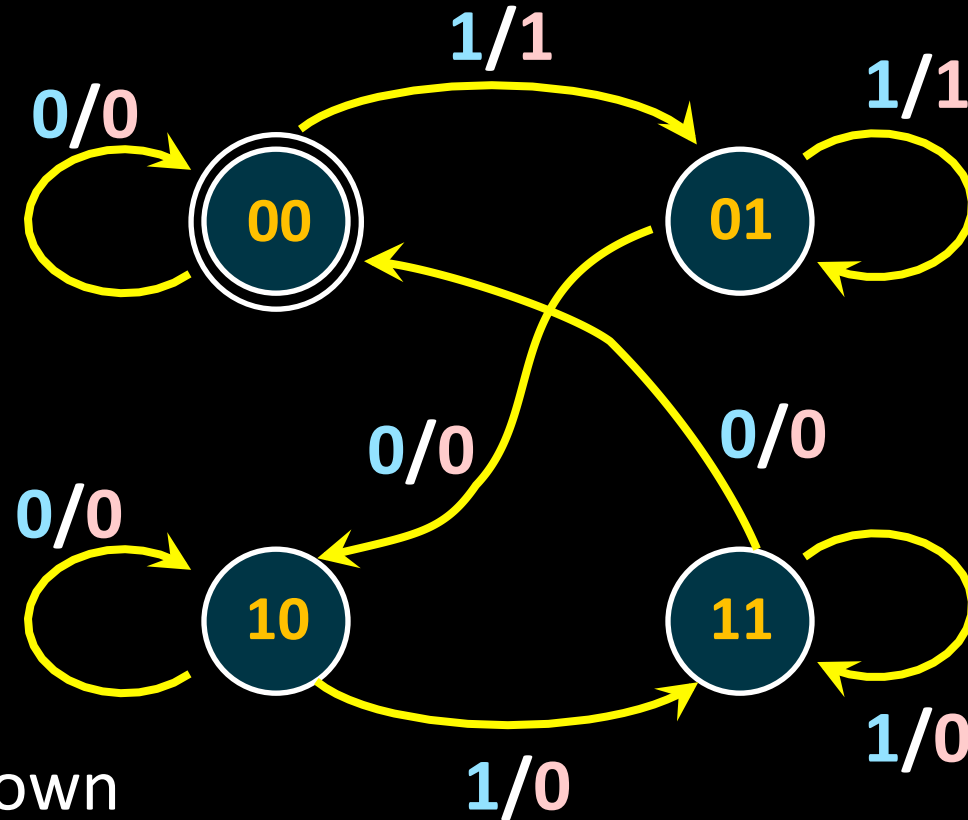
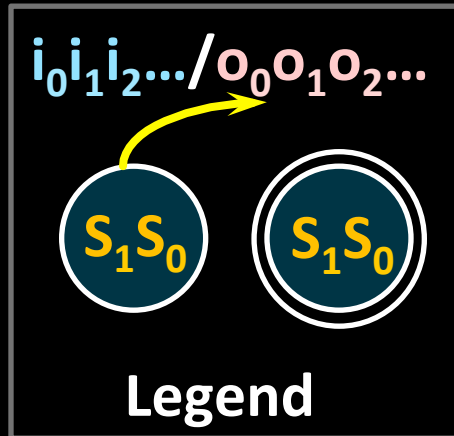
Input: **up** or **down**

Output: **on** or **off**

States: **A**, **B**, **C**, or **D**

In: D U D D U  
State: B C D D A  
Out: ON OFF OFF OFF OFF

# FSM Example Details



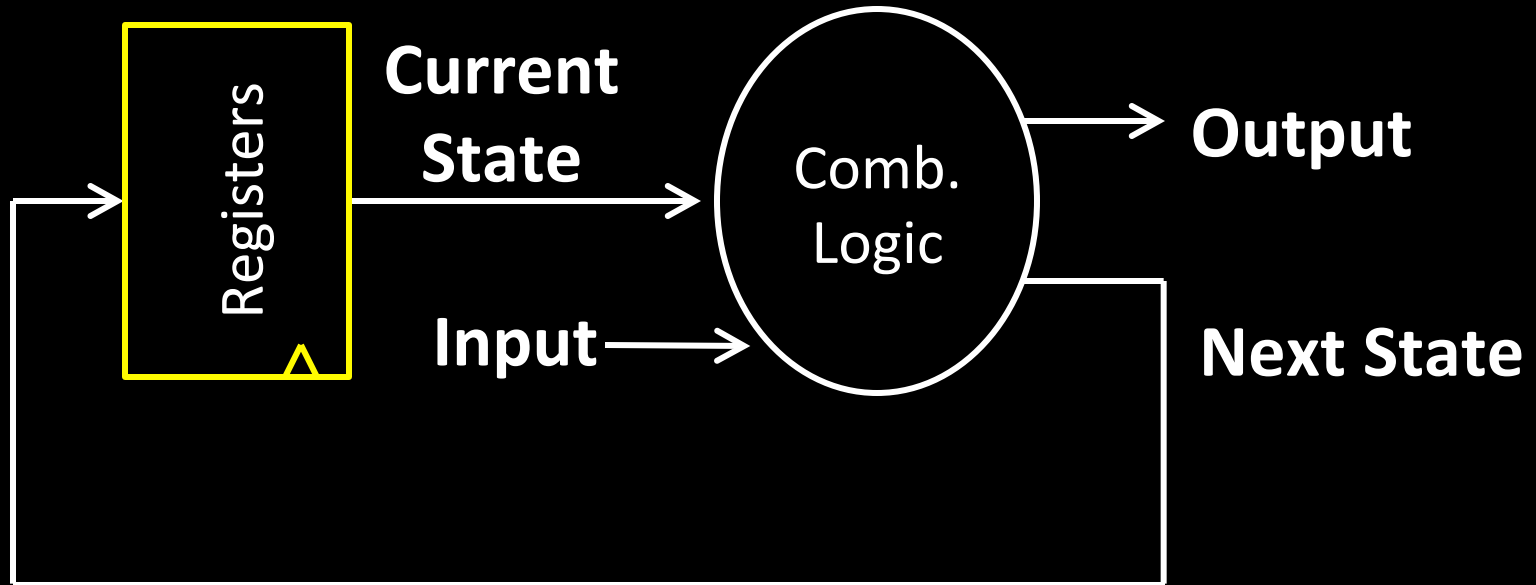
Input: **0**=up or **1**=down

Output: **1**=on or **1**=off

States: **00**=A, **01**=B, **10**=C, or **11**=D

# Mealy Machine

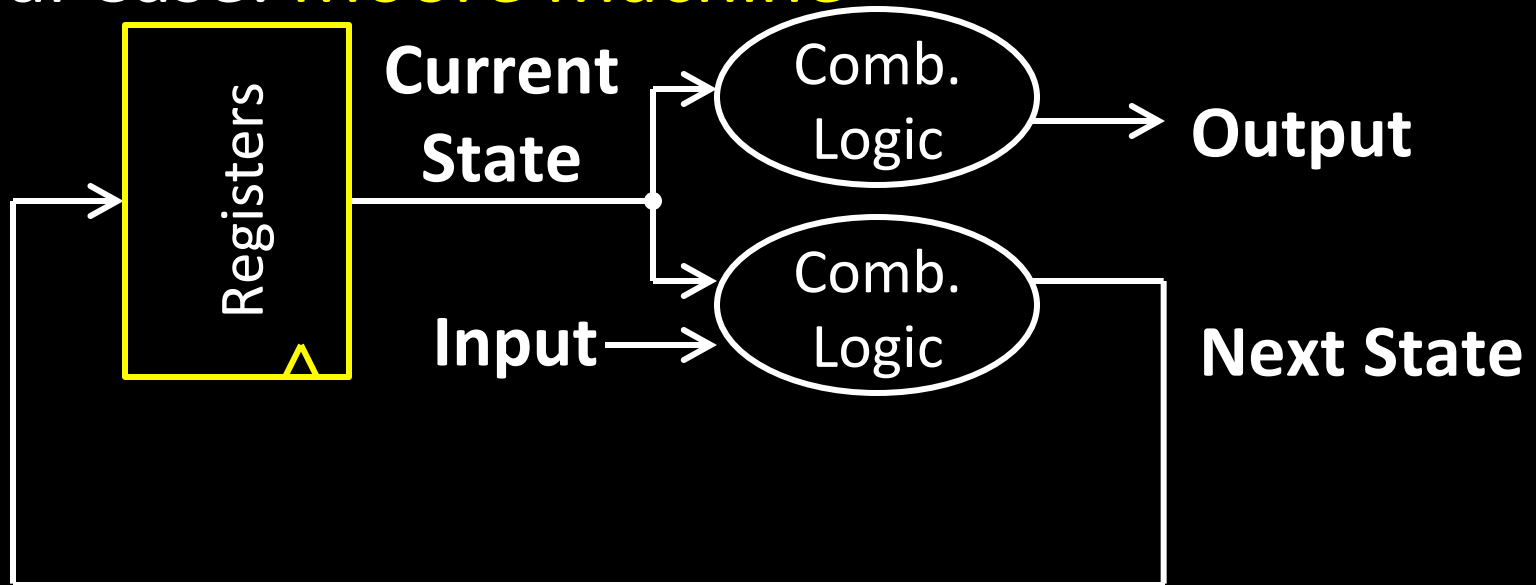
## General Case: Mealy Machine



Outputs and next state depend on both current state and input

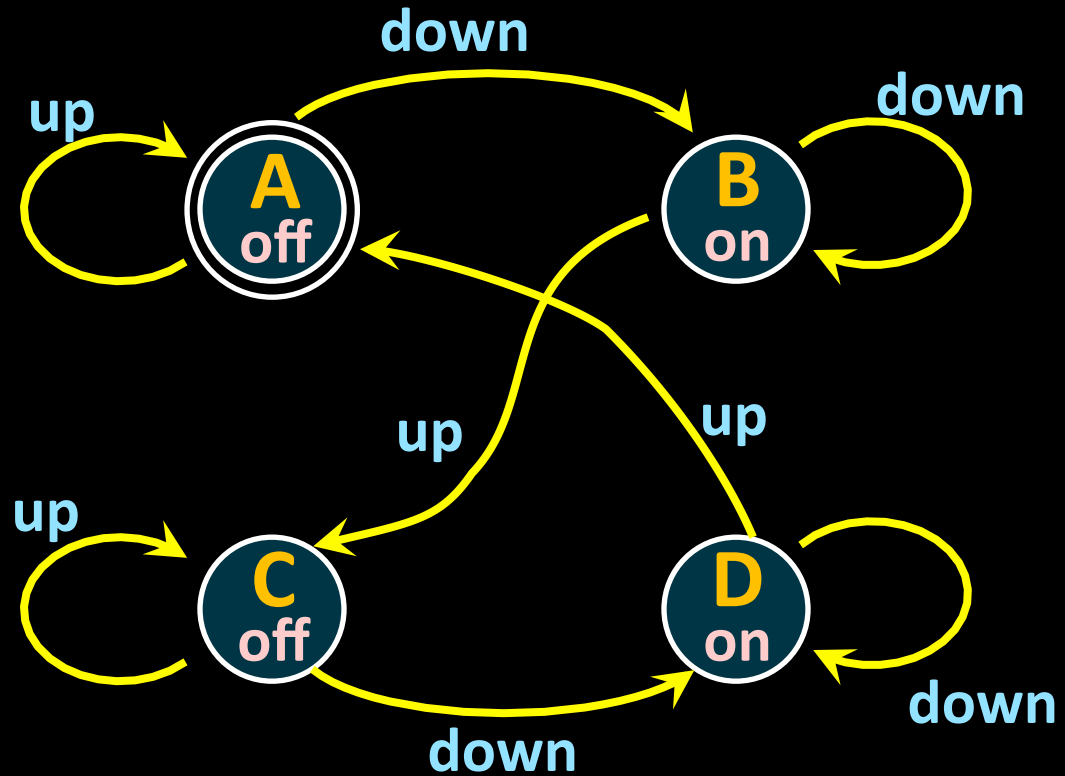
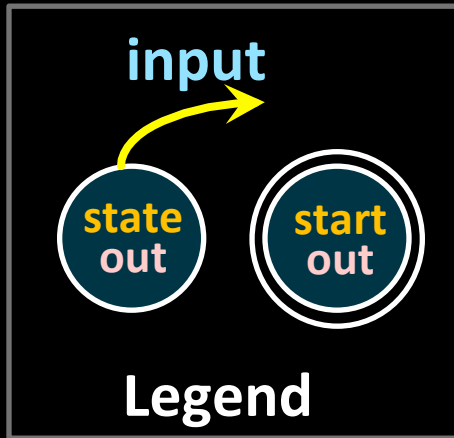
# Moore Machine

## Special Case: Moore Machine



Outputs depend only on current state

# Moore Machine Example



Input: **up** or **down**

Output: **on** or **off**

States: **A**, **B**, **C**, or **D**

# Digital Door Lock

---



## Digital Door Lock

### Inputs:

- keycodes from keypad
- clock

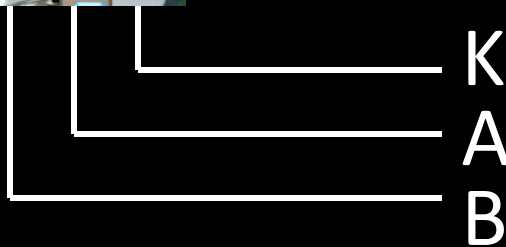
### Outputs:

- “unlock” signal
- display how many keys pressed so far

# Door Lock: Inputs

Assumptions:

- signals are synchronized to clock
- Password is B-A-B



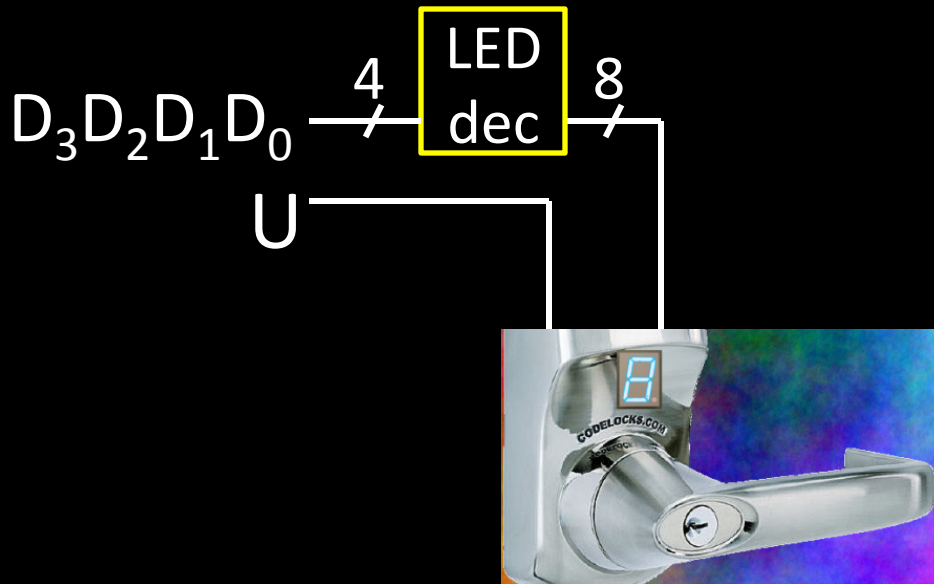
K	A	B	Meaning
0	0	0	$\emptyset$ (no key)
1	1	0	'A' pressed
1	0	1	'B' pressed



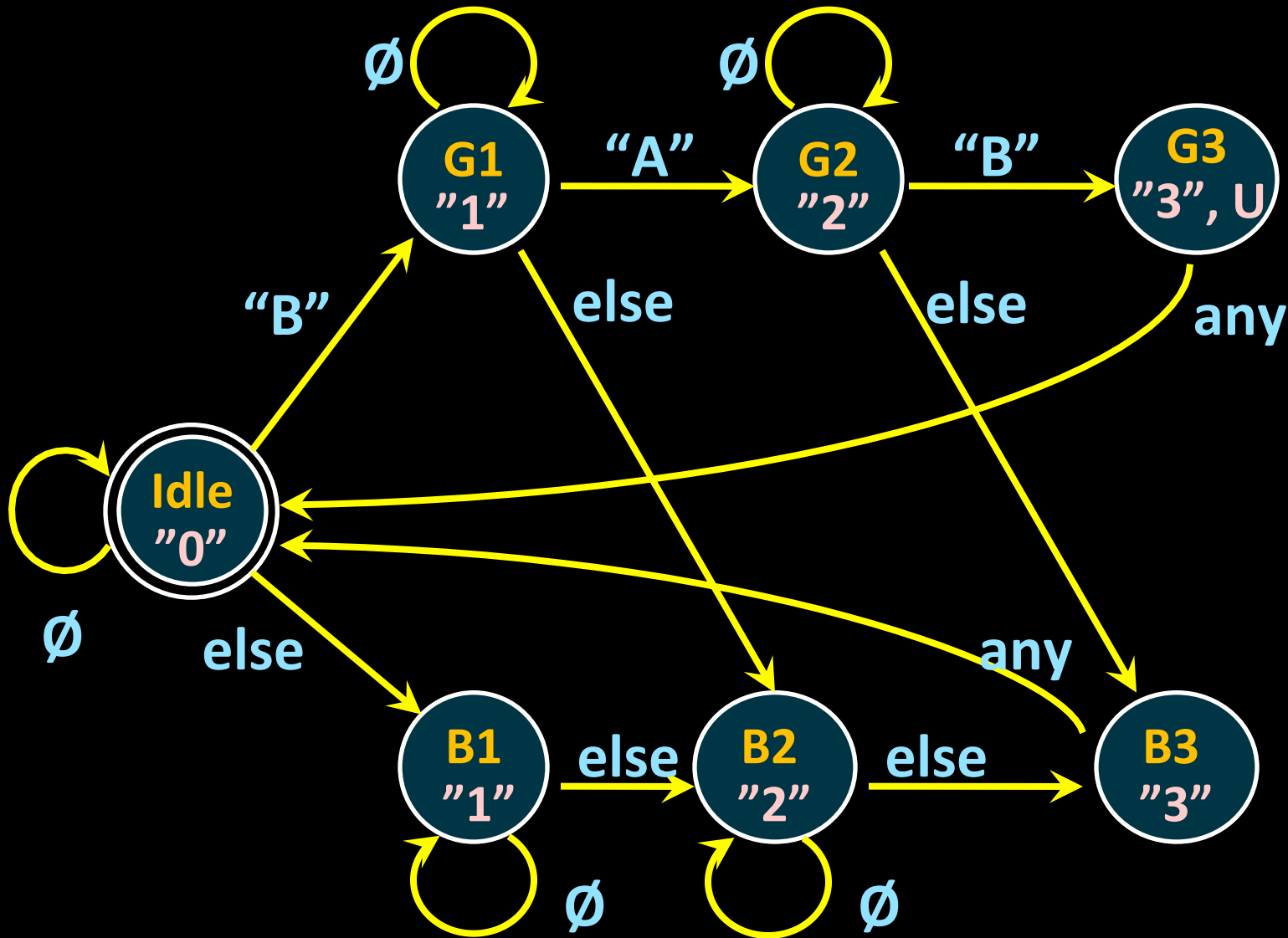
# Door Lock: Outputs

Assumptions:

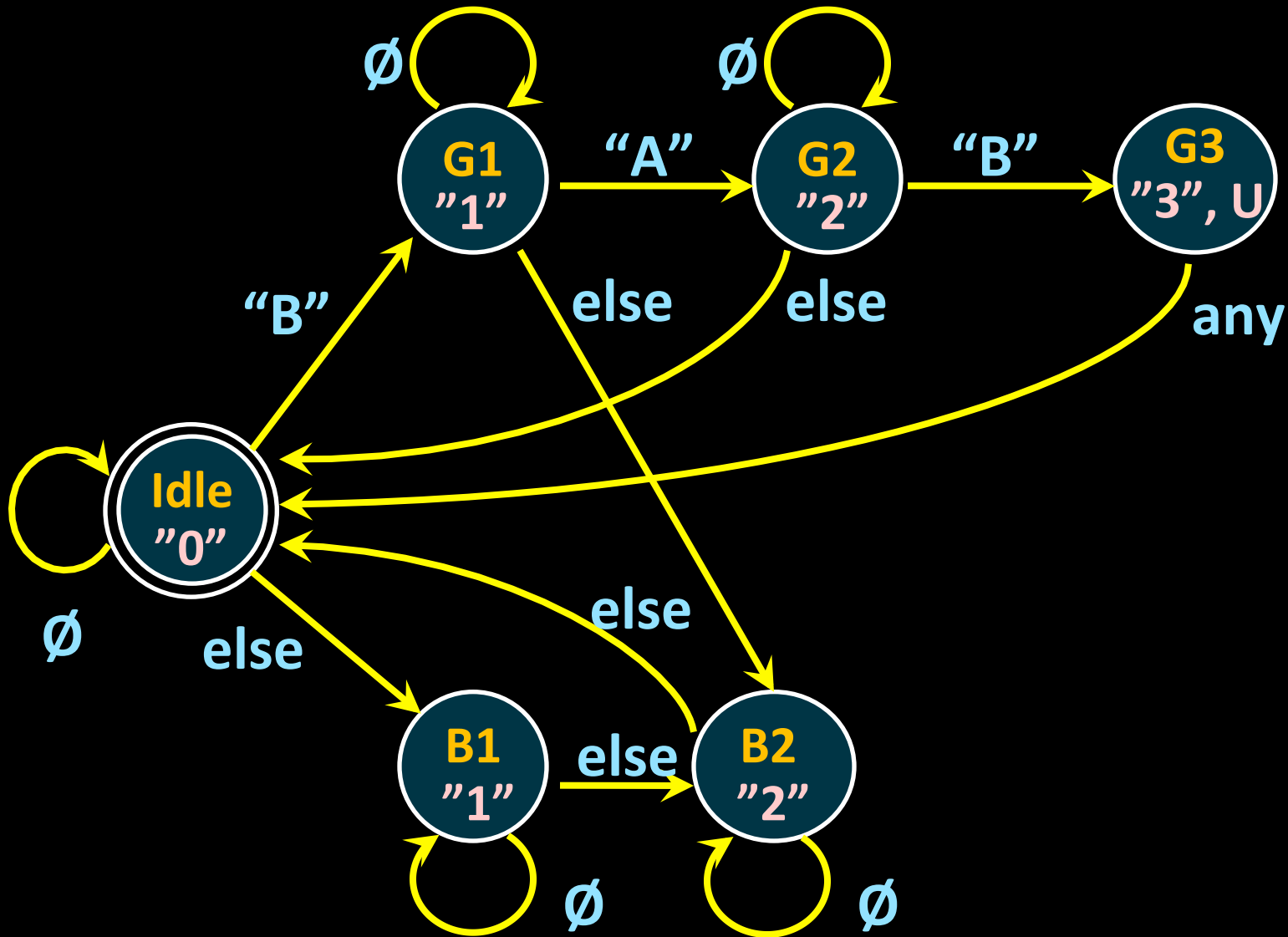
- High pulse on U unlocks door



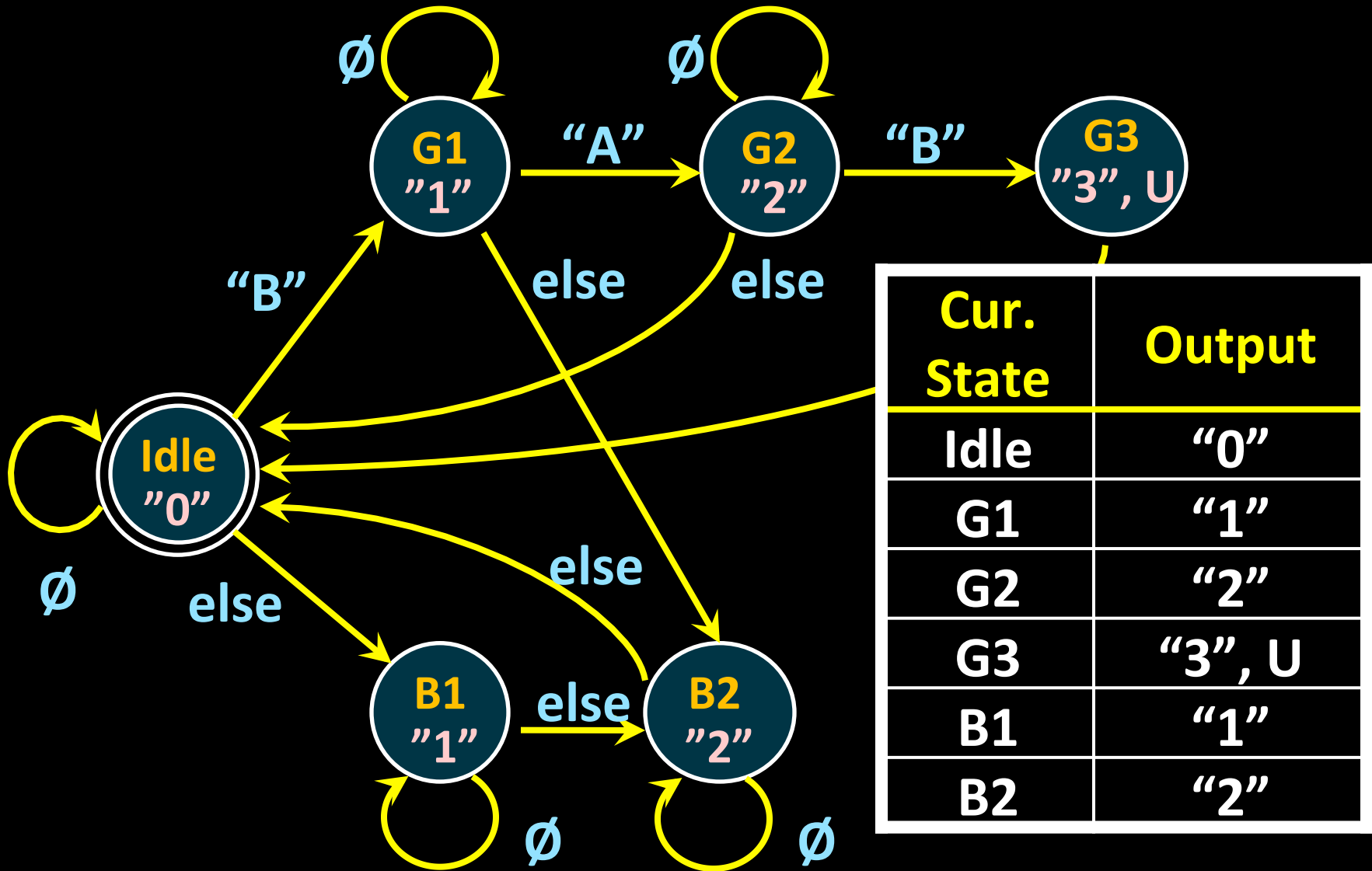
# Door Lock: Simplified State Diagram



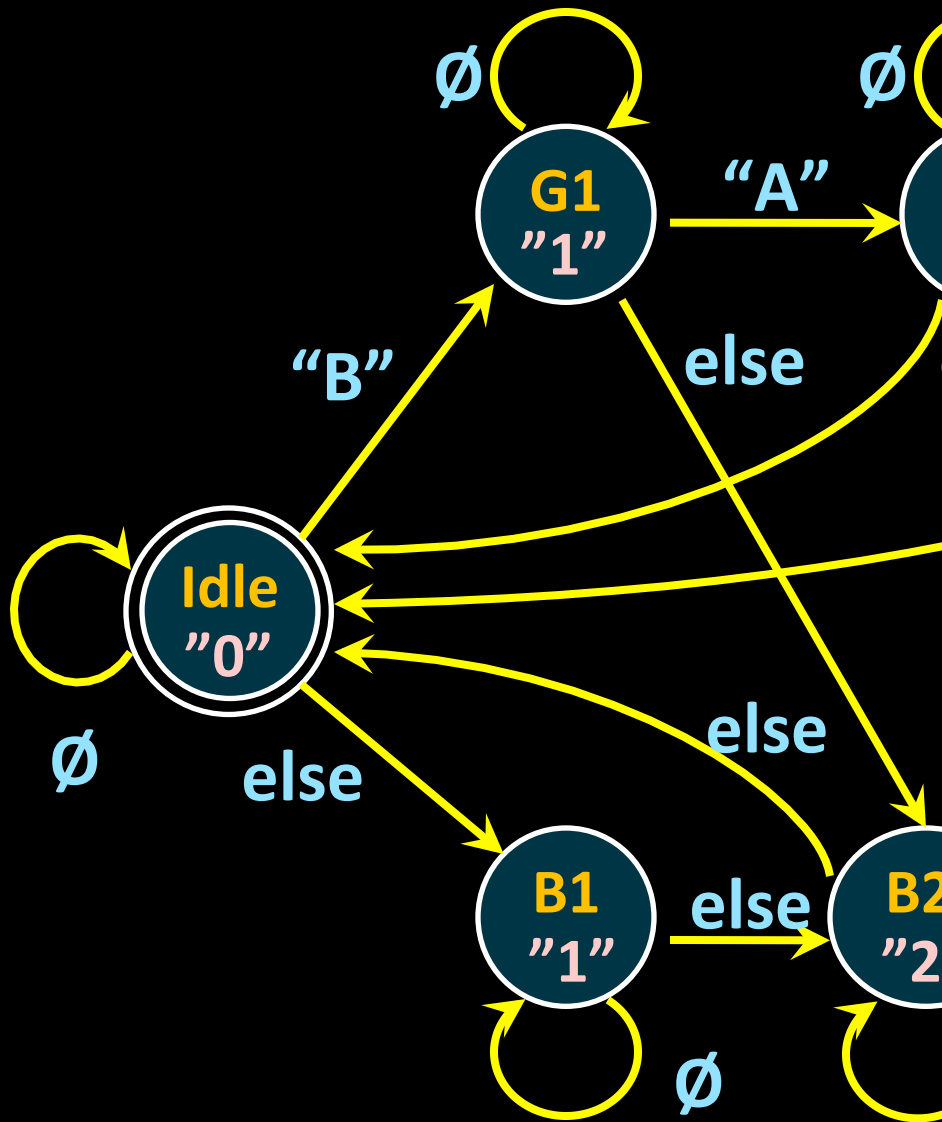
# Door Lock: Simplified State Diagram



# Door Lock: Simplified State Diagram



# Door Lock: Simplified State Diagram



Cur. State	Input	Next State
Idle	$\emptyset$	Idle
Idle	"B"	G1
Idle	"A"	B1
G1	$\emptyset$	G1
G1	"A"	G2
G1	"B"	B2
G2	$\emptyset$	B2
G2	"B"	G3
G2	"A"	Idle
G3	any	Idle
B1	$\emptyset$	B1
B1	K	B2
B2	$\emptyset$	B2
B2	K	Idle

# State Table Encoding

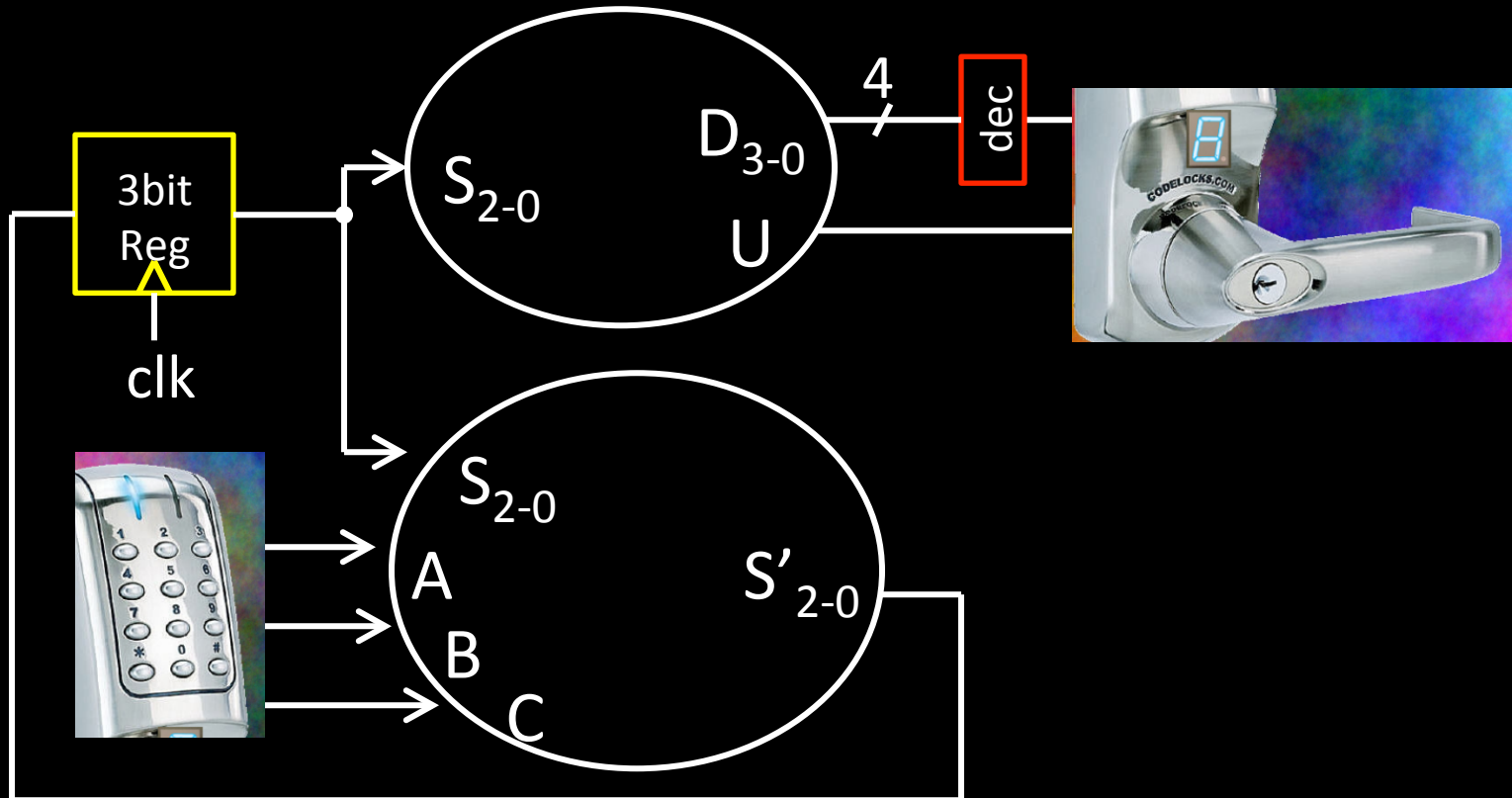
$S_2$	$S_1$	$S_0$	$D_3$	$D_2$	$D_1$	$D_0$	U
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	0	0	0	1	0
1	0	1	0	0	1	0	0

$D_3$

State	$S_2$	$S_1$	$S_0$
Idle	0	0	0
G1	0	0	1
G2	0	1	0
G3	0	1	1
B1	1	0	0
B2	1	0	1

$S_2$	$S_1$	$S_0$	K	A	B	$S'_2$	$S'_1$	$S'_0$
0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	1
0	0	0	1	1	0	1	0	0
0	0	1	0	0	0	0	0	1
0	0	1	1	1	0	0	1	0
0	0	1	1	0	1	1	0	1
0	1	0	0	0	0	0	1	0
0	1	0	1	0	1	0	1	1
0	1	0	1	1	0	0	0	0
0	1	1	x	x	x	0	0	0
1	0	0	0	0	0	1	0	0
1	0	0	1	x	x	1	0	1
1	0	1	0	0	0	1	0	1
1	0	1	1	x	x	0	0	0

# Door Lock: Implementation



## Strategy:

- (1) Draw a state diagram (e.g. Moore Machine)
- (2) Write output and next-state tables
- (3) Encode states, inputs, and outputs as bits
- (4) Determine logic equations for next state and outputs

# Summary

---

We can now build interesting devices with sensors

- Using combinational logic

We can also store data values

- Stateful circuit elements (D Flip Flops, Registers, ...)
- Clock to synchronize state changes
- But be wary of asynchronous (un-clocked) inputs
- State Machines or Ad-Hoc Circuits