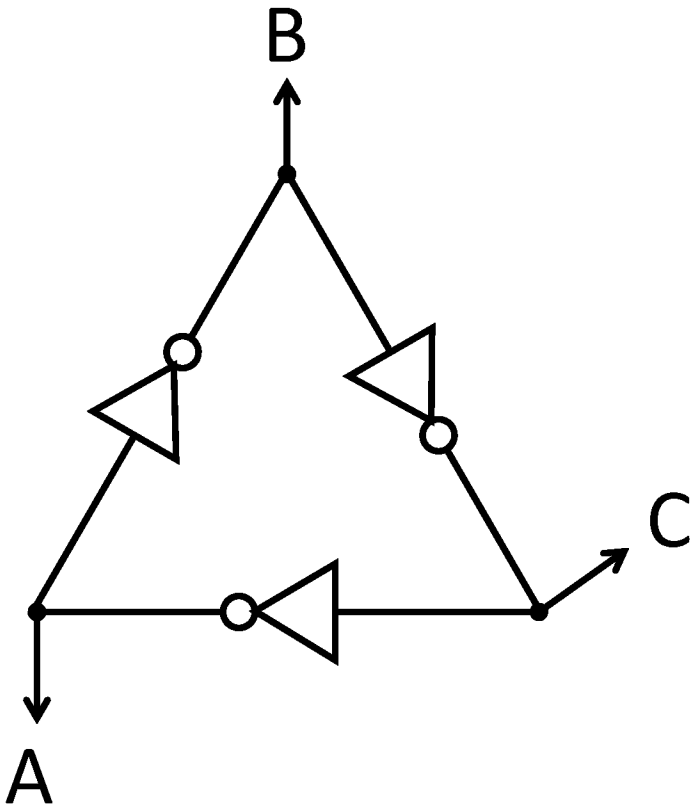
State & Finite State Machines

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

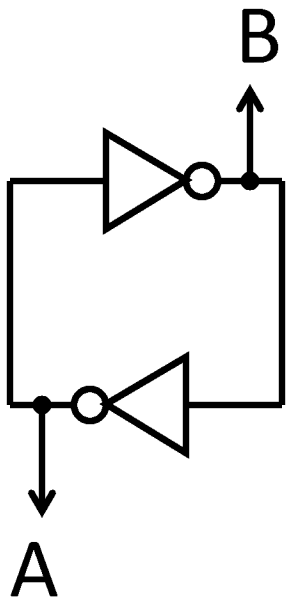
See: P&H Appendix C.7. C.8, C.10, C.11

Prelim 1: 3/18/2010 (evening)

Prelim 2: 4/27/2010 (evening)

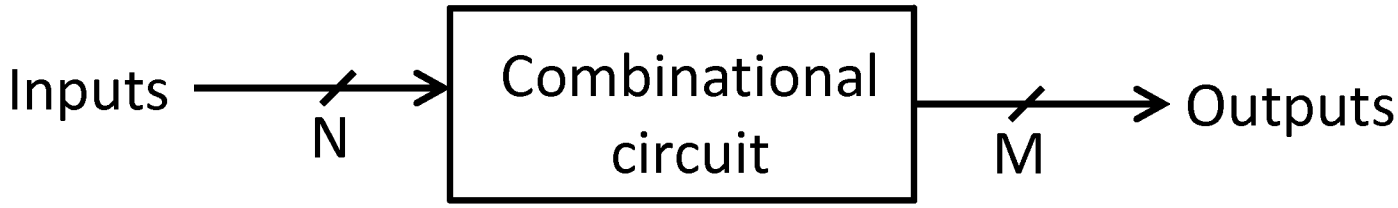

B
C
A

Stable and unstable equilibria?



So far:

- Current output depends only on current input (no internal state)

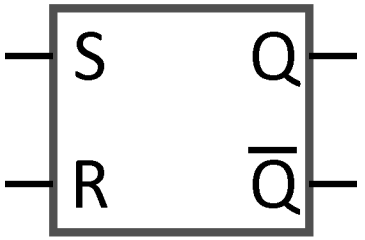


Need a way to record data

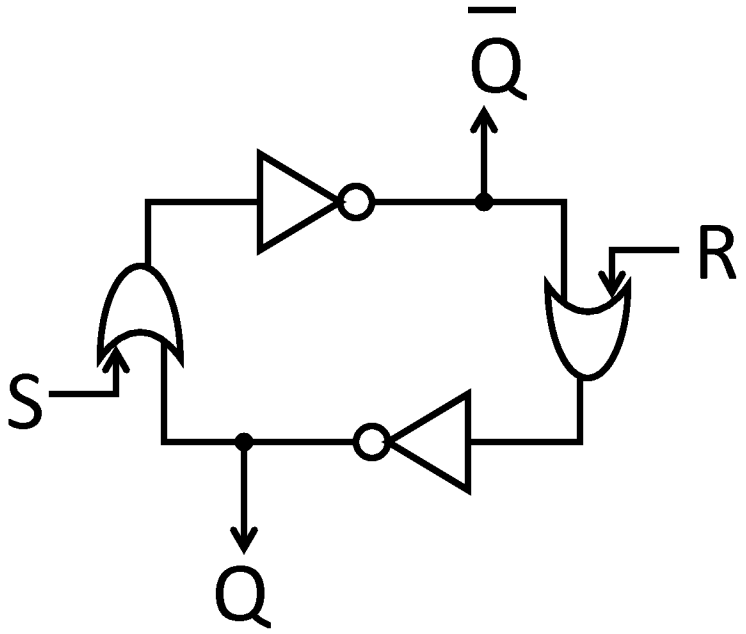
- … a way to build stateful circuits
- … a state-holding device

Set-Reset (SR) Latch

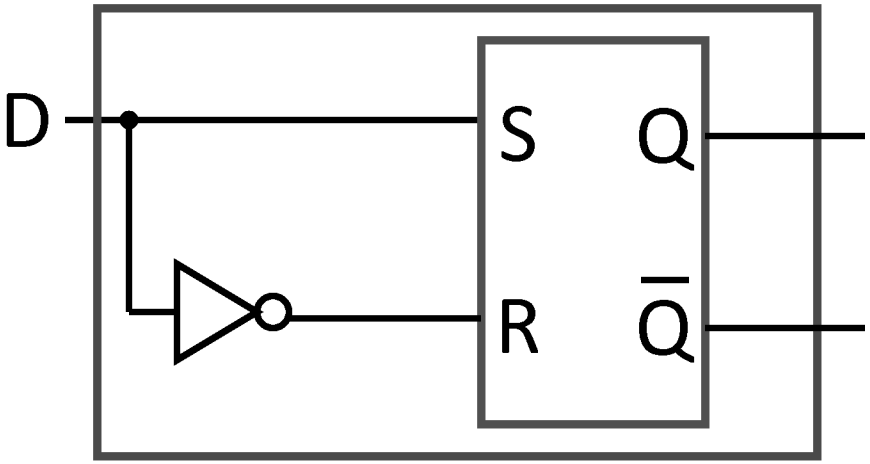
Stores a value Q and its complement Q



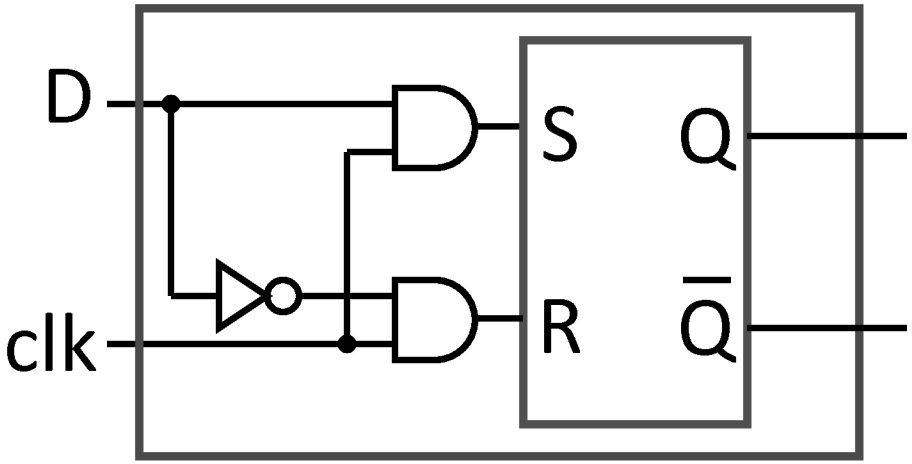
S	R	Q	$\overline{Q}$
0	0		
0	1		
1	0		
1	1		



Data (D) Latch



D	Q	$\overline{Q}$
0		
1		



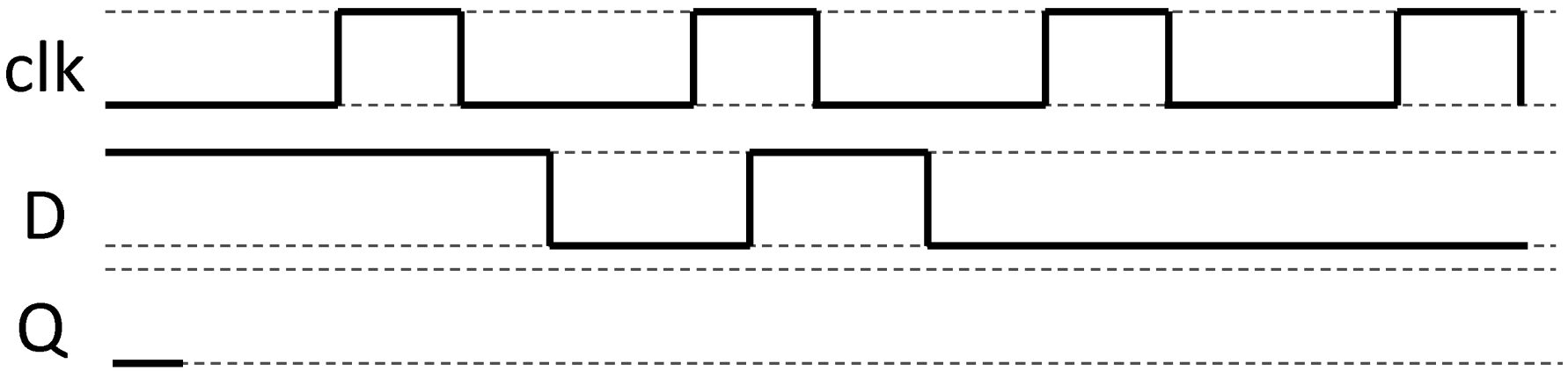
Level Sensitive D Latch

Clock high:

set/reset (according to D)

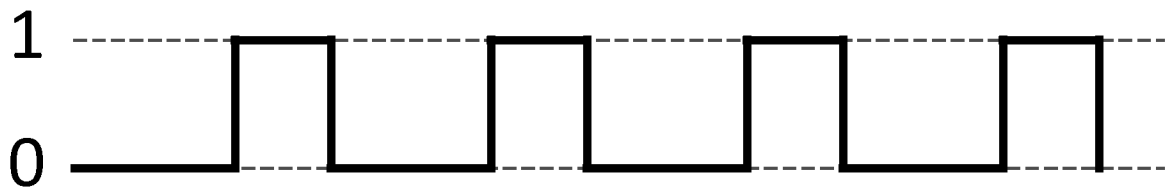
Clock low:

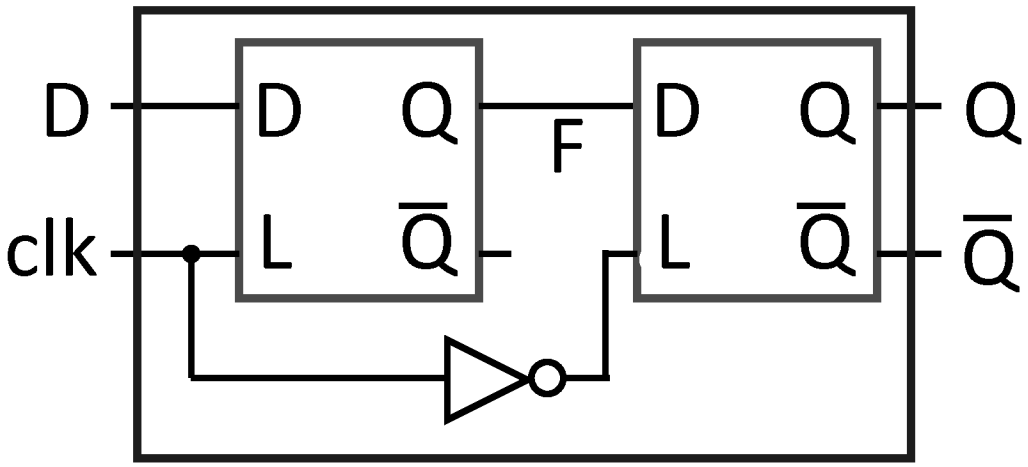
keep state (ignore D)



Clock helps coordinate state changes

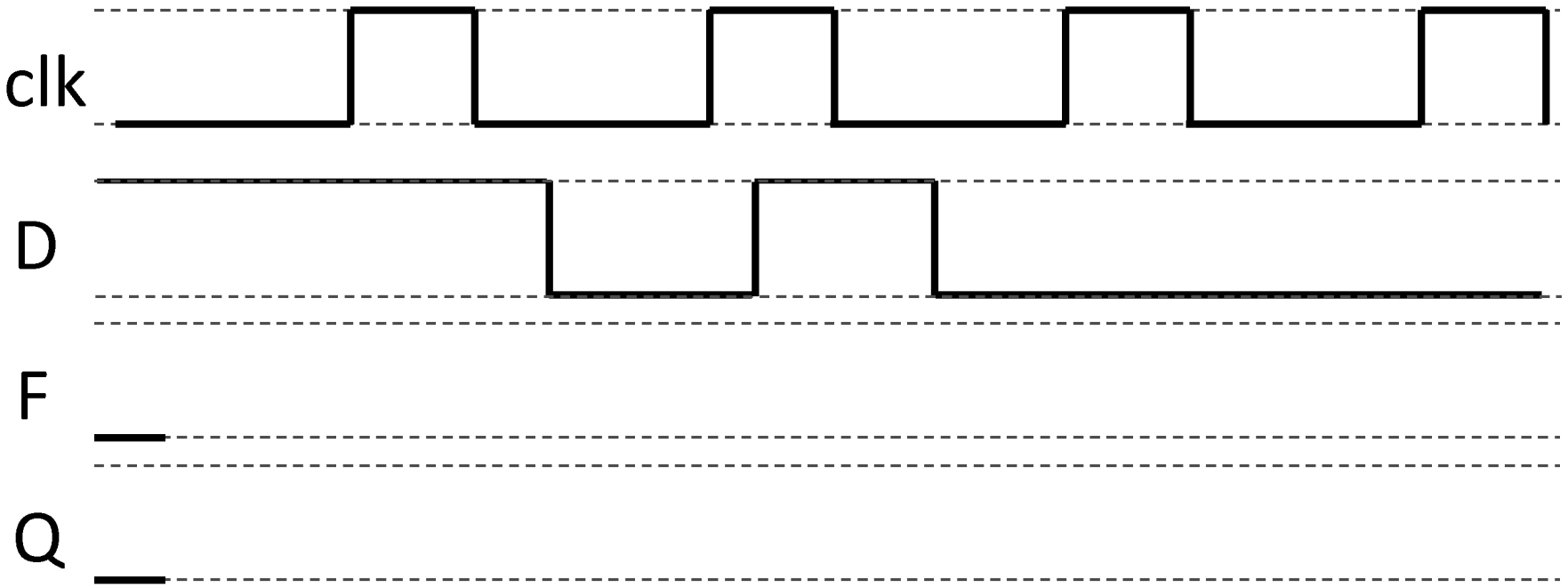
- Usually generated by an oscillating crystal
- Fixed period; frequency = 1/period





D Flip-Flop

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



Level sensitive

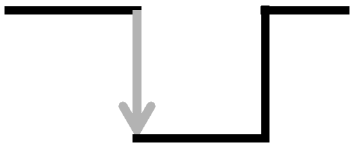
- State changes when clock is high (or low)

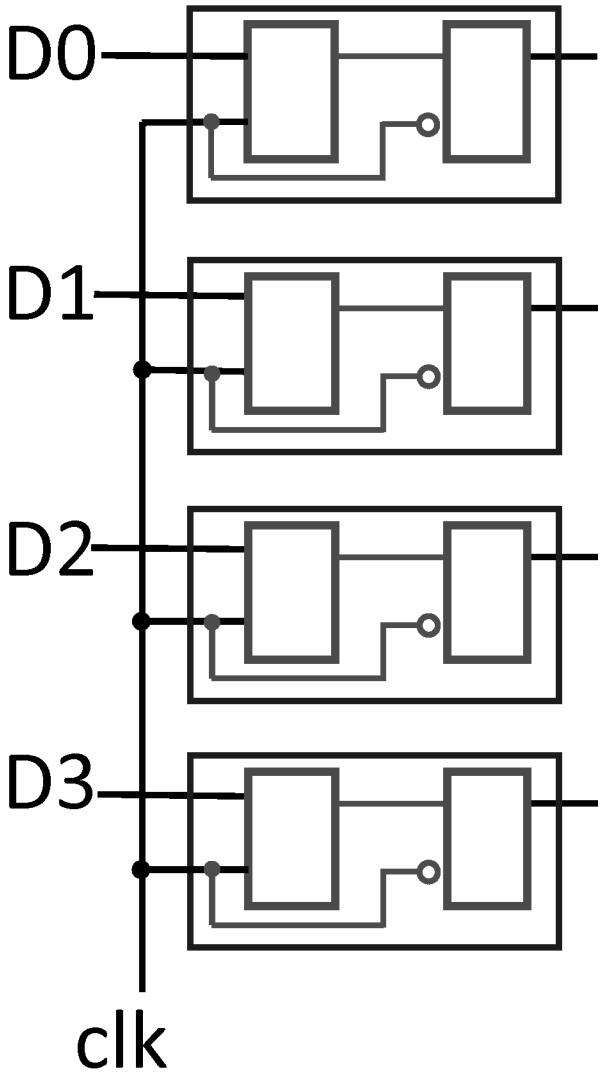
Edge triggered

- State changes at clock edge

positive edge-triggered

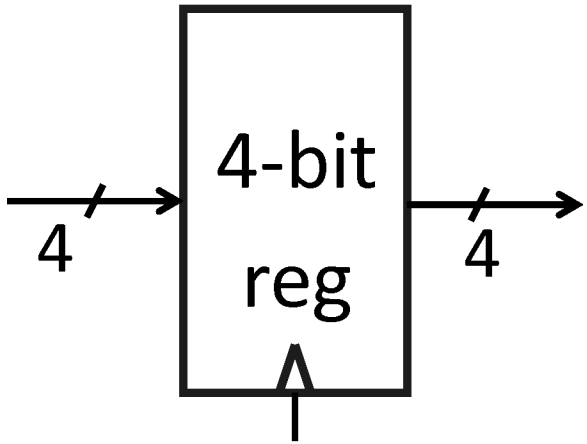
negative edge-triggered

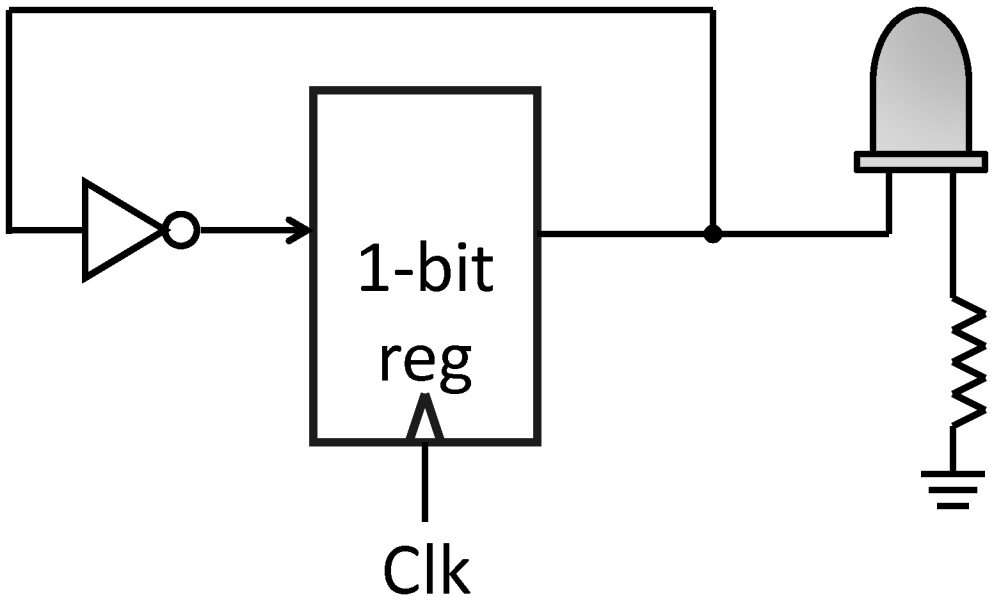




Register

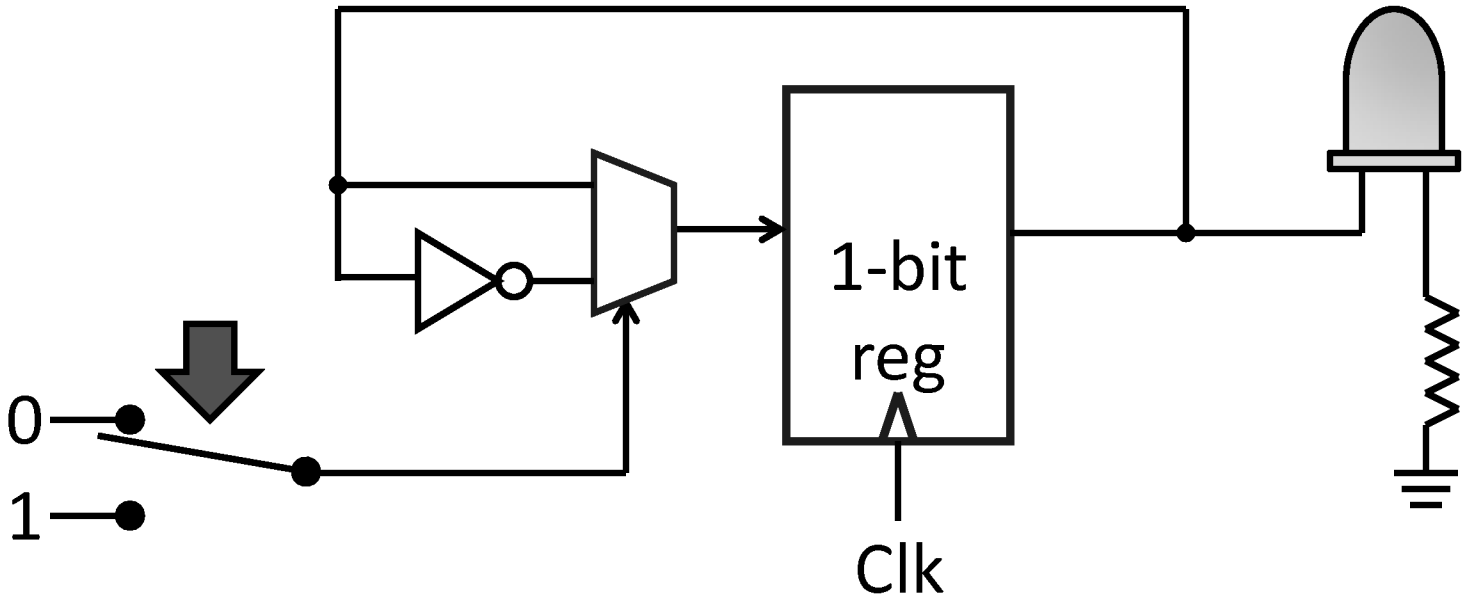
- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, …

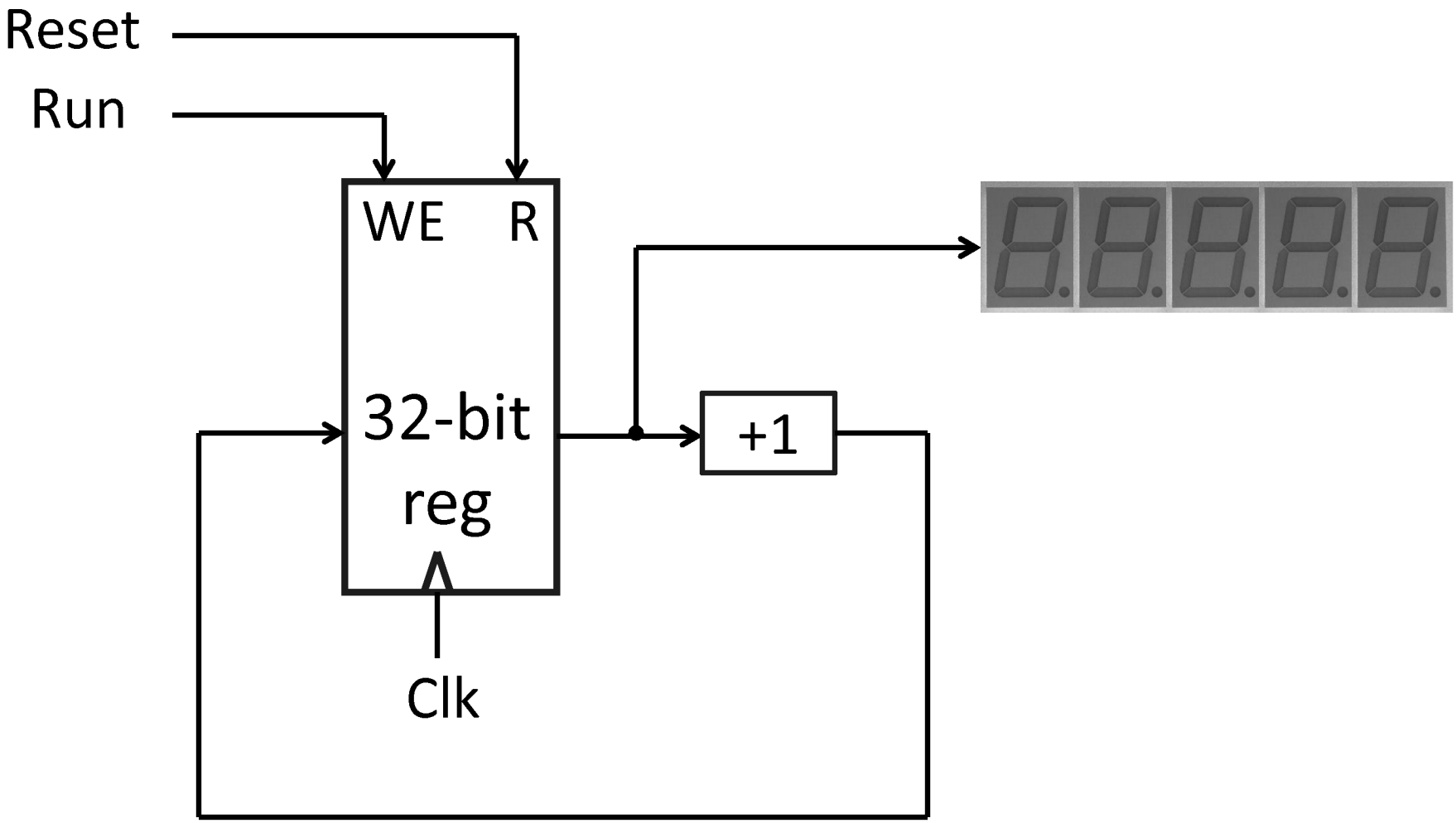



1-bit
reg
Clk

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

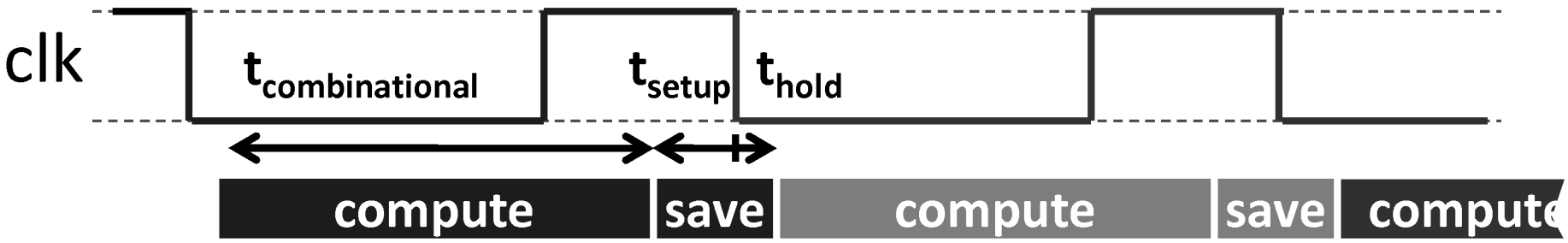
A: Google “Buridan’s Principle” by Leslie Lamport




Reset
Run
WE
R
32-bit
reg
Clk
+1

Clock Methodology

- Negative edge, synchronous



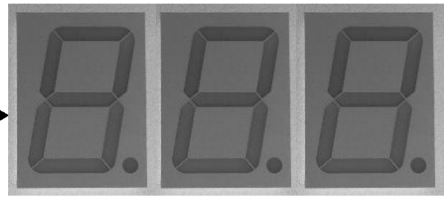
  - Signals must be stable near falling clock edge

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

3

mux

LED dec



mux

+1

32

32

32

32

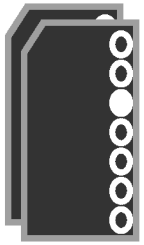
reg WE

reg WE

reg WE

reg WE

decoder (3-to-8)

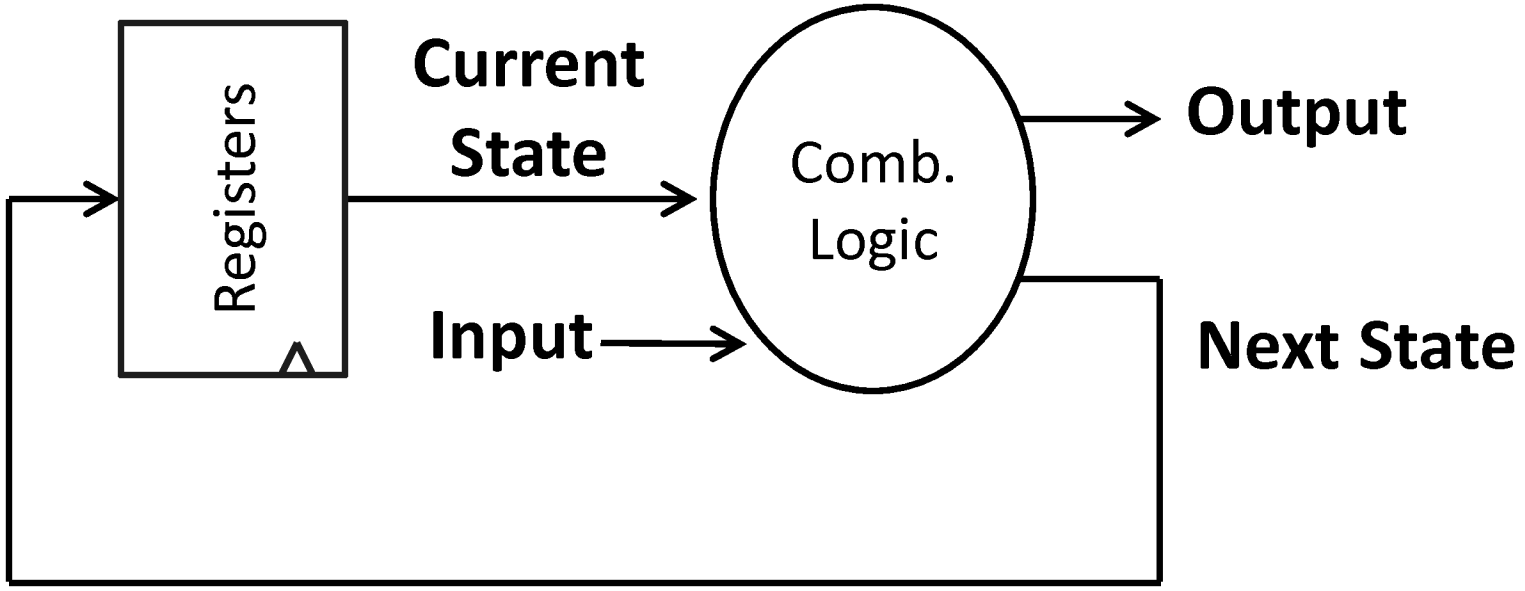


detect

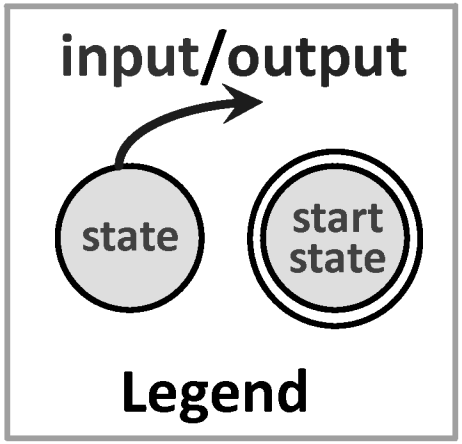
enc

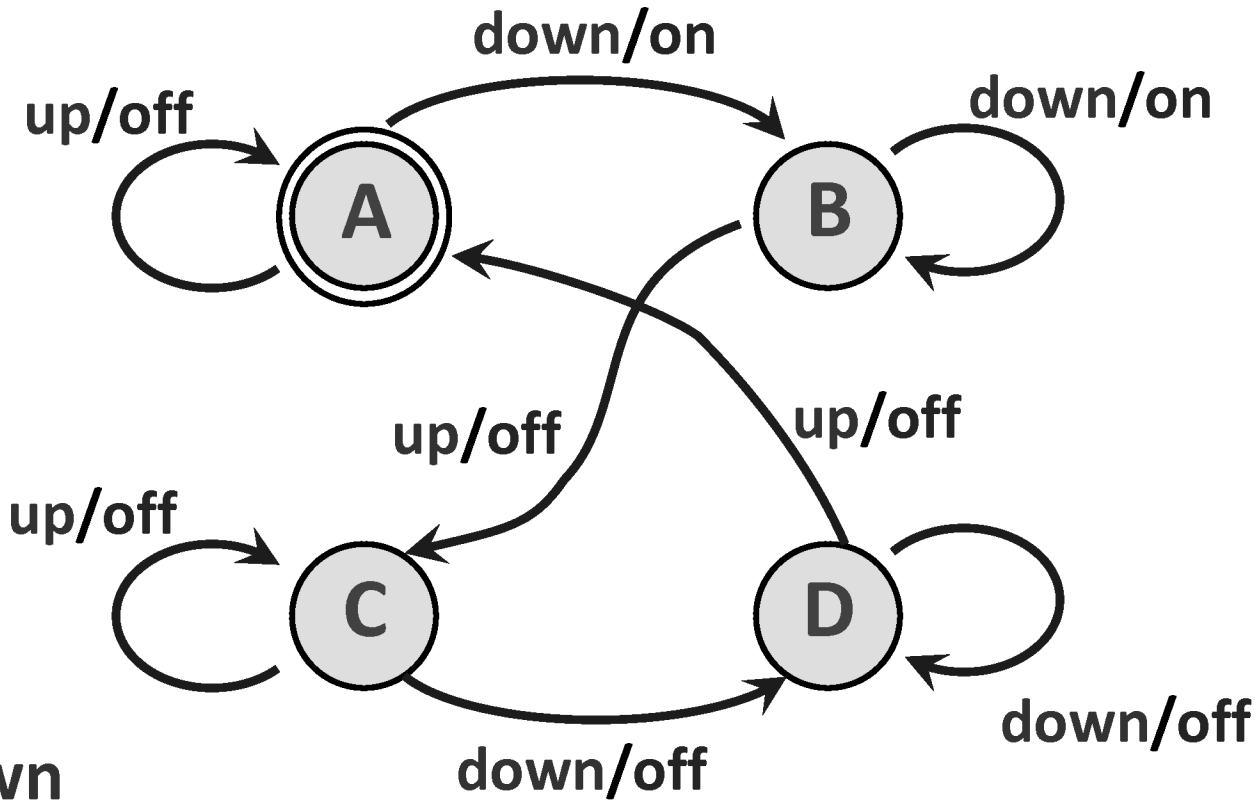
3

Finite State Machine



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

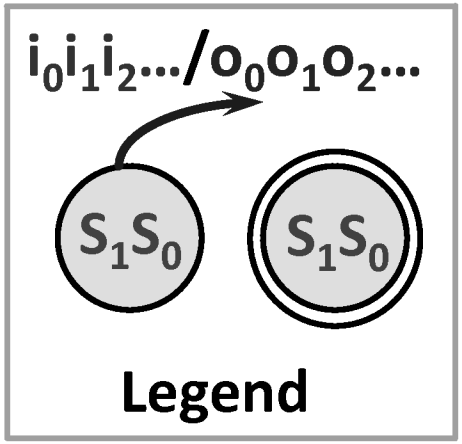


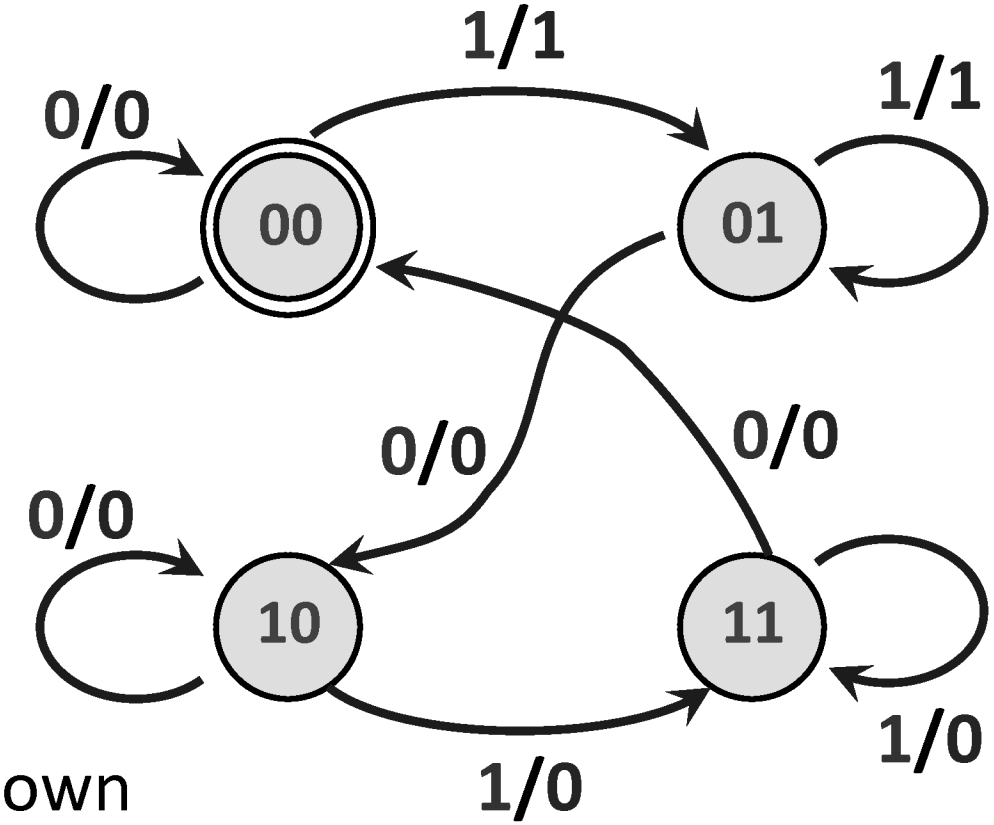


Input: **up** or **down**

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

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



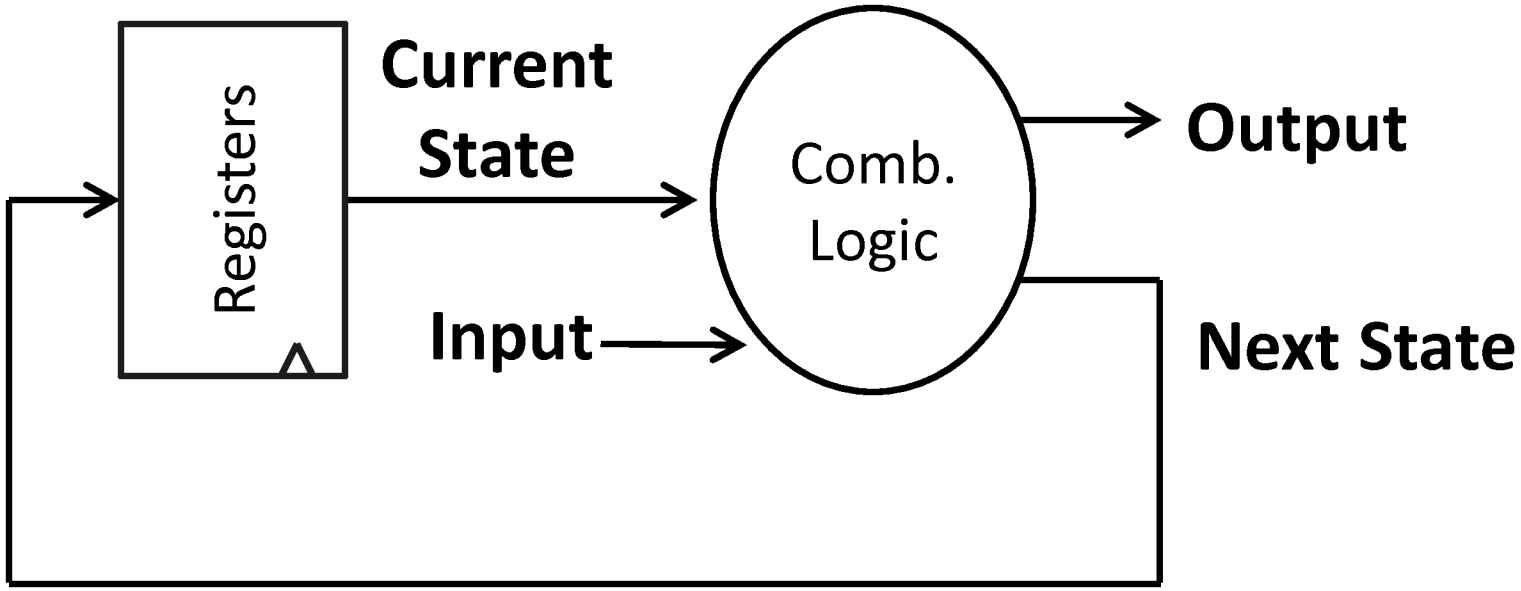


Input: **0**=up or **1**=down

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

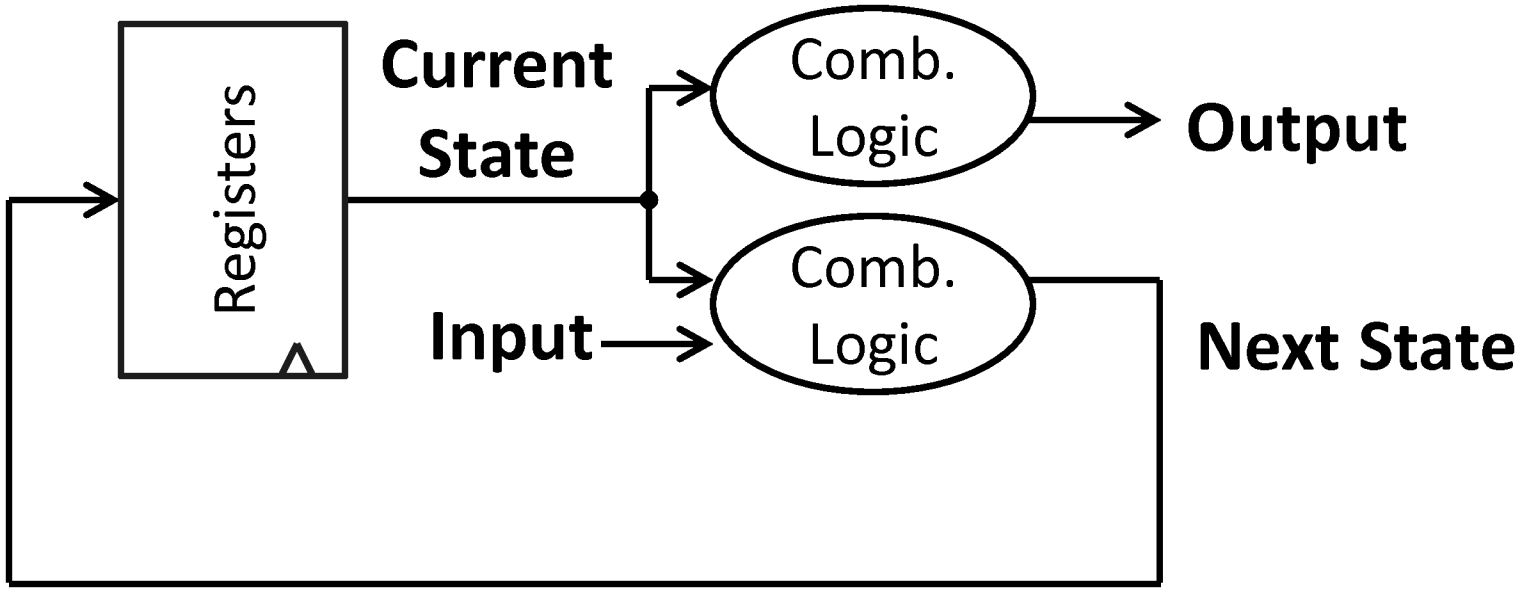
States: **00**=A, **01**=B, **10**=C, or **11**=D

General Case: Mealy Machine

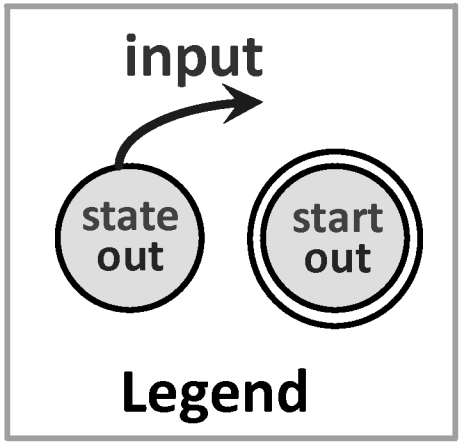


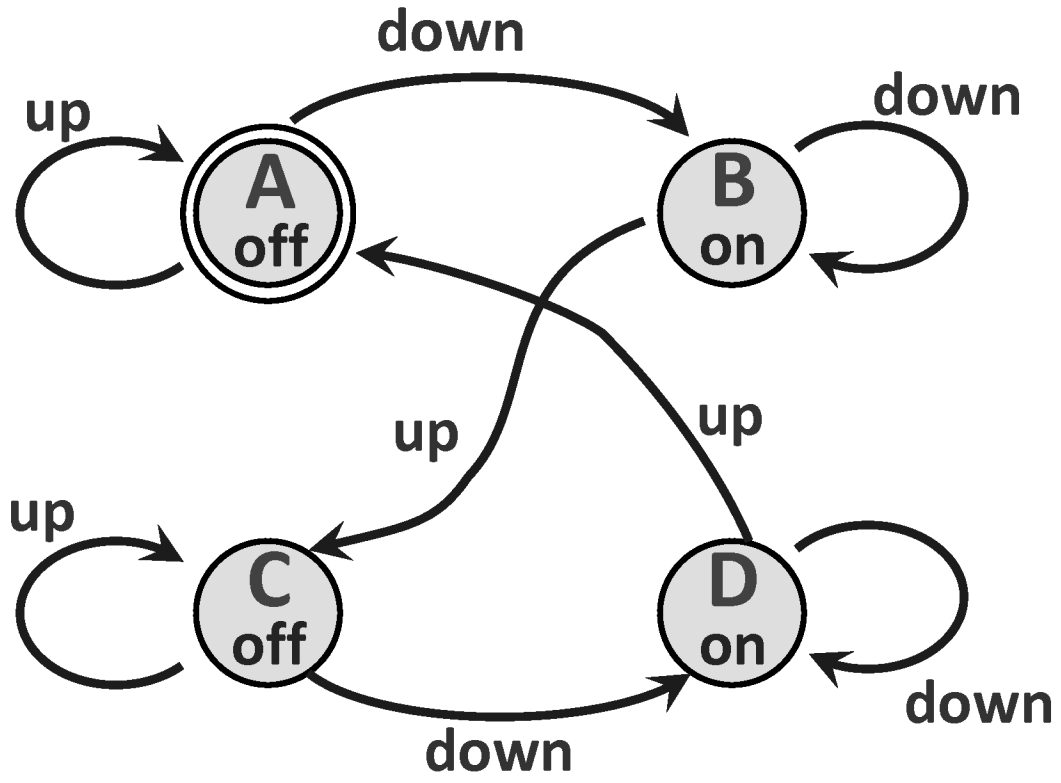
Outputs and next state depend on both current state and input

Special Case: Moore Machine



Outputs depend only on current state





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

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

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

Digital Door Lock

Inputs:

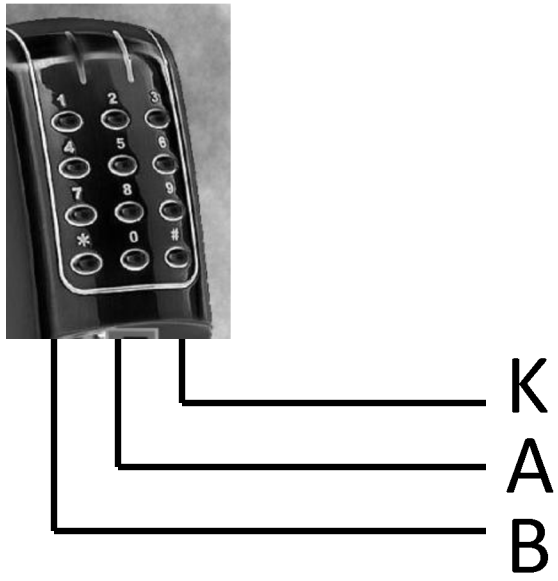
- keycodes from keypad
- clock

Outputs:

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

Assumptions:

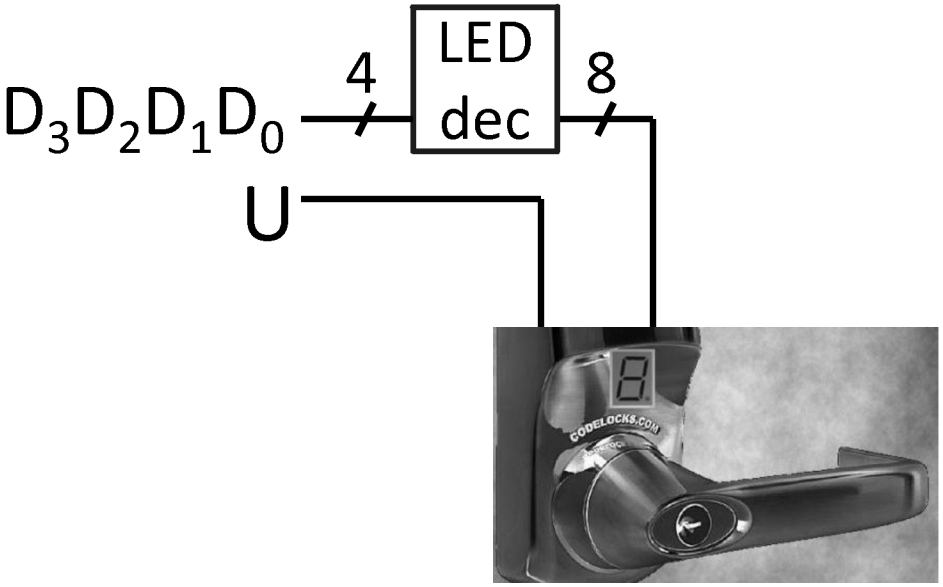
- signals are synchronized to clock
- Password is B-A-B

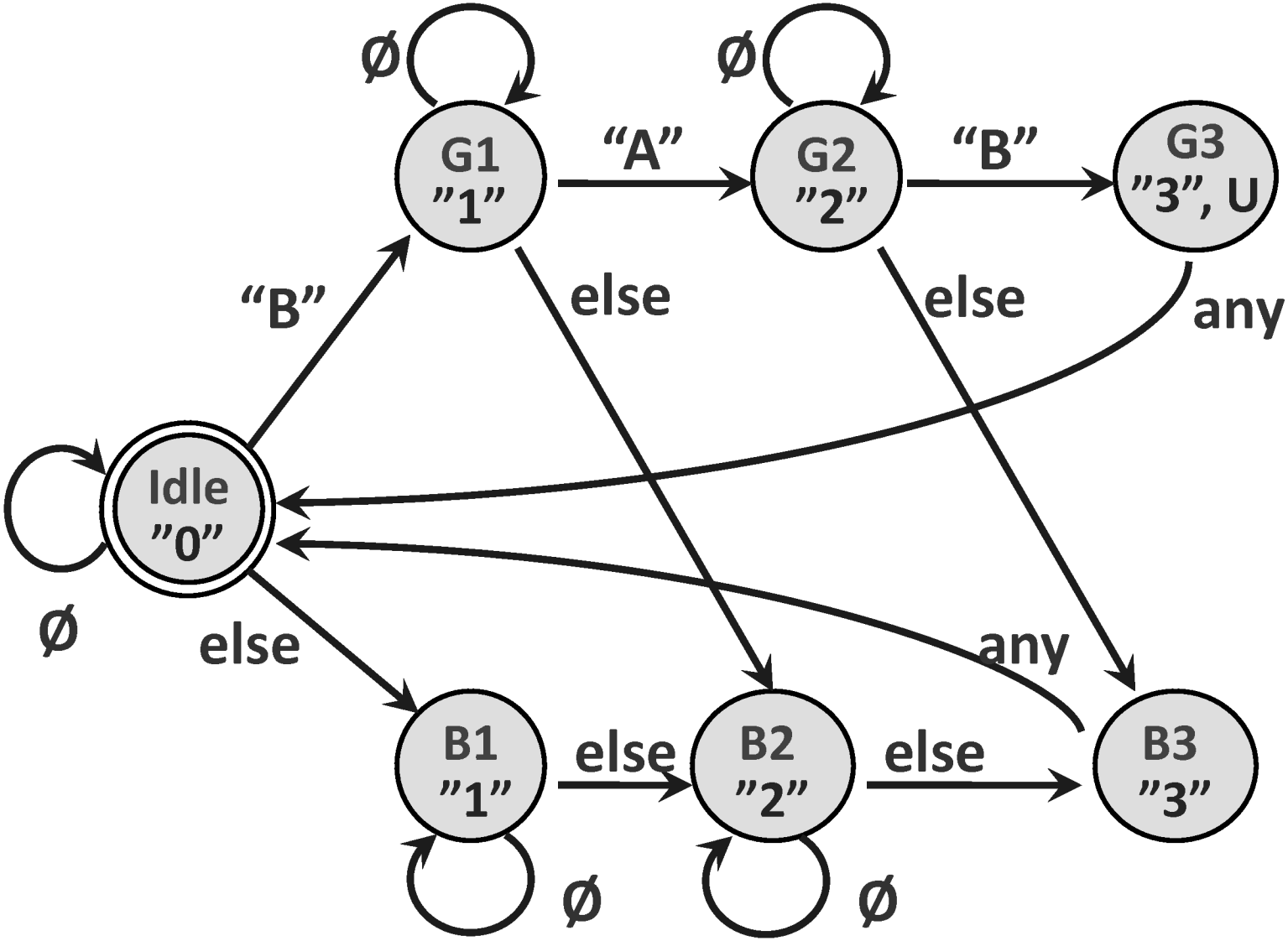


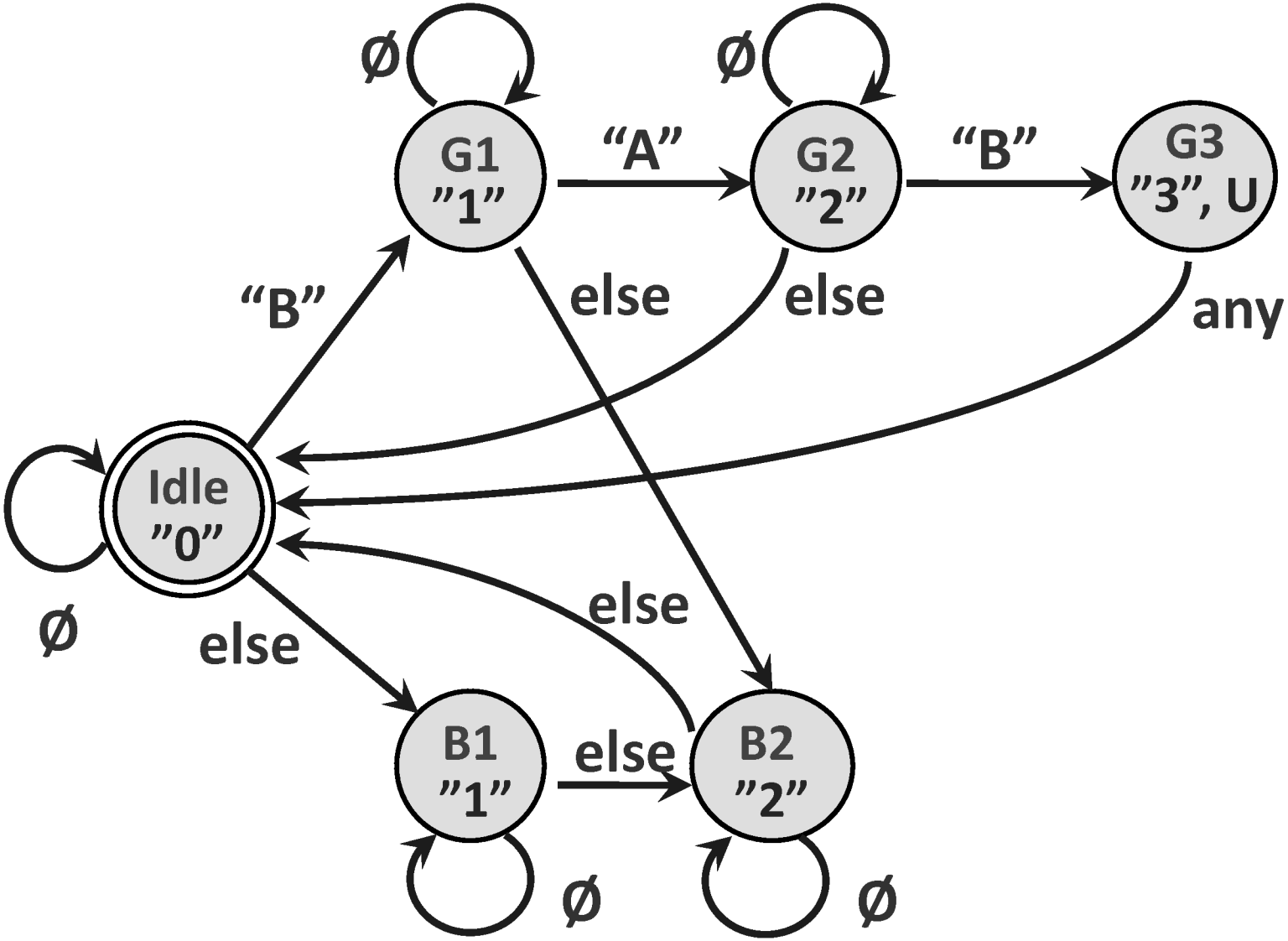
K	A	B	Meaning
0	0	0	Ø (no key)
1	1	0	‘A’ pressed
1	0	1	‘B’ pressed

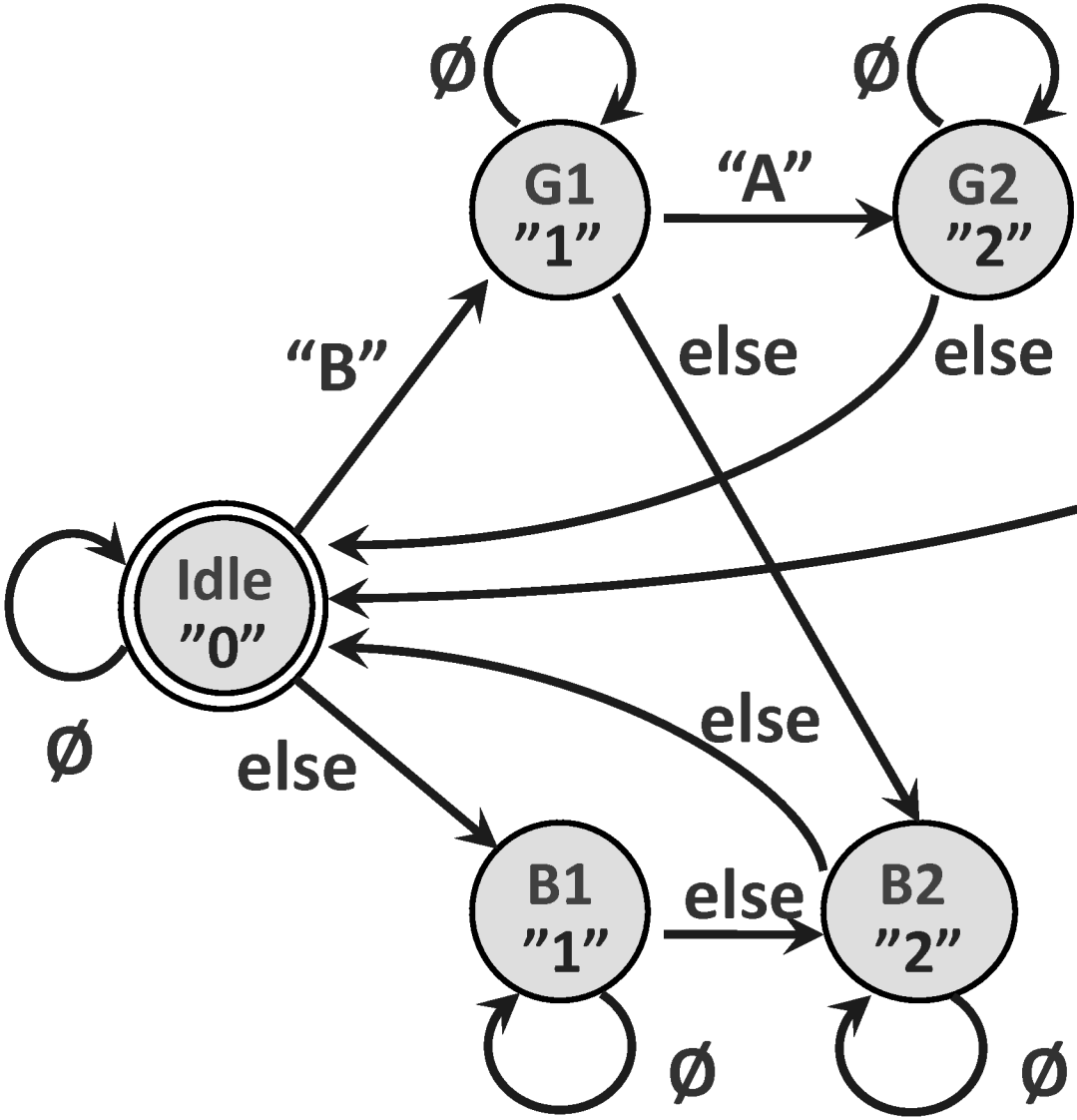
Assumptions:

- High pulse on U unlocks door

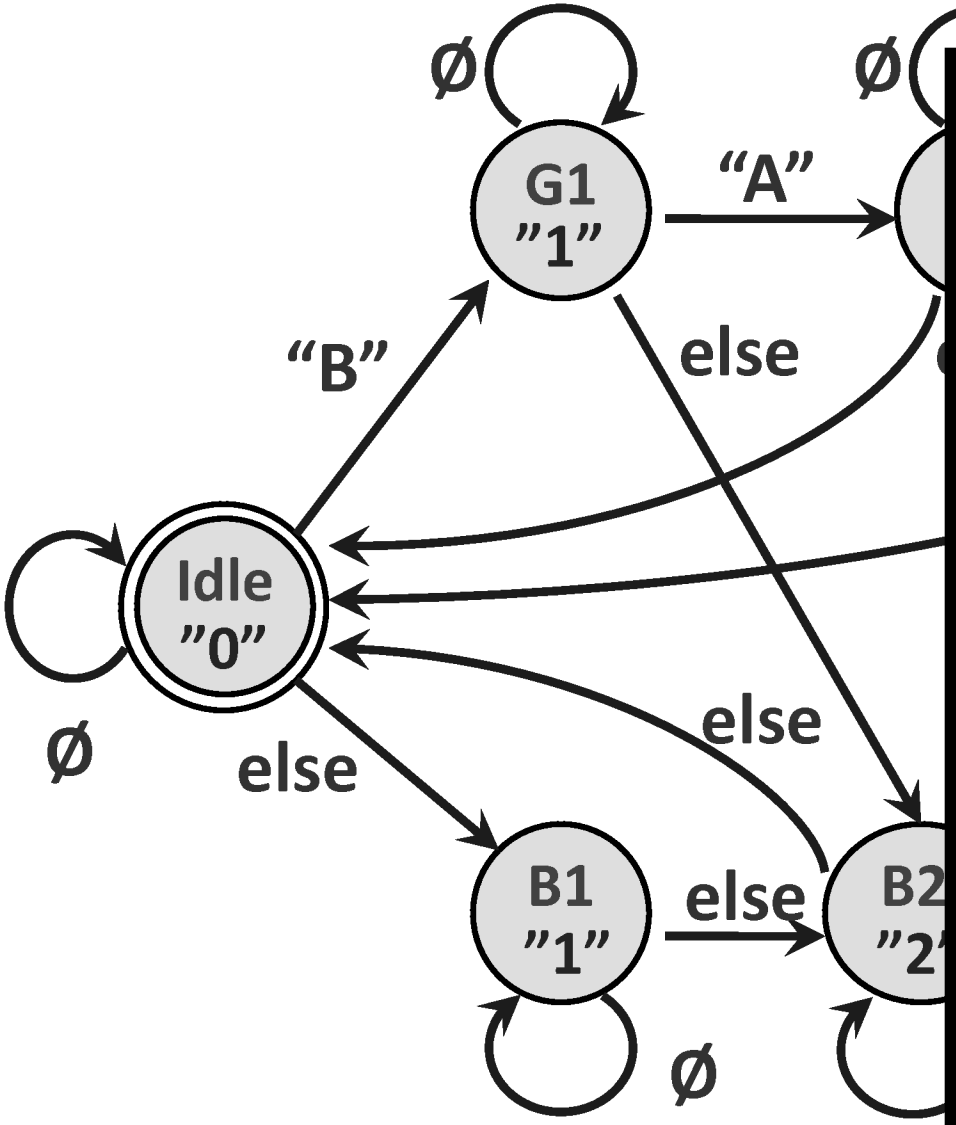



Ø
Ø
G1
"1"
"A"
G2
"2"
"B"
G3
"3", U
"B"
else
else
any
Idle
"0"
Ø
else
any
B1
"1"
else
B2
"2"
else
B3
"3"
Ø
Ø


Ø
Ø
G1
"1"
"A"
G2
"2"
"B"
G3
"3", U
"B"
else
else
any
Idle
"0"
Ø
else
else
B1
"1"
else
B2
"2"
Ø
Ø



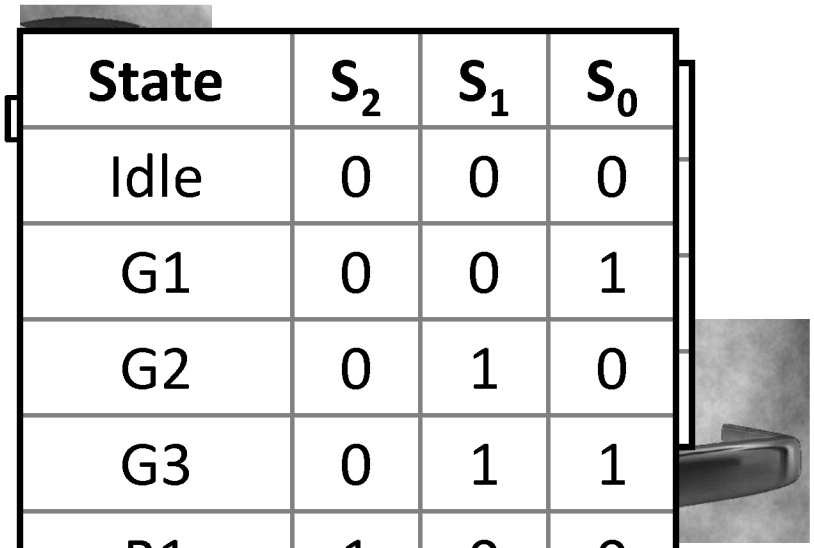
Cur. State	Output
Idle	"0"
G1	"1"
G2	"2"
G3	"3", U
B1	"1"
B2	"2"



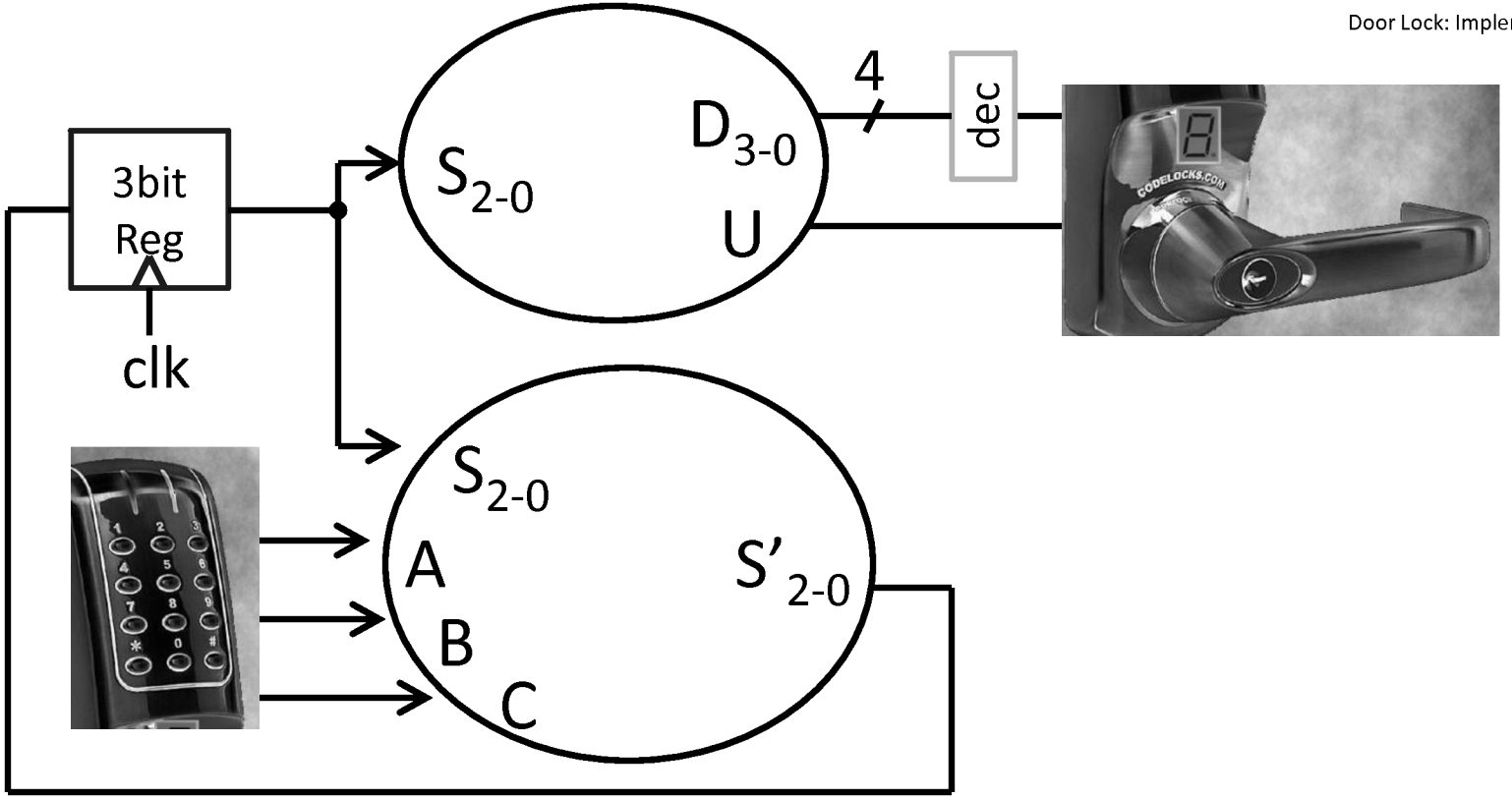
Cur. State	Input	Next State
Idle	Ø	Idle
Idle	"B"	G1
Idle	"A"	B1
G1	Ø	G1
G1	"A"	G2
G1	"B"	B2
G2	Ø	B2
G2	"B"	G3
G2	"A"	Idle
G3	any	Idle
B1	Ø	B1
B1	K	B2
B2	Ø	B2
B2	K	Idle

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

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	0	0	1
0	0	1	0	0	0	0	0	1
0	0	1	1	1	0	0	1	0
0	0	1	0	0	1	0	1	0
0	1	0	0	0	0	0	1	0
0	1	0	0	0	1	0	1	1
0	1	0	0	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



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

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