Computer Architecture

Week 3: Combinational and Sequential Circuits

Fenerbahçe University

Professor and TAs

Prof: Dr. Vecdi Emre Levent

Office: 311

Email: emre.levent@fbu.edu.tr

TA: Arş. Gör. Uğur Özbalkan

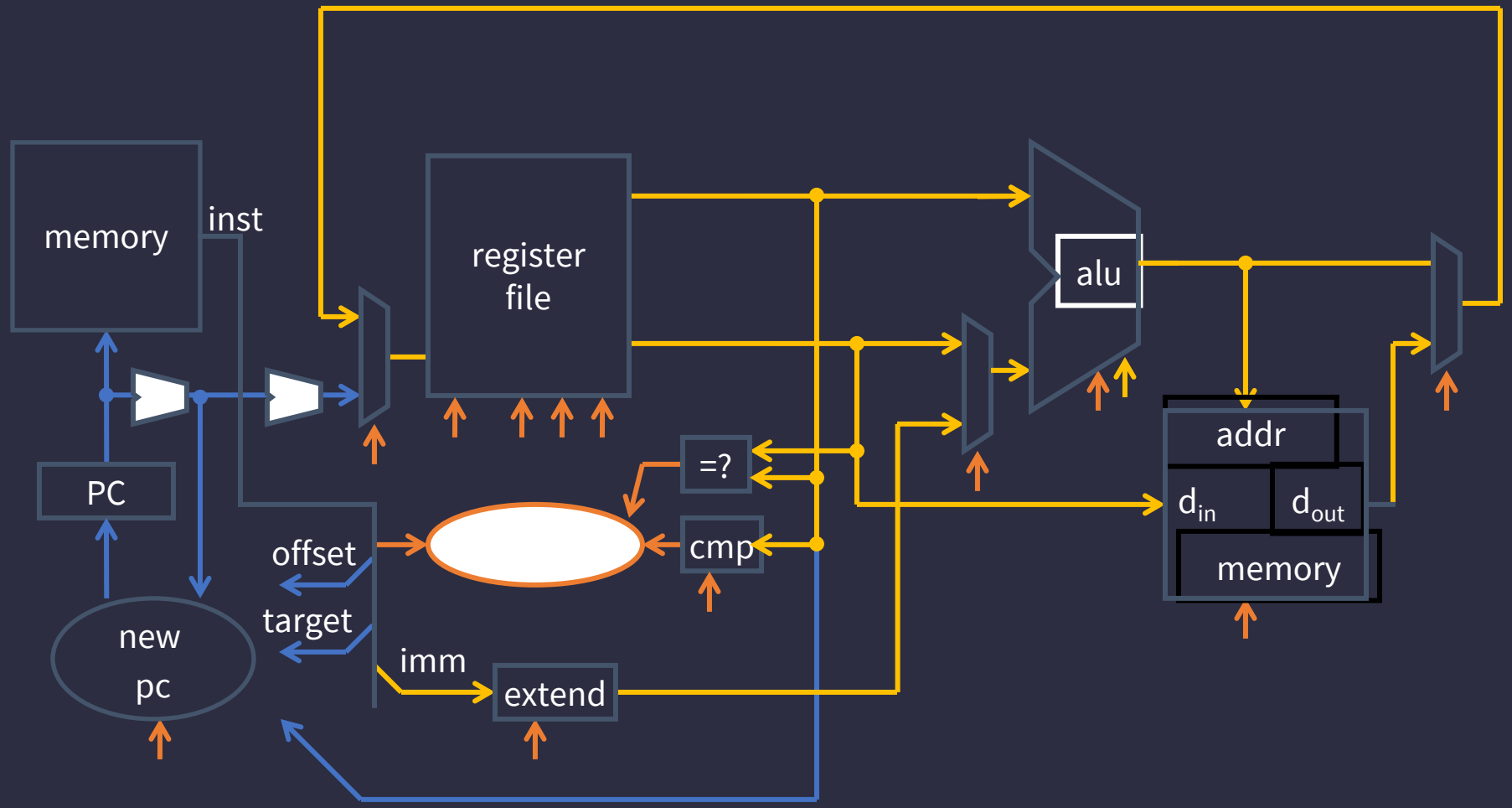
Office: 311

Email: ugur.ozbalkan@fbu.edu.tr

Course Plan

- Combinational and Sequential Circuits

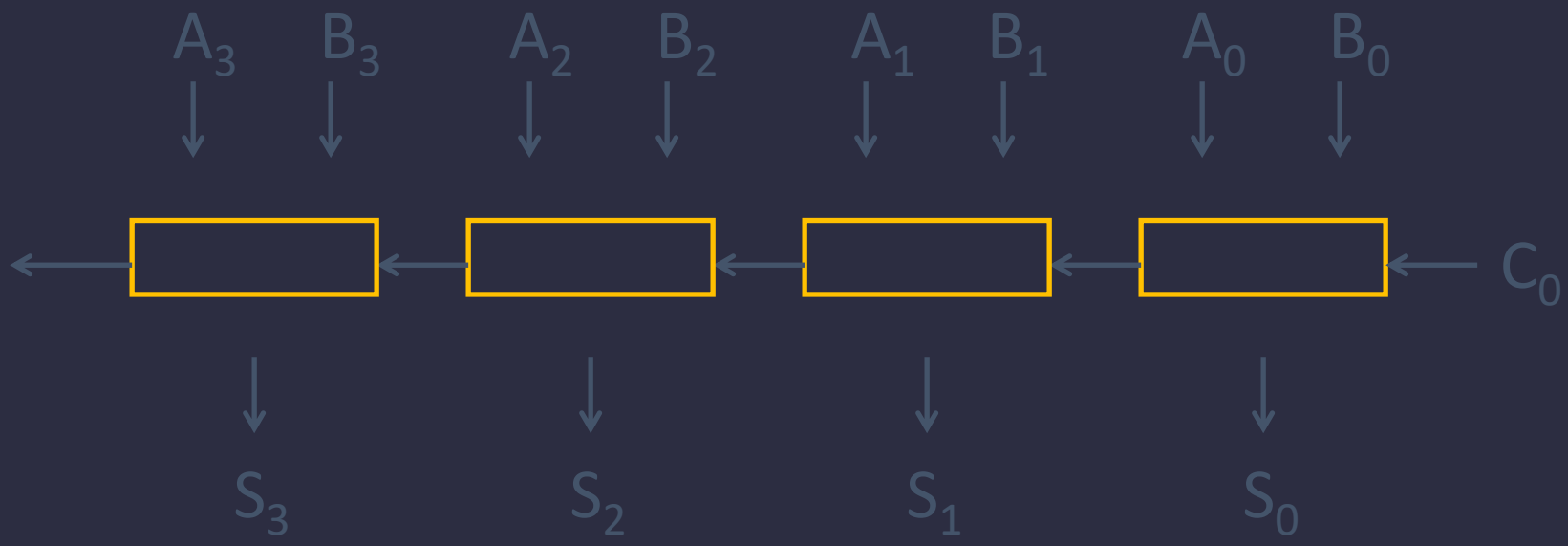
Big Picture: Building a Processor



A single cycle processor

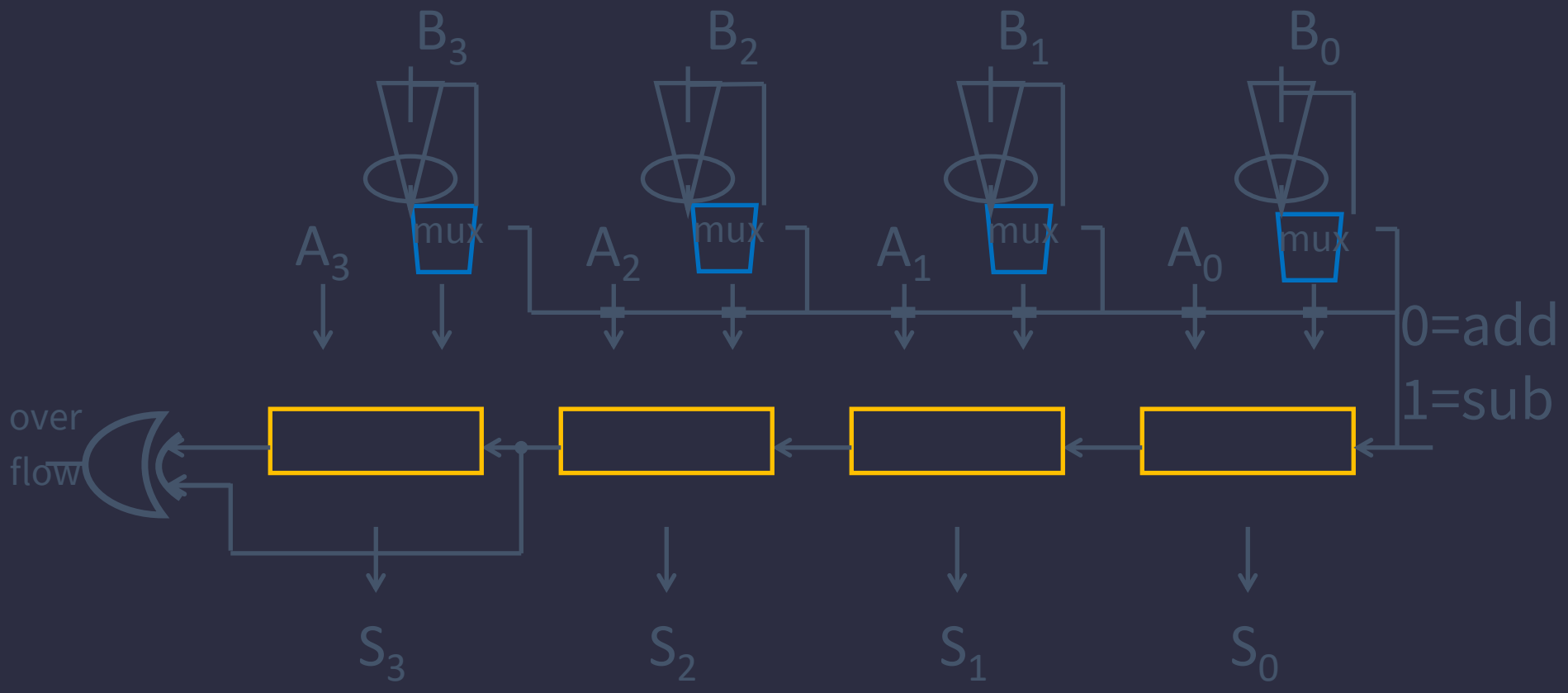
Review

- We can generalize 1-bit Full Adders to 32 bits, 64 bits …



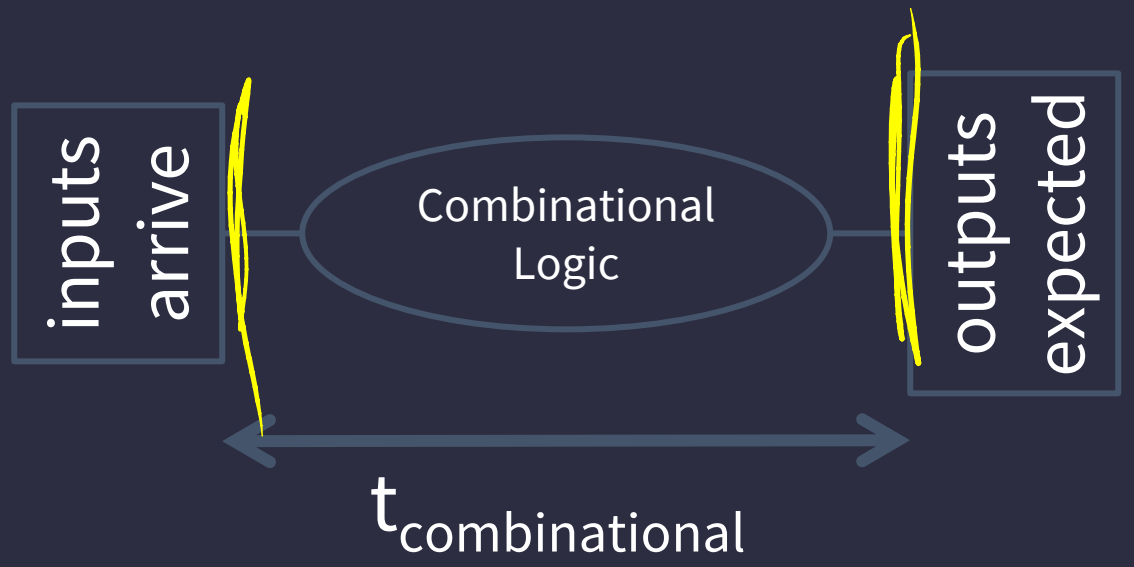
Review

- We can generalize 1-bit Full Adders to 32 bits, 64 bits …

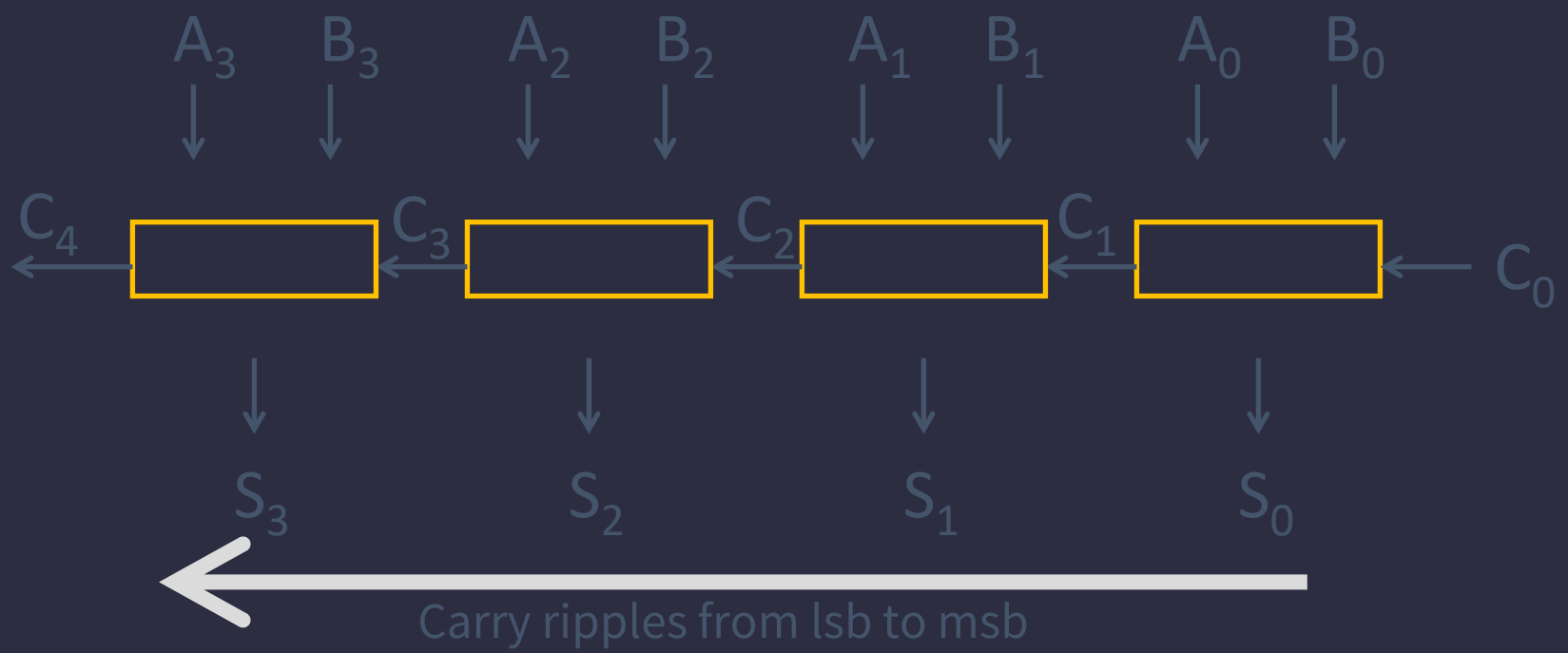


Performance

Speed of a circuit is affected by the number of gates in series (on the *critical path* or the *deepest level of logic*)



4-bit Ripple Carry Adder

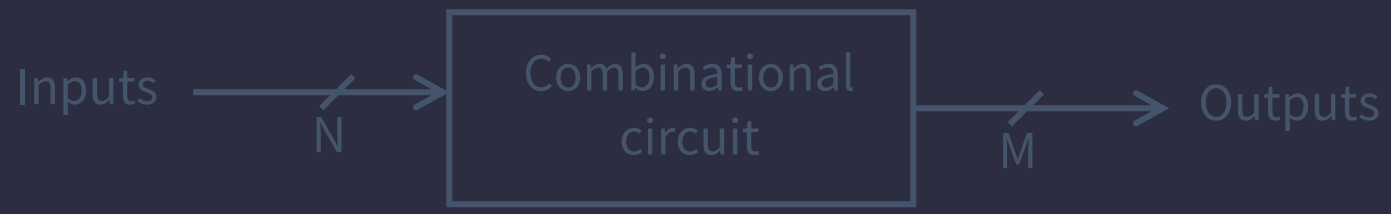


- First full adder, 2 gate delay
- Second full adder, 2 gate delay

Stateful Components

Until now is combinationial 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

Goals for Today

State

- How do we store ***one*** bit?
- Attempts at storing (and changing) one bit
  - Set-Reset Latch
  - D Latch
  - D Flip-Flops
  - Master-Slave Flip-Flops
- Register: storing more than one bit, N-bits

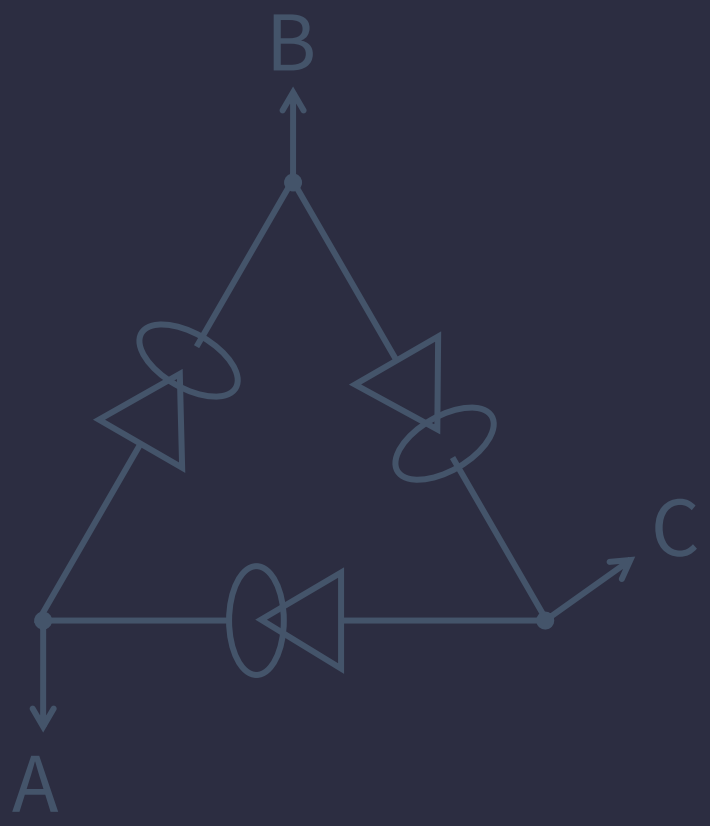
Basic Building Blocks

- Decoders and Encoders

Goal

How do we store store ***one*** bit?

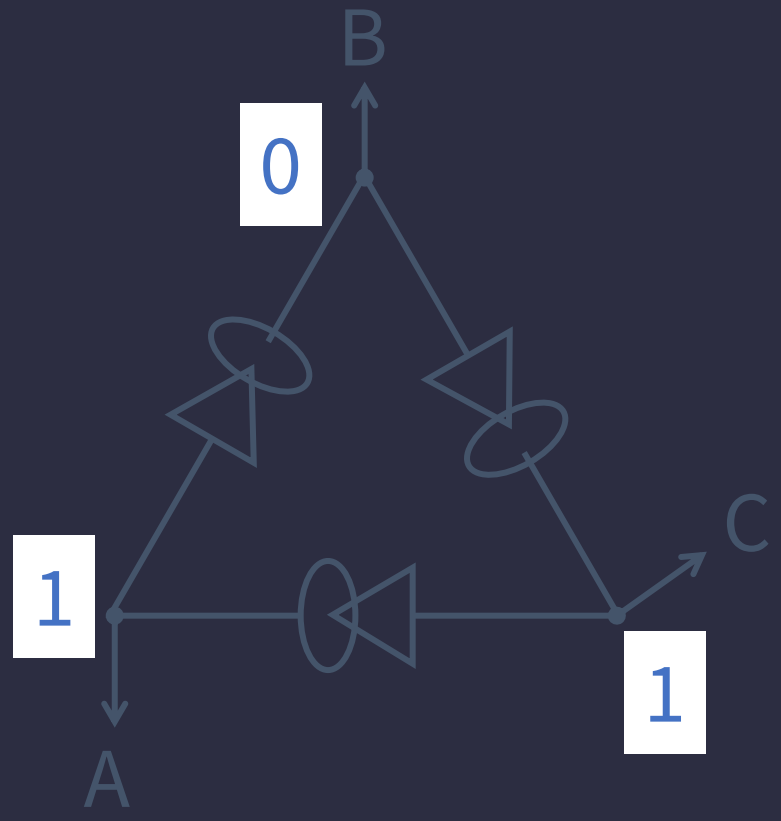
First Attempt: Unstable Devices



First Attempt: Unstable Devices

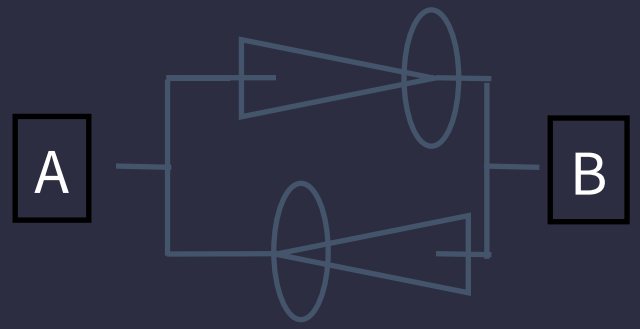
Does not work!

- Unstable
- Oscillates wildly!



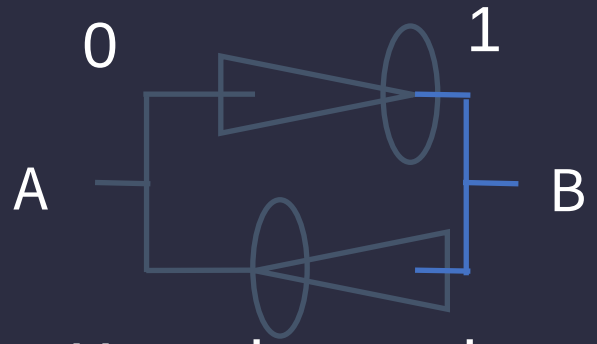
Second Attempt: Bistable Devices

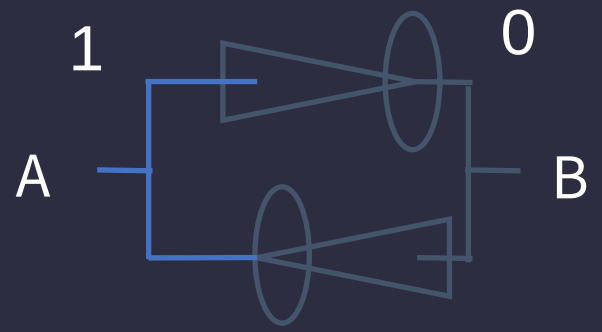
- Stable and unstable equilibria?



A Simple Device

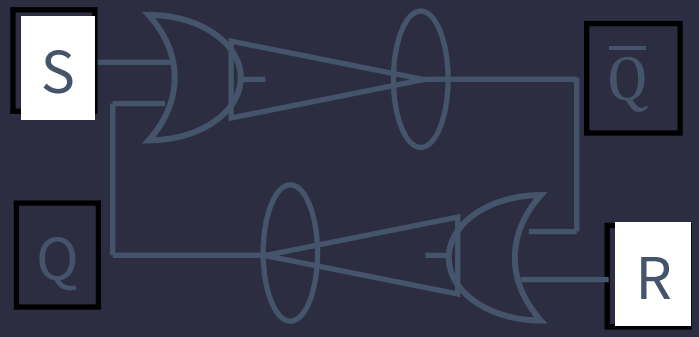
In stable state, $\bar{A} = B$



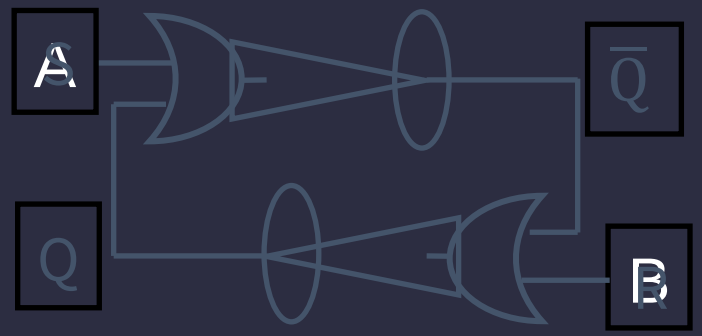


How do we change the state?

Third Attempt: Set-Reset Latch



Third Attempt: Set-Reset Latch

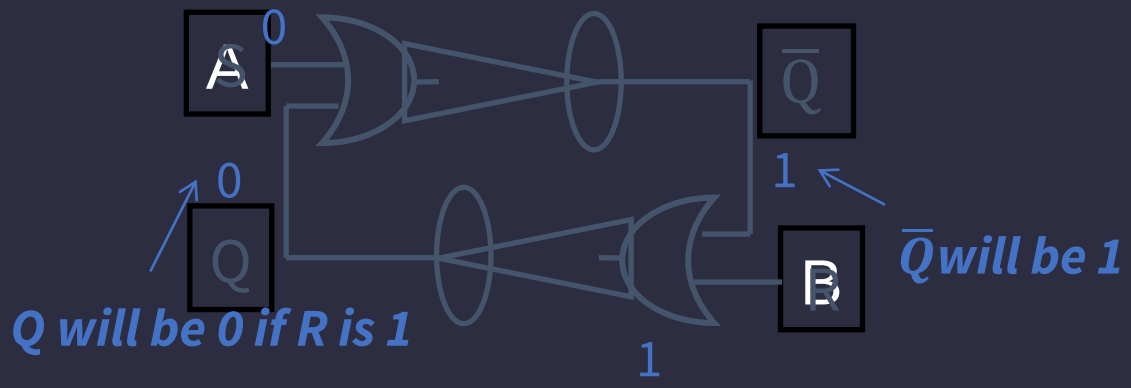


A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

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

Set-Reset (S-R) Latch
Stores a value Q and its complement

Third Attempt: Set-Reset Latch

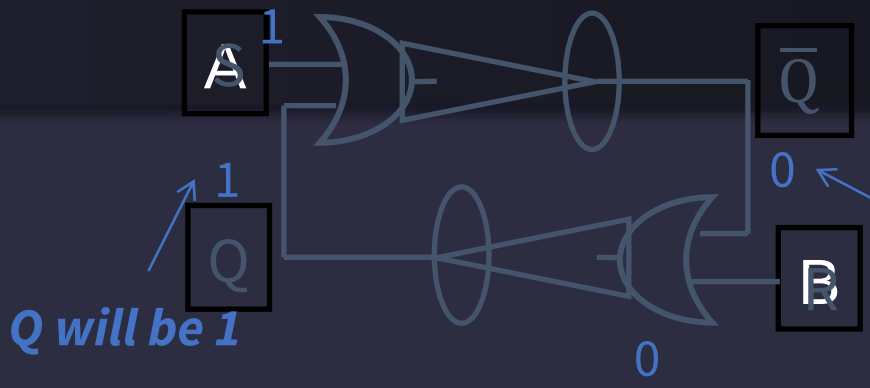


A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

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

Set-Reset (S-R) Latch
Stores a value Q and its complement

Third Attempt: Set-Reset Latch



A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Set-Reset (S-R) Latch
Stores a value Q and its complement

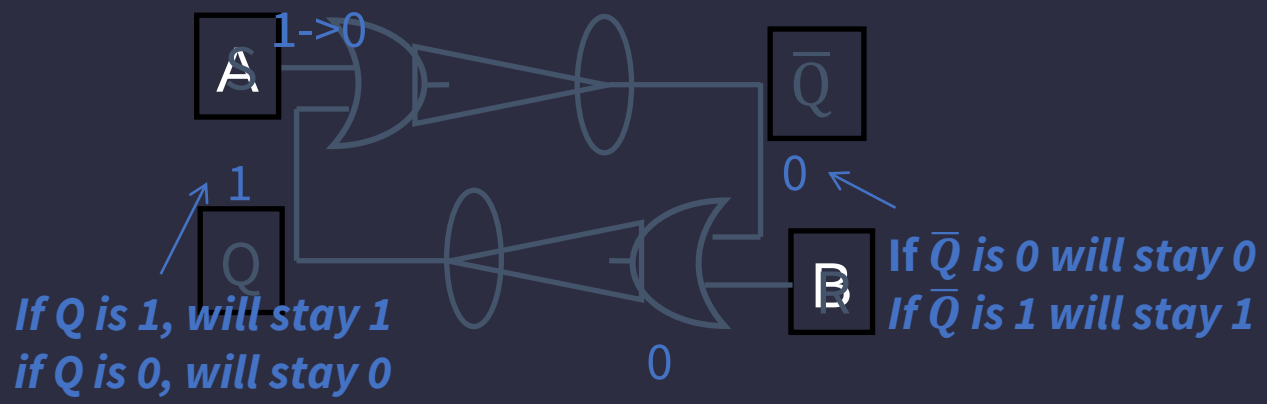
S	R	Q	$\overline{Q}$
0	0		
0	1	0	1
1	0	1	0
1	1		

What are the values for Q and $\overline{Q}$?

a) 0 and 0
b) 0 and 1
c) ~~1 and 0~~
d) 1 and 1

iClicker Question

Third Attempt: Set-Reset Latch

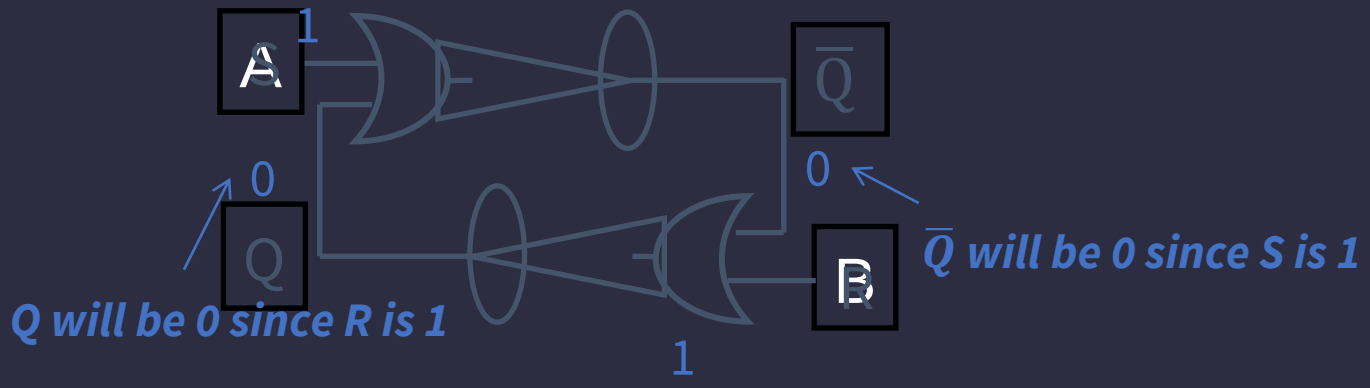


A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

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

Set-Reset (S-R) Latch
Stores a value Q and its complement

Third Attempt: Set-Reset Latch



A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

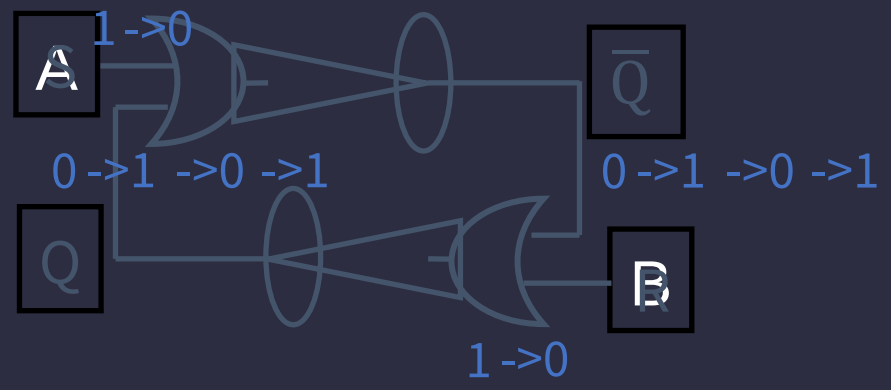
S	R	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
1	1	?	?

Set-Reset (S-R) Latch

Stores a value Q and its complement

What happens when S,R changes from 1,1 to 0,0?

Third Attempt: Set-Reset Latch



A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

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

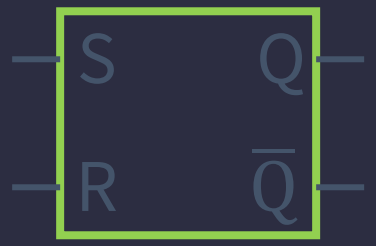
Set-Reset (S-R) Latch

Stores a value Q and its complement

What happens when S,R changes from 1,1 to 0,0?

Q and $\overline{Q}$ become unstable and will oscillate wildly between values 0,0 to 1,1 to 0,0 to 1,1 …

Third Attempt: Set-Reset Latch



S	R	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	hold
0	1	0	1	reset
1	0	1	0	set
1	1	forbidden		

Set-Reset (S-R) Latch
Stores a value Q and its complement

Takeaway

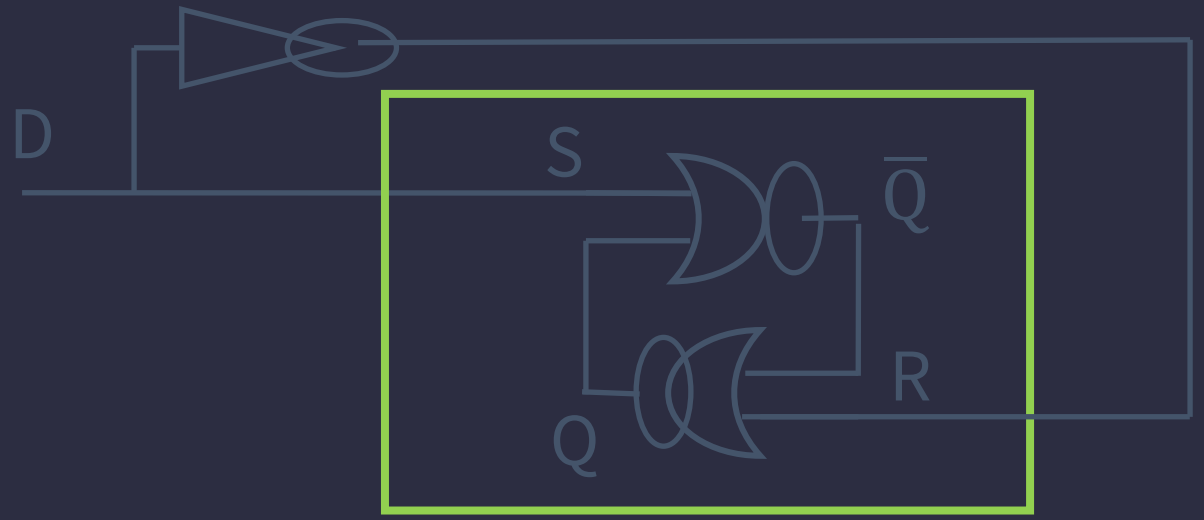
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit.

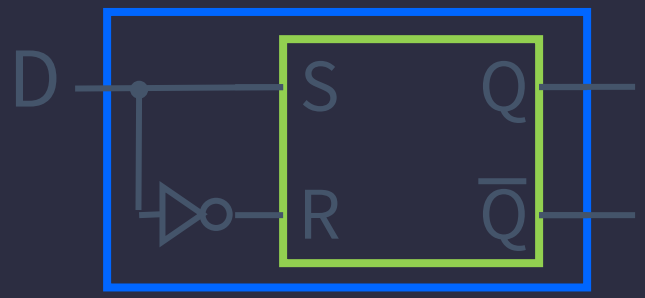
But, SR Latch has a forbidden state.

Next Goal

How do we avoid the forbidden state of S-R Latch?

Fourth Attempt: (Unclocked) D Latch





Data (D) Latch

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

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

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

When D changes, Q changes

- … immediately (…after a delay of 2 Ors and 2 NOTs)

Need to control when the output changes

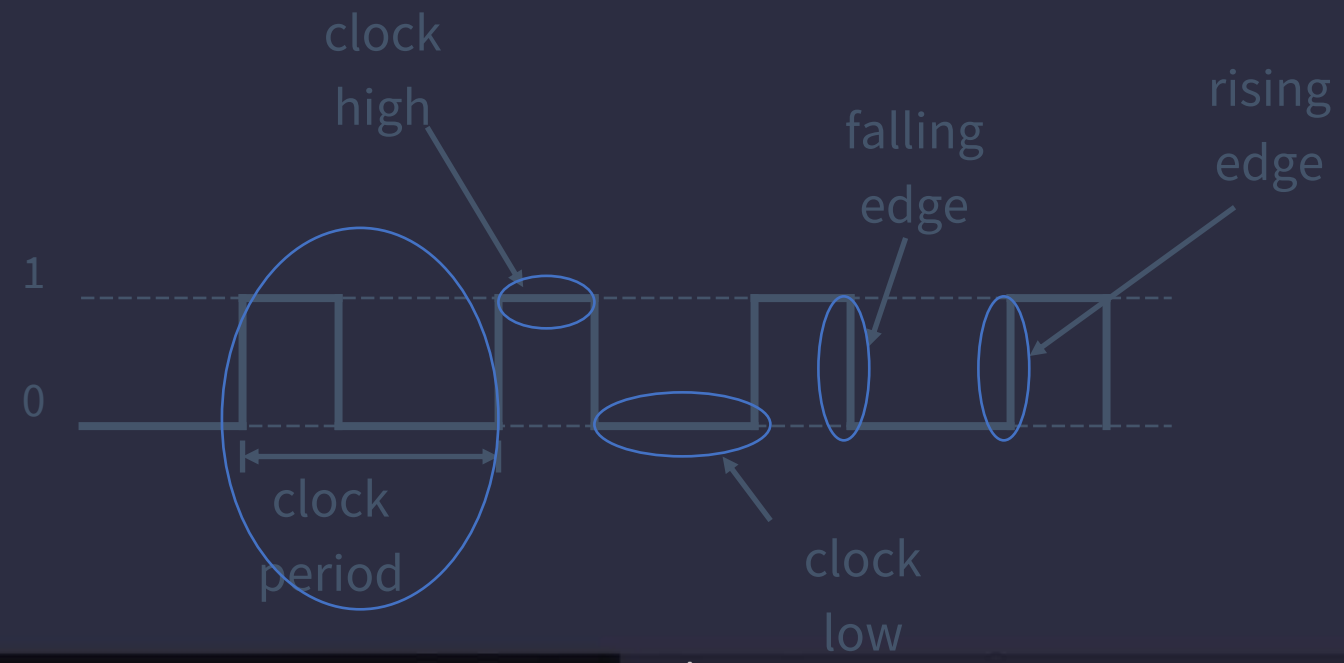
Next Goal

How do we coordinate state changes to a D Latch?

Aside: Clocks

Clock helps coordinate state changes

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



Clock Disciplines

Level sensitive

- State changes when clock is high (or low)

Edge triggered

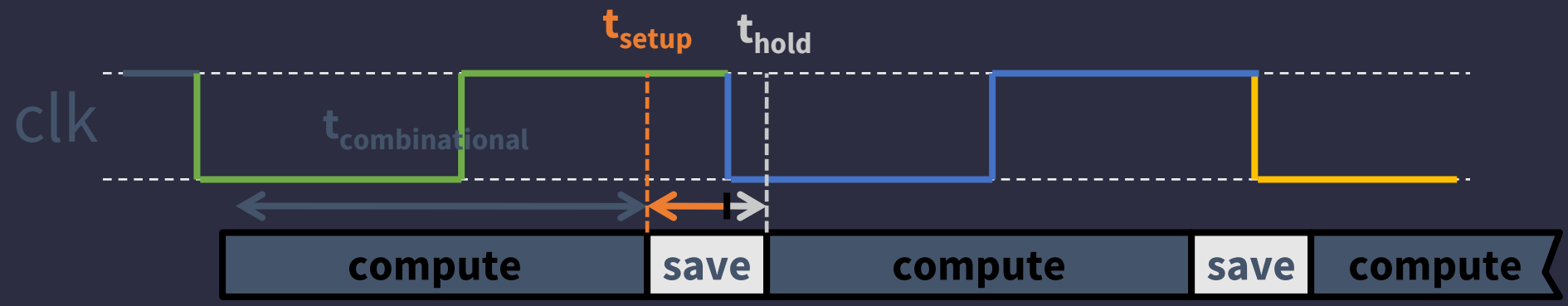
- State changes at clock edge

positive edge-triggered

negative edge-triggered

Clock Methodology

- Negative edge, synchronous

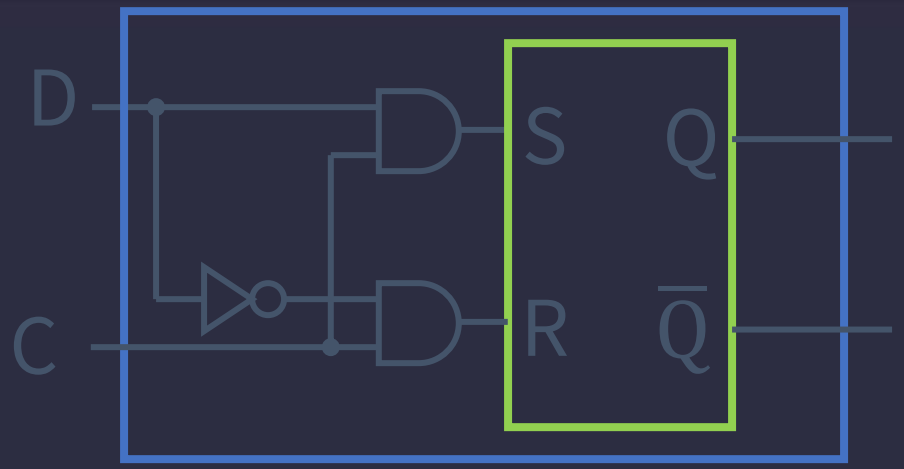


Edge-Triggered → signals must be stable near falling edge

"near" = before and after

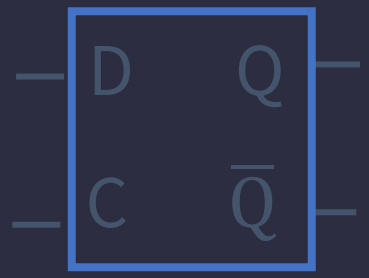
t_{setup} t_{hold}

Round 2: D Latch (1)



C = 1, D Latch *transparent*:
set/reset (according to D)
C = 0, D Latch *opaque*:
keep state (ignore D)

S	R	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	hold
0	1	0	1	reset
1	0	1	0	set
1	1	forbidden		

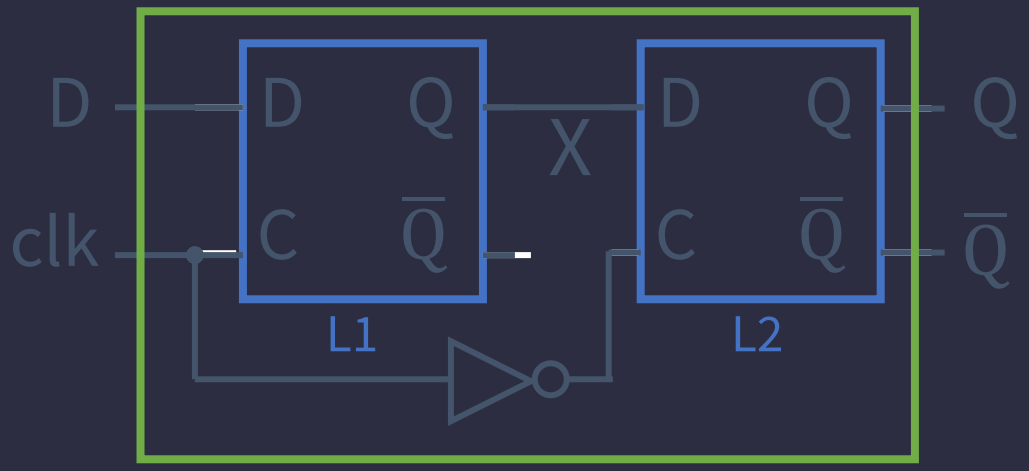


- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- C enables changes

C	D	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	*No Change*
0	1	Q	$\overline{Q}$	
1	0	0	1	*Reset*
1	1	1	0	*Set*

Round 3: D Flip-Flop

- Edge-Triggered
- Data captured when clock high
- Output changes only on falling edges



Round 3: D Flip-Flop

Clock = 1: L1 *transparent*

L2 *opaque*

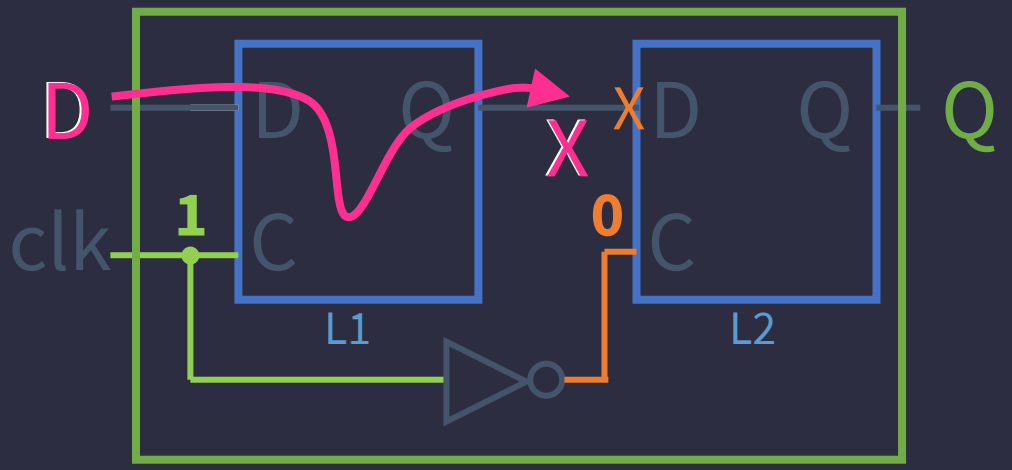
When CLK rises (0→1), now X can change, Q does not change

Clock = 0: L1 *opaque*

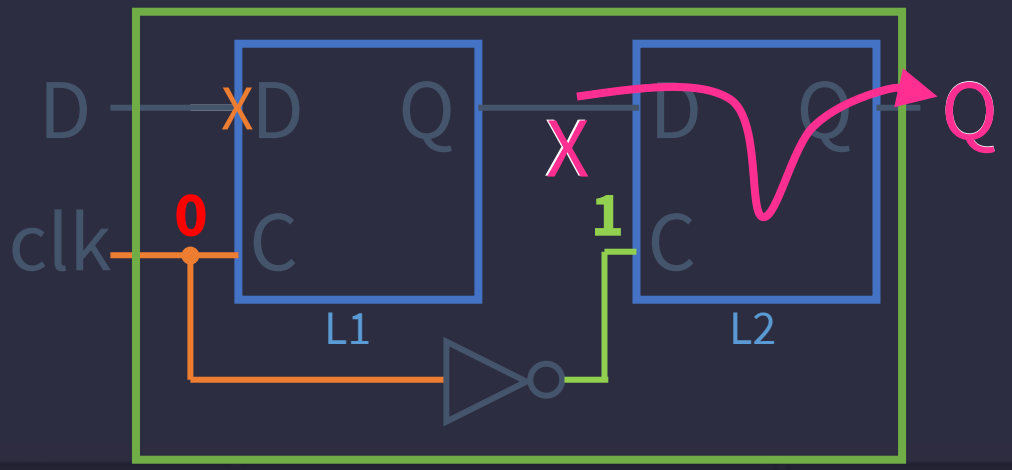
L2 *transparent*

When ***CLK* falls** (1→0), *Q gets X, X cannot change*

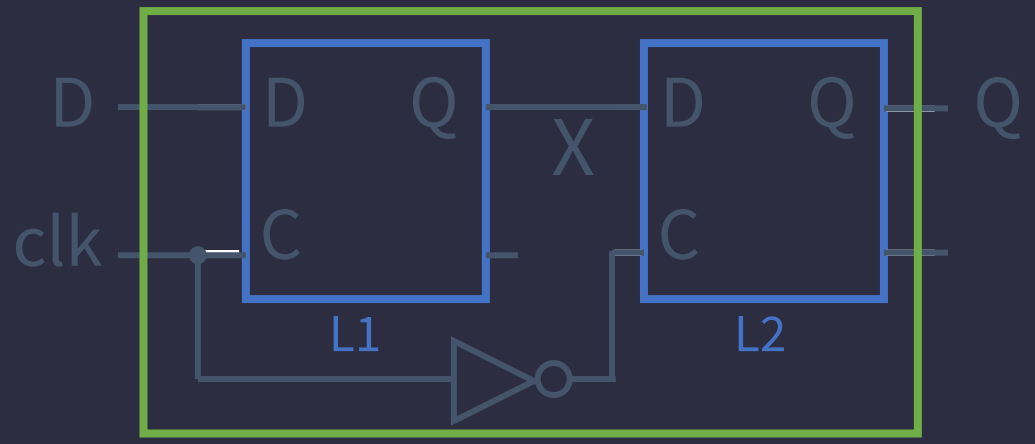
D passes through L1 to X



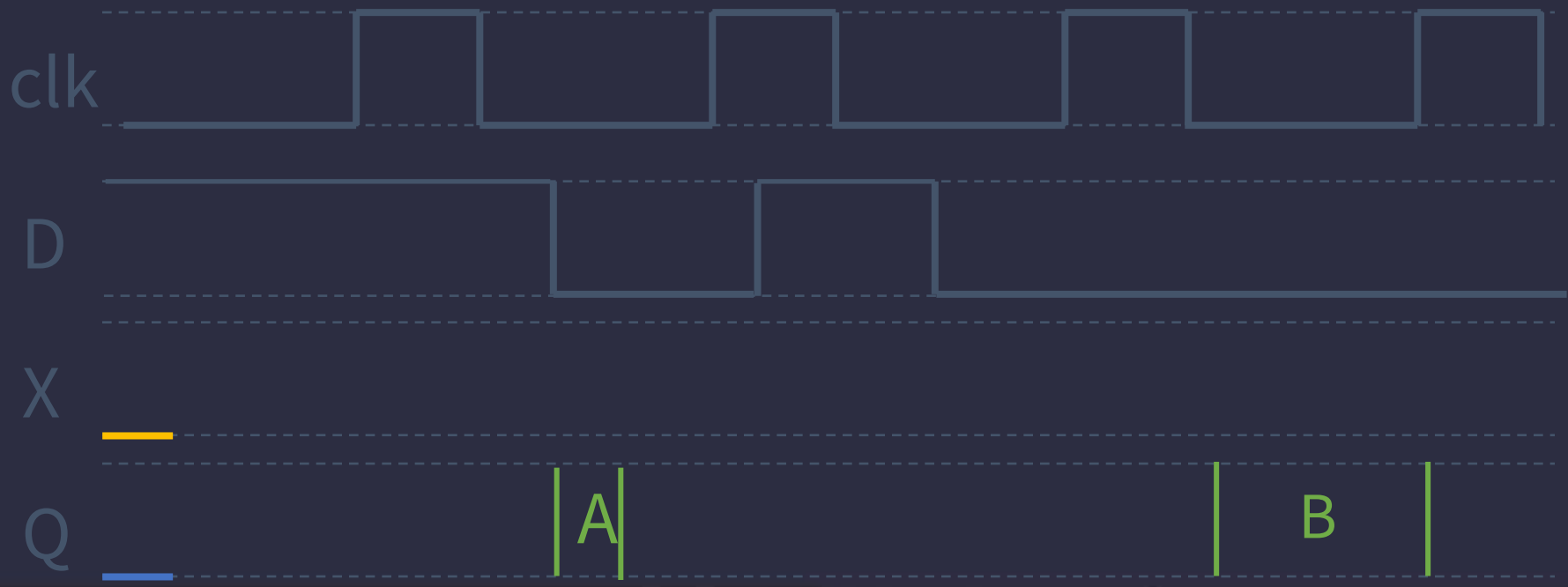
X passes through L2 to Q



Example



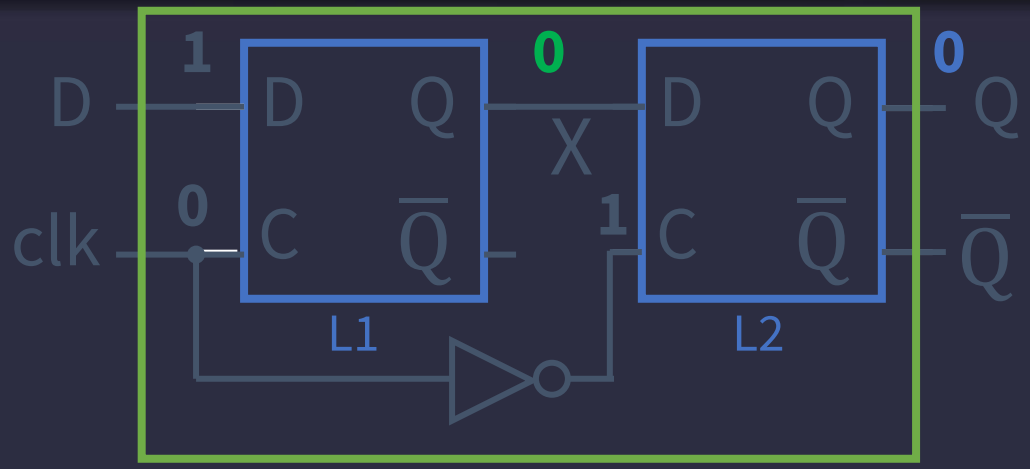
A = 1, B = 0

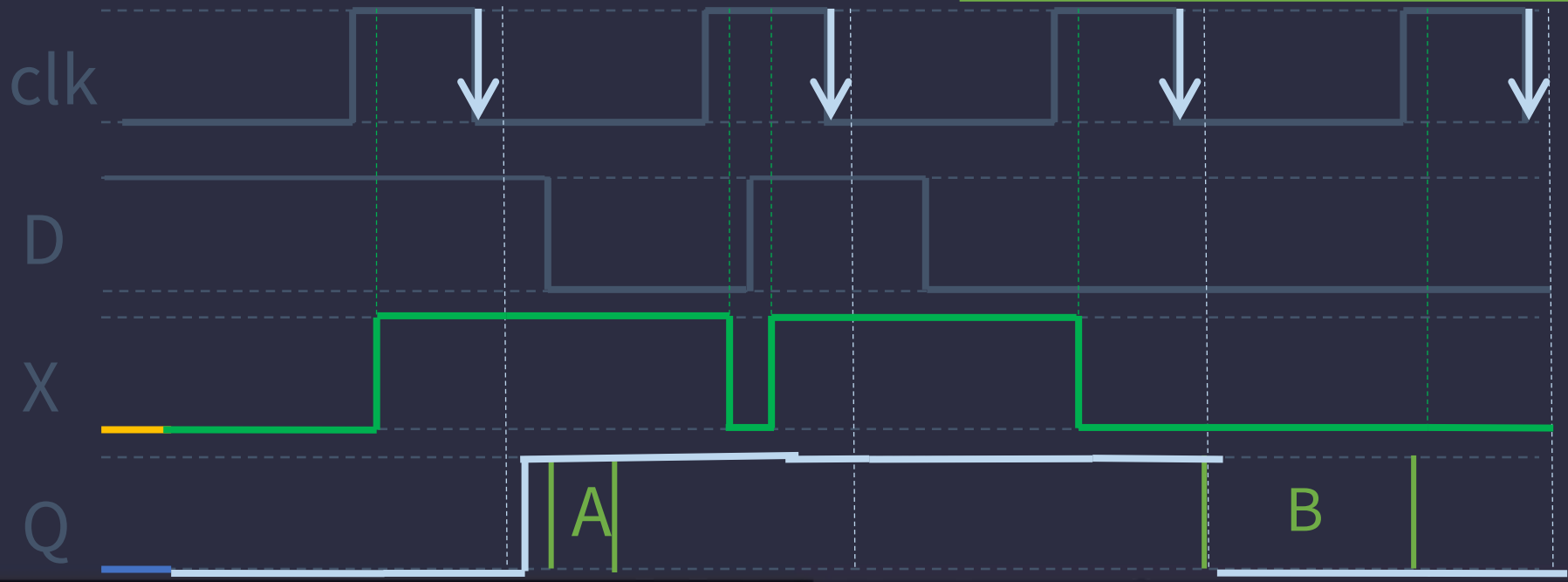


Edge-Triggered D Flip-Flop

D Flip-Flop

- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges





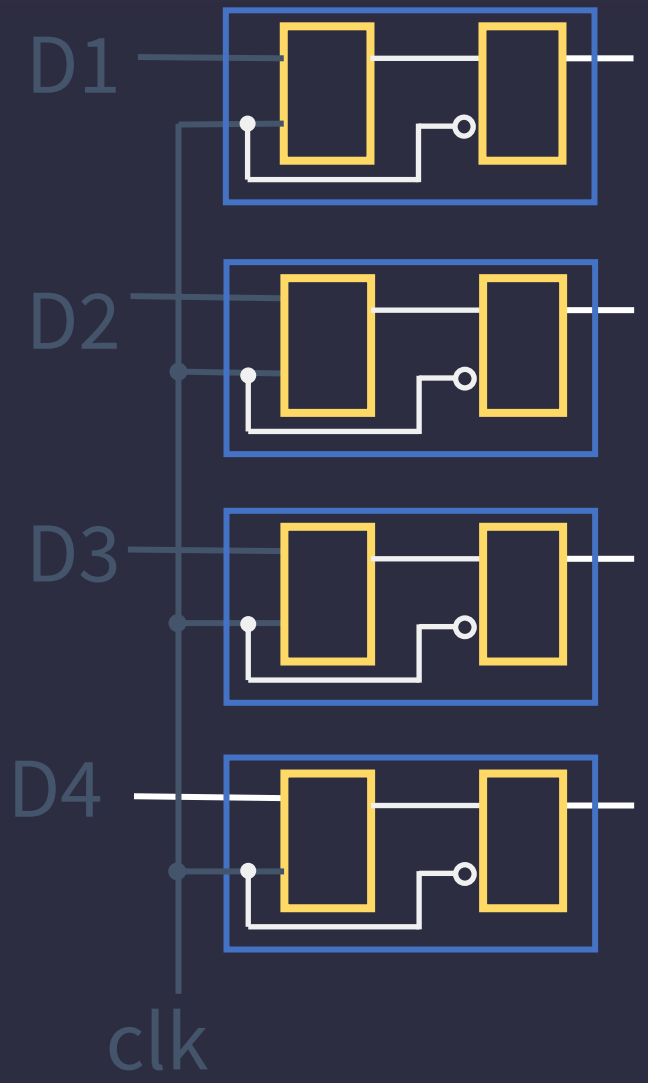
Next Goal

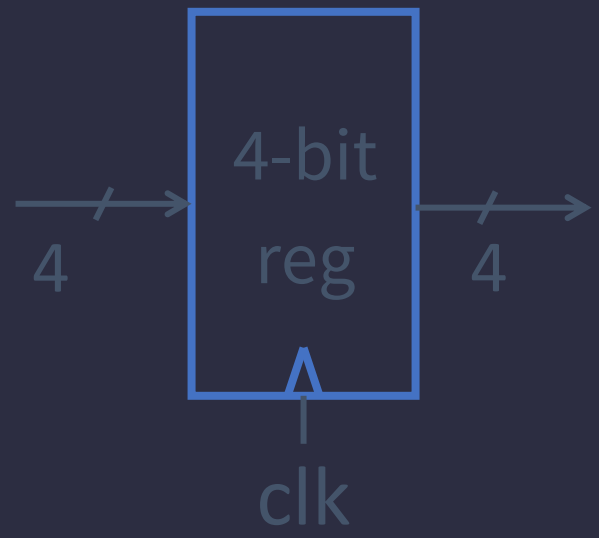
How do we store more than one bit, N bits?

Registers

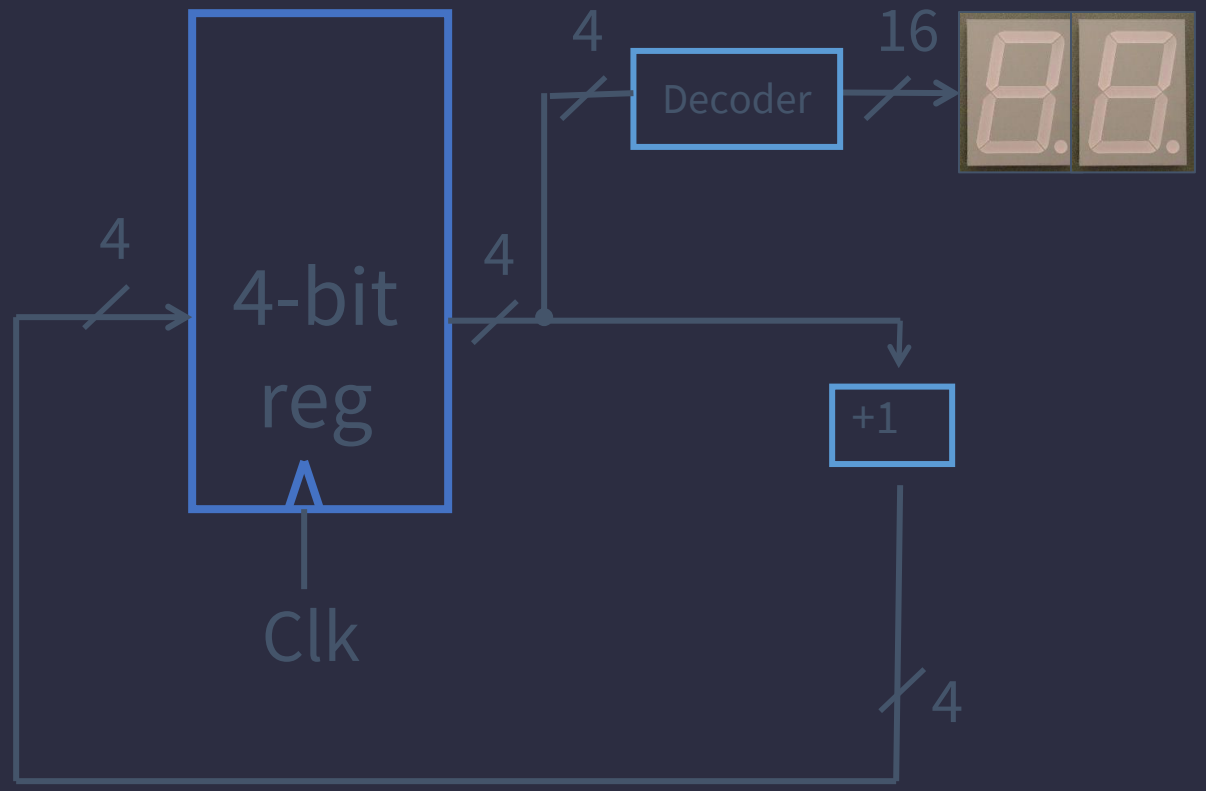
Register

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

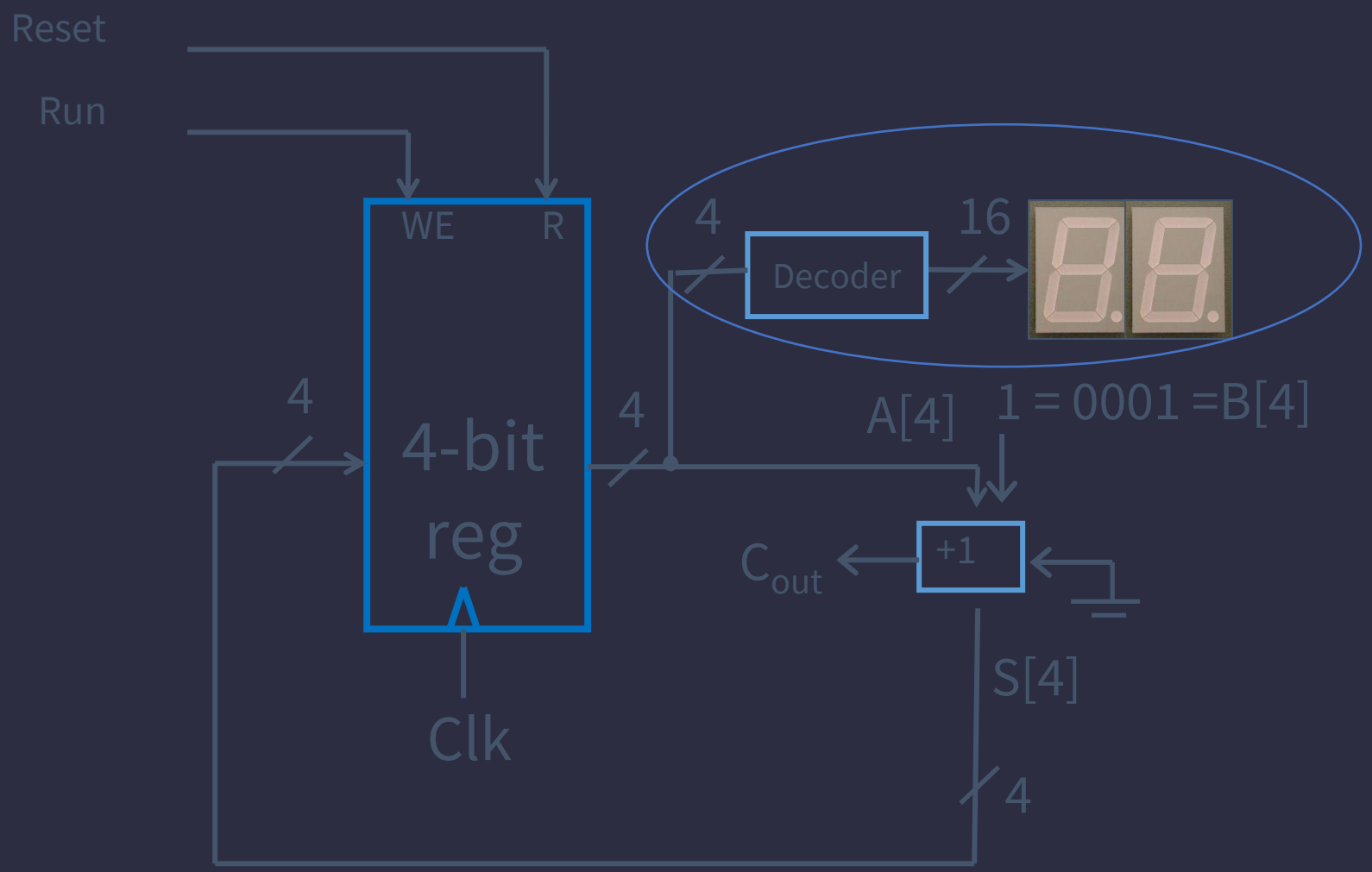




An Example: What will this circuit do?



An Example: What will this circuit do?



Decoder Example: 7-Segment LED

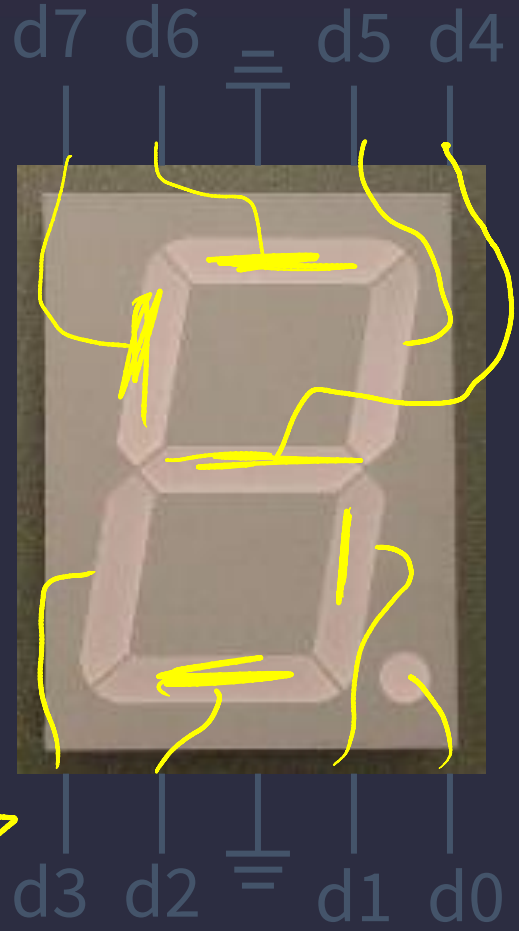
7-Segment LED

- photons emitted when electrons fall into holes

Decoder Example: 7-Segment LED

7-Segment LED

- photons emitted when electrons fall into holes



d0 1 2 3 4 5 6 7

5 = 0 1 1 0 1 0 1 1

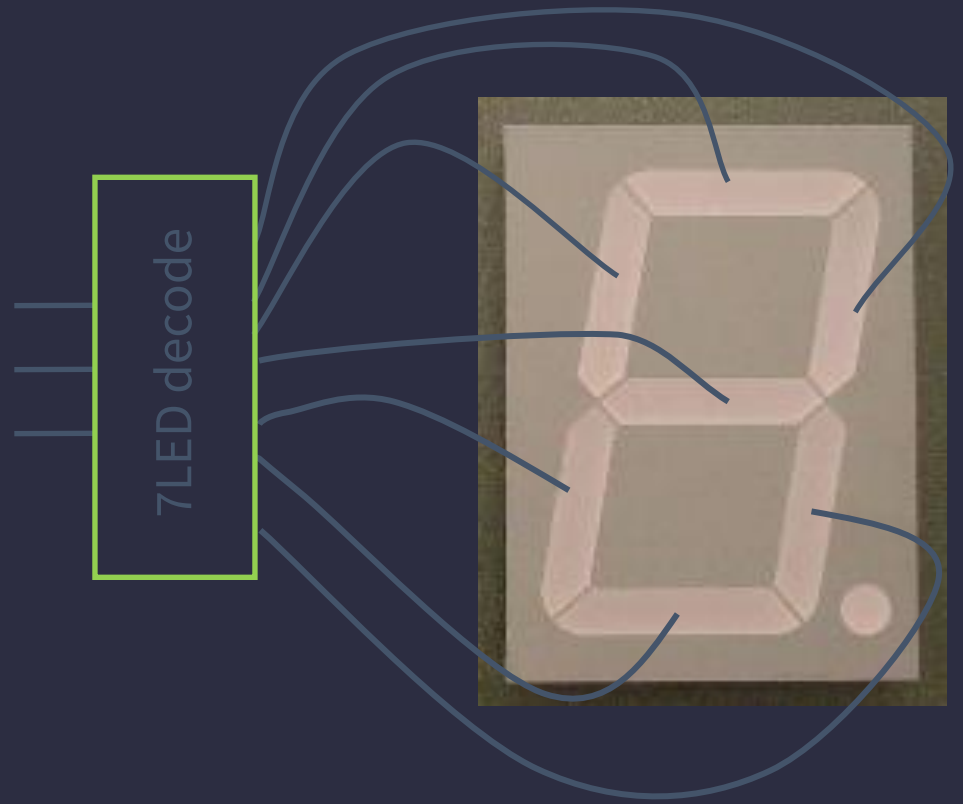
Decoder Example: 7-Segment LED Decoder

3 inputs

- encode 0 – 7 in binary

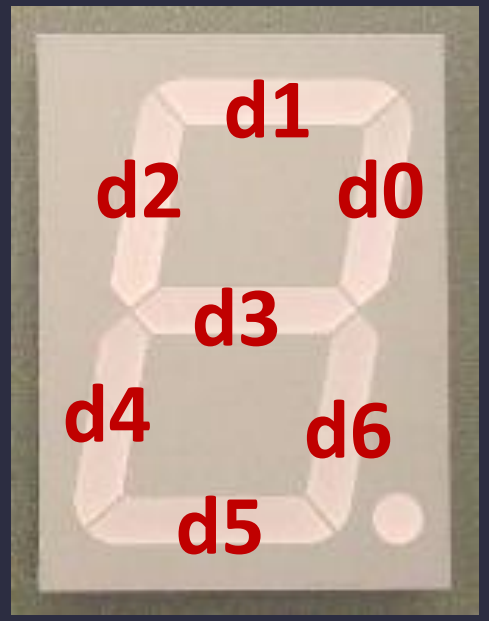
7 outputs

- one for each LED

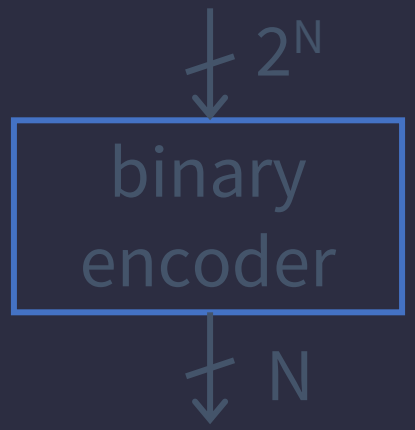


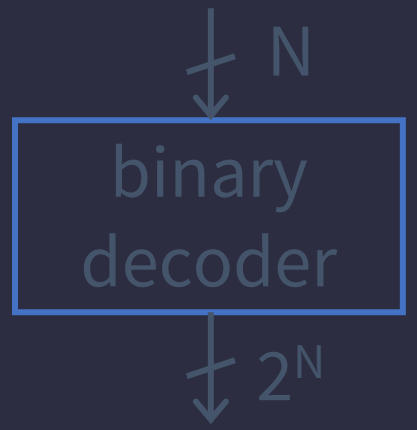
7 Segment LED Decoder Implementation

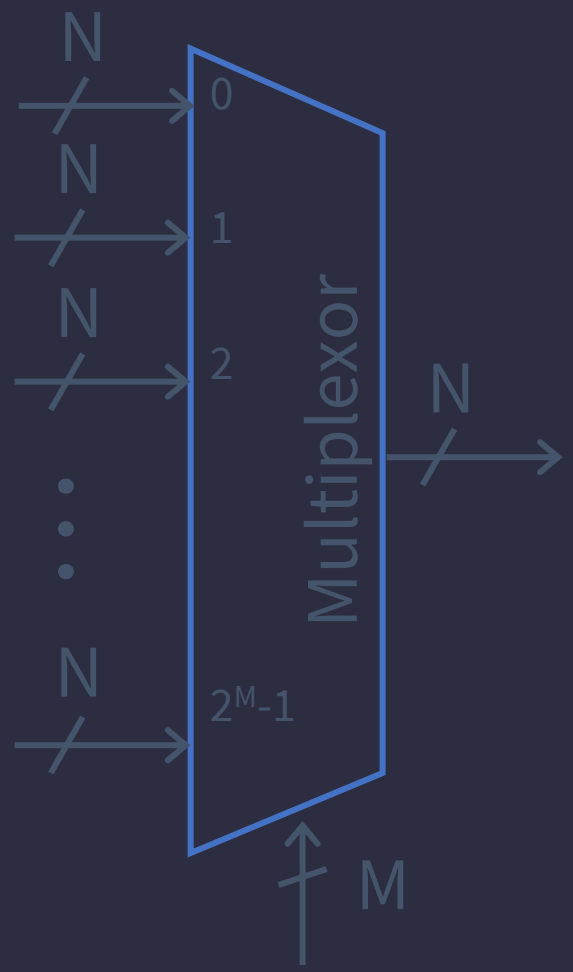
b2	b1	b0	d6	d5	d4	d3	d2	d1	d0
0	0	0	1	1	1	0	1	1	1
0	0	1	1	0	0	0	0	0	1
0	1	0	0	1	1	1	0	1	1
0	1	1	1	1	0	1	0	1	1
1	0	0	1	0	0	1	1	0	1
1	0	1	1	1	0	1	1	1	0
1	1	0	1	1	1	1	1	1	0
1	1	1	1	0	0	0	0	1	1



Basic Building Blocks We have Seen







Encoders

N Input wires

$Log_2(N)$ outputs wires