

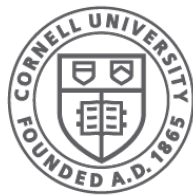
State

Prof. Hakim Weatherspoon

CS 3410

Computer Science

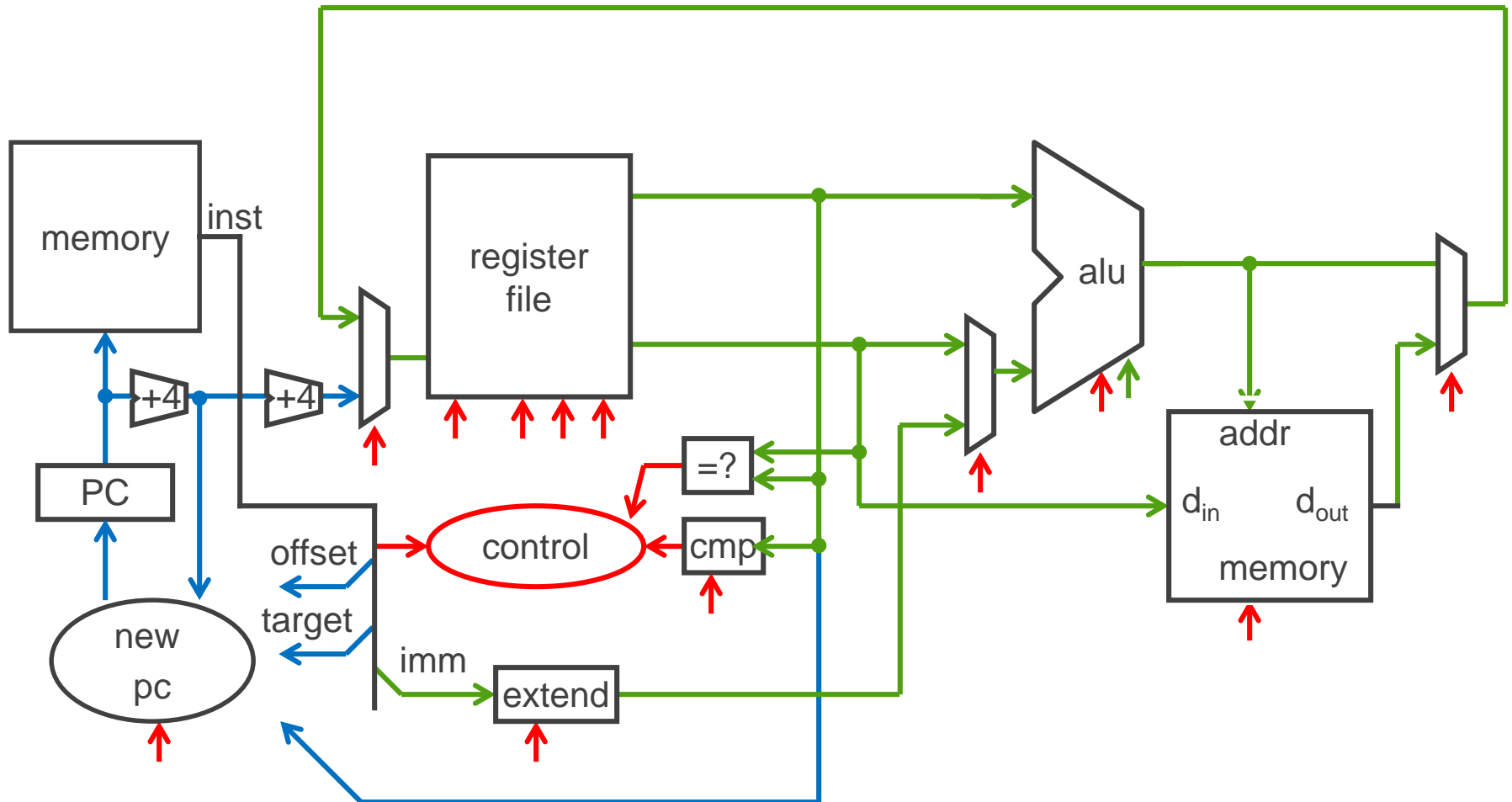
Cornell University



Cornell CIS
COMPUTING AND INFORMATION SCIENCE

[Weatherspoon, Bala, Bracy, and Sirer]

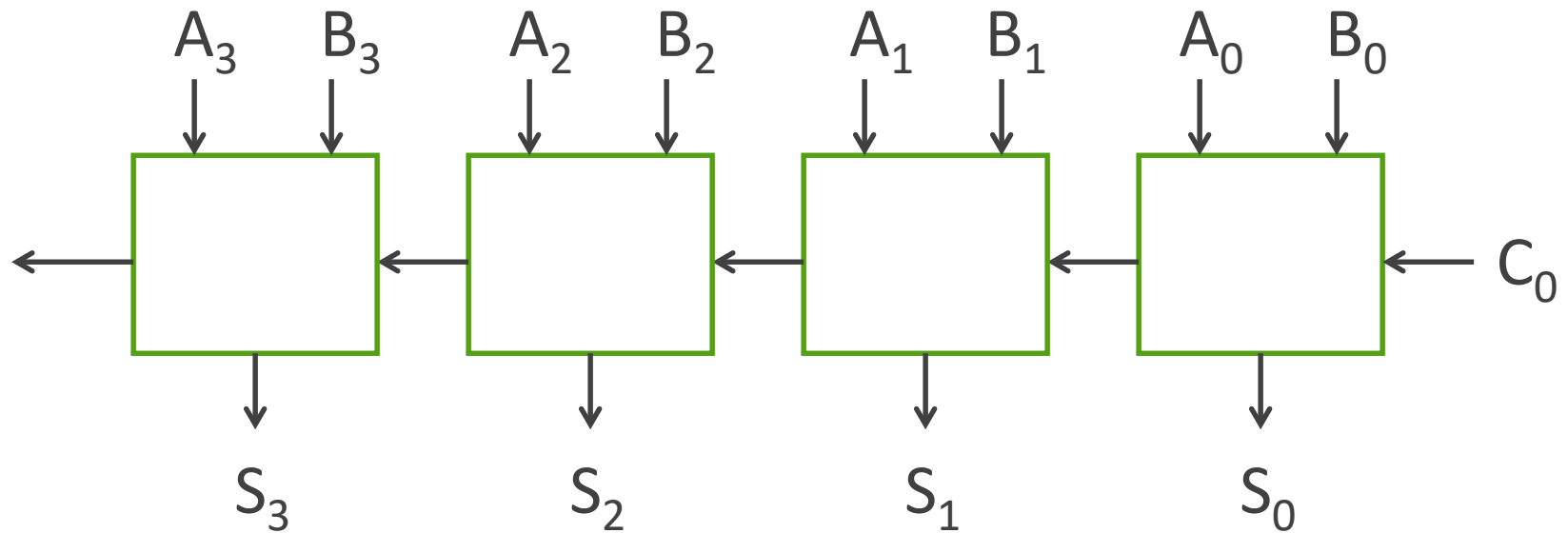
Big Picture: Building a Processor



A single cycle processor

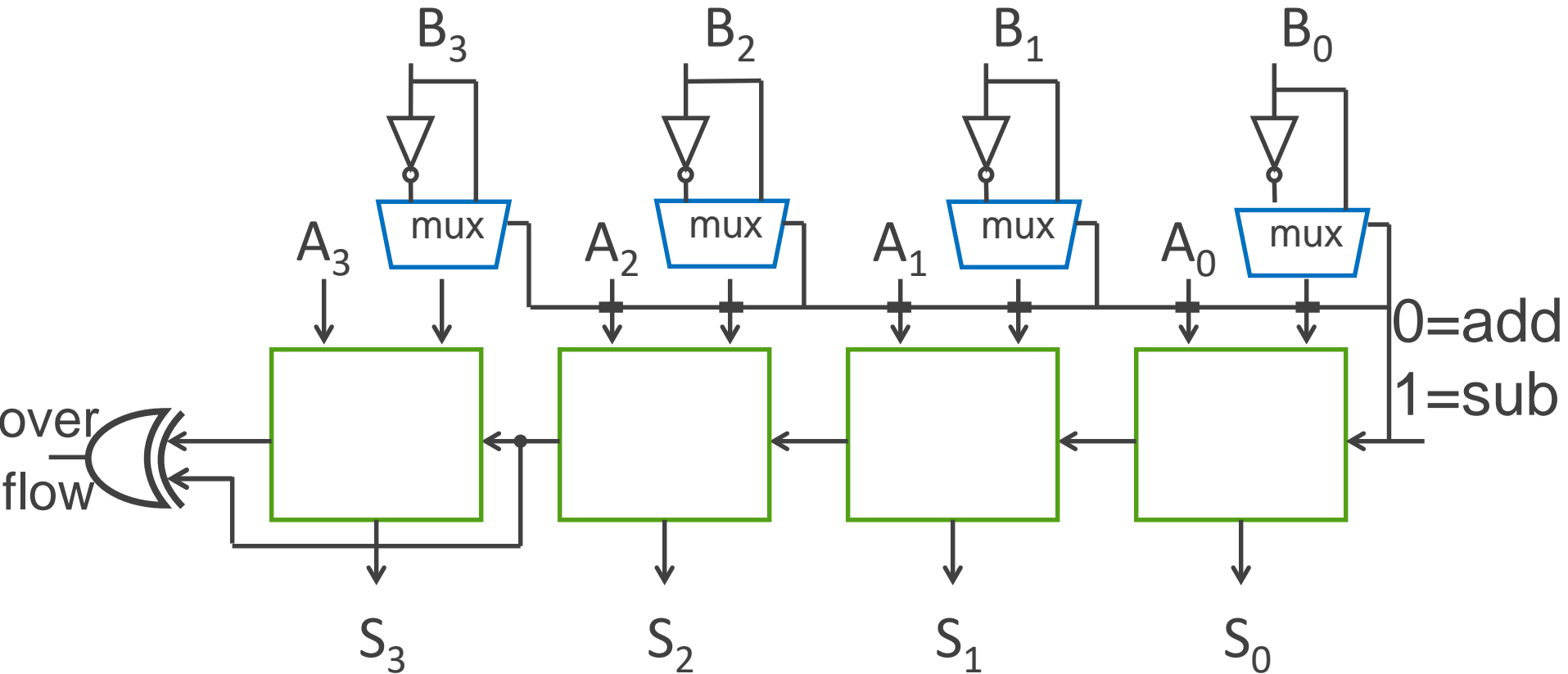
Review

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

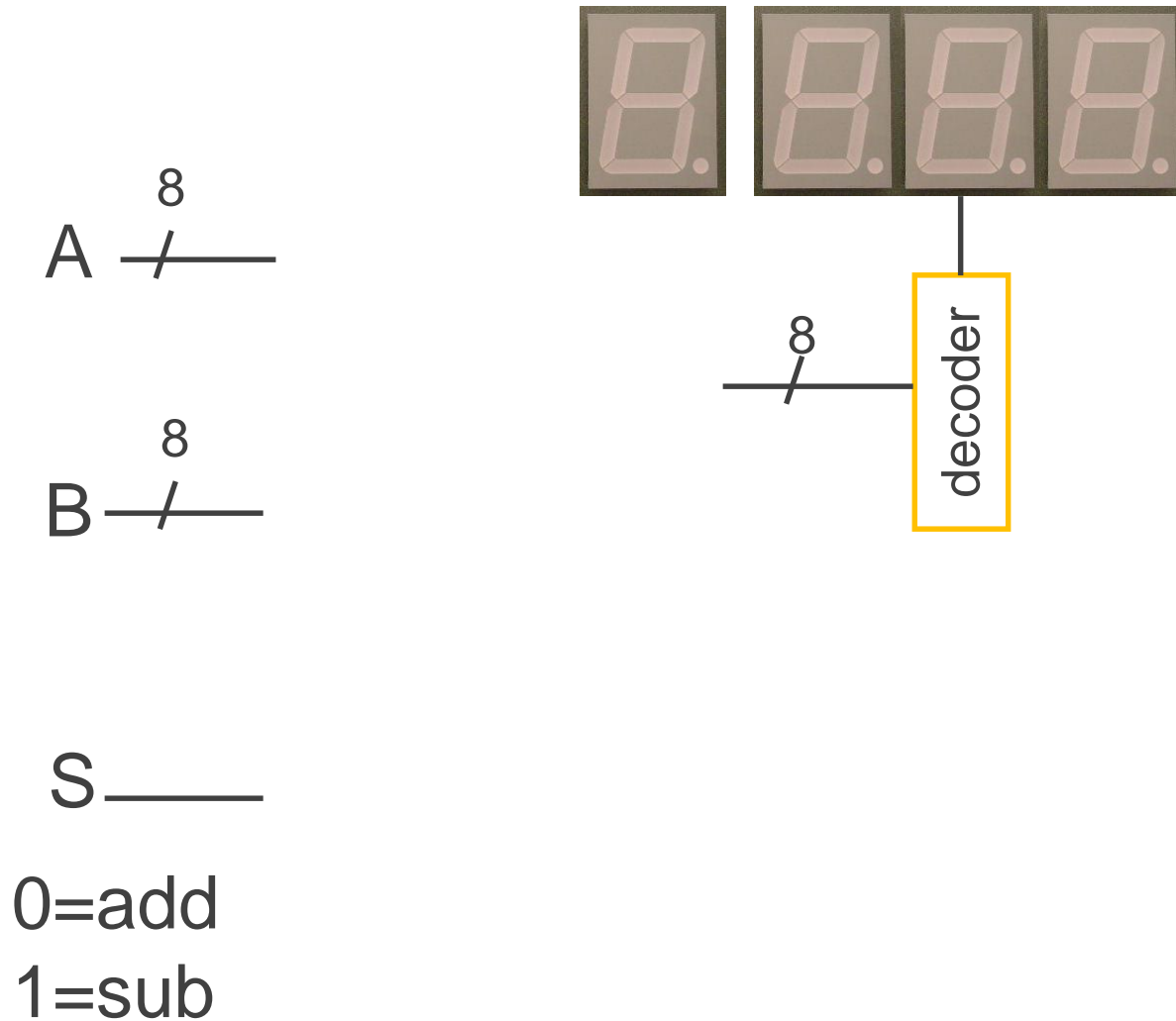


Review

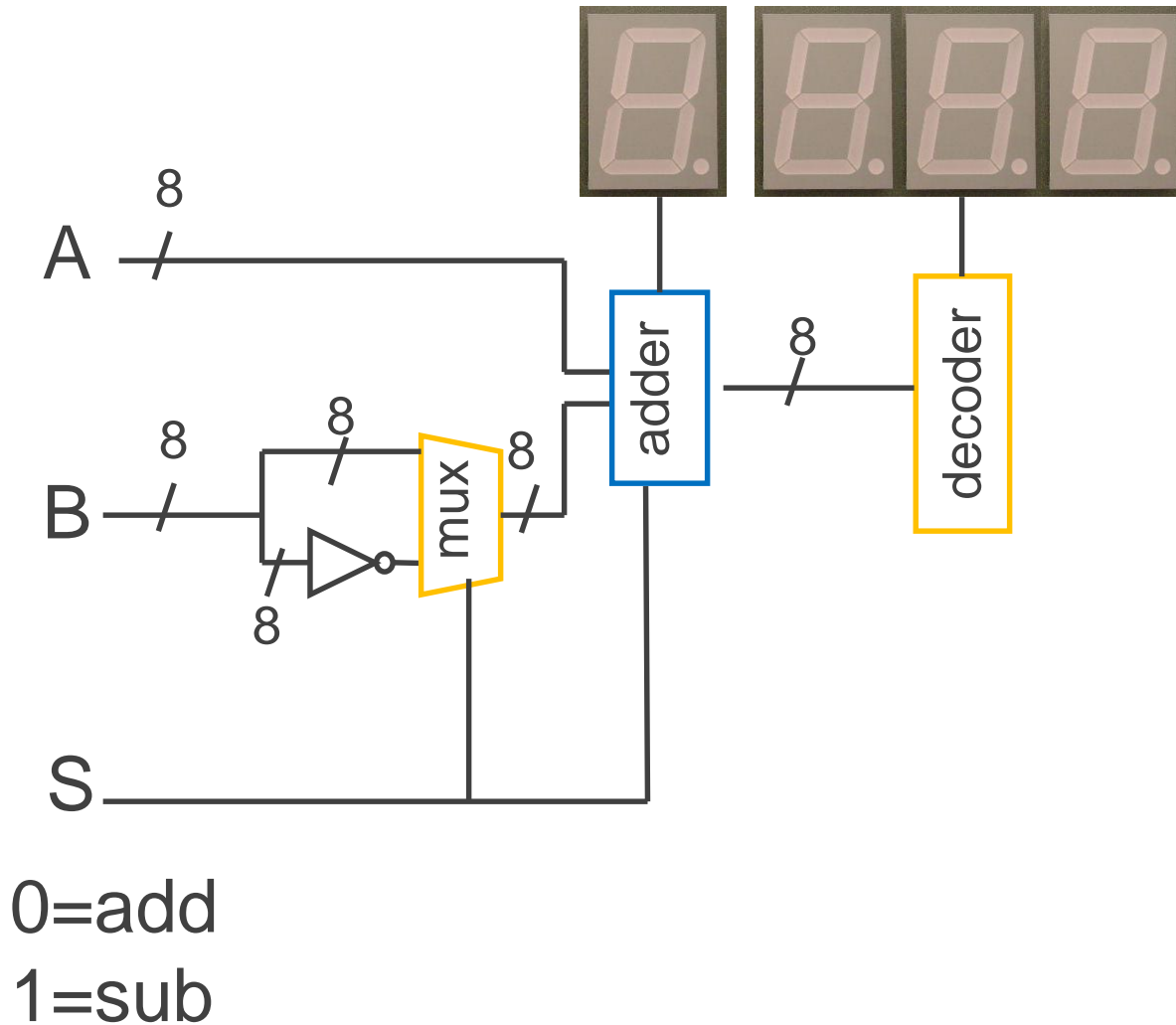
- We can generalize 1-bit Full Adders to 32 bits, 64 bits ...



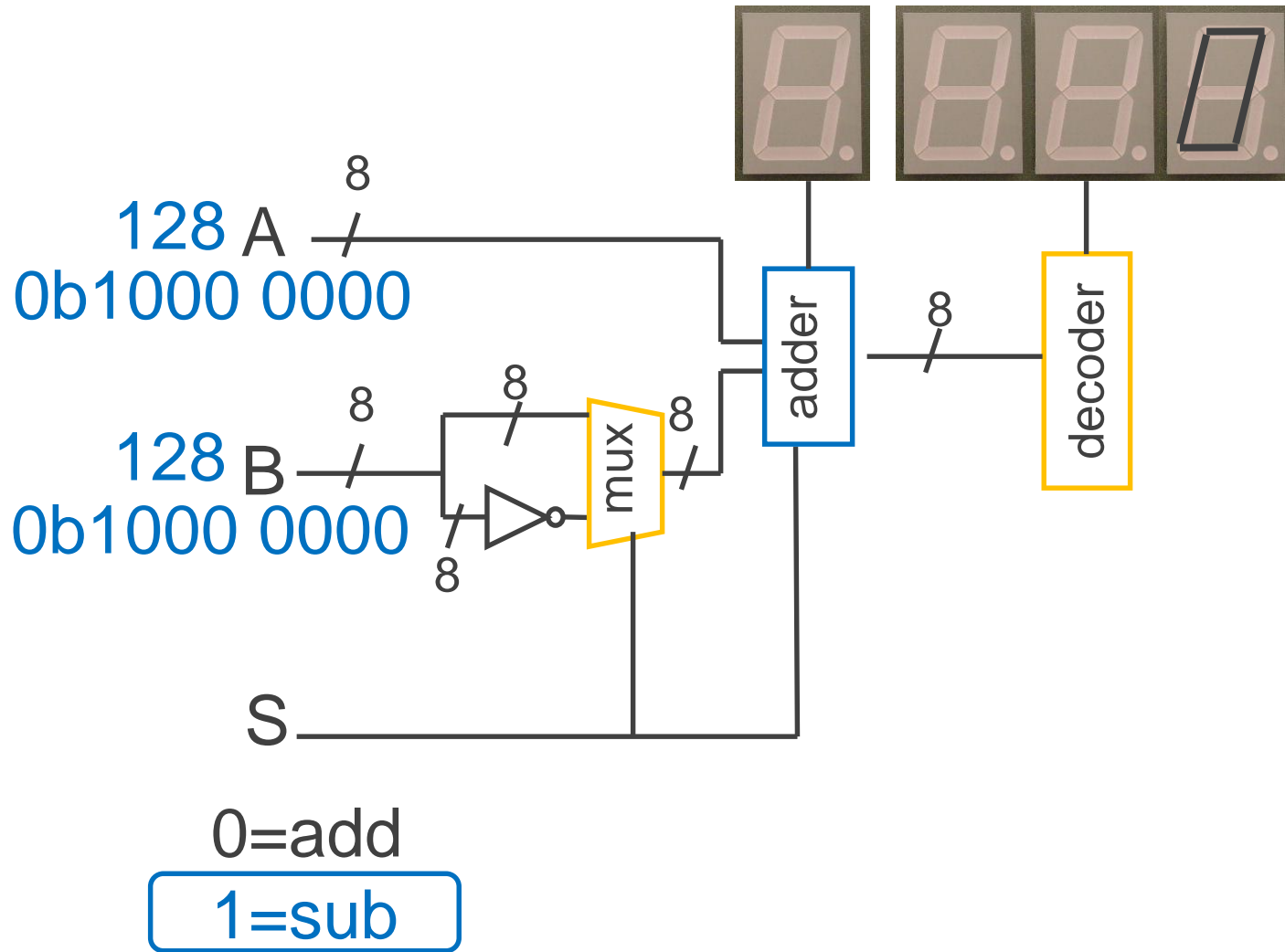
Example: A Calculator



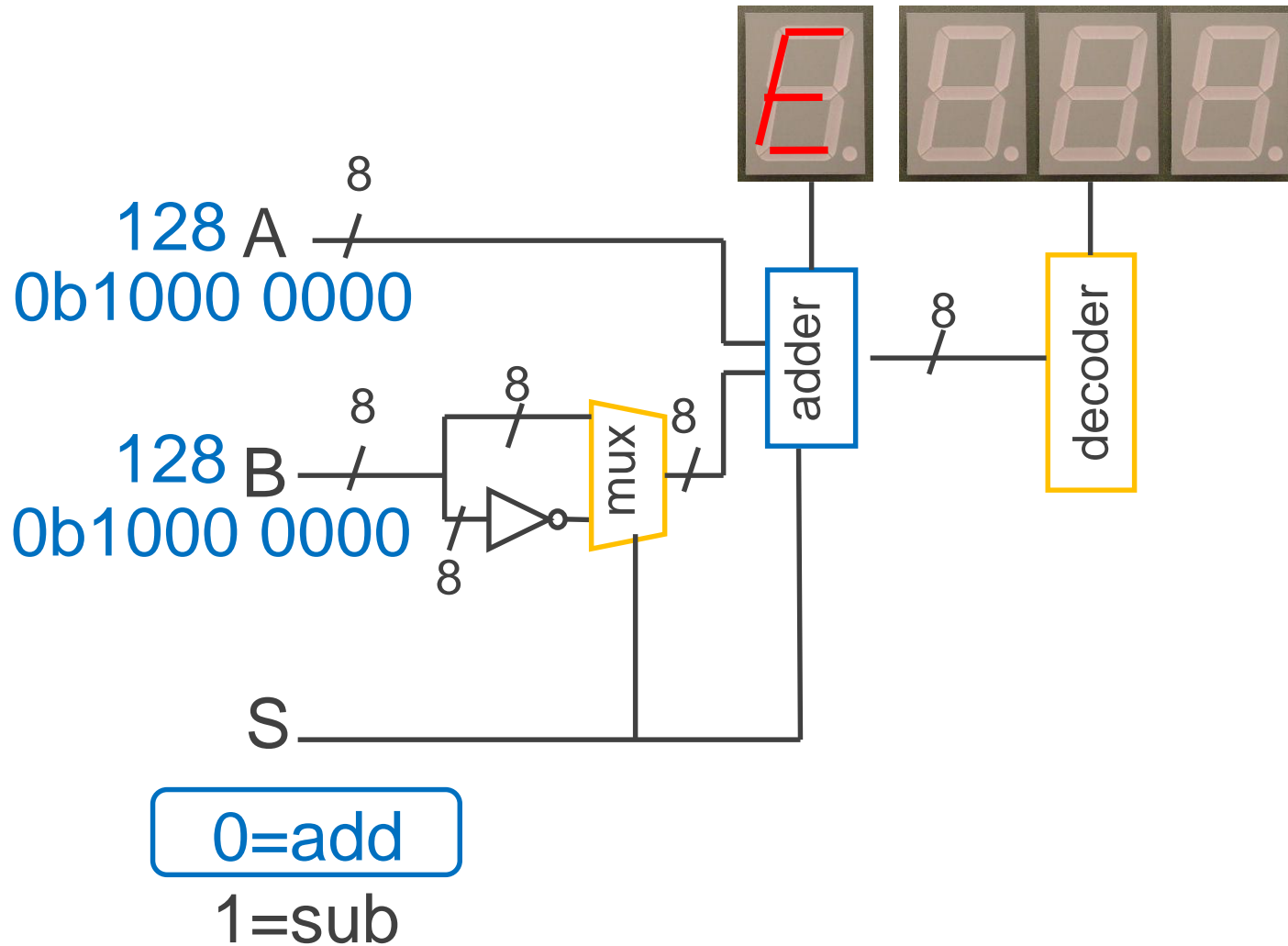
Example: A Calculator



Example: A Calculator

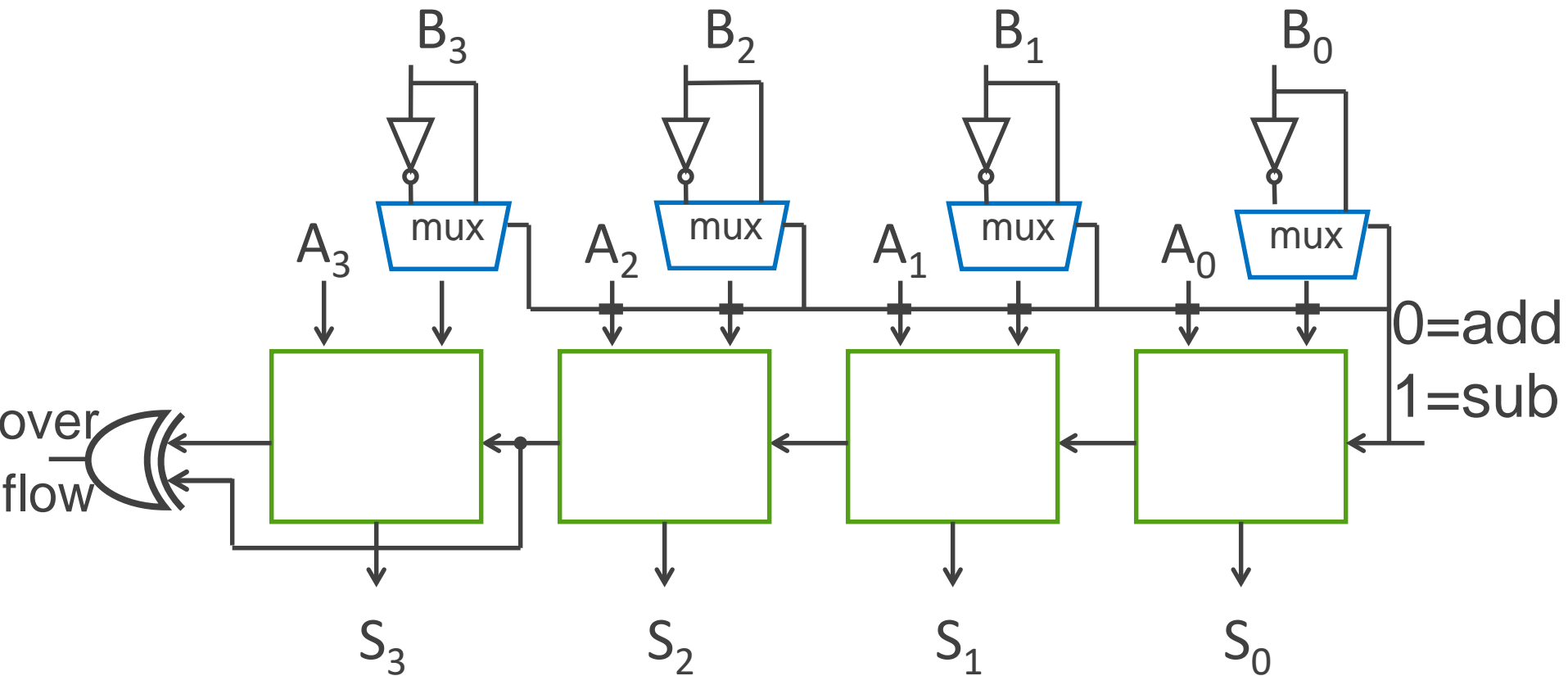


Example: A Calculator



Review: Efficiency and Generality

- We can generalize 1-bit Full Adders to 32 bits, 64 bits
...
- How long does it take to compute a result?

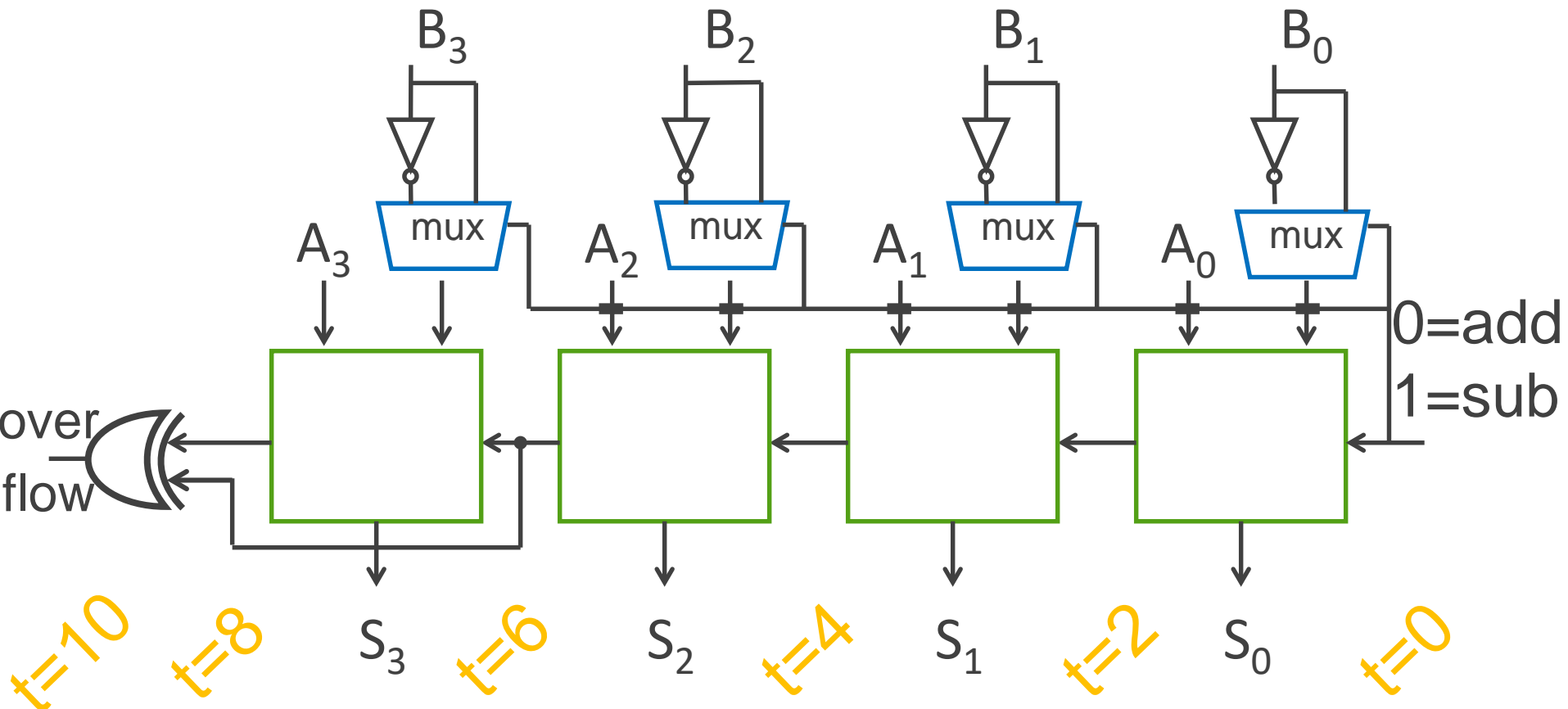


Review: Efficiency and Generality

- We can generalize 1-bit Full Adders to 32 bits, 64 bits
...
- How long does it take to compute a result?
 - A) 2 ns
 - B) 2 gate delays
 - C) 10 ns
 - D) 10 gate delays
 - E) 8 gate delays

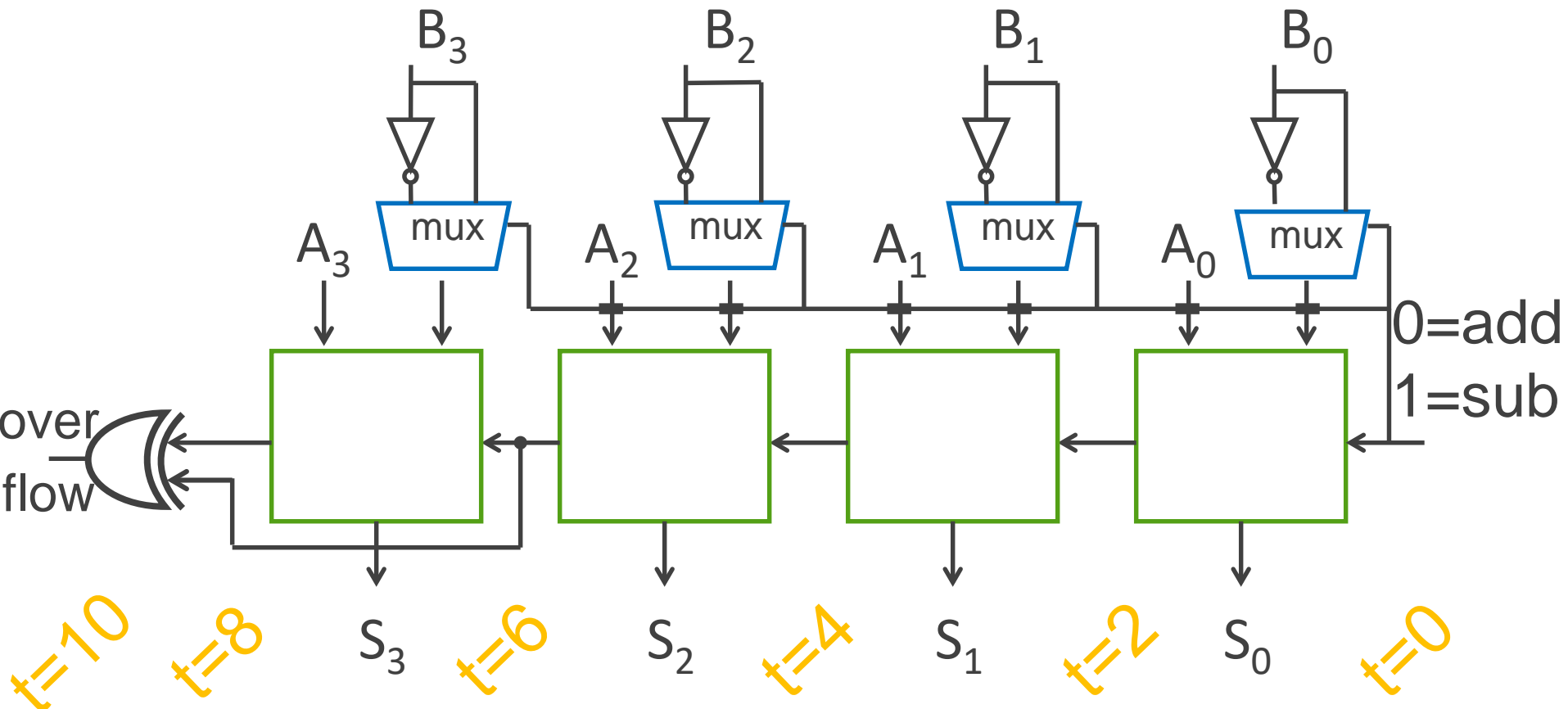
Review: Efficiency and Generality

- We can generalize 1-bit Full Adders to 32 bits, 64 bits
...
- How long does it take to compute a result?



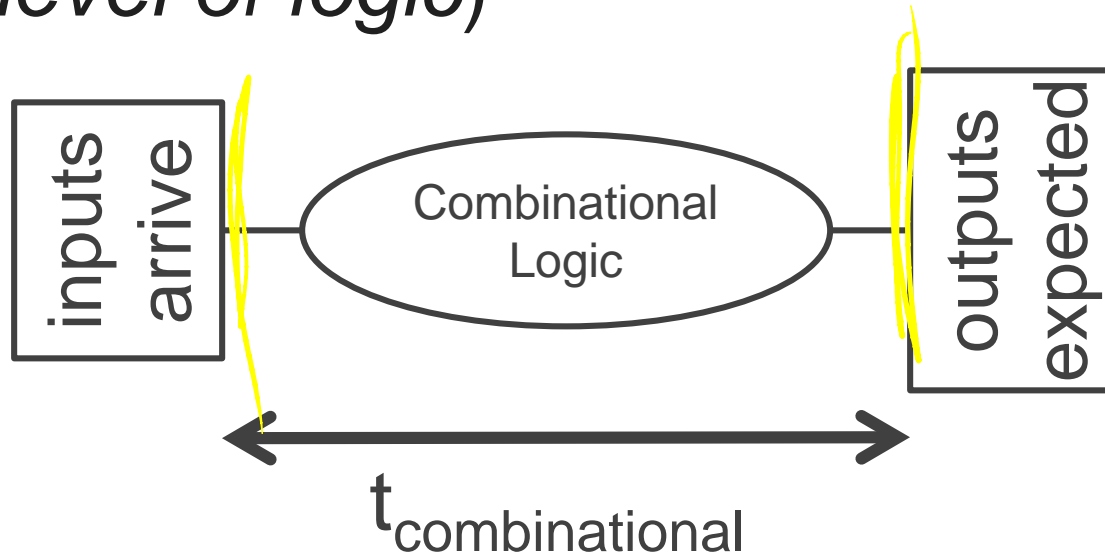
Review: Efficiency and Generality

- We can generalize 1-bit Full Adders to 32 bits, 64 bits ...
- How long does it take to compute a result?
- Can we **store** the result?

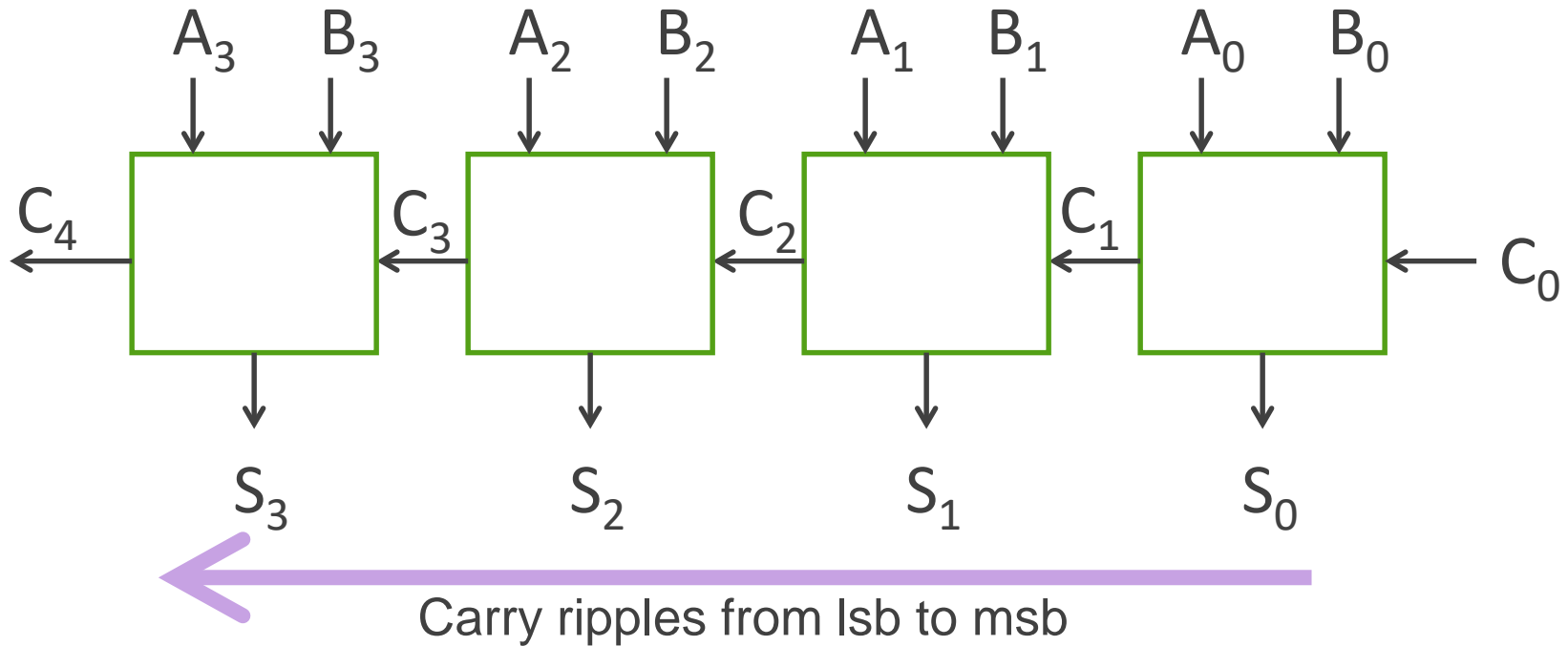


Performance

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

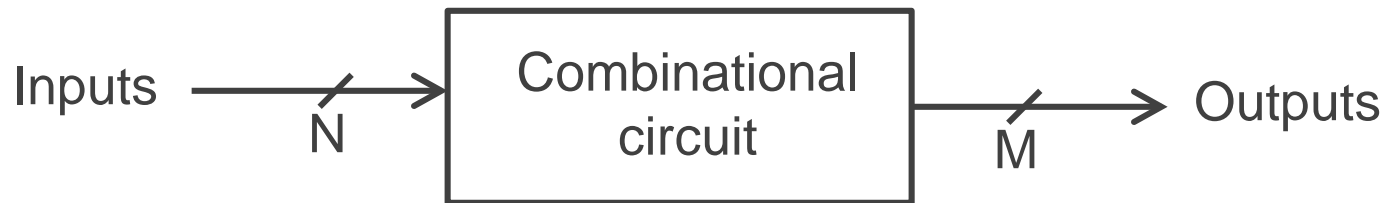


- First full adder, 2 gate delay
- Second full adder, 2 gate delay
- ...

Stateful Components

Until now is **combinational** 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

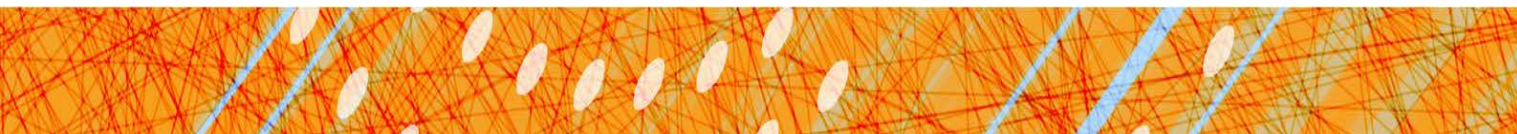
Goals for Today

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

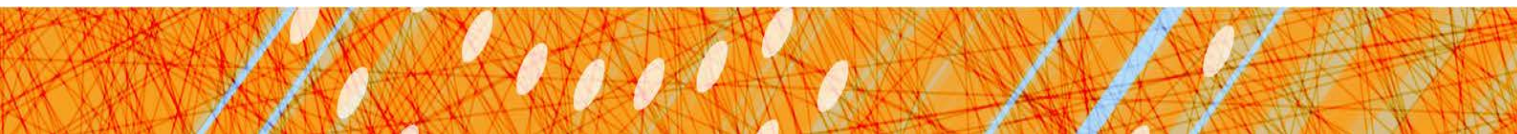
Basic Building Blocks

- Decoders and Encoders

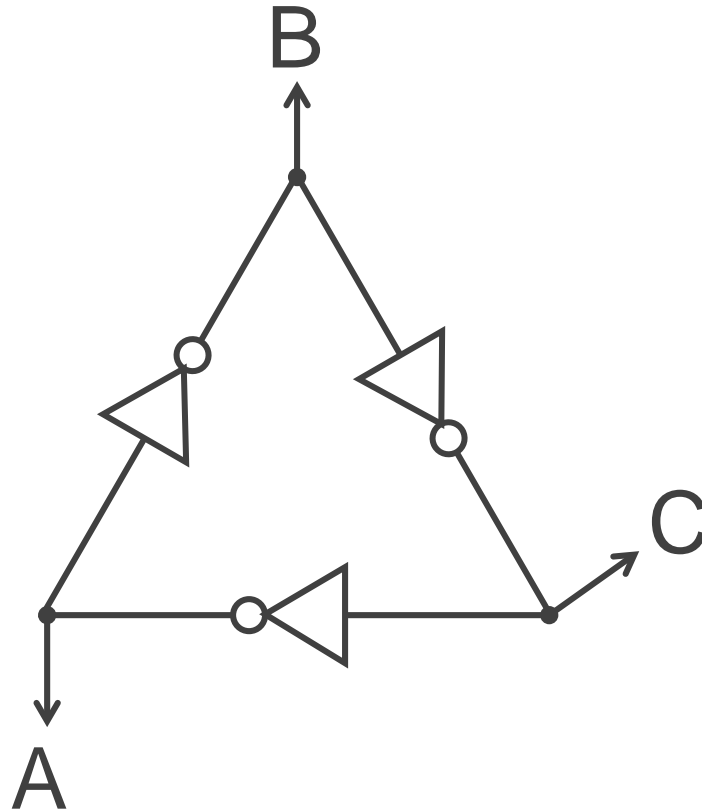


Goal

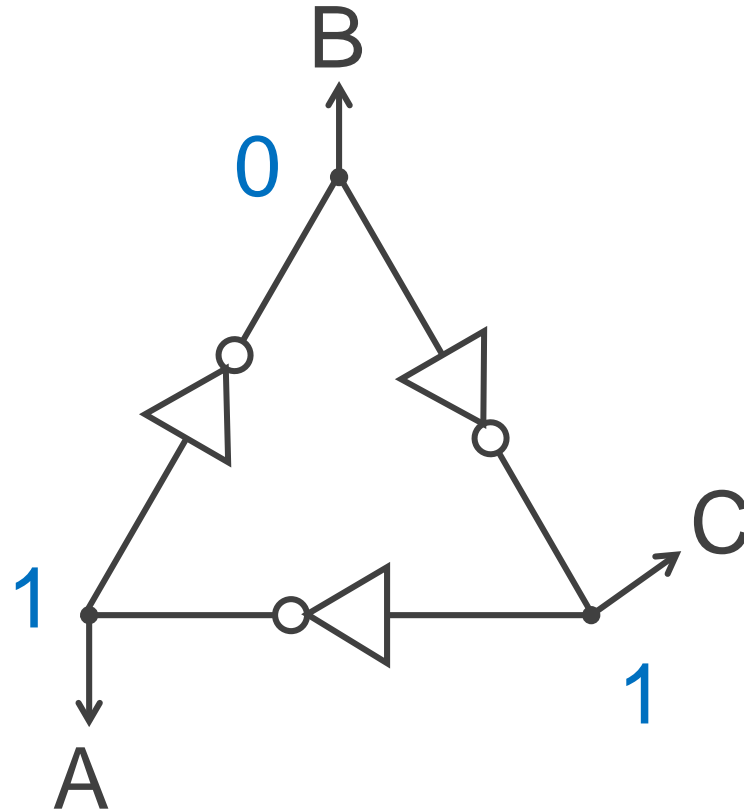
How do we store store **one** bit?



First Attempt: Unstable Devices



First Attempt: Unstable Devices

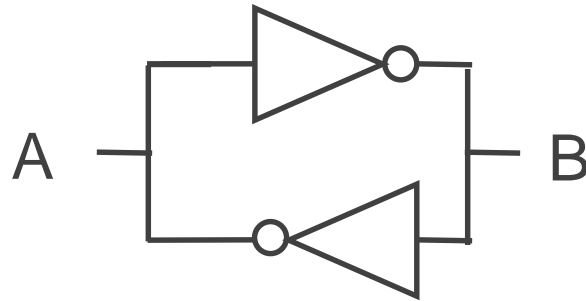


Does not work!

- Unstable
- Oscillates wildly!

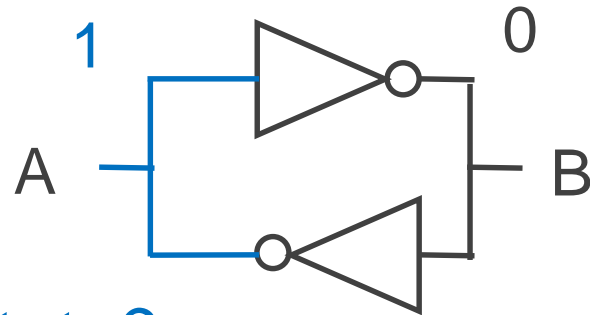
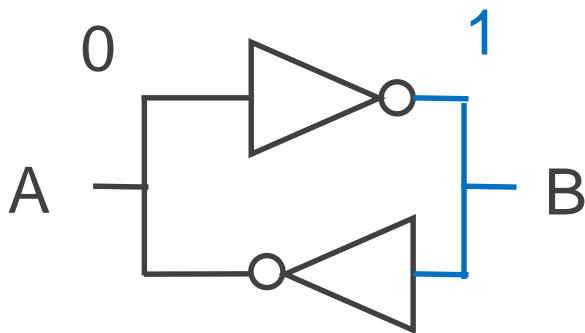
Second Attempt: Bistable Devices

- Stable and unstable equilibria?



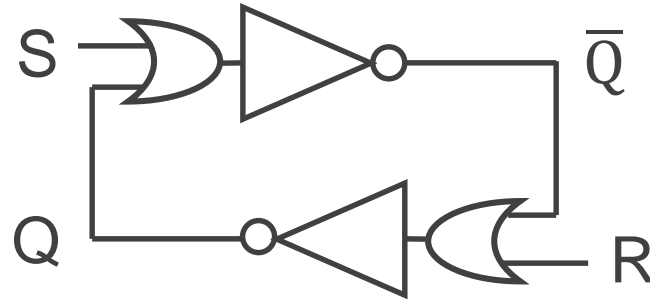
A Simple Device

In stable state, $A = B$

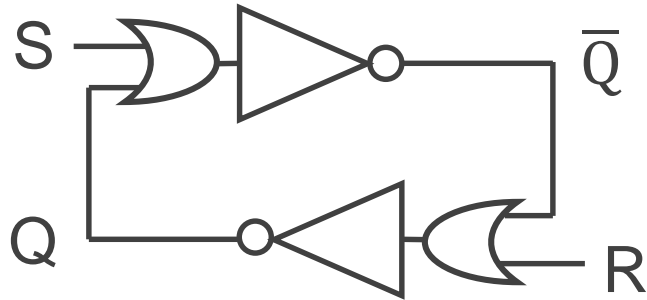


How do we change the state?

Third Attempt: Set-Reset Latch



Third Attempt: Set-Reset Latch



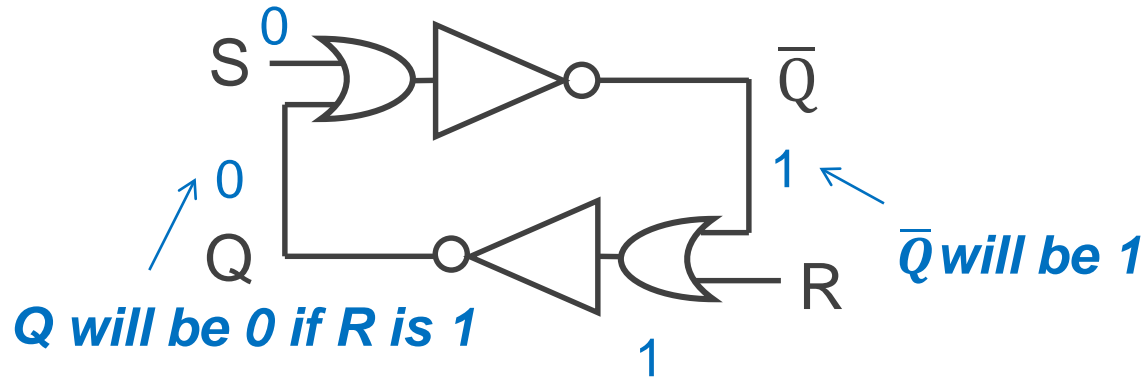
S	R	Q	\bar{Q}
0	0		
0	1		
1	0		
1	1		

Set-Reset (S-R) Latch

Stores a value Q and its complement

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

Third Attempt: Set-Reset Latch



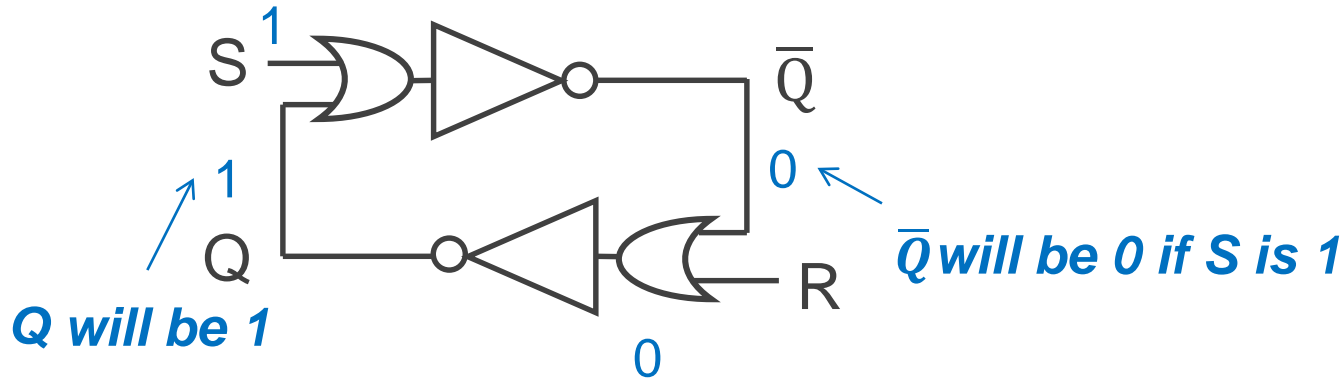
A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

S	R	Q	\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



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

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

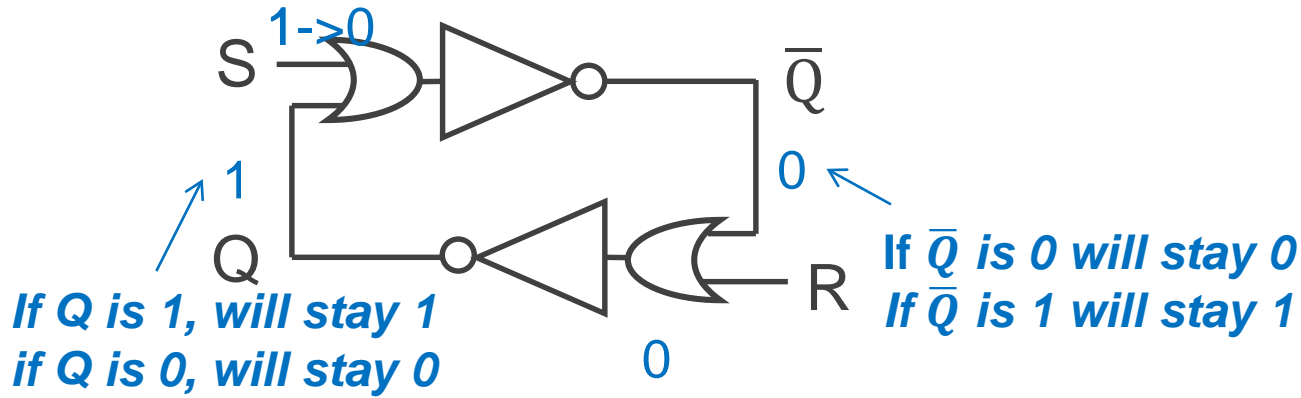
S	R	Q	\bar{Q}
0	0		
0	1	0	1
1	0	1	0
1	1		

What are the values for Q and \bar{Q} ?

- a) 0 and 0
- b) 0 and 1
- c) 1 and 0**
- d) 1 and 1

iClicker Question

Third Attempt: Set-Reset Latch



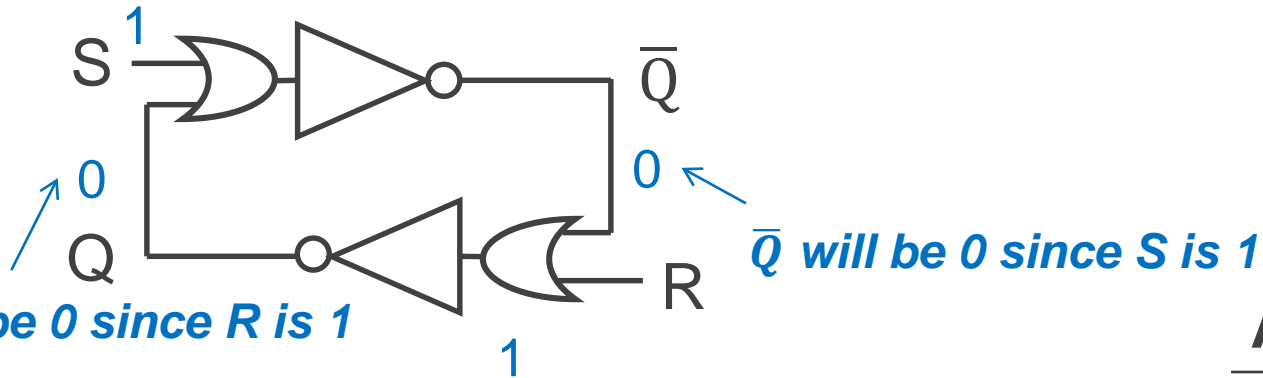
S	R	Q	\bar{Q}
0	0	Q	\bar{Q}
0	1	0	1
1	0	1	0
1	1		

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

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

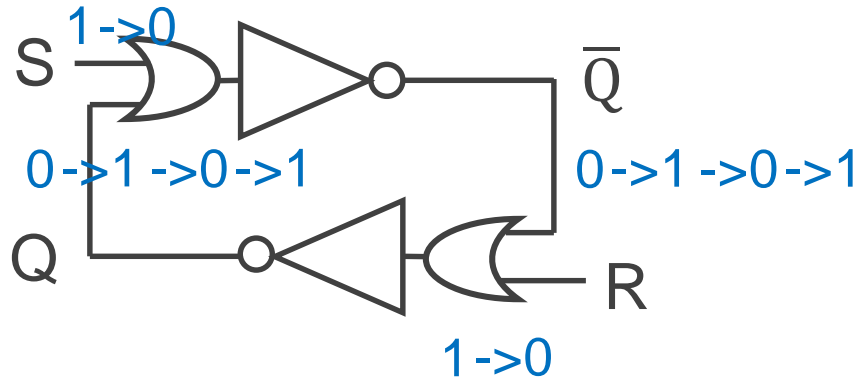
What happens when S,R changes from 1,1 to 0,0?

iClicker Question

What's wrong with the SR Latch?

- A. Q is undefined when $S=1$ and $R=1$
(That's why this is called the forbidden state.)
- B. Q oscillates between 0 and 1 when the inputs transition from $1,1 \rightarrow 0,0$
- C. The SR Latch is problematic b/c it has two outputs to store a single bit.
- D. There is nothing wrong with the SR 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	Q̄
0	0	Q	Q̄
0	1	0	1
1	0	1	0
1	1	forbidden	

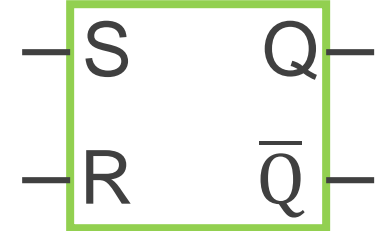
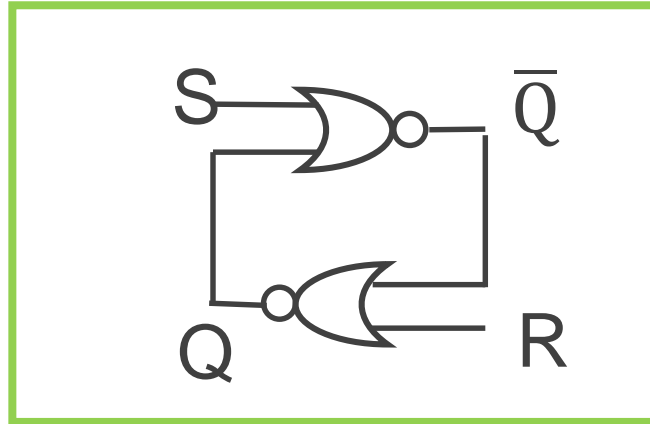
Set-Reset (S-R) Latch

Stores a value Q and its complement

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

Q and Q̄ become unstable and will oscillate wildly between values 0,0 to 1,1 to 0,0 to 1,1 ...

Third Attempt: Set-Reset Latch



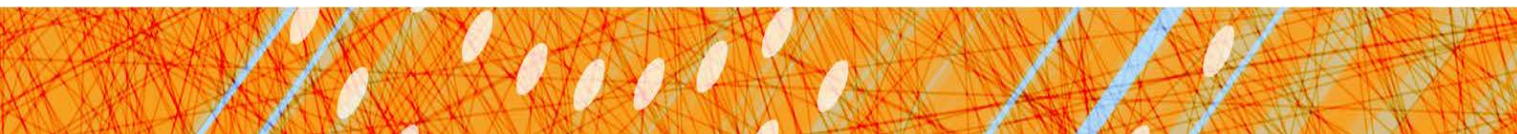
S	R	Q	\bar{Q}	
0	0	Q	\bar{Q}	hold
0	1	0	1	reset
1	0	1	0	set
1	1	forbidden		

Set-Reset (S-R) Latch

Stores a value Q and its complement

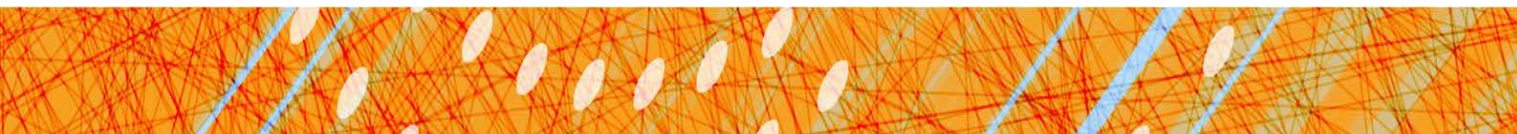
Takeaway

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.

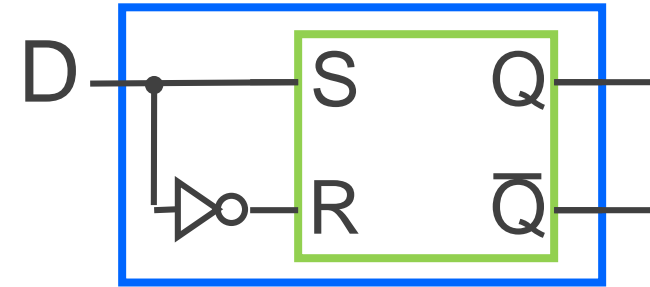
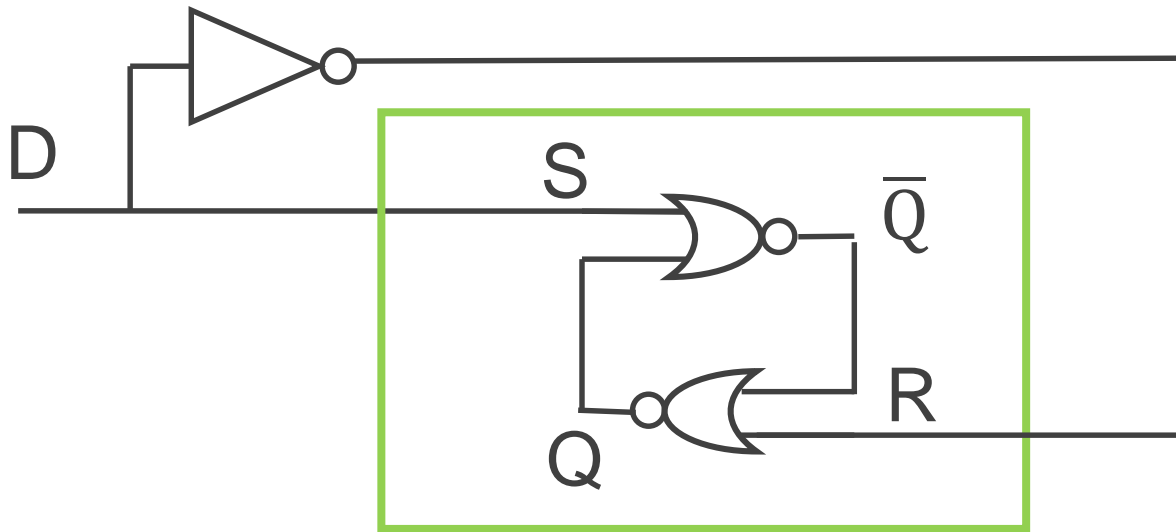


Next Goal

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



Fourth Attempt: (Unclocked) D Latch

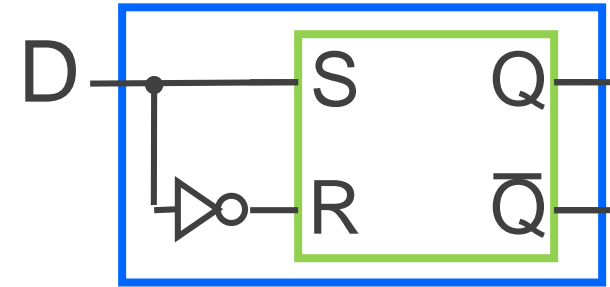
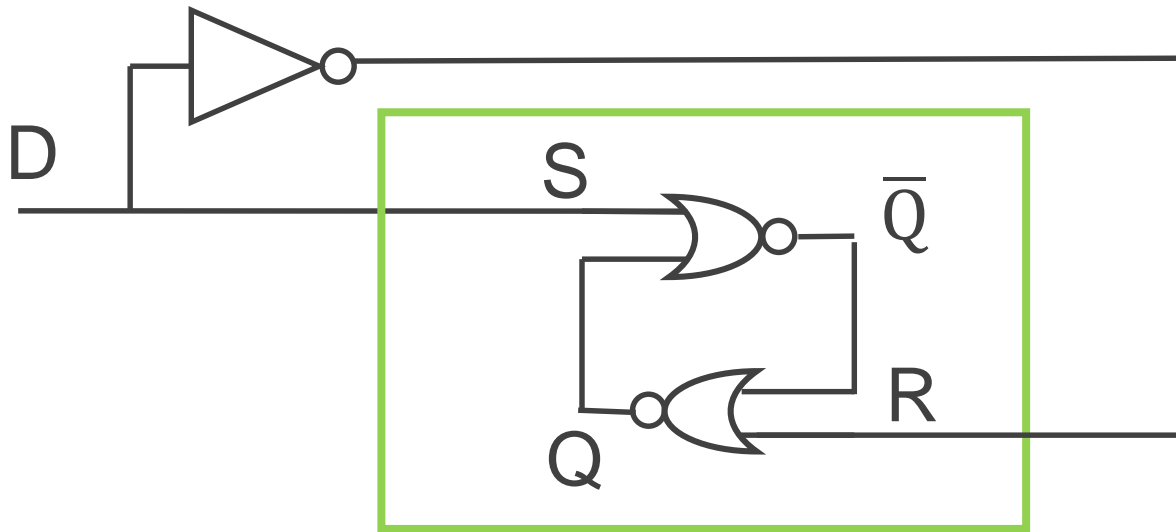


Fill in the truth table?

D	Q	\bar{Q}
0		
1		

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

Fourth Attempt: (Unclocked) D Latch



Fill in the truth table?

Data (D) 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

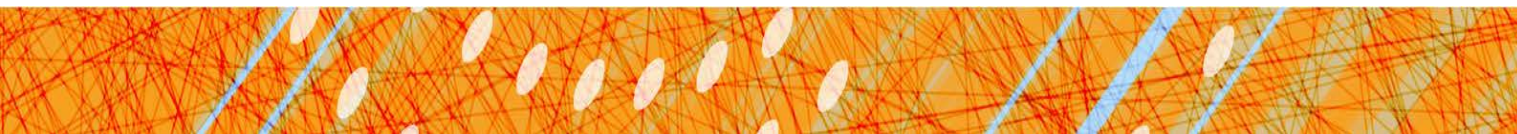
D	Q	\bar{Q}
0	0	1
1	1	0

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

Takeaway

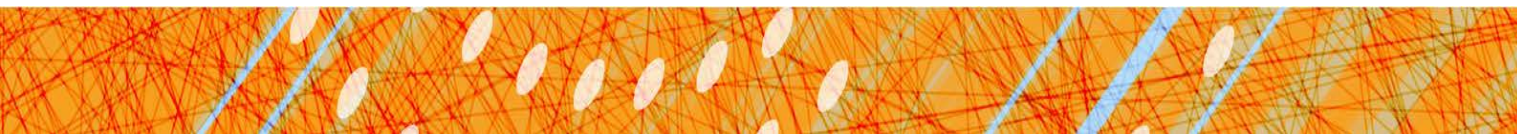
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding the forbidden state.



Next Goal

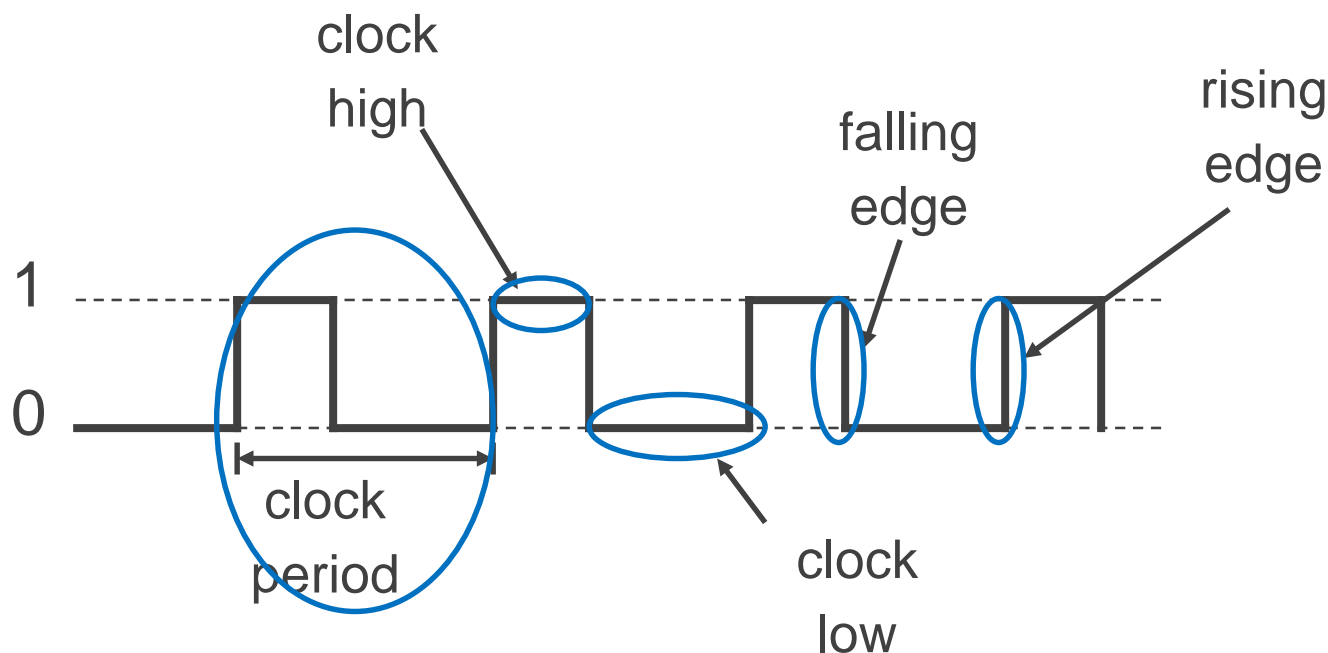
How do we coordinate state changes to a D Latch?



Aside: Clocks

Clock helps coordinate state changes

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



Clock Disciplines

Level sensitive

- State changes when clock is high (or low)



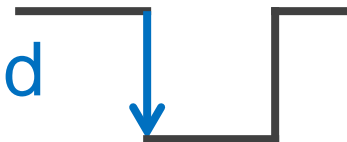
Edge triggered

- State changes at clock edge

positive edge-triggered



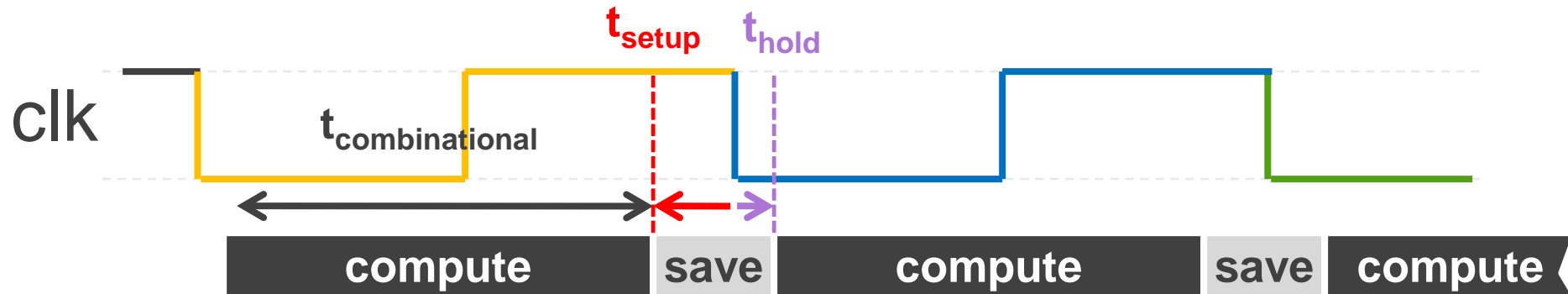
negative edge-triggered



Clock Methodology

Clock Methodology

- Negative edge, synchronous

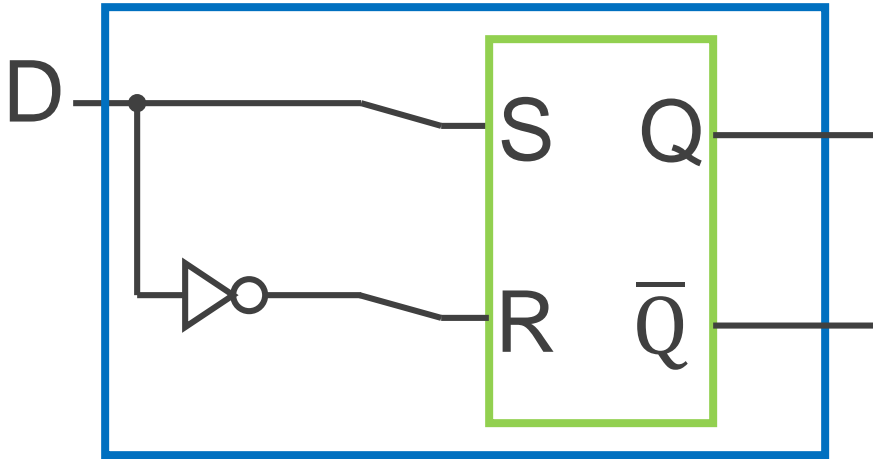


Edge-Triggered \rightarrow signals must be stable near falling edge

“near” = before and after

t_{setup} t_{hold}

Round 2: D Latch (1)

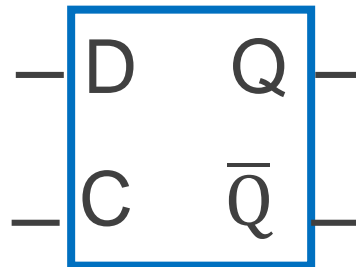


- Inverter prevents SR Latch from entering 1,1 state

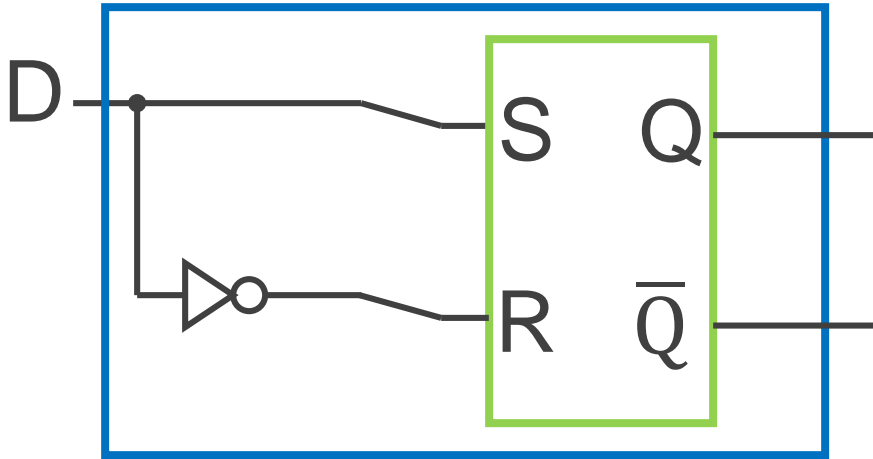
	D	Q	Q
0			
1			

Reset

Set



Round 2: D Latch (1)

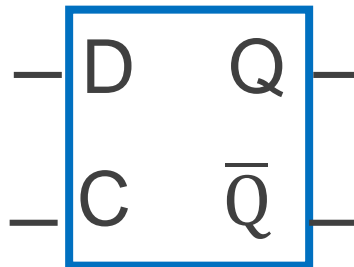


- Inverter prevents SR Latch from entering 1,1 state

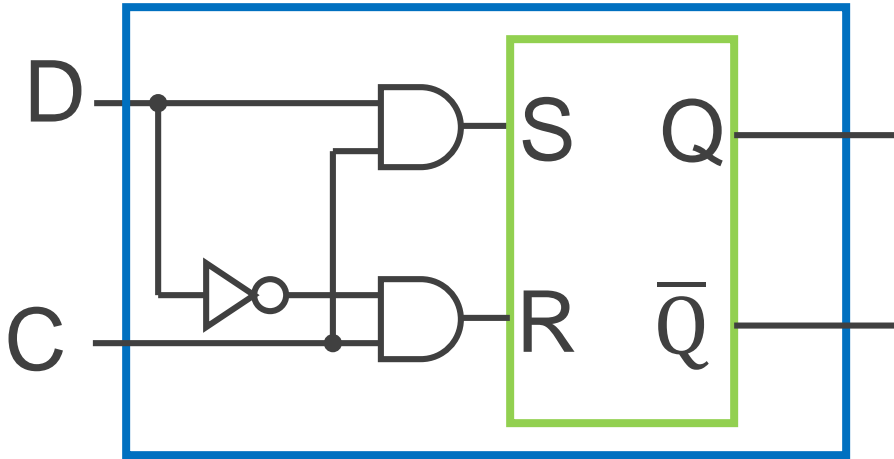
	D	Q	\bar{Q}
	0	0	1
	1	1	0

Reset

Set



Round 2: D Latch (1)



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

C = 1, D Latch *transparent*:
set/reset (according to D)

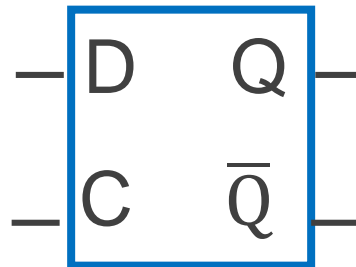
C = 0, D Latch *opaque*:
keep state (ignore D)

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

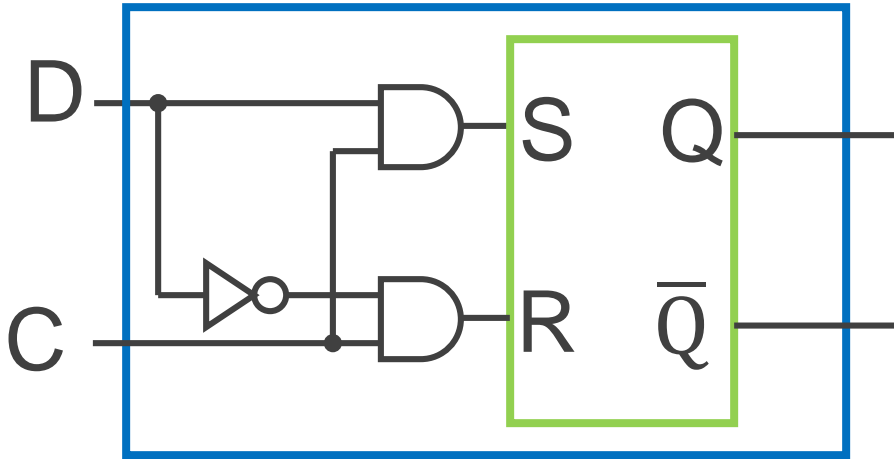
No
Change

Reset

Set



Round 2: D Latch (1)

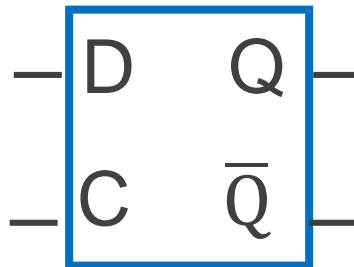


- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- C enables changes

C = 1, D Latch *transparent*: set/reset (according to D)

C = 0, D Latch *opaque*: keep state (ignore D)

S	R	Q	Q	
0	0	Q	Q	hold
0	1	0	1	reset
1	0	1	0	set
1	1	forbidden		



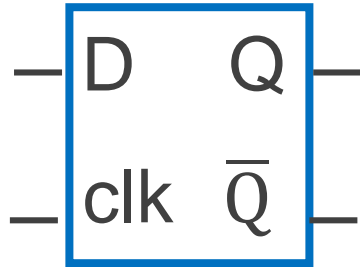
C	D	Q	Q
0	0	Q	Q
0	1	Q	Q
1	0	0	1
1	1	1	0

No
Change

Reset

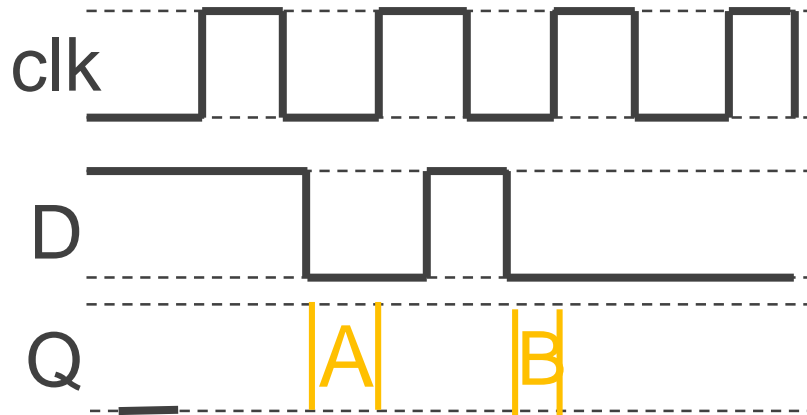
Set

iClicker Question



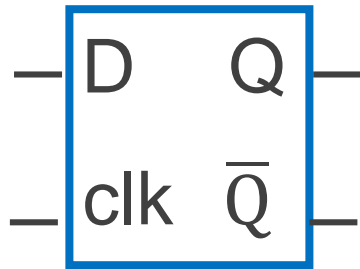
What is the value of Q at A & B?

- a) $A = 0, B = 0$
- b) $A = 0, B = 1$
- c) $A = 1, B = 0$
- d) $A = 1, B = 1$



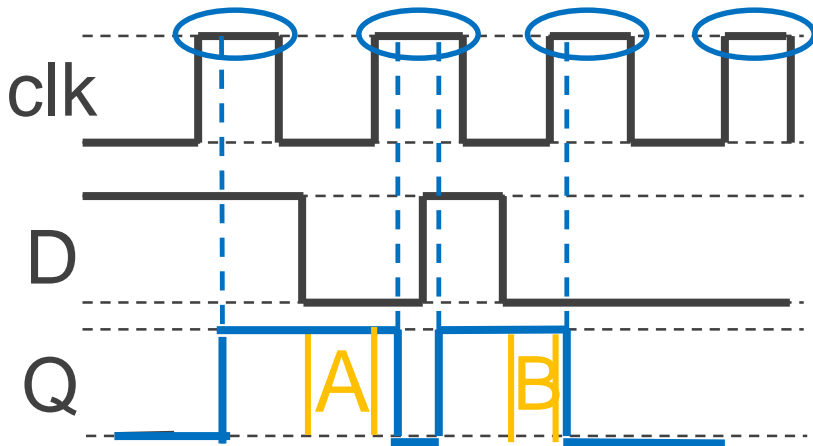
clk	D	Q	Q
0	0	Q	Q
0	1	Q	Q
1	0	0	1
1	1	1	0

iClicker Question



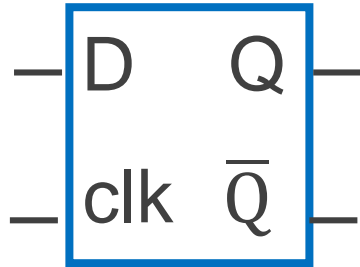
What is the value of Q at A & B?

- a) $A = 0, B = 0$
- b) $A = 0, B = 1$
- c) $A = 1, B = 0$
- d) $A = 1, B = 1$



clk	D	Q	Q
0	0	Q	Q
0	1	Q	Q
1	0	0	1
1	1	1	0

iClicker Question



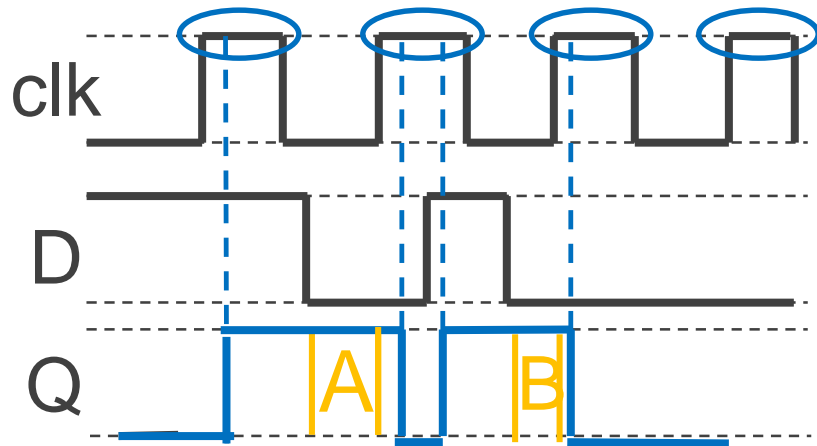
Level Sensitive D Latch

Clock high:

set/reset (according to D)

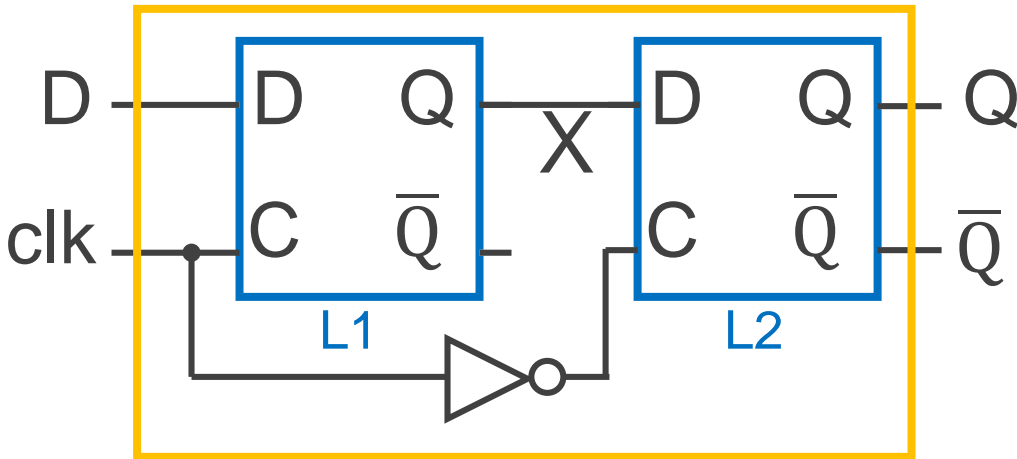
Clock low:

keep state (ignore D)



clk	D	Q	Q
0	0	Q	Q
0	1	Q	Q
1	0	0	1
1	1	1	0

Round 3: D Flip-Flop

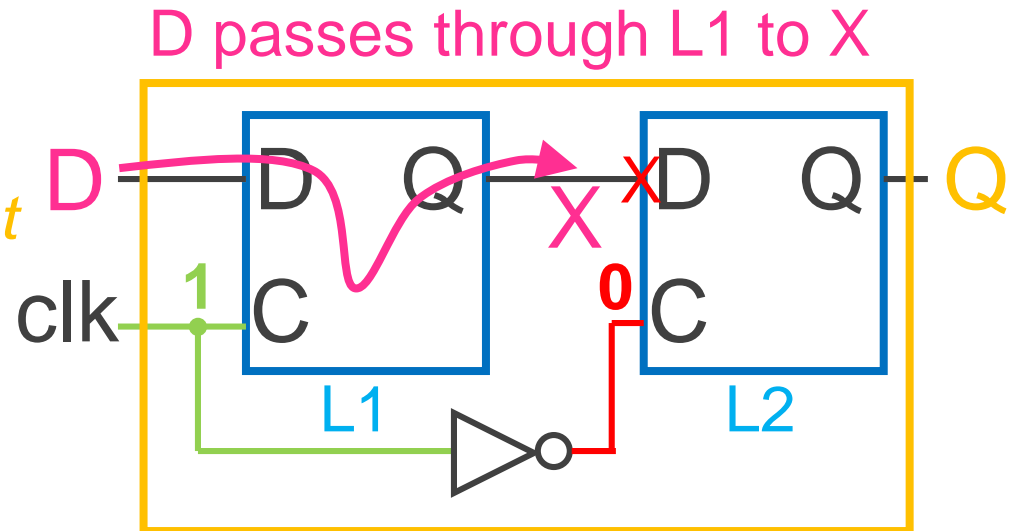


- Edge-Triggered
- Data captured when clock high
- Output changes only on falling edges

Round 3: D Flip-Flop

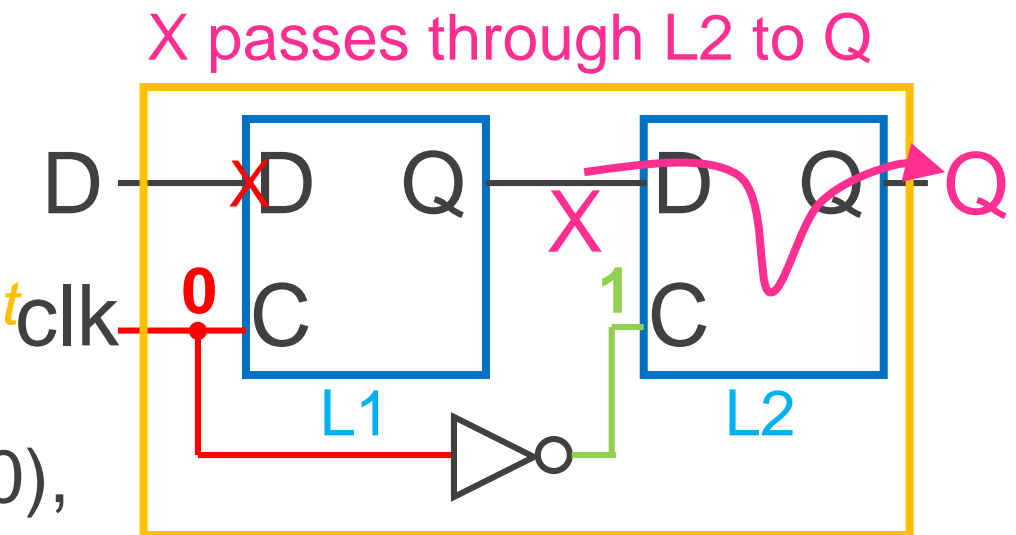
Clock = 1: L1 *transparent*
L2 *opaque*

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

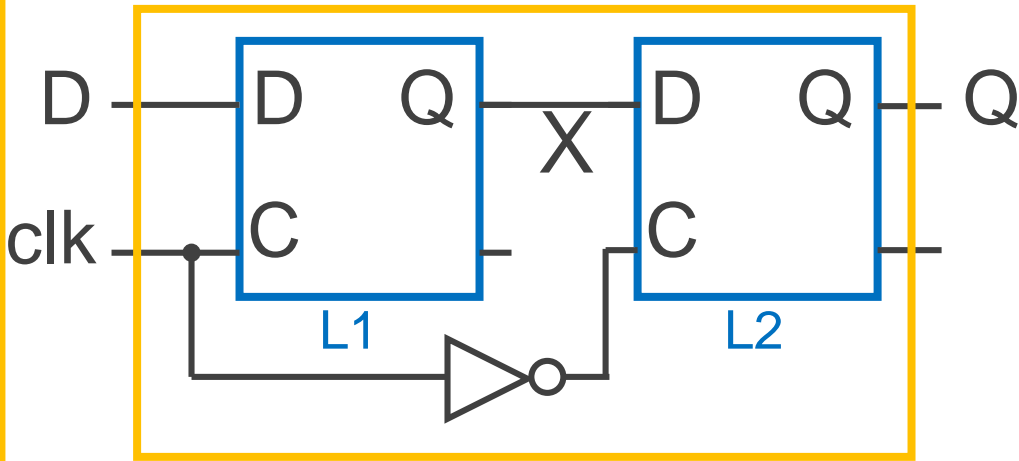


Clock = 0: L1 *opaque*
L2 *transparent*

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

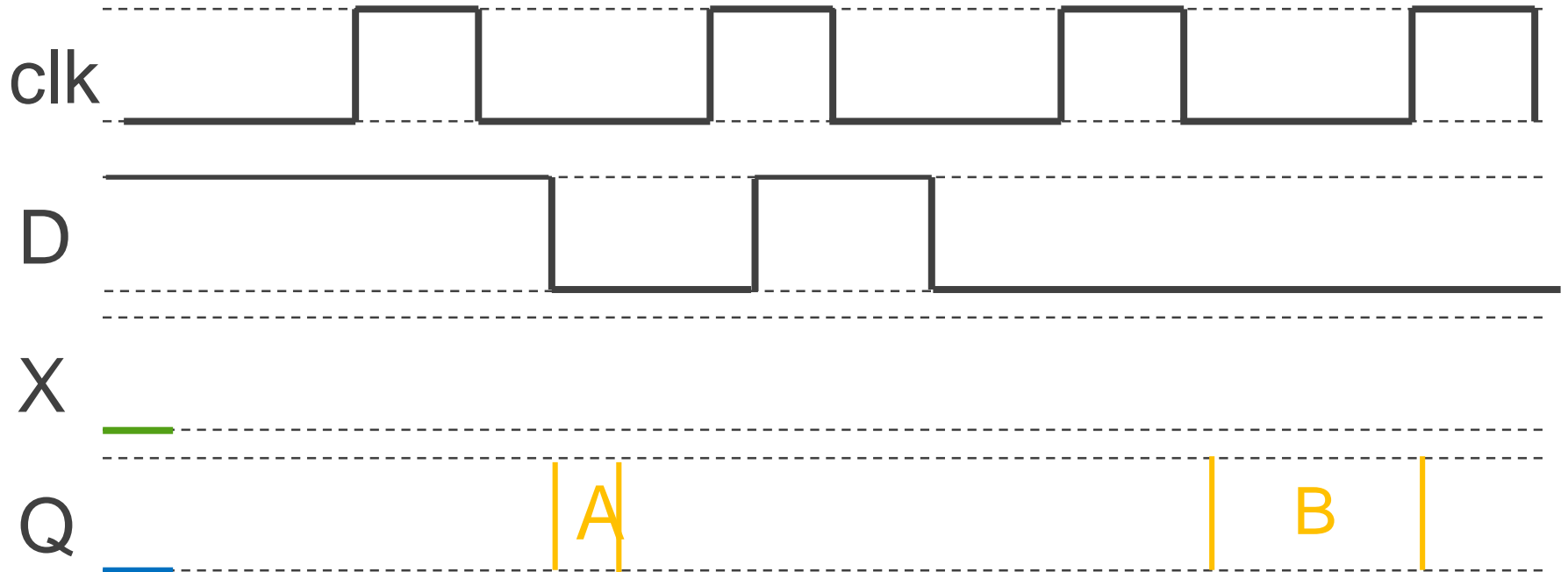


iClicker Question – start here

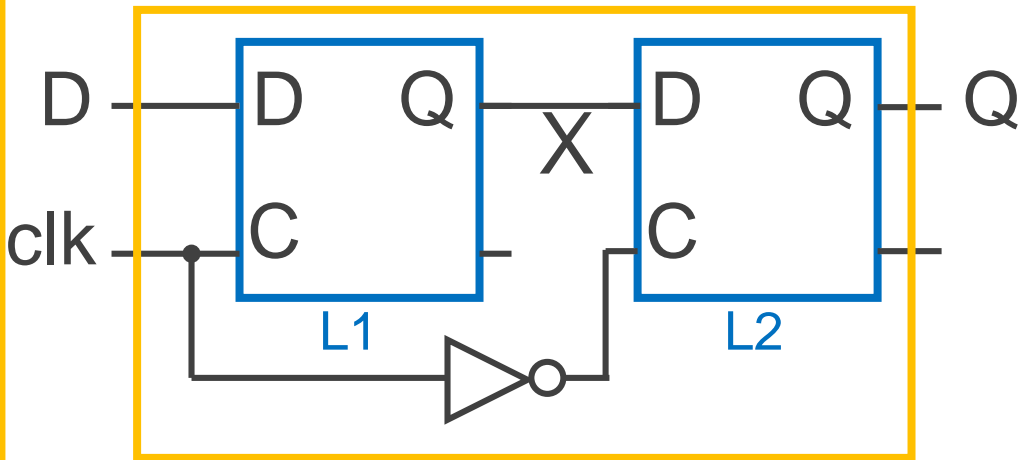


What is the value of Q at A & B?

- a) $A = 0, B = 0$
- b) $A = 0, B = 1$
- c) $A = 1, B = 0$
- d) $A = 1, B = 1$

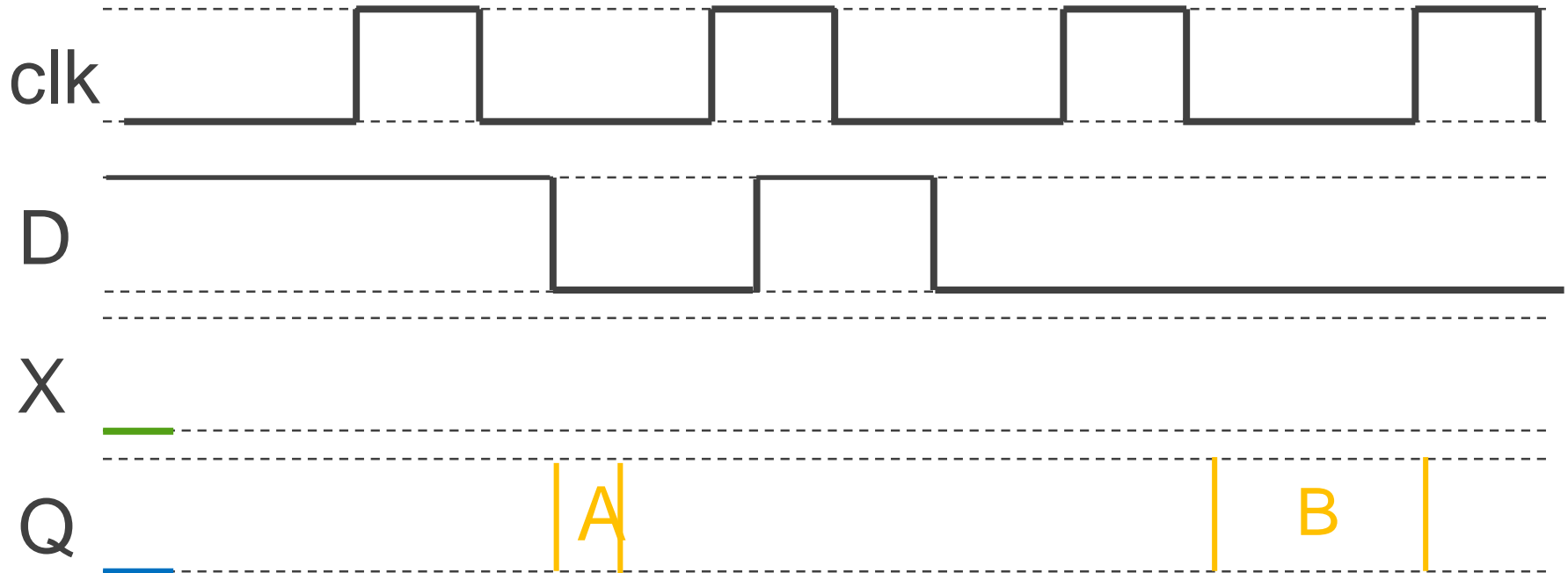


iClicker Question – start here



What is the value of Q at A & B?

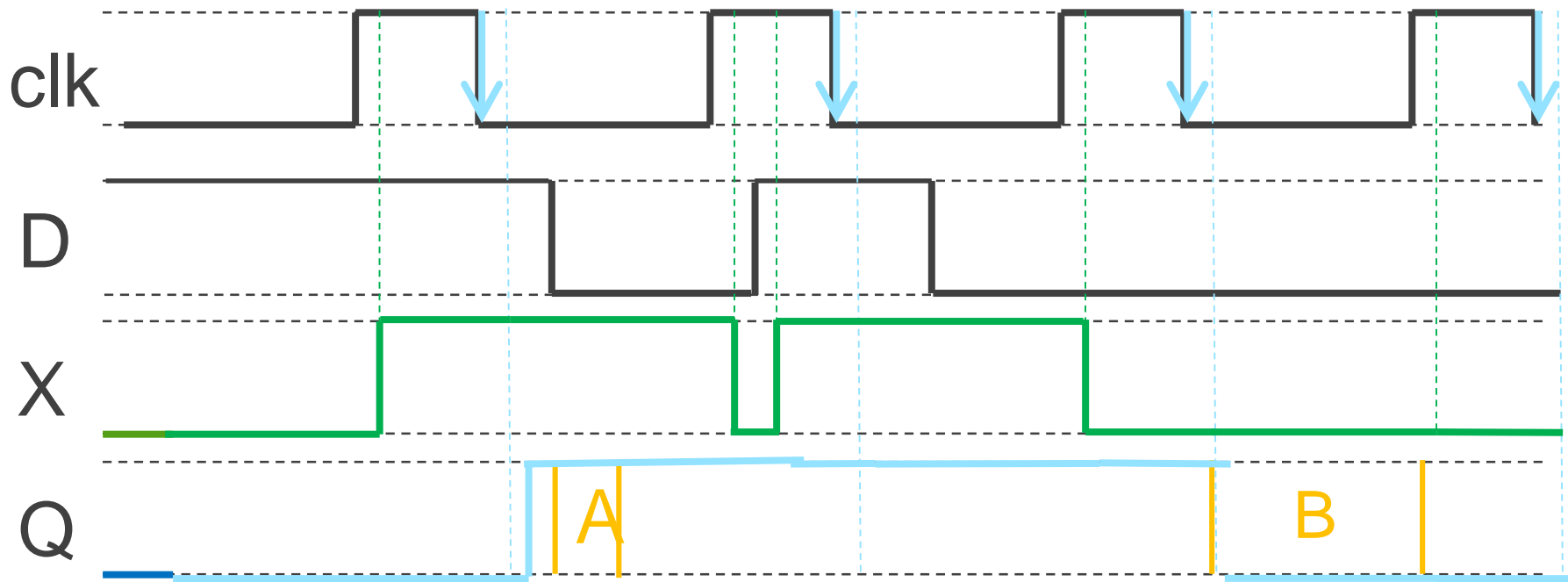
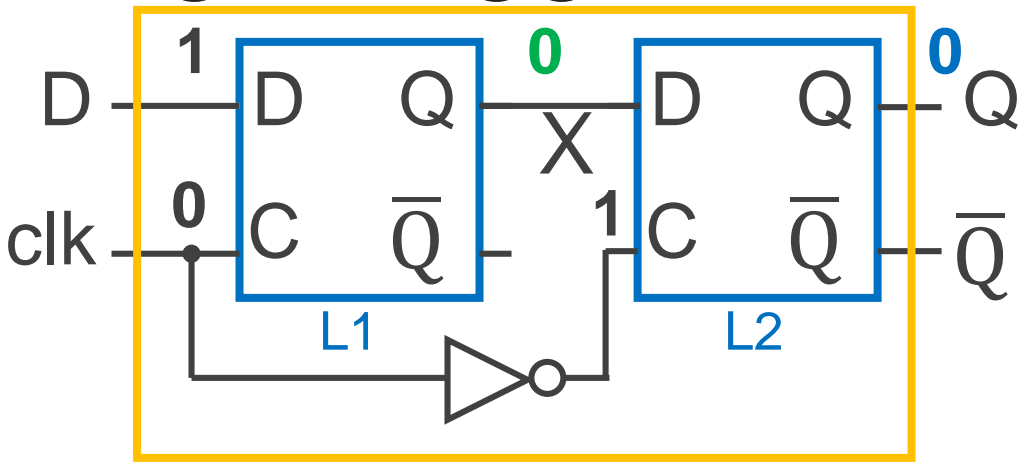
- a) $A = 0, B = 0$
- b) $A = 0, B = 1$
- c) $A = 1, B = 0$
- d) $A = 1, B = 1$



Edge-Triggered D Flip-Flop

D Flip-Flop

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



Takeaway

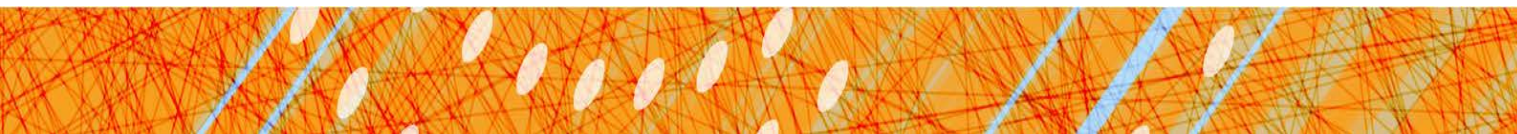
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

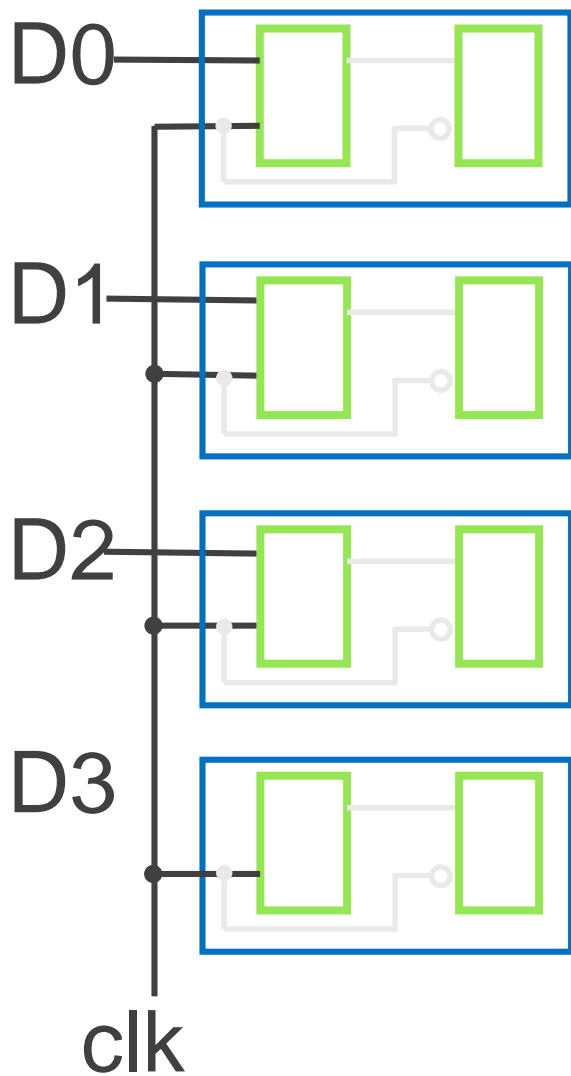
An Edge-Triggered D Flip-Flop (aka Master-Slave D Flip-Flop) stores one bit. The bit can be changed in a synchronized fashion on the edge of a clock signal.

Next Goal

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

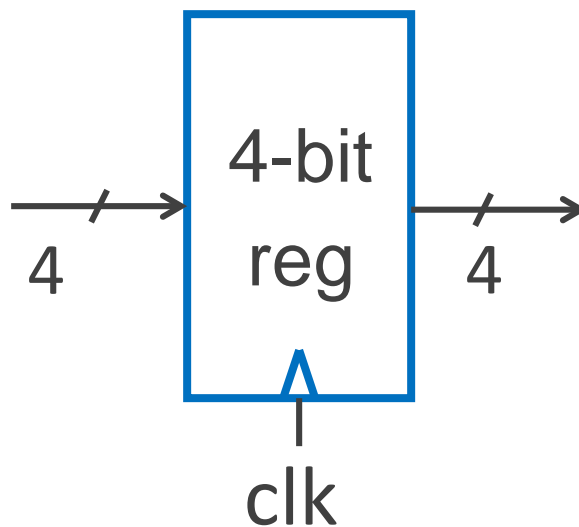


Registers



Register

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



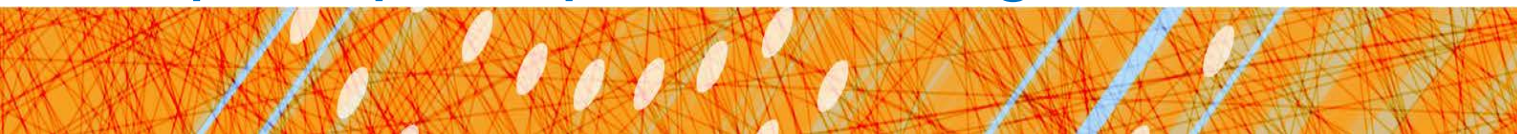
Takeaway

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.

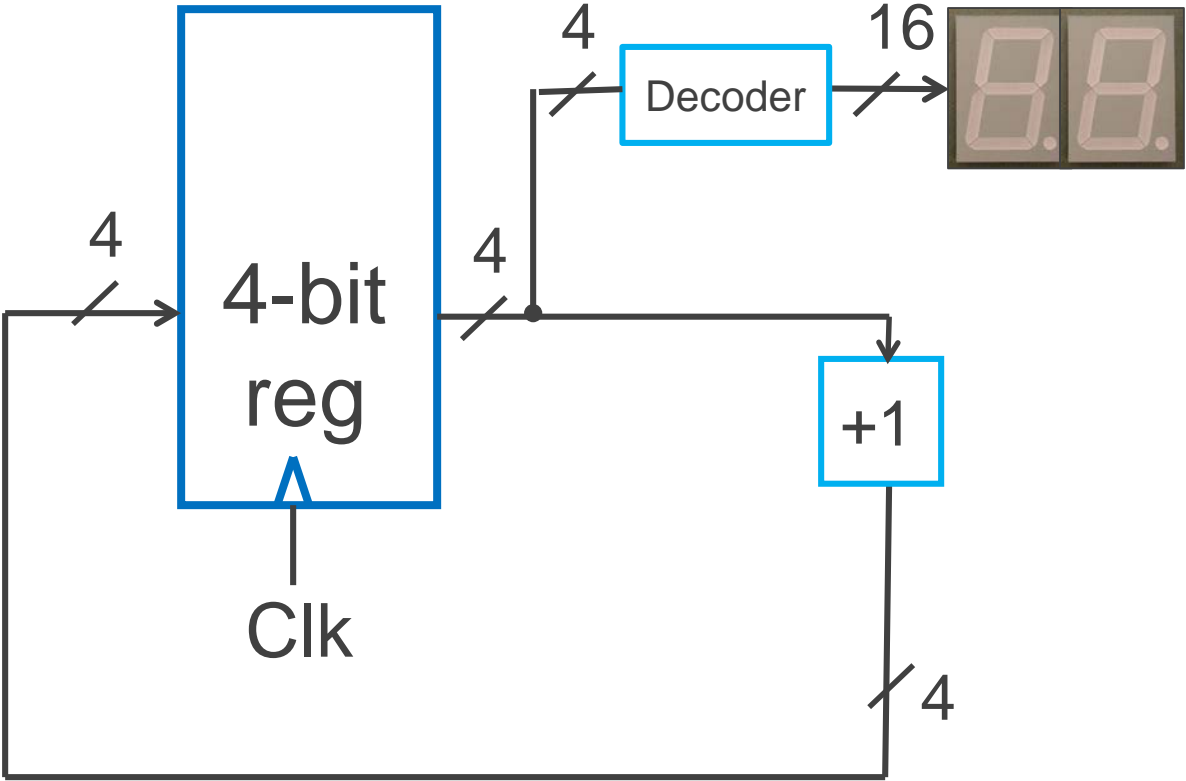
(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

An Edge-Triggered D Flip-Flop (aka Master-Slave D Flip-Flop) stores one bit. The bit can be changed in a synchronized fashion on the edge of a clock signal.

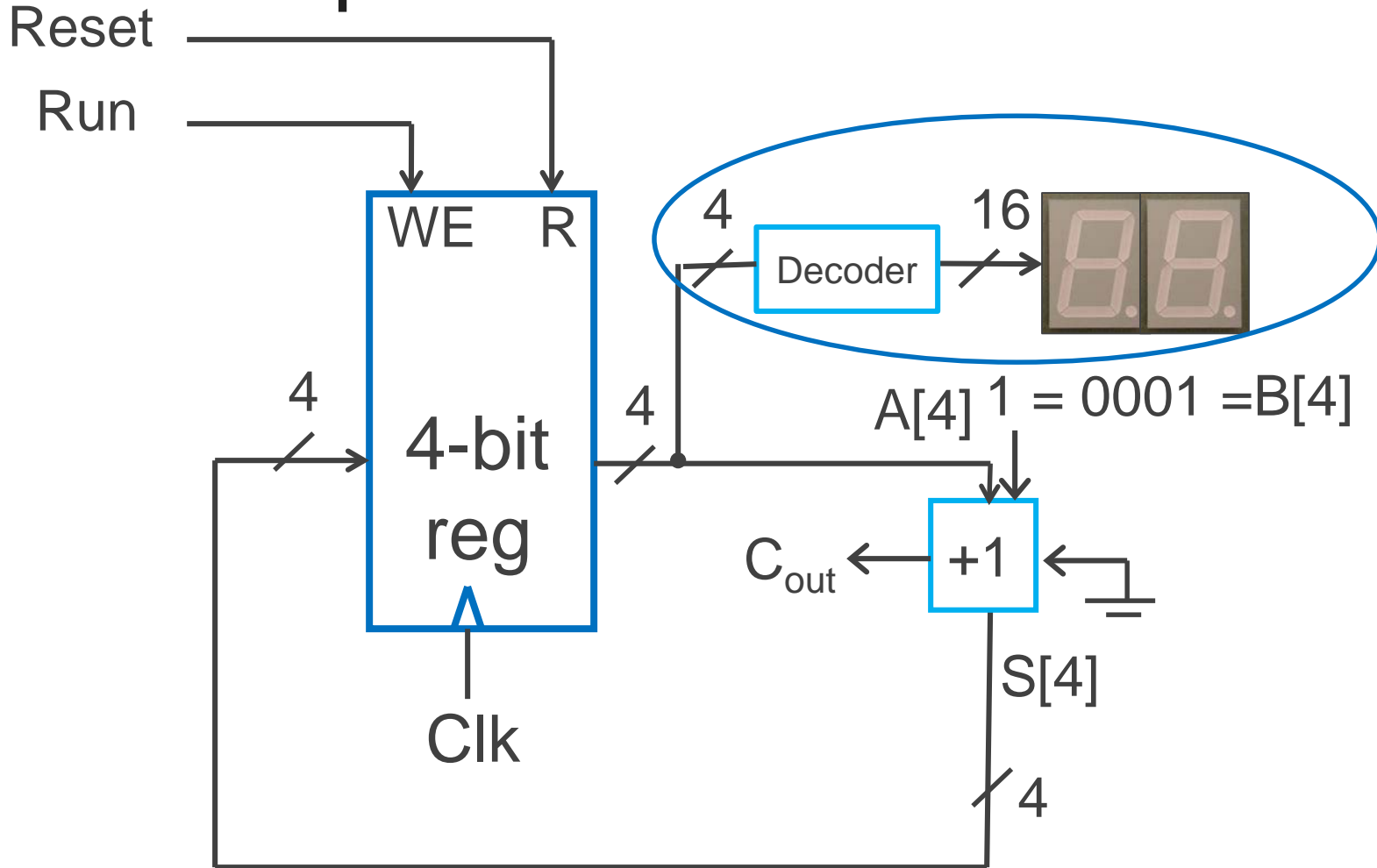
An N -bit **register** stores N -bits. It is created with N D-Flip-Flops in parallel along with a shared clock.



An Example: What will this circuit do?



An Example: What will this circuit do?



Decoder Example: 7-Segment LED

7-Segment LED

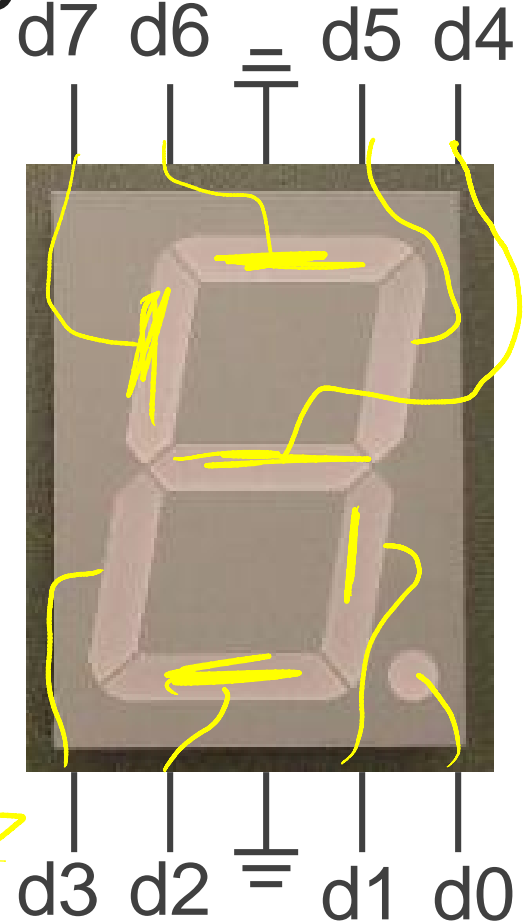
- photons emitted when electrons fall into holes



Decoder Example: 7-Segment LED

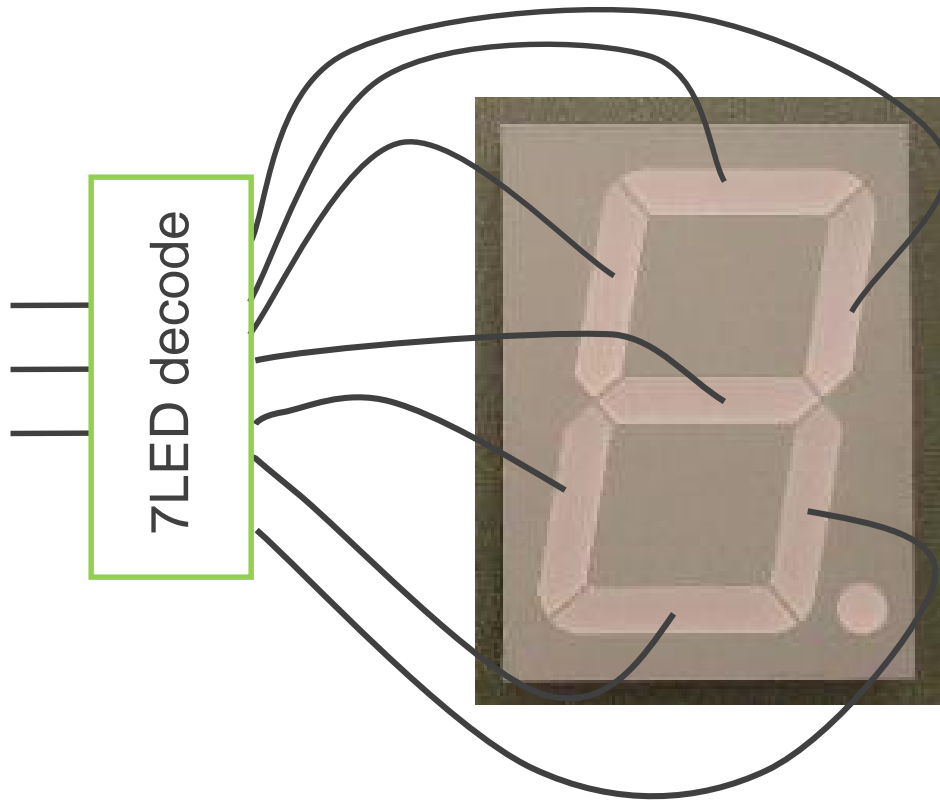
7-Segment LED

- photons emitted when electrons fall into holes



$5 = 01101011$

Decoder Example: 7-Segment LED Decoder



3 inputs

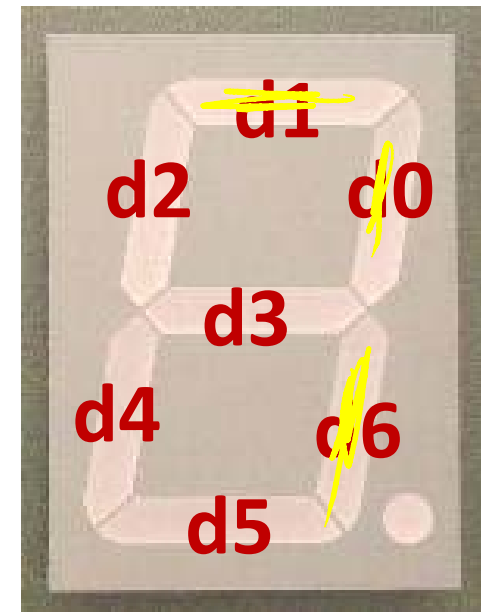
- encode 0 – 7 in binary

7 outputs

- one for each LED

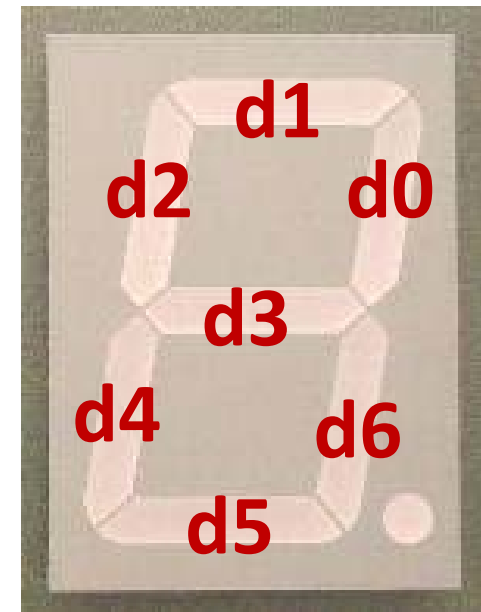
7 Segment LED Decoder Implementation

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

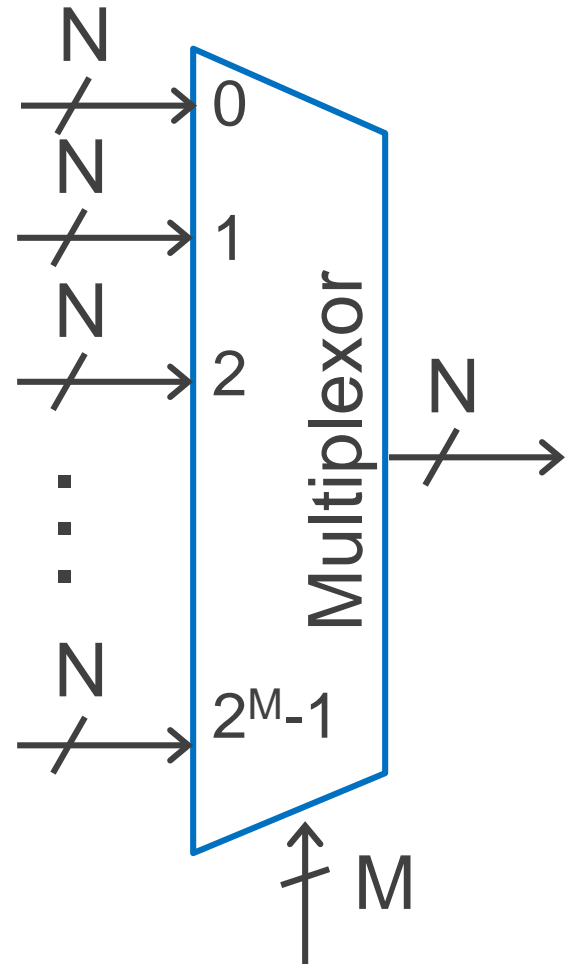
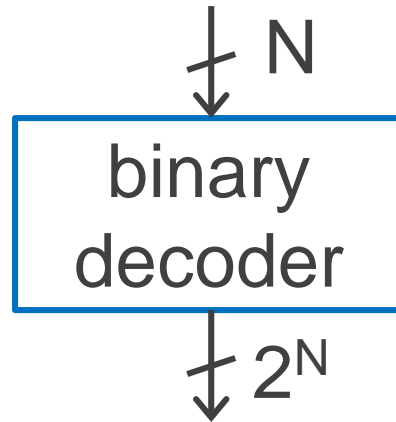
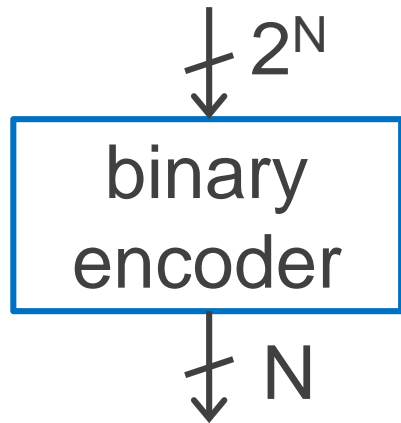


7 Segment LED Decoder Implementation

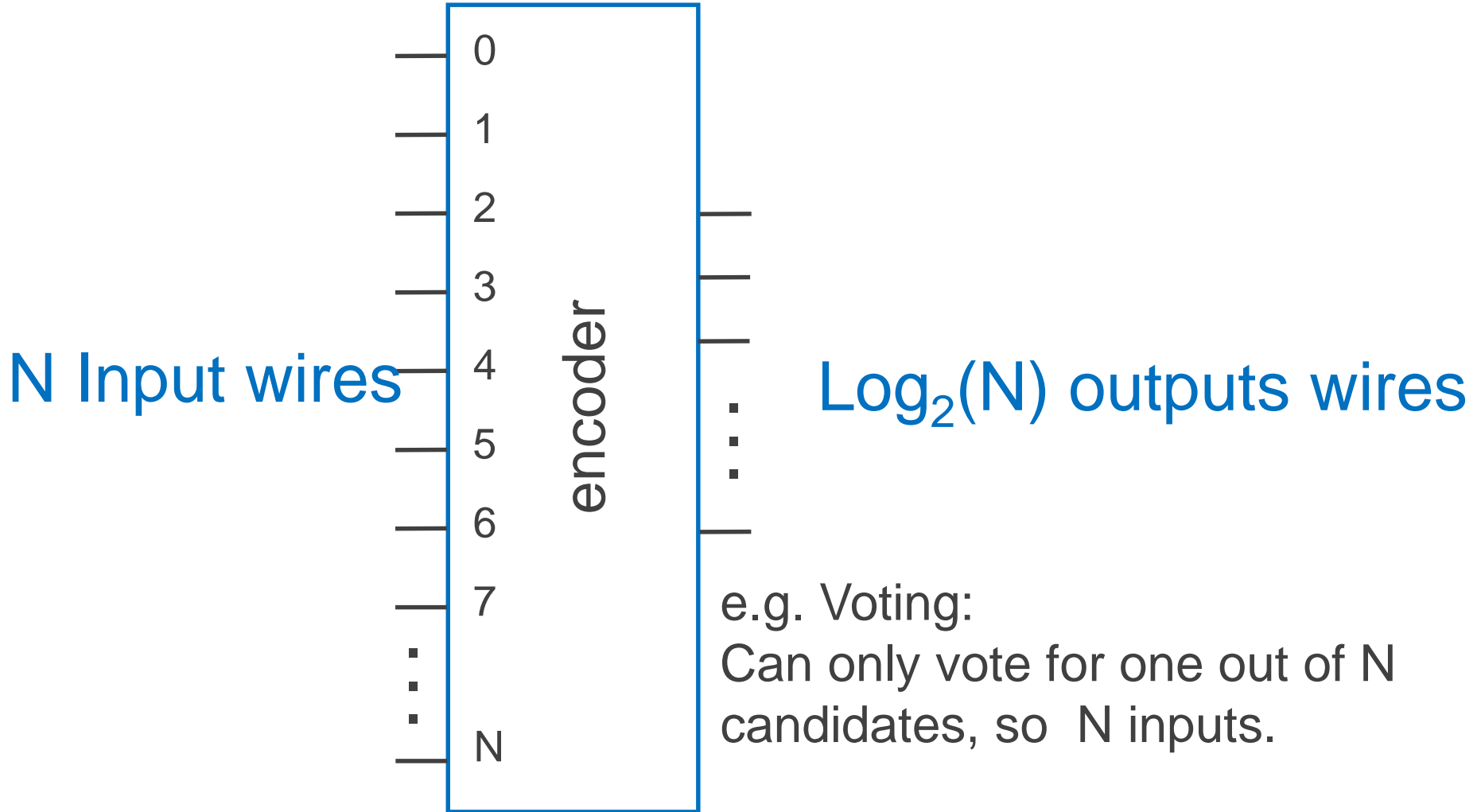
b2	b1	b0	d6	d5	d4	d3	d2	d1	d0
0	0	0	1	1	1	0	1	1	1
0	0	1	1	0	0	0	0	0	1
0	1	0	0	1	1	1	0	1	1
0	1	1	1	1	0	1	0	1	1
1	0	0	1	0	0	1	1	0	1
1	0	1	1	1	0	1	1	1	0
1	1	0	1	1	1	1	1	1	0
1	1	1	1	0	0	0	0	1	1



Basic Building Blocks We have Seen



Encoders

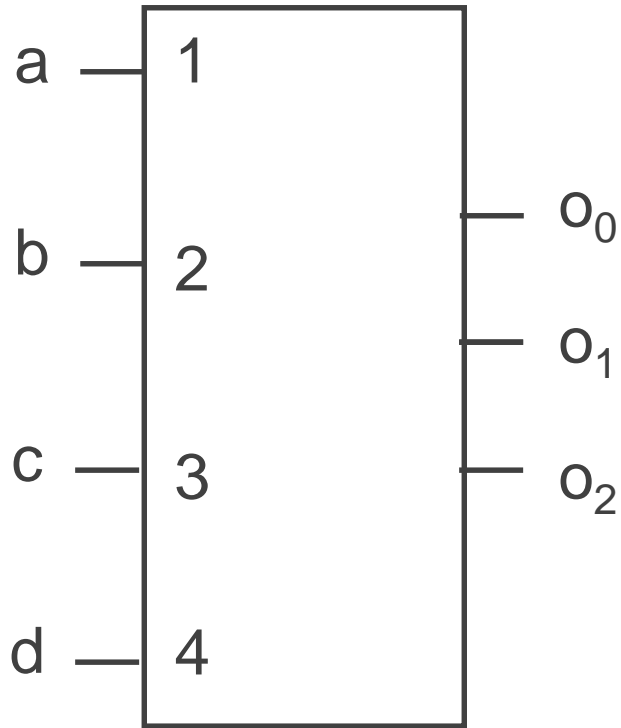


e.g. Voting:

Can only vote for one out of N candidates, so N inputs.

But can encode vote efficiently with binary encoding.

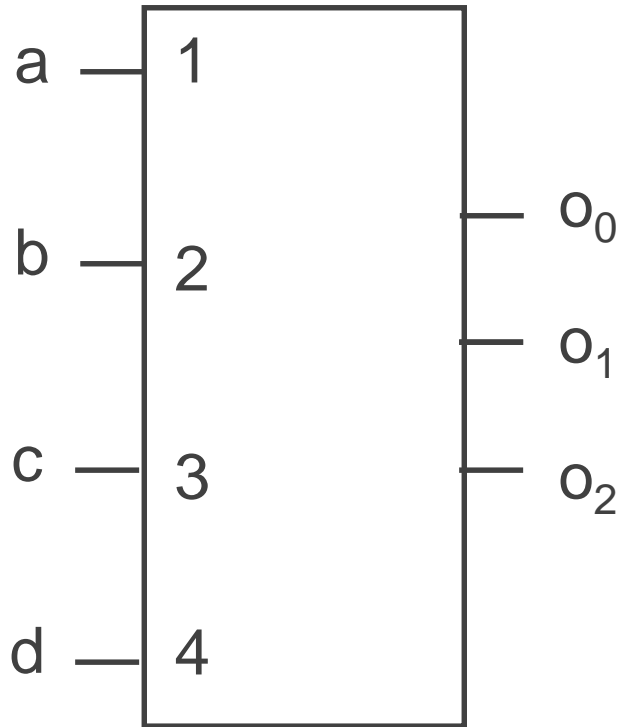
Example Encoder Truth Table



A 3-bit
encoder
with 4 inputs
for simplicity

a	b	c	d				
0	0	0	0				
1	0	0	0				
0	1	0	0				
0	0	1	0				
0	0	0	1				

Example Encoder Truth Table

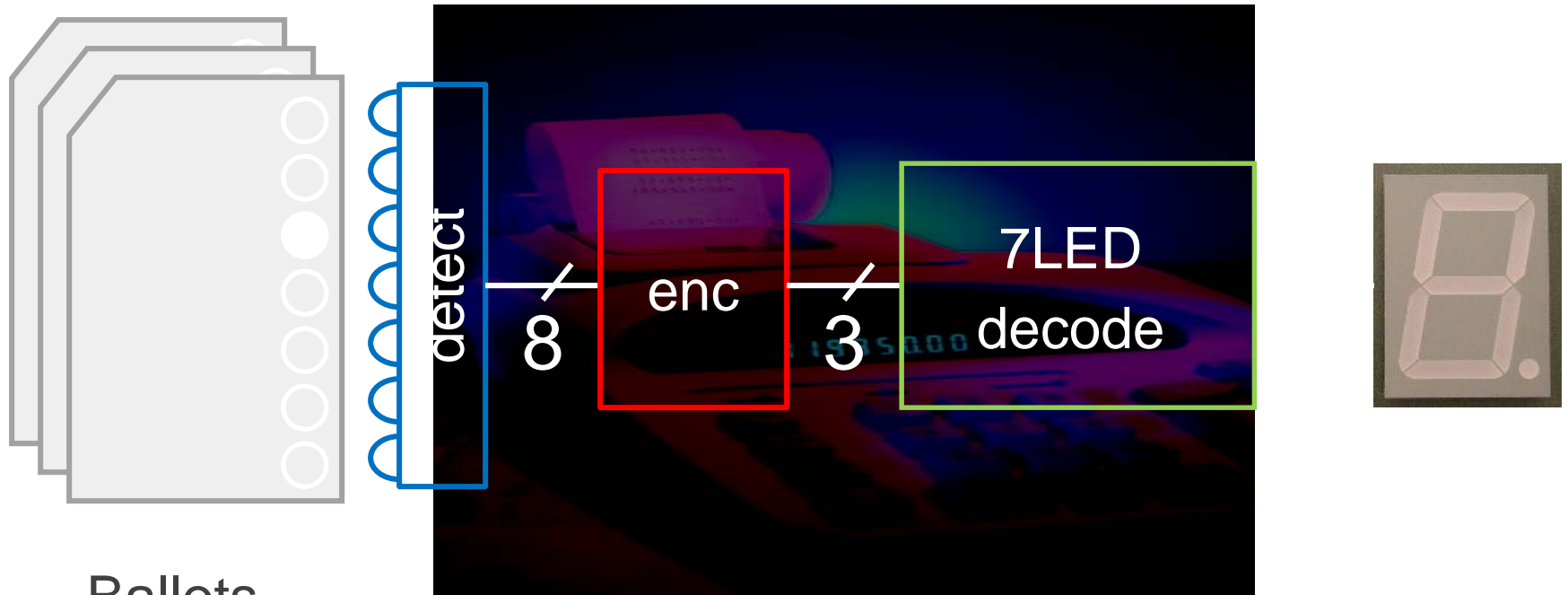


A 3-bit
encoder
with 4 inputs
for simplicity

a	b	c	d		o ₂	o ₁	o ₀
0	0	0	0		0	0	0
1	0	0	0		0	0	1
0	1	0	0		0	1	0
0	0	1	0		0	1	1
0	0	0	1		1	0	0

- $o_2 = \overline{abcd}$
- $o_1 = \overline{a}bcd + a\overline{b}cd$
- $o_0 = ab\overline{c}d + abcd\overline{d}$

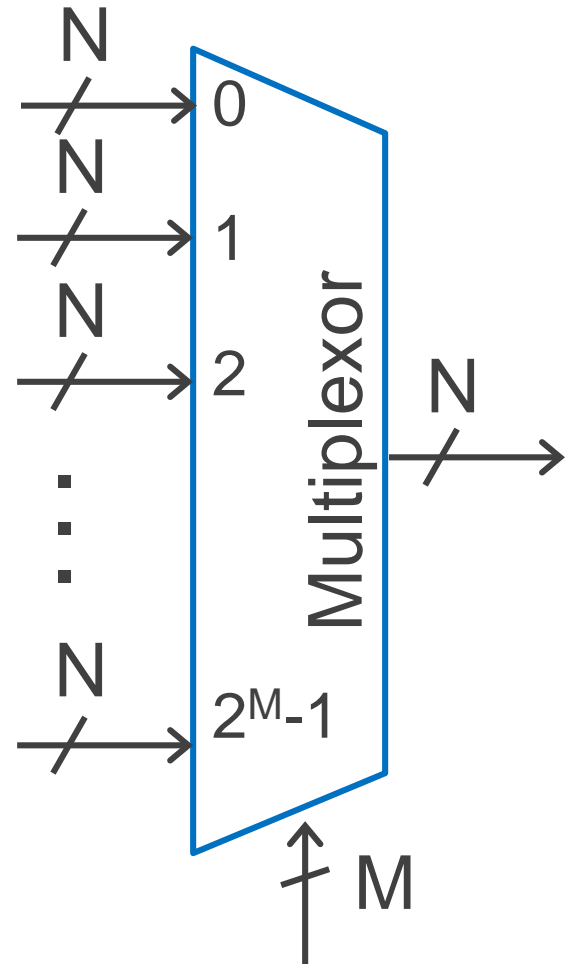
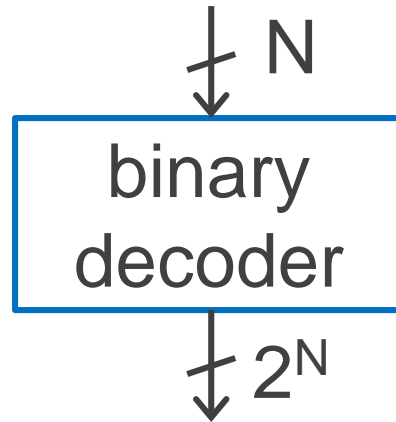
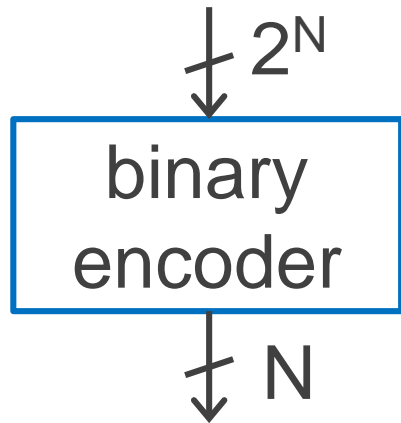
Basic Building Blocks Example: Voting



Ballots

The 3410 optical scan
vote reader
machine

Basic Building Blocks We have Seen



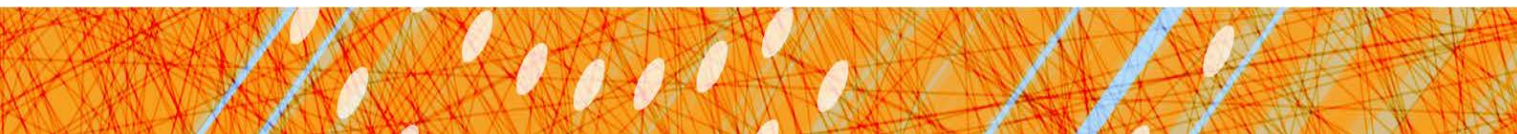
Recap

We can now build interesting devices with sensors

- Using combinational logic

We can also store data values (aka Sequential Logic)

- In state-holding elements
- Coupled with clocks



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

