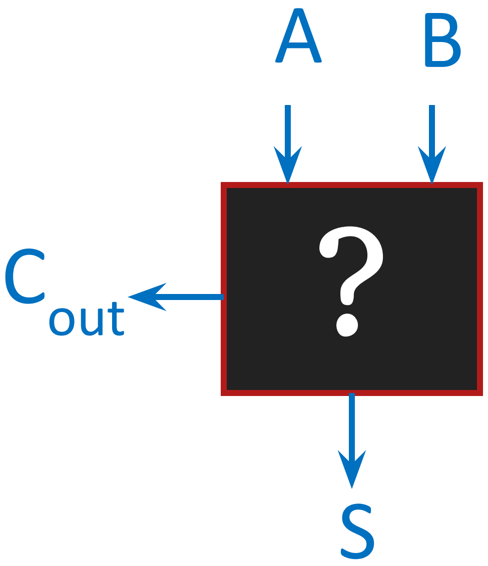
Review: Adder Circuit

CS 3410: Computer System Organization and Programming

Fall 2025

1-bit Half Adder



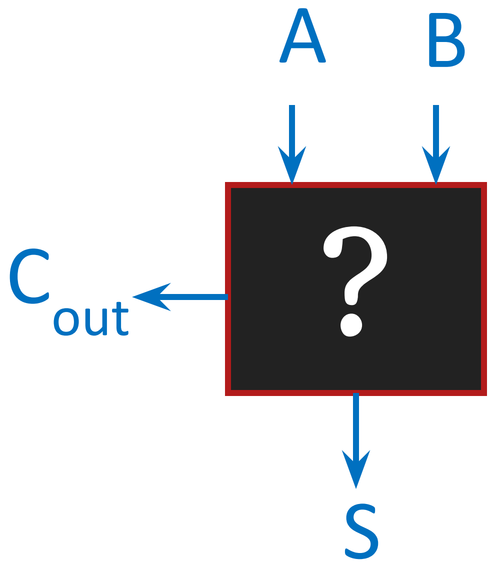
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- No carry-in

A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

S = one input equals 1

C_{out} = two inputs equal 1

1-bit Half Adder



- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- No carry-in

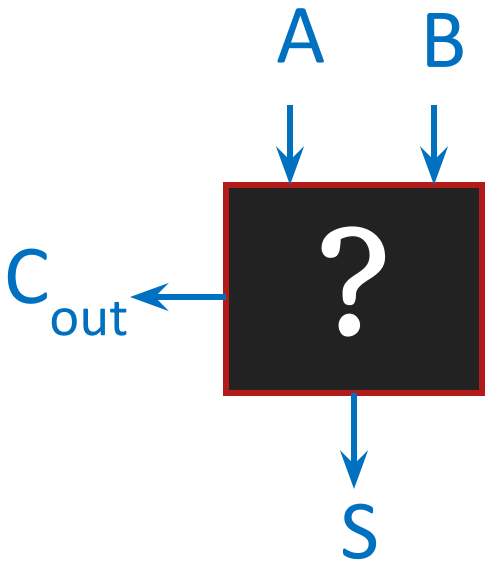
A	B	C_{out}	S
0	0	0	0
0	1		
1	0		
1	1		

$$\begin{array}{r} 0 \\ 0 \\ \underline{+\,0} \\ 0 \end{array}$$

S = one input equals 1

C_{out} = two inputs equal 1

1-bit Half Adder



- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- No carry-in

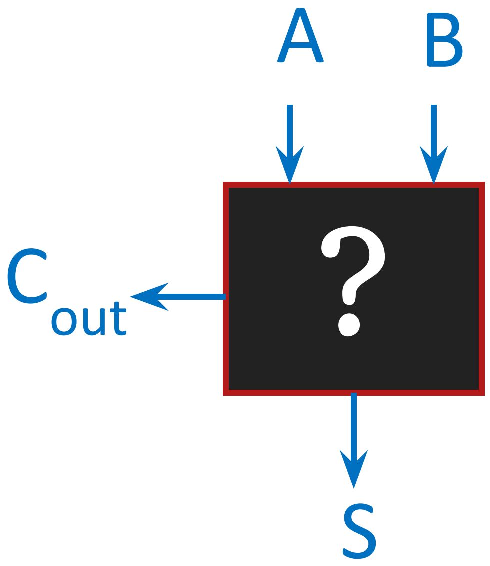
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0		
1	1		

$$\begin{array}{rr} 0 & \\ & 0 \\ + & 0 \\ \hline & 0 \end{array} \qquad \begin{array}{rr} 0 & \\ & 0 \\ + & 1 \\ \hline & 1 \end{array}$$

S = one input equals 1

C_{out} = two inputs equal 1

1-bit Half Adder



- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- No carry-in

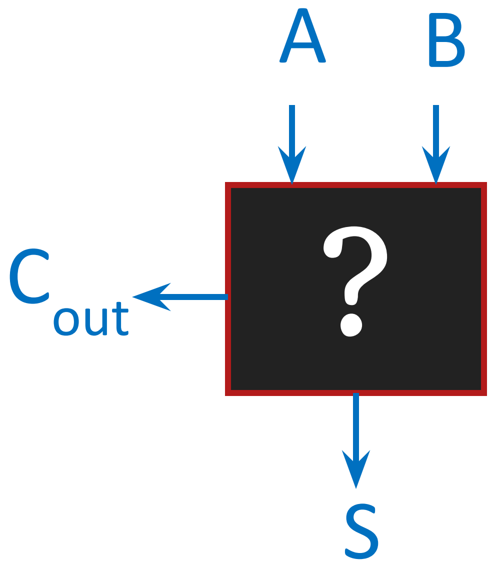
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1		

$$\begin{array}{r} 0 \\ 0 \\ \underline{+\,0} \\ 0 \end{array} \qquad \begin{array}{r} 0 \\ 0 \\ \underline{+\,1} \\ 1 \end{array} \qquad \begin{array}{r} 0 \\ 1 \\ \underline{+\,0} \\ 1 \end{array}$$

S = one input equals 1

C_{out} = two inputs equal 1

1-bit Half Adder



- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- No carry-in

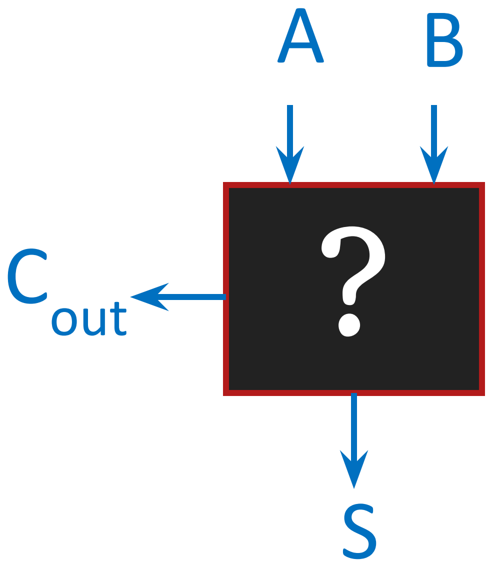
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

0	0	0	1
0	0	1	1
+ 0	+ 1	+ 0	+ 1
0	1	1	0

S = one input equals 1

C_{out} = two inputs equal 1

1-bit Half Adder

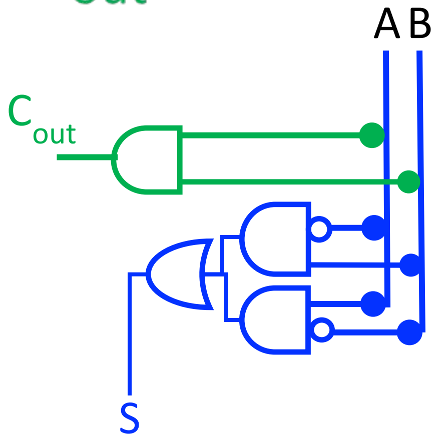


- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- No carry-in

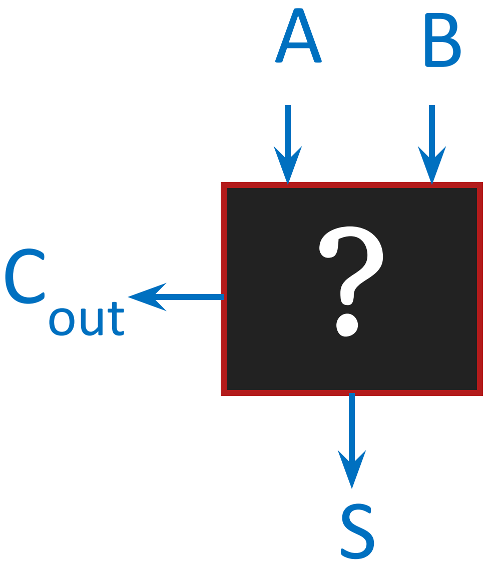
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = \overline{A}B + A\overline{B}$$

$$C_{out} = AB$$



1-bit Half Adder

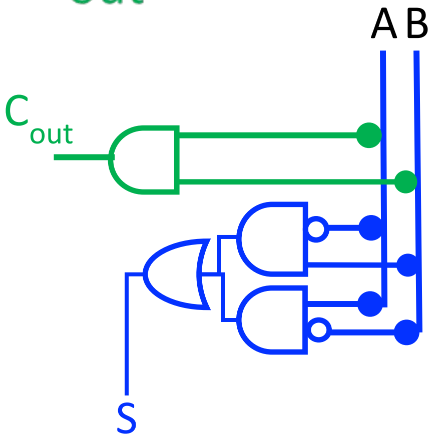


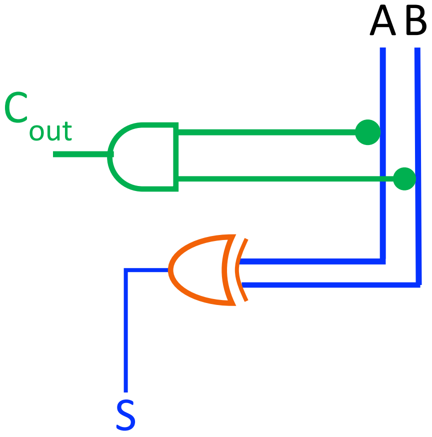
- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- No carry-in

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

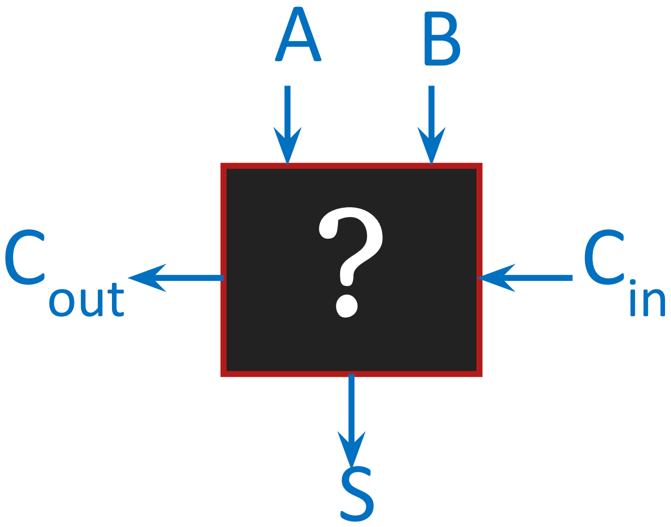
$$S = \overline{A}B + A\overline{B} \quad = A \oplus B$$

$$C_{out} = AB$$





1-bit Full Adder



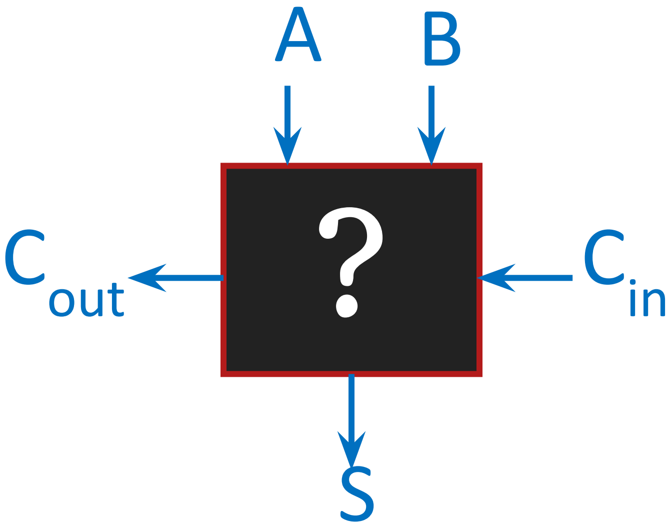
A	B	C_{in}	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- Can be cascaded

- Fill in Truth Table
- Create Sum-of-Product Form
- Draw the Circuits

Cornell Bowers CIS
Computer Science

1-bit Full Adder



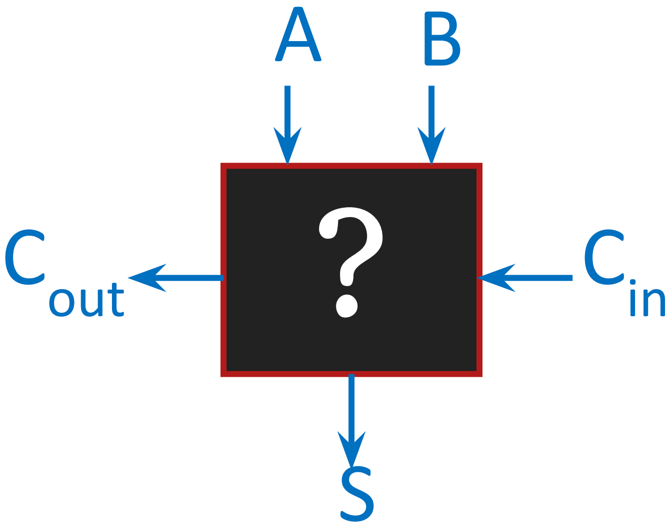
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- Can be cascaded

- Fill in Truth Table
- Create Sum-of-Product Form
- Draw the Circuits

$$0 + 0 + 0 = 0$$

1-bit Full Adder



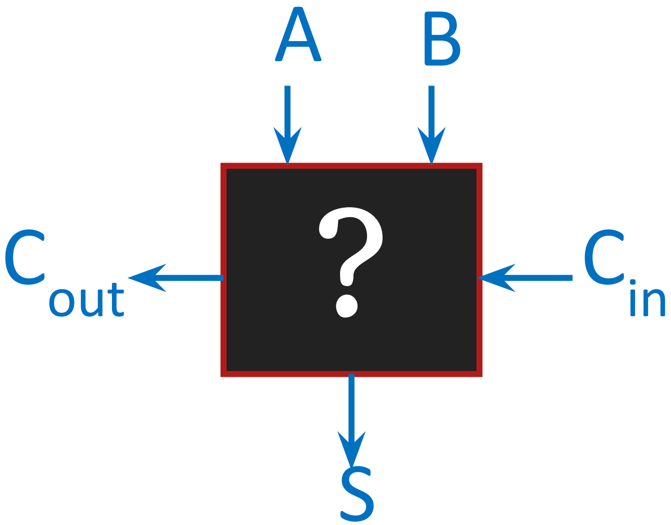
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1		
1	0	0	0	1
1	0	1		
1	1	0		
1	1	1		

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- Can be cascaded

- Fill in Truth Table
- Create Sum-of-Product Form
- Draw the Circuits

$$1 + 0 + 0 = 1$$

1-bit Full Adder



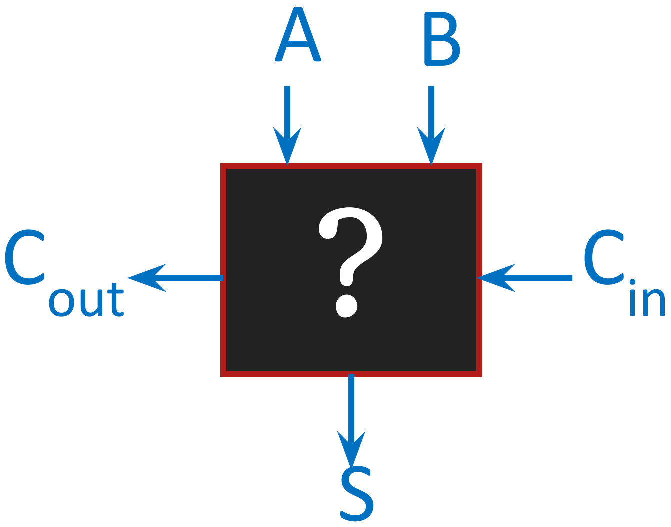
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1		

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- Can be cascaded

- Fill in Truth Table
- Create Sum-of-Product Form
- Draw the Circuits

$$1 + 1 + 0 = 2_{10} = 10_2$$

1-bit Full Adder



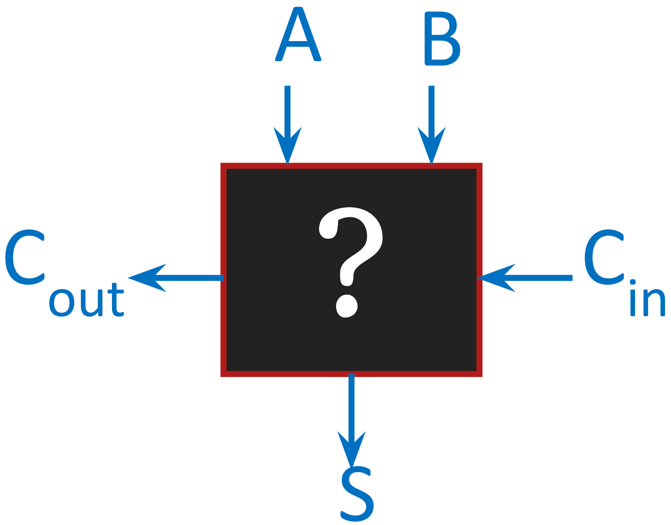
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- Can be cascaded

- Fill in Truth Table
- Create Sum-of-Product Form
- Draw the Circuits

$$1 + 1 + 1 = 3_{10} = 11_2$$

1-bit Full Adder



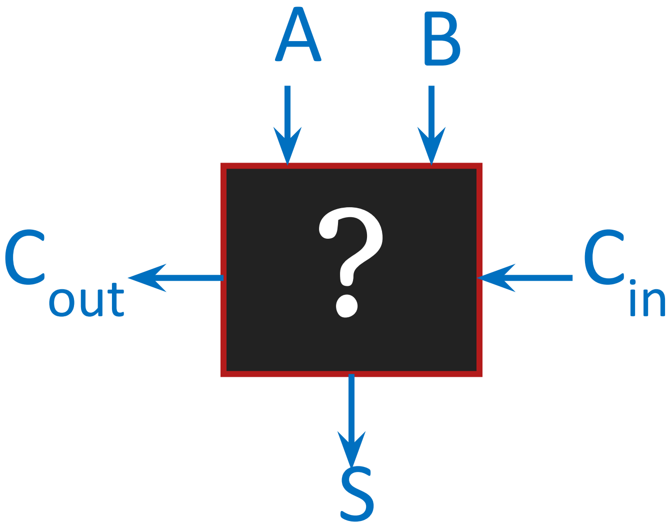
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- Can be cascaded

- Fill in Truth Table
- <u>Create Sum-of-Product Form</u>
- Draw the Circuits

$$S = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$$

1-bit Full Adder



A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

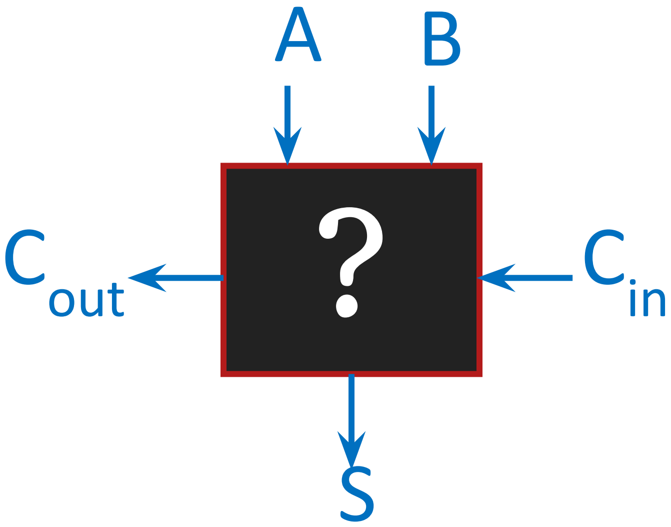
- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry-out
- Can be cascaded

- Fill in Truth Table
- Create Sum-of-Product Form
- Draw the Circuits

$$S = \overline{AB}C + \overline{A}B\overline{C} + A\overline{BC} + ABC$$

$$C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

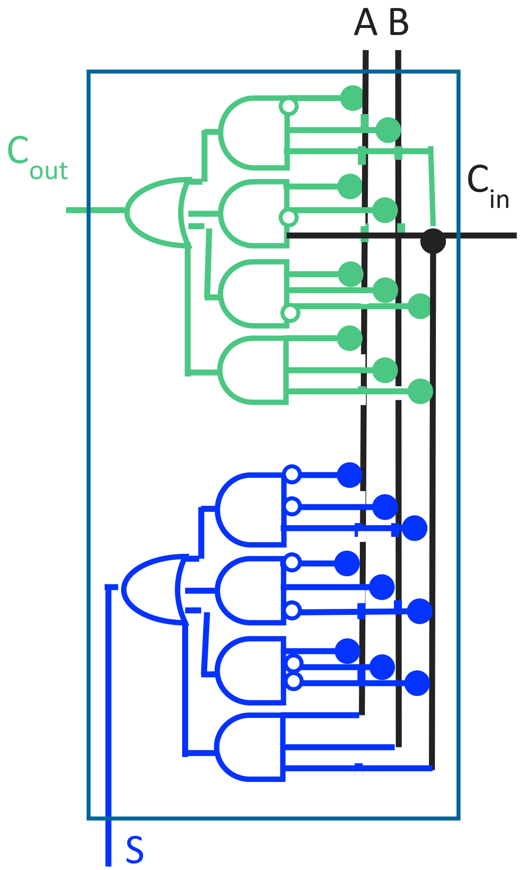
1-bit Full Adder



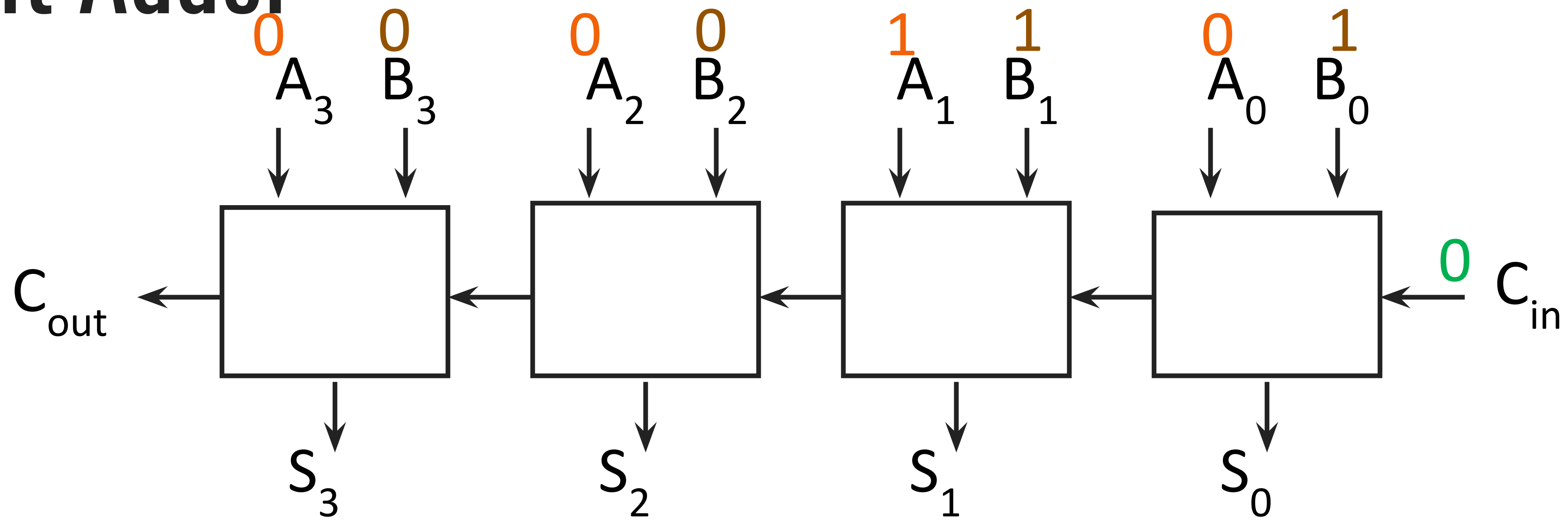
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = \overline{AB}C + \overline{A}B\overline{C} + A\overline{BC} + ABC$$

$$C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

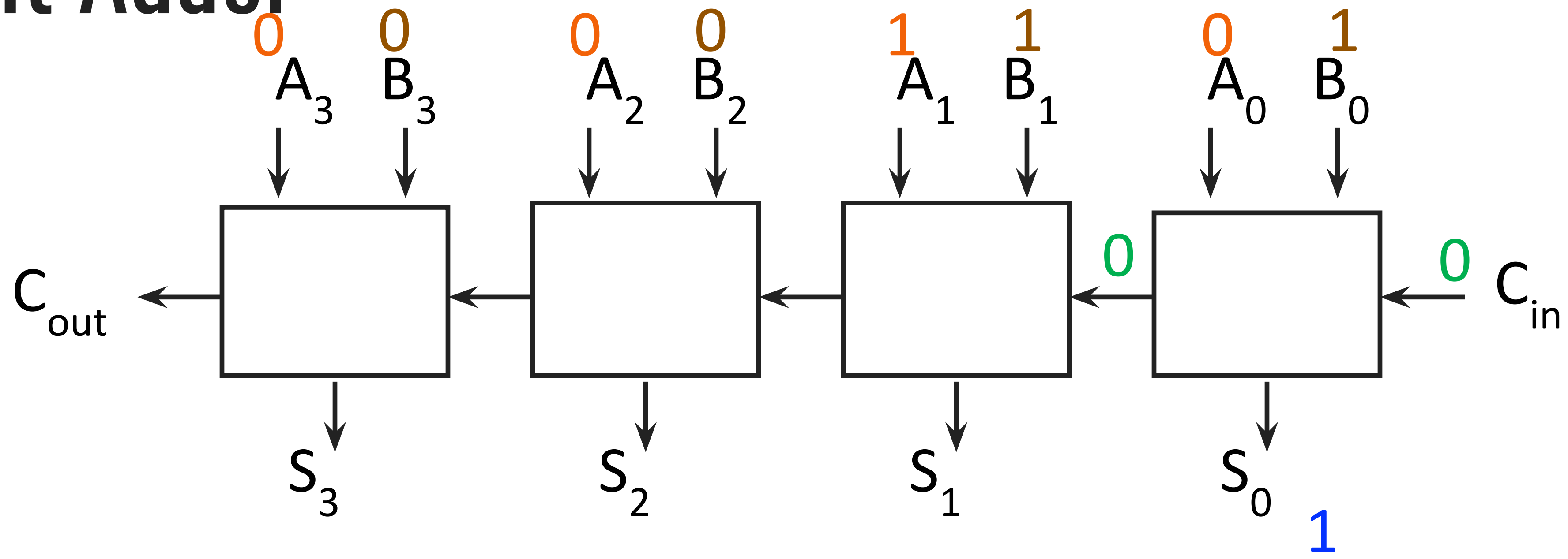


4-bit Adder



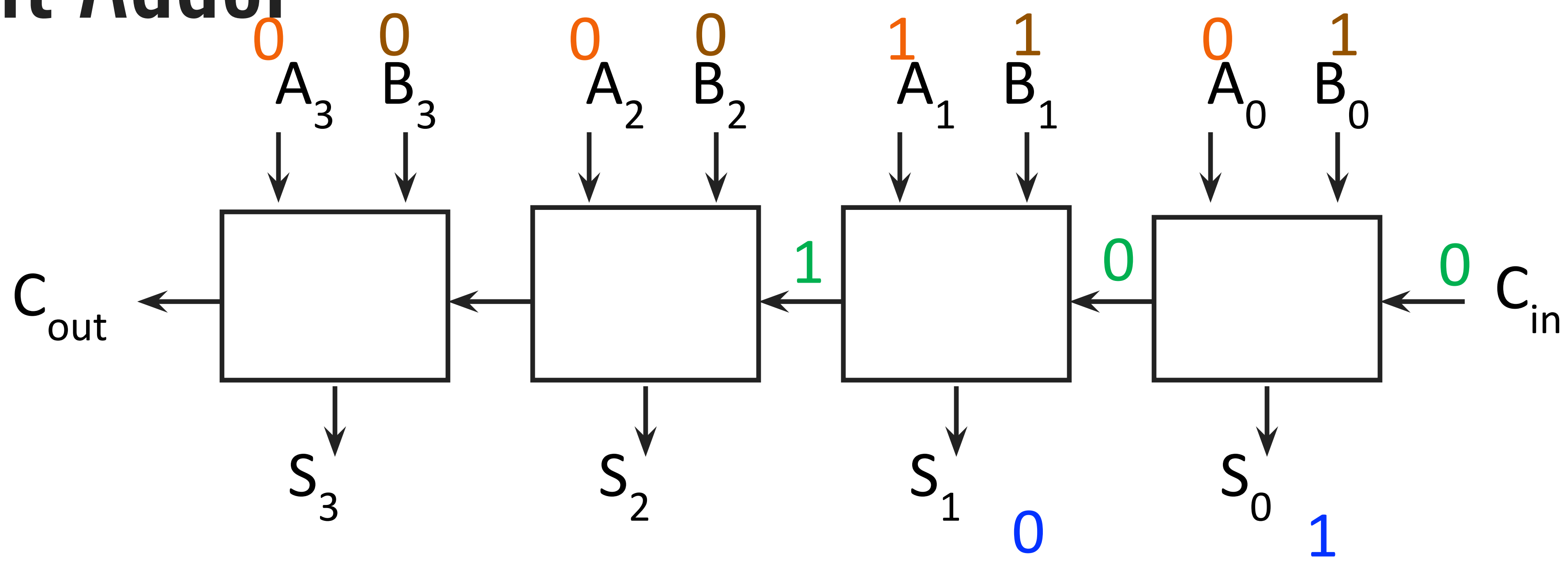
- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out
- 3 + 2 = 5
- Carry-out □ result > 4 bits

4-bit Adder



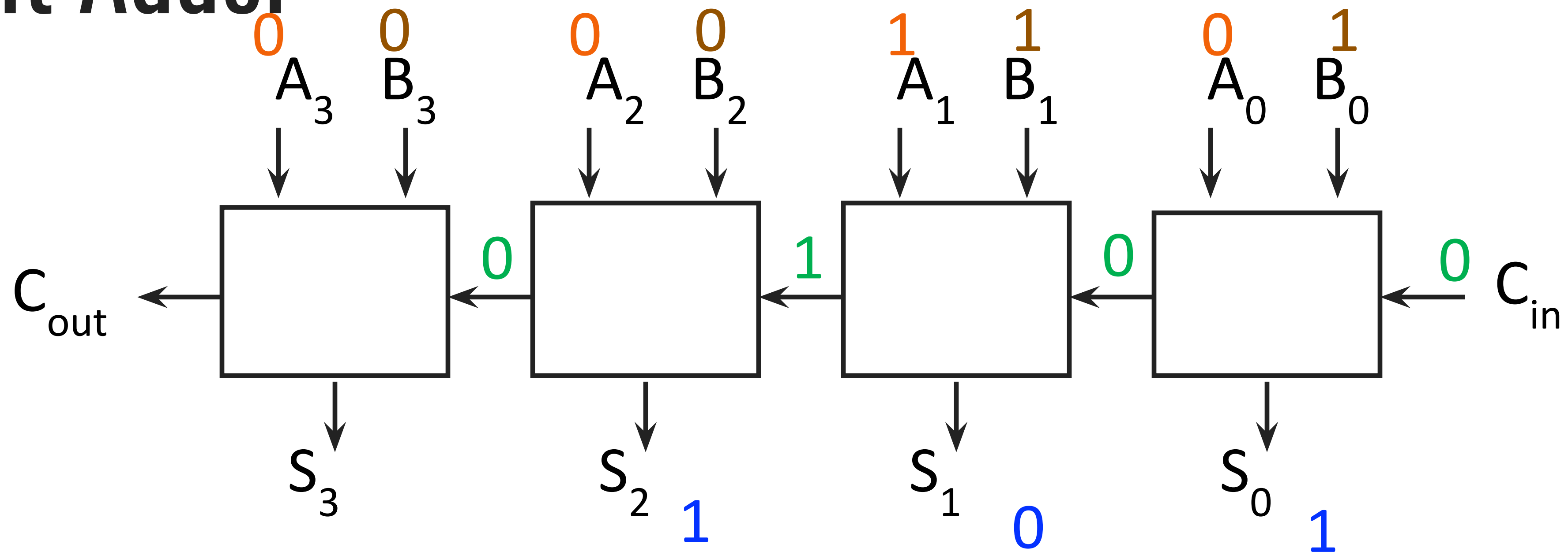
- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out
- 3 + 2 = 5
- Carry-out □ result > 4 bits

4-bit Adder



- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out
- 3 + 2 = 5
- Carry-out □ result > 4 bits

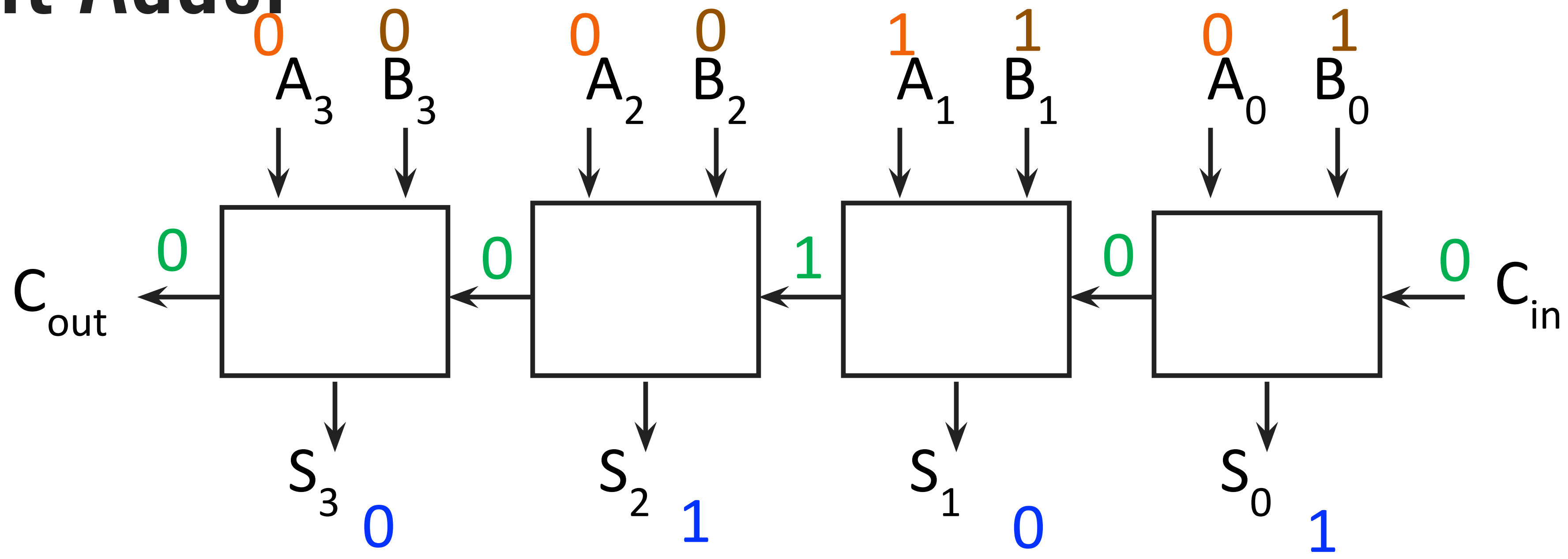
4-bit Adder



- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out
- 3 + 2 = 5
- Carry-out □ result > 4 bits

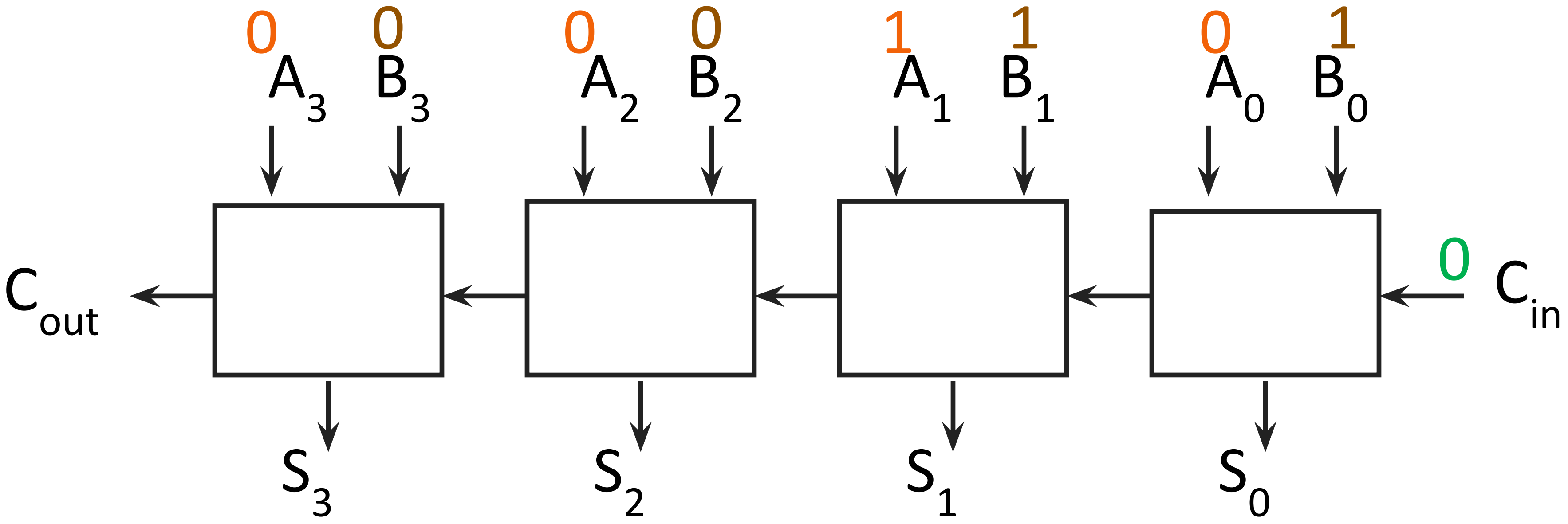
Cornell Bowers CIS
Computer Science

4-bit Adder



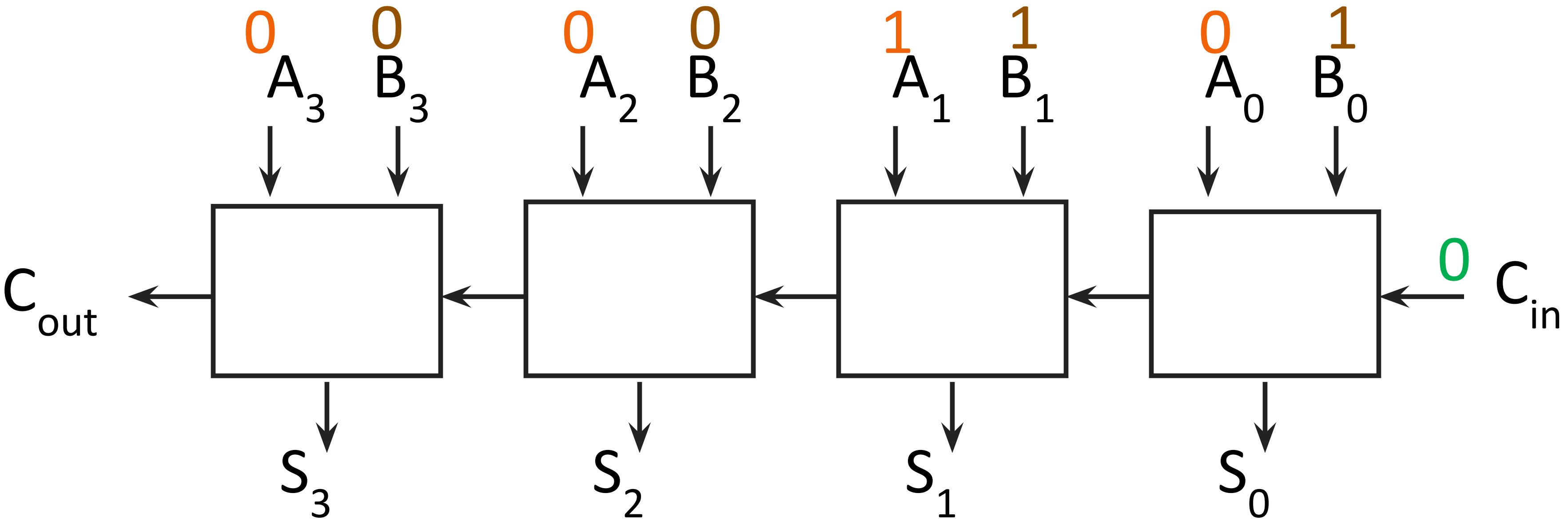
- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out
- 3 + 2 = 5
- Carry-out □ result > 4 bits

4-bit Adder to 4-bit Subtractor



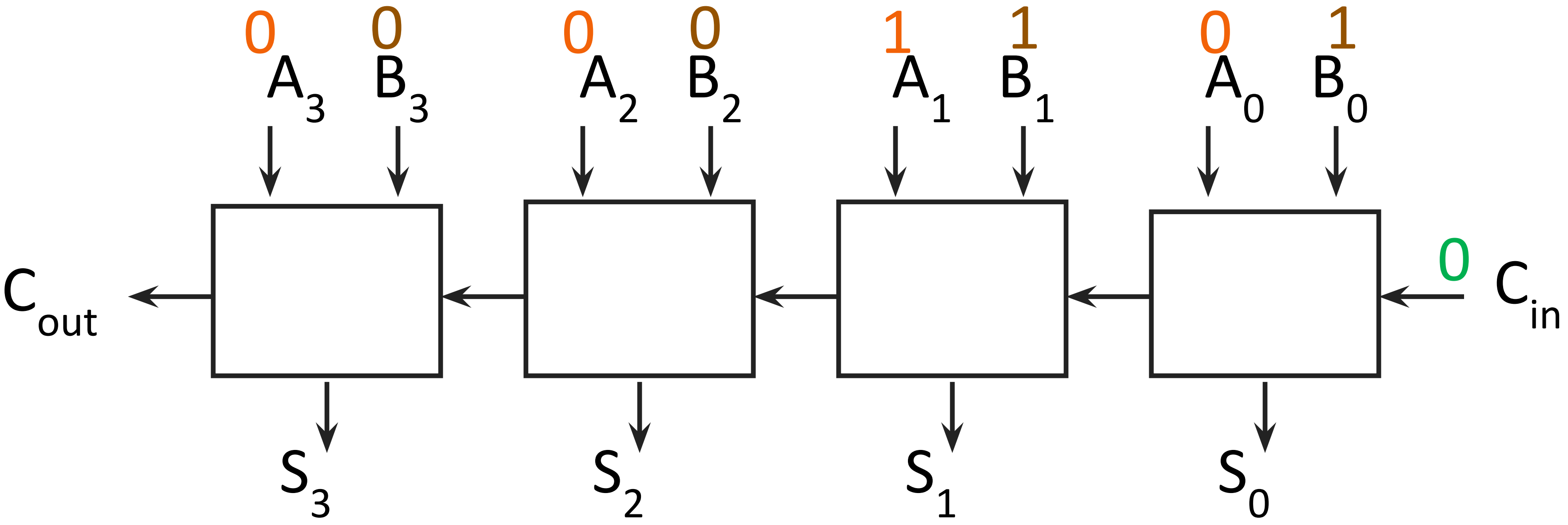
- What if we want to subtract instead?

4-bit Adder to 4-bit Subtractor



- What if we want to subtract instead?
- How do we calculate 3 - 2 = 1?

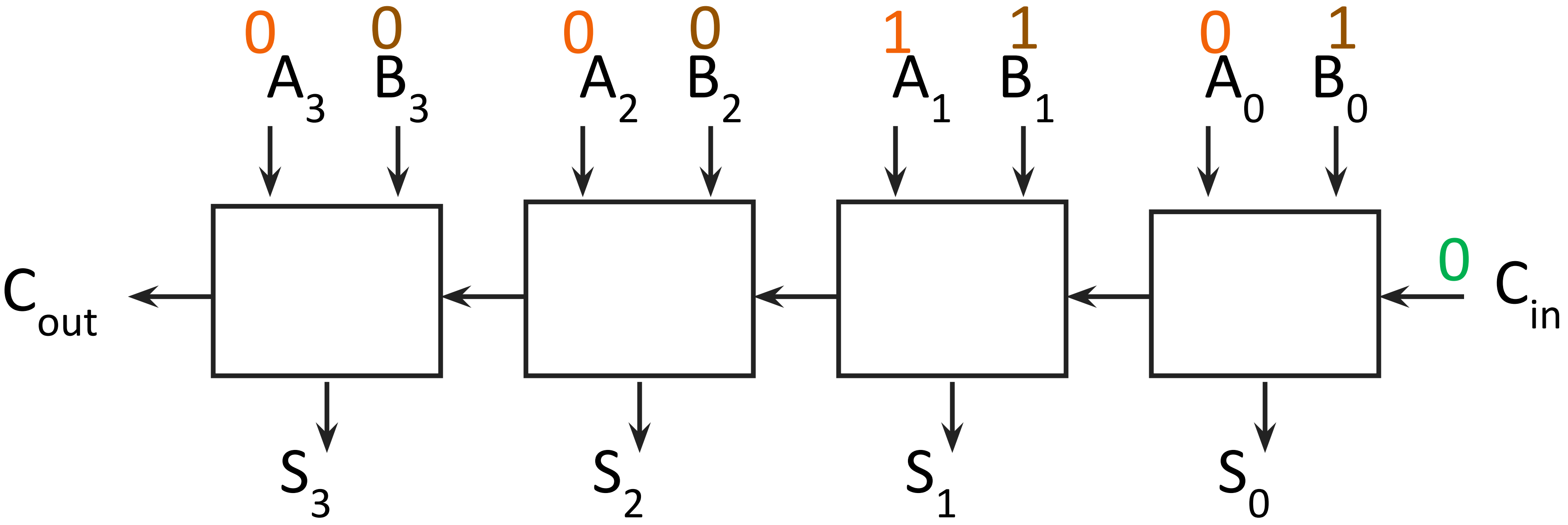
4-bit Adder to 4-bit Subtractor



- What if we want to subtract instead?
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1

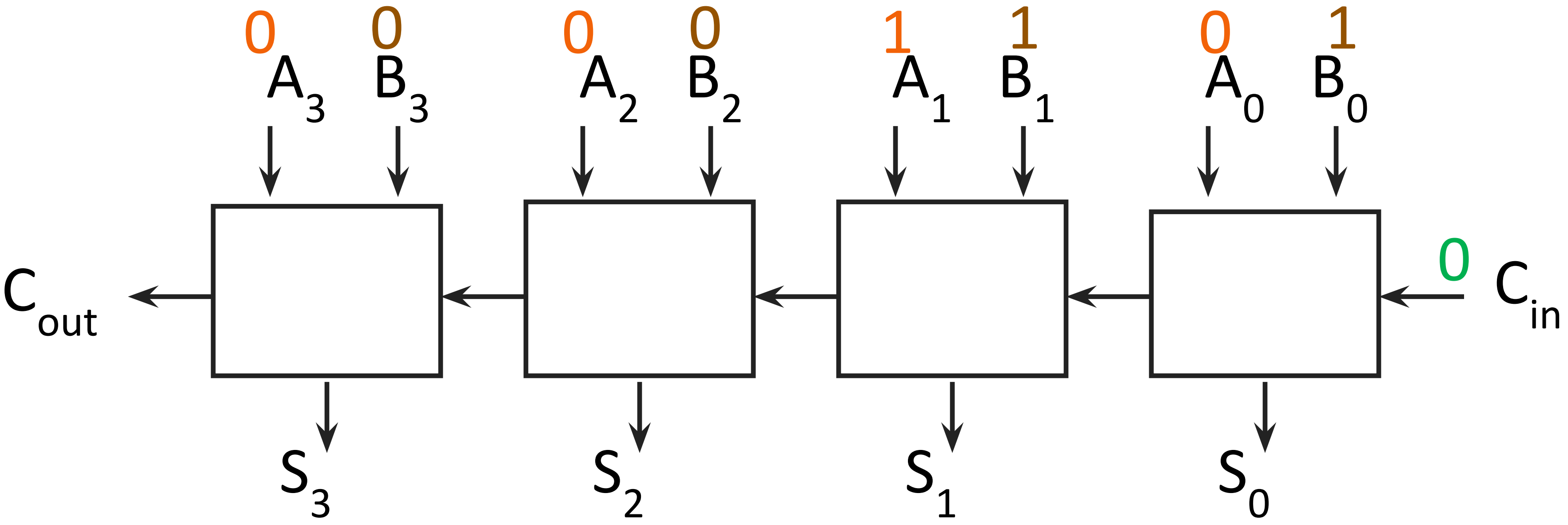
Cornell Bowers CIS
Computer Science

4-bit Adder to 4-bit Subtractor



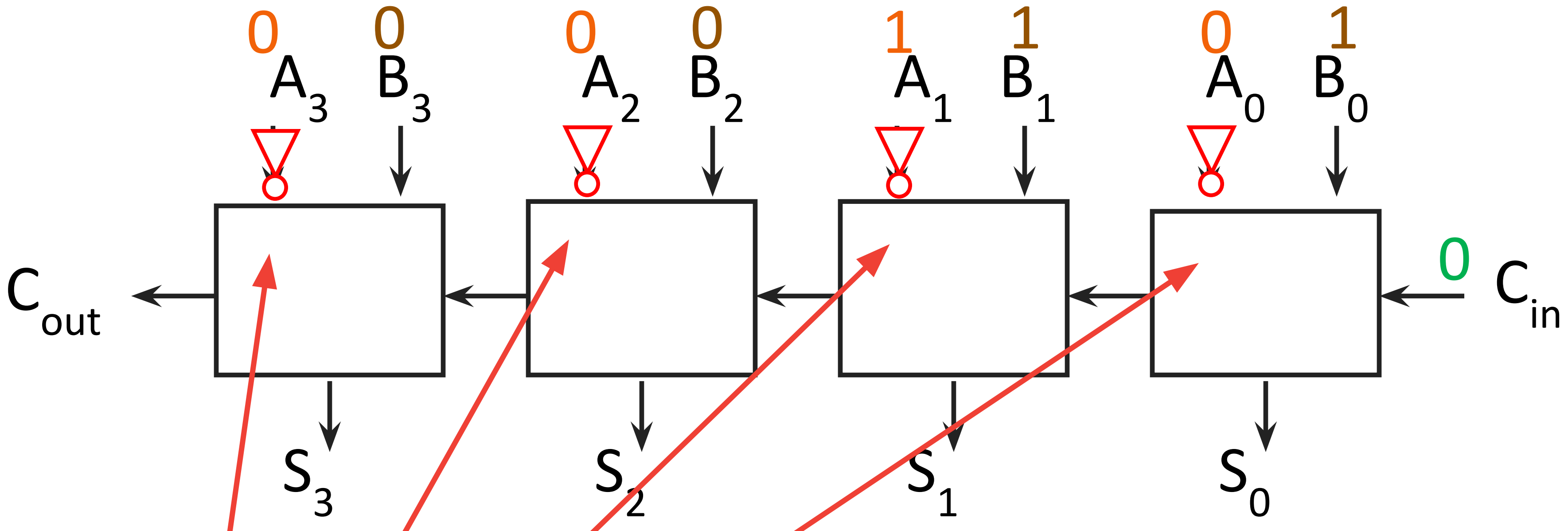
- What if we want to subtract instead?
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- How do we negate a two's complement number?

4-bit Adder to 4-bit Subtractor



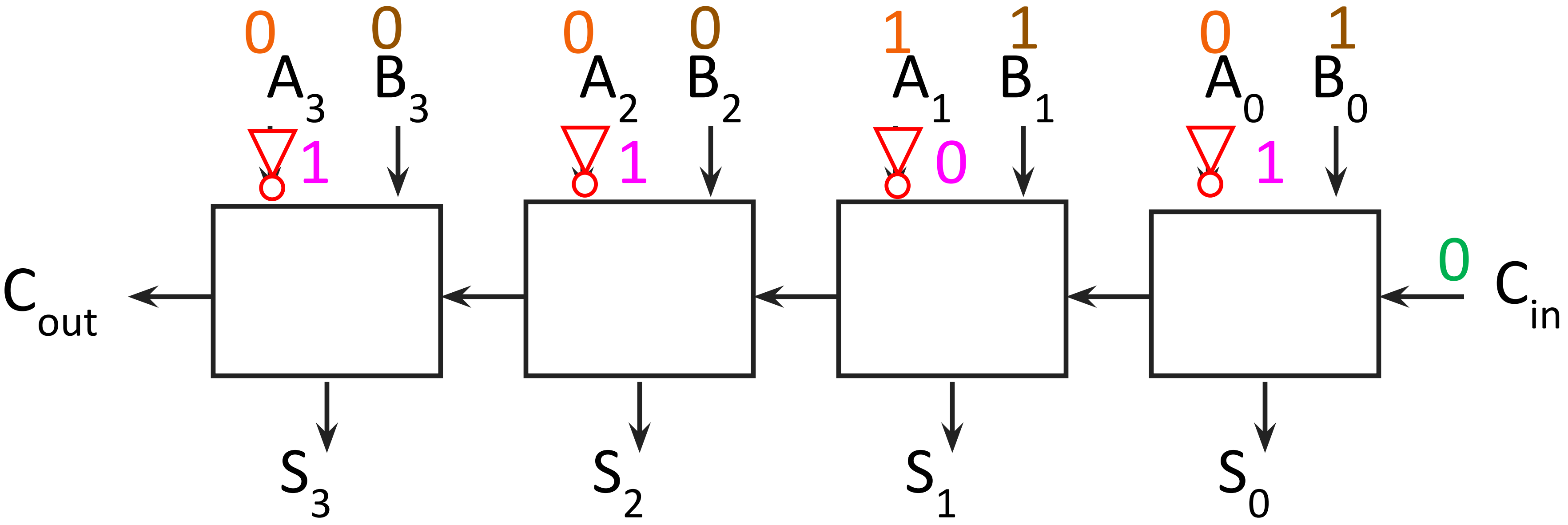
- What if we want to subtract instead?
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- How do we negate a two's complement number?
- -2: !(0010) + 1 = 1101 + 1 = 1110

4-bit Adder to 4-bit Subtractor



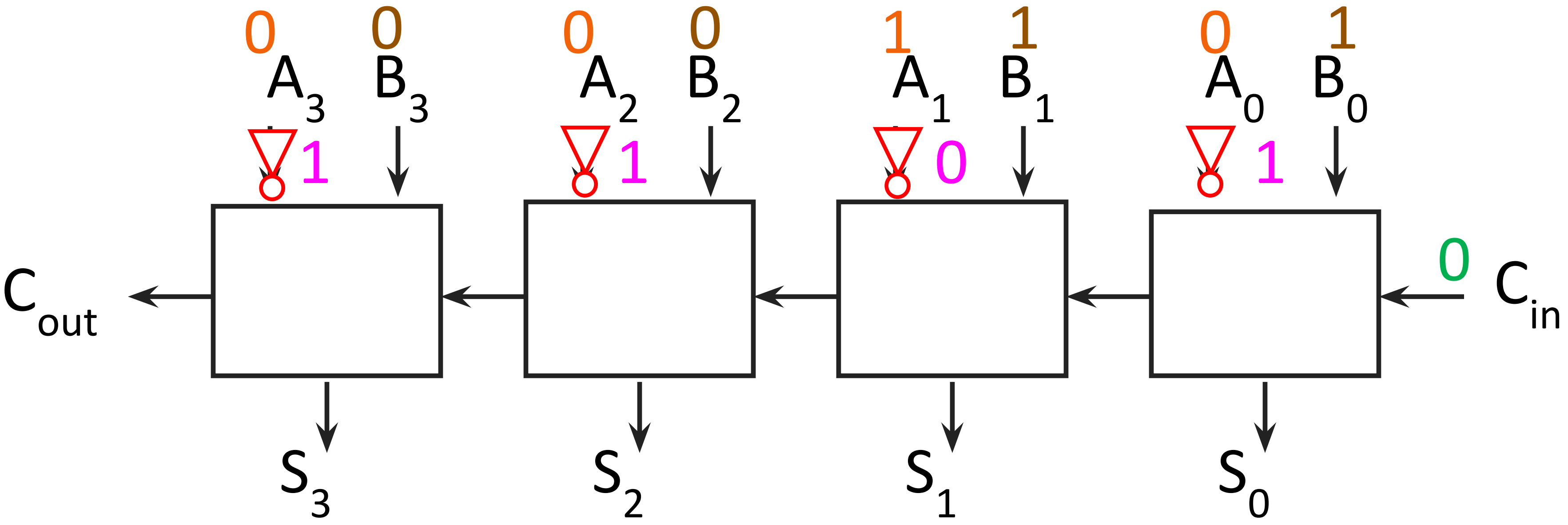
- What if we want to subtract instead?
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- How do we negate a two’s complement number?
- -2: !(0010) + 1 = 1101 + 1 = 1110

4-bit Adder to 4-bit Subtractor



- What if we want to subtract instead?
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- How do we negate a two's complement number?
- -2: !(0010) + 1 = 1101 + 1 = 1110

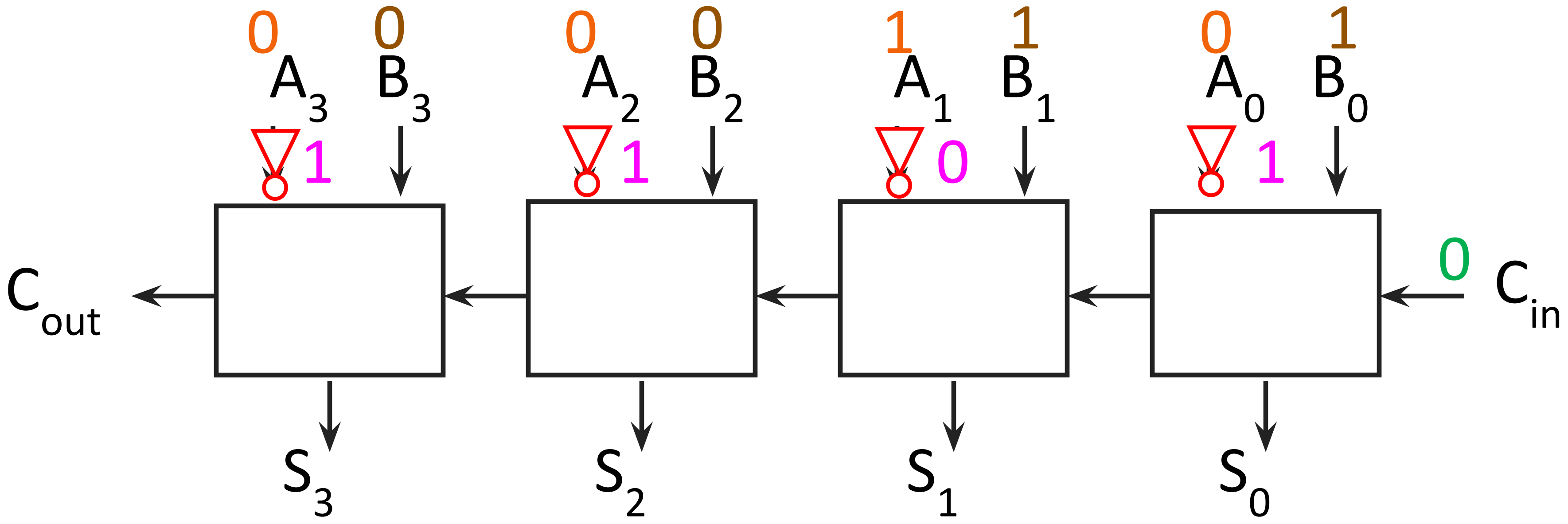
4-bit Adder to 4-bit Subtractor



- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2)

Cornell Bowers CIS
Computer Science

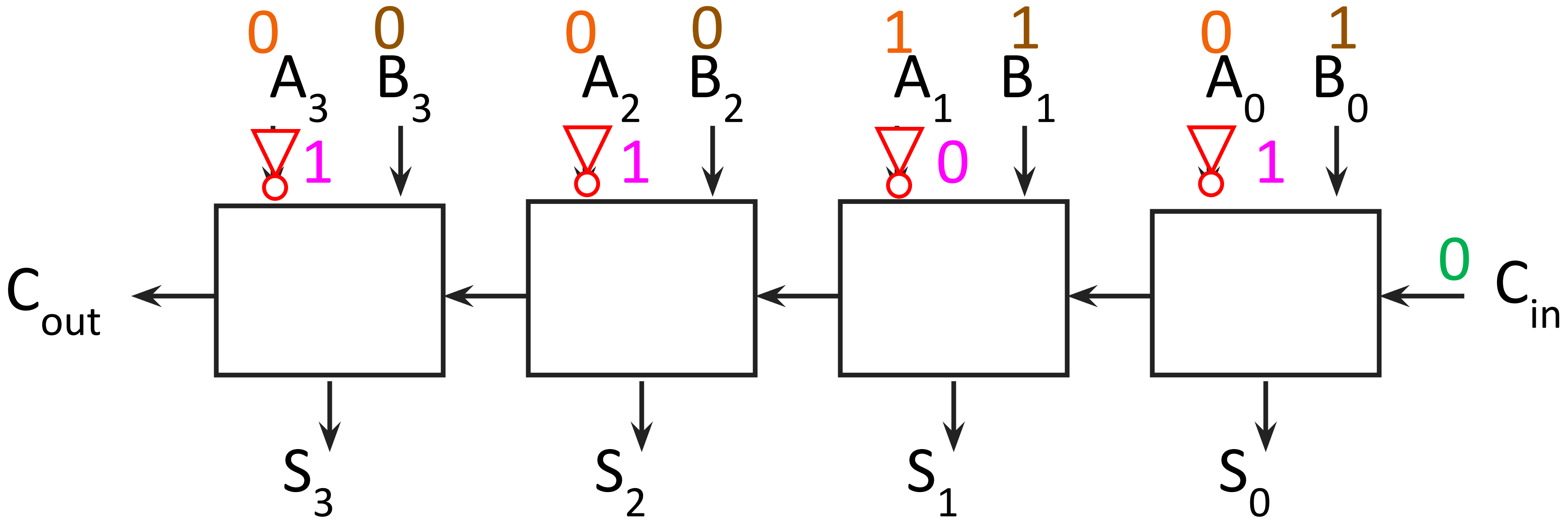
4-bit Adder to 4-bit Subtractor



- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 + 1

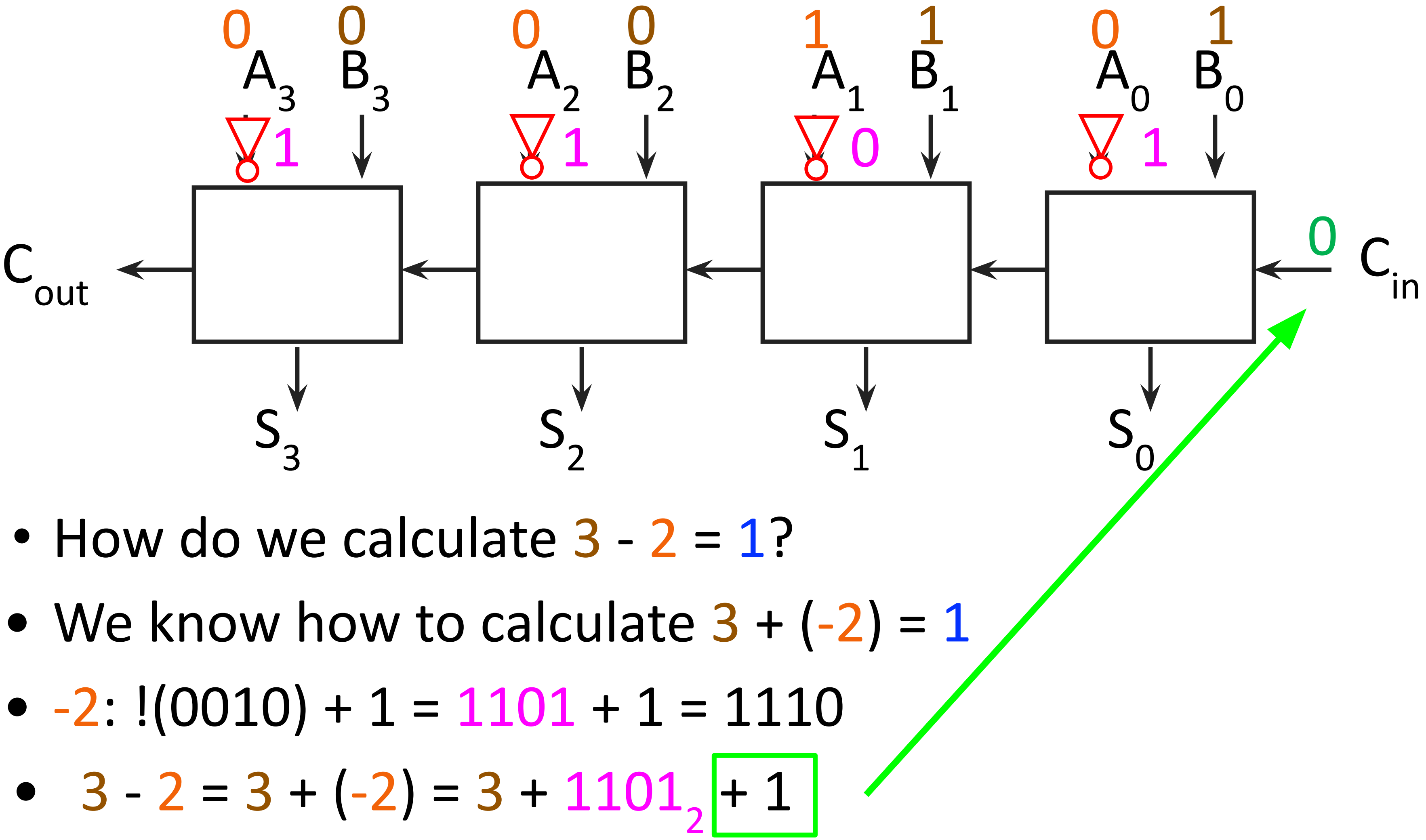
Cornell Bowers CIS
Computer Science

4-bit Adder to 4-bit Subtractor



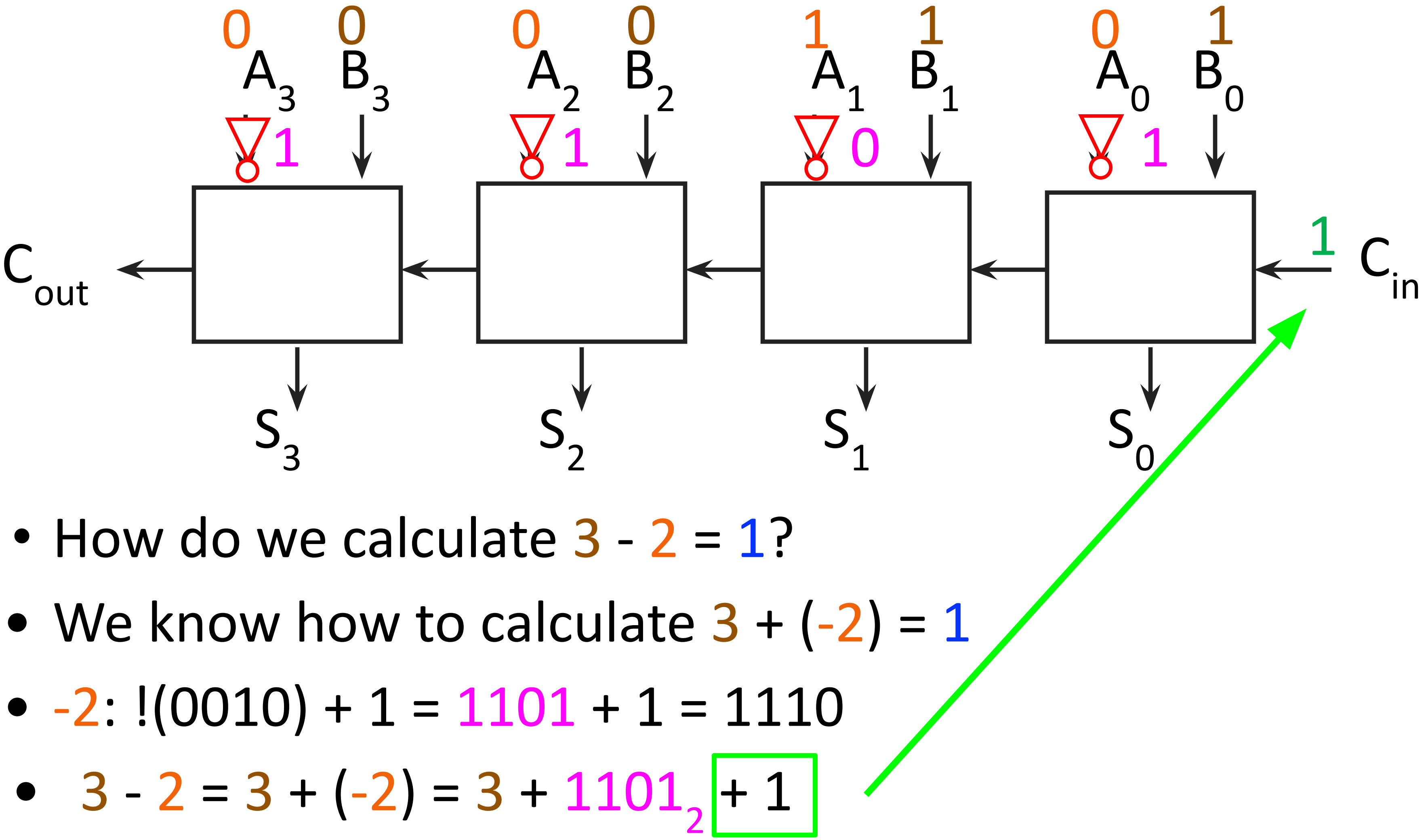
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 $\boxed{+ 1}$???

4-bit Adder to 4-bit Subtractor



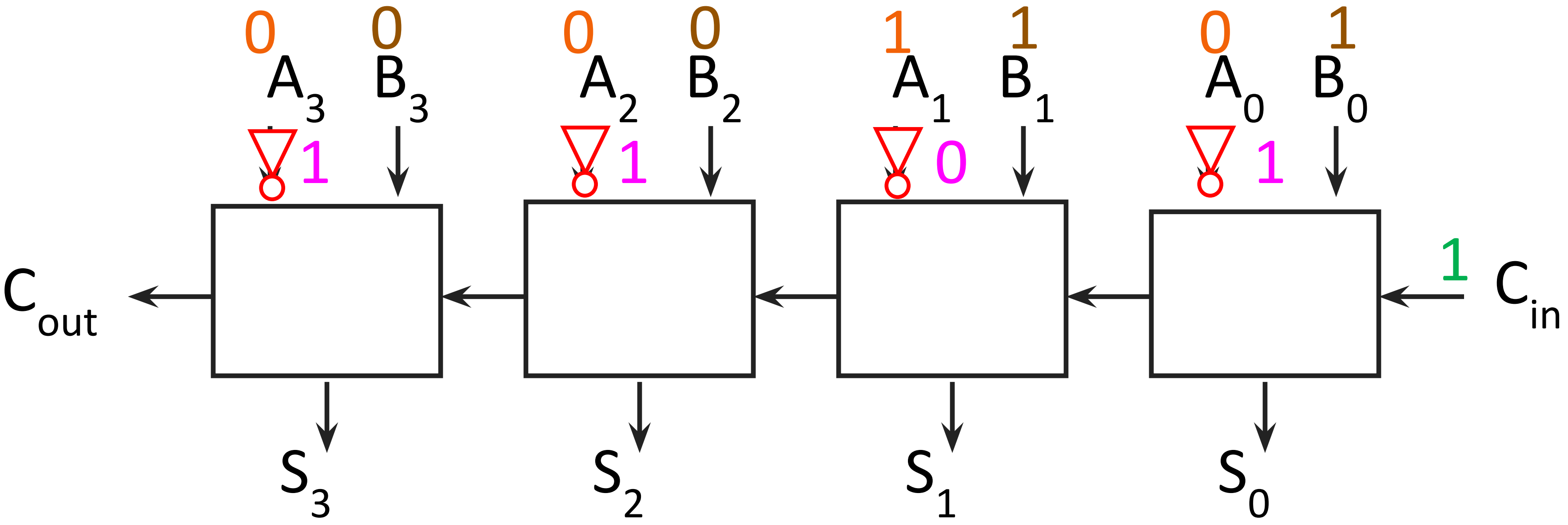
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 $\boxed{+ 1}$

4-bit Adder to 4-bit Subtractor



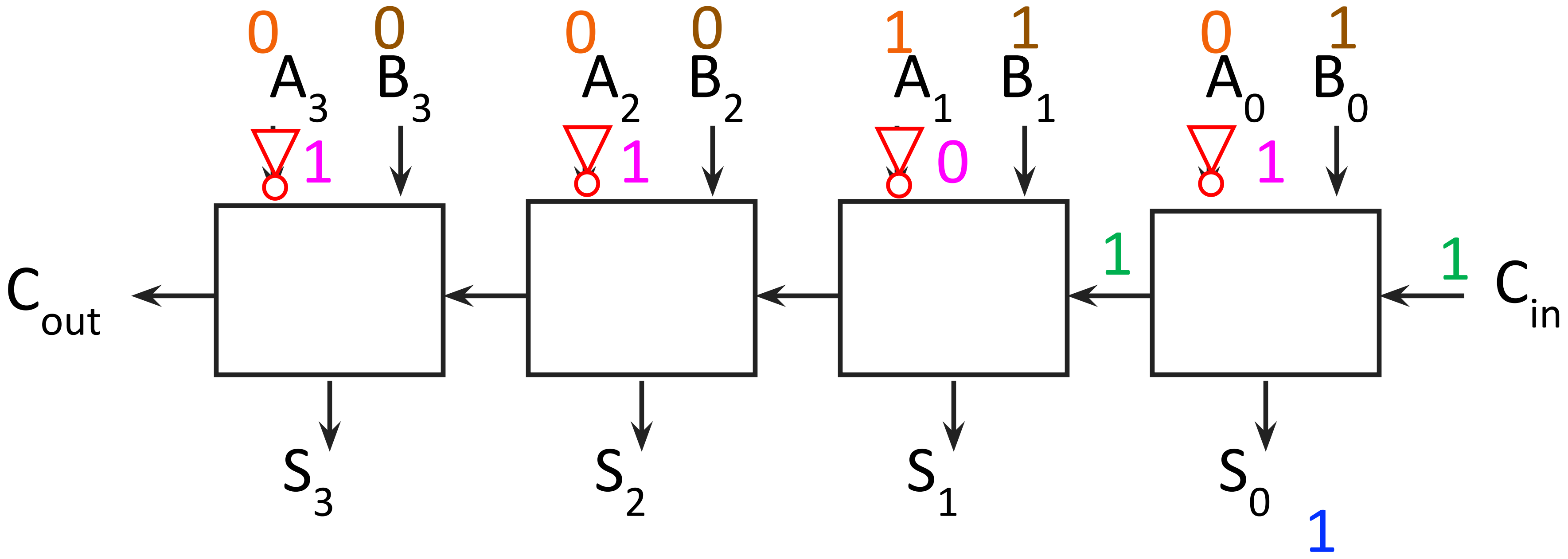
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 + 1

4-bit Adder to 4-bit Subtractor



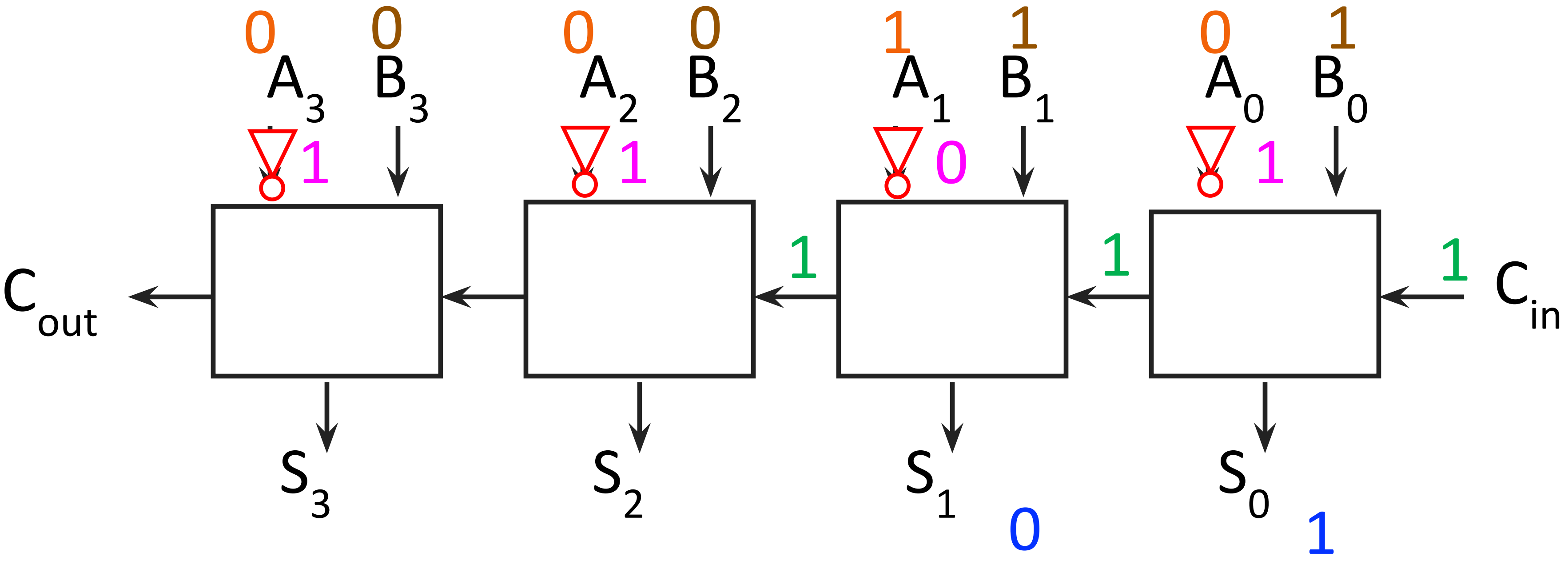
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 $\boxed{+ 1}$

4-bit Adder to 4-bit Subtractor



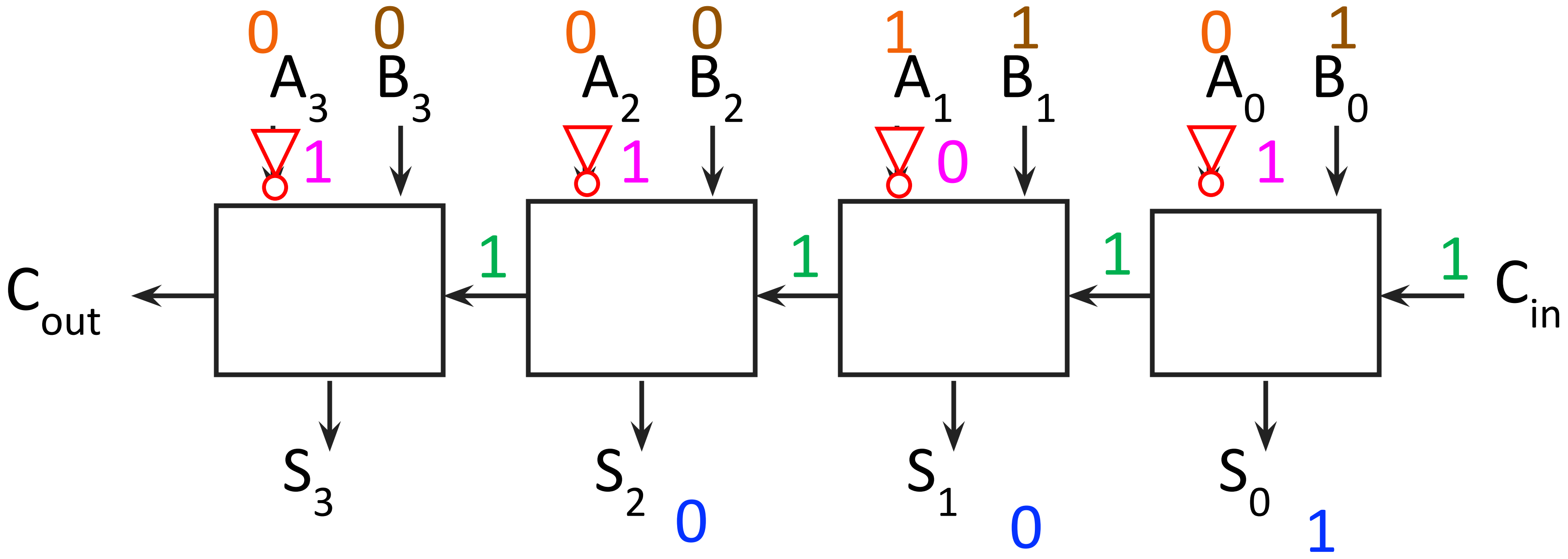
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 $\boxed{+ 1}$

4-bit Adder to 4-bit Subtractor



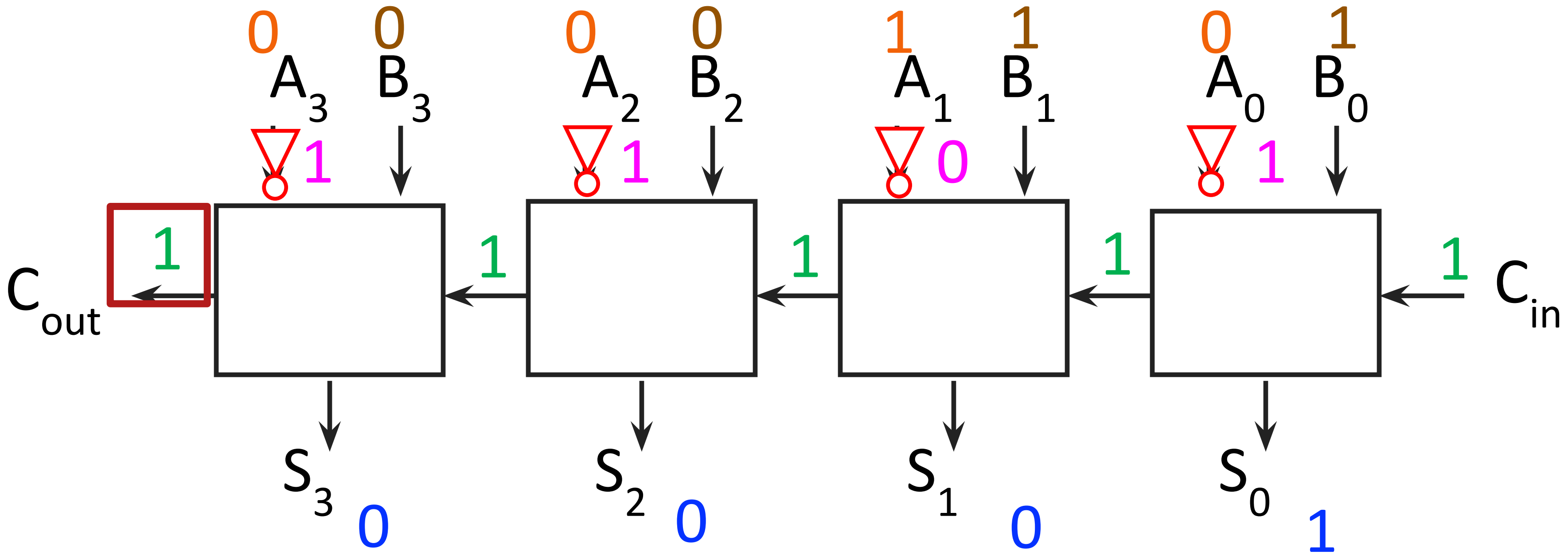
- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 $\boxed{+\ 1}$

4-bit Adder to 4-bit Subtractor



- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 $\boxed{+ 1}$

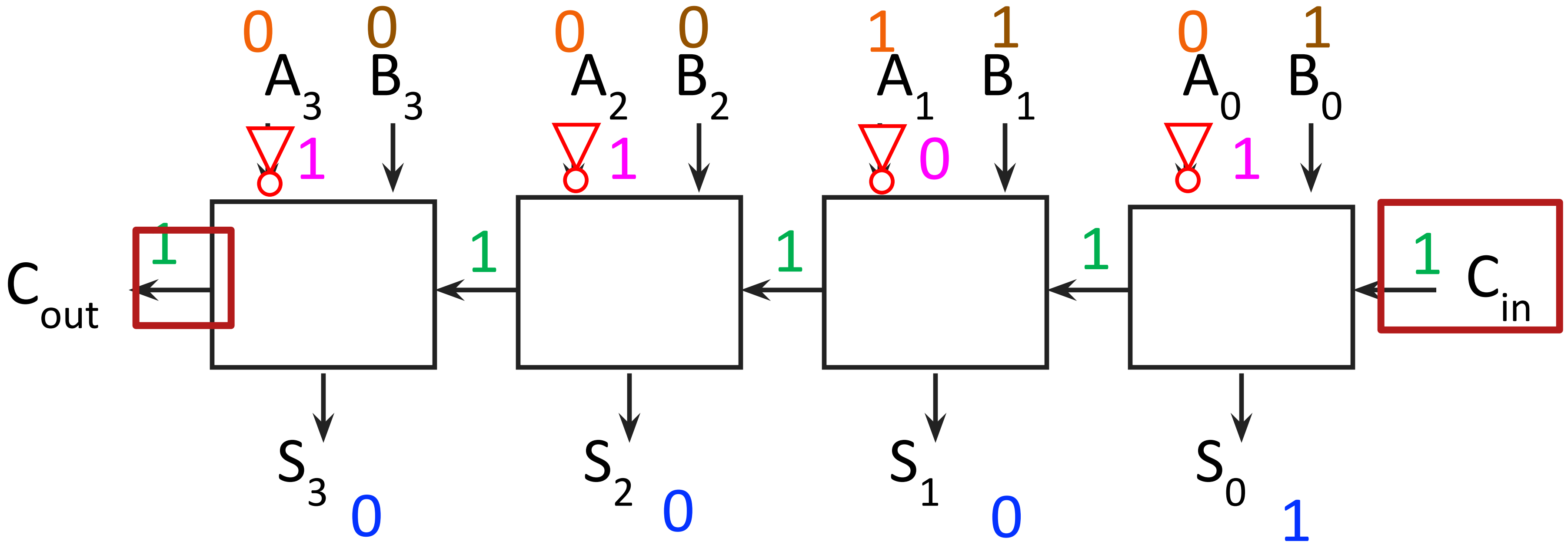
4-bit Adder to 4-bit Subtractor



- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 + 1

Overflow?

4-bit Adder to 4-bit Subtractor

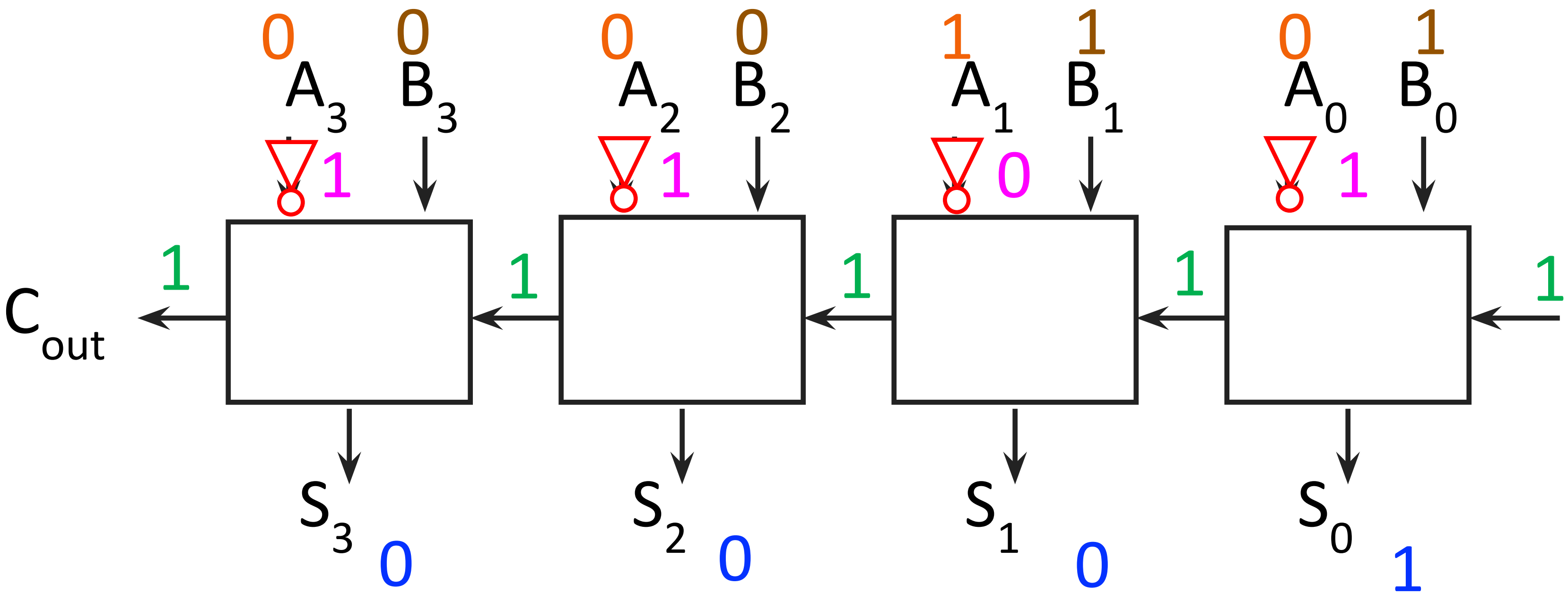


- How do we calculate 3 - 2 = 1?
- We know how to calculate 3 + (-2) = 1
- -2: !(0010) + 1 = 1101 + 1 = 1110
- 3 - 2 = 3 + (-2) = 3 + 1101_2 $\boxed{+ 1}$

Overflow?

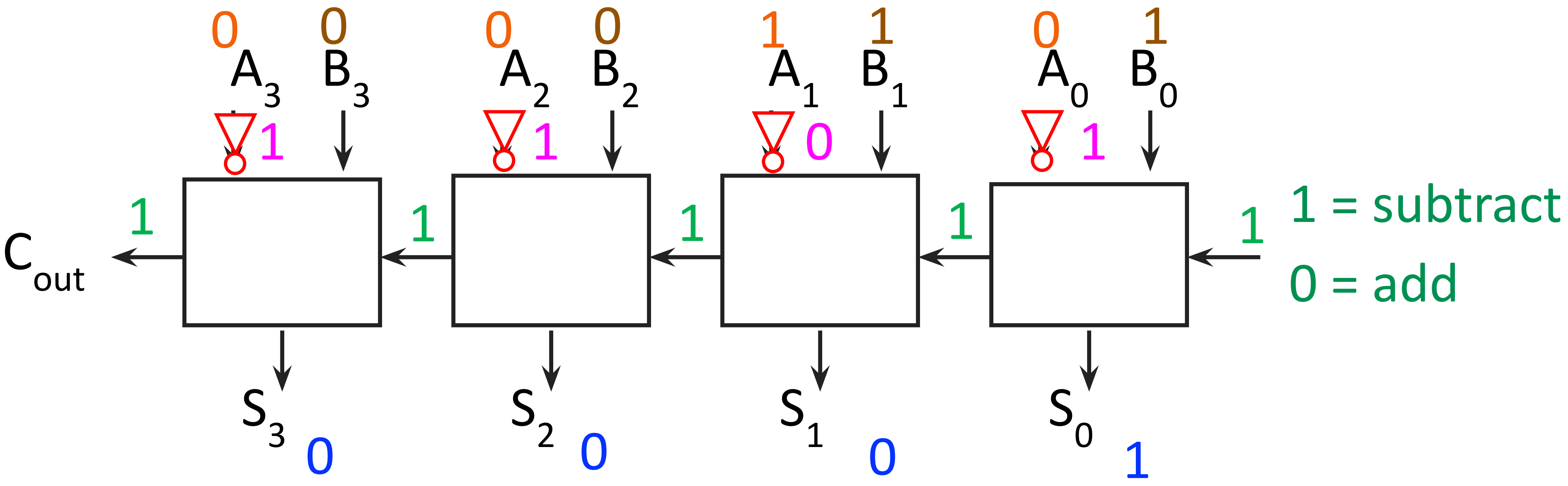
No. C_{in} = C_{out}

4-bit Adder to 4-bit Subtractor



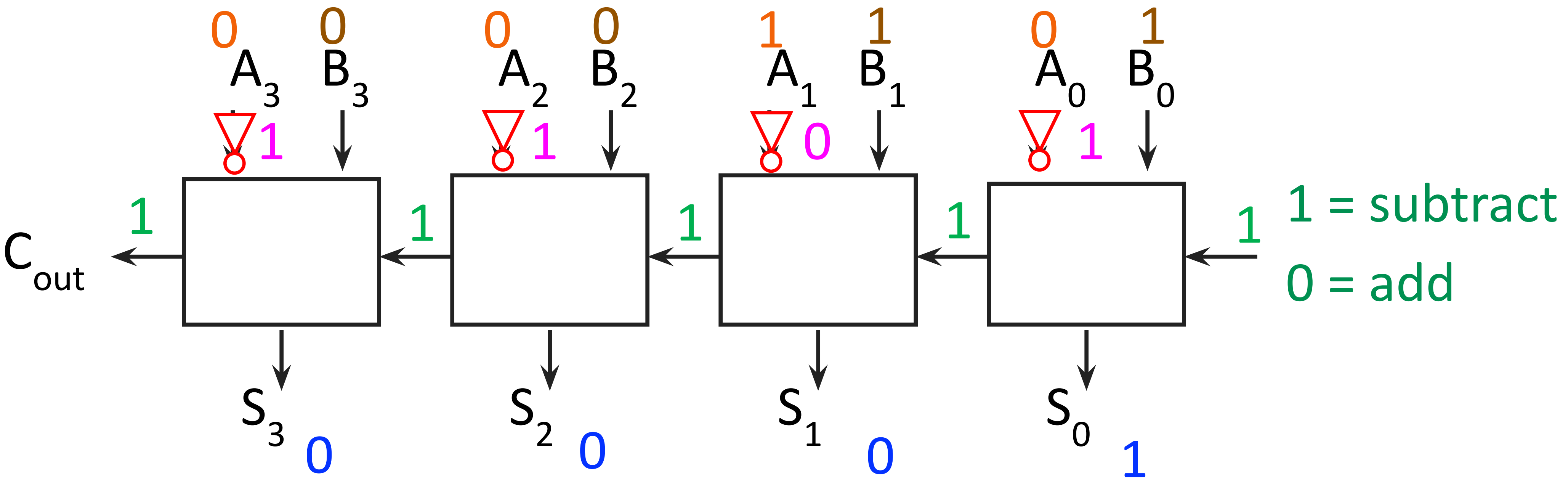
- Can we add and subtract with the same circuit?

4-bit Adder to 4-bit Subtractor



- Can we add and subtract with the same circuit?

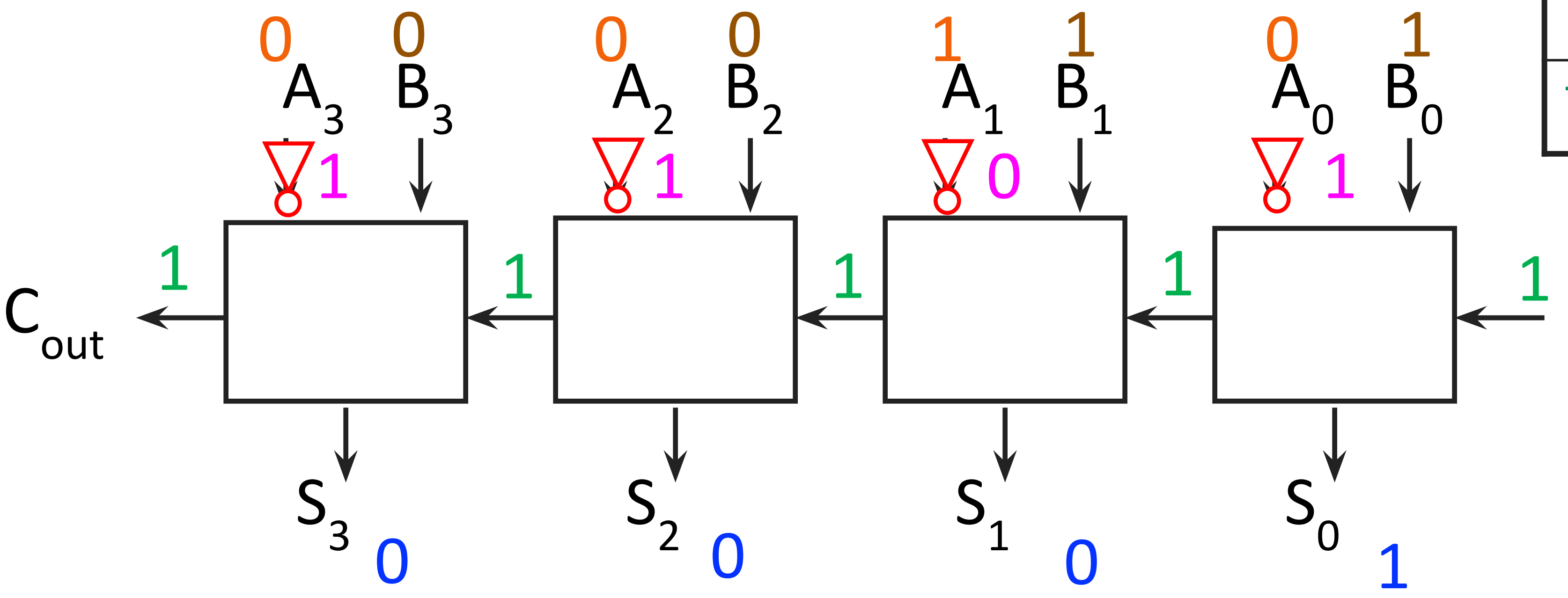
4-bit Adder to 4-bit Subtractor



- Can we add and subtract with the same circuit?
- How can we disable the inverter?

4-bit Adder to 4-bit Subtractor

sub?	in	out
0	0	
0	1	
1	0	
1	1	



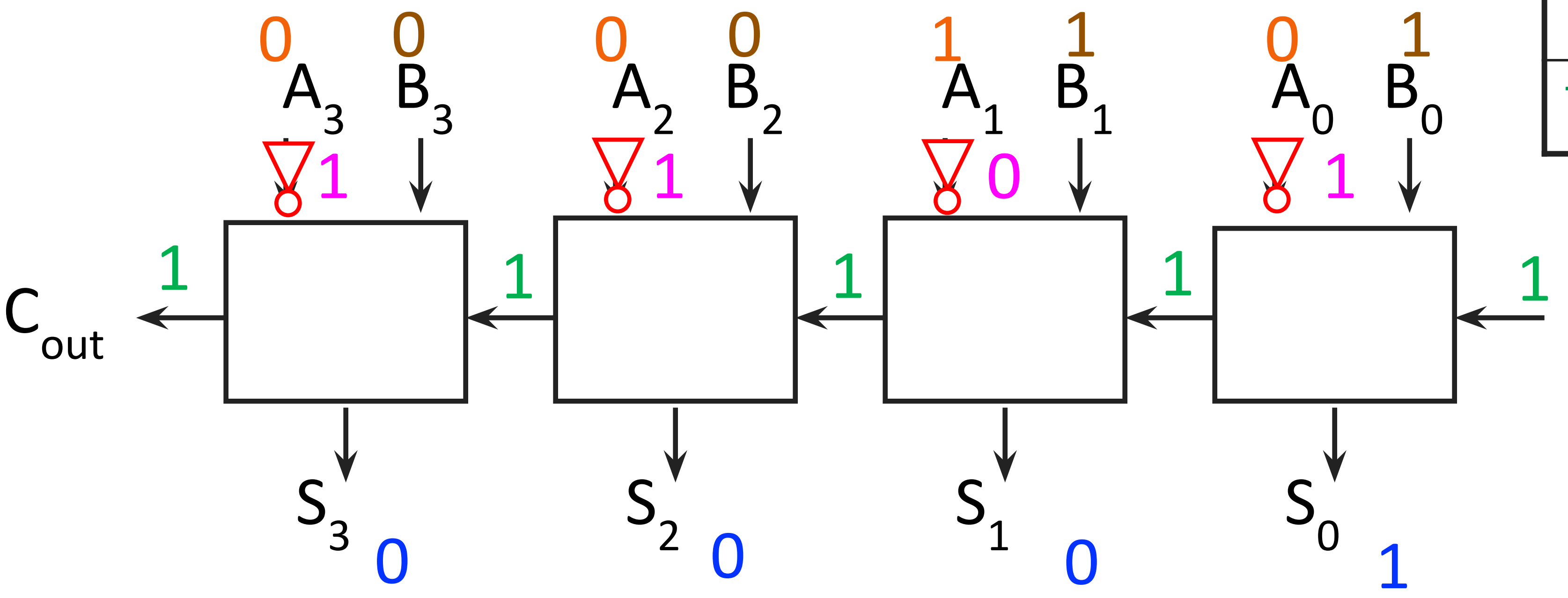
1 = subtract

0 = add

- Can we add and subtract with the same circuit?
- How can we disable the inverter?

4-bit Adder to 4-bit Subtractor

sub?	in	out
0	0	0
0	1	1
1	0	
1	1	



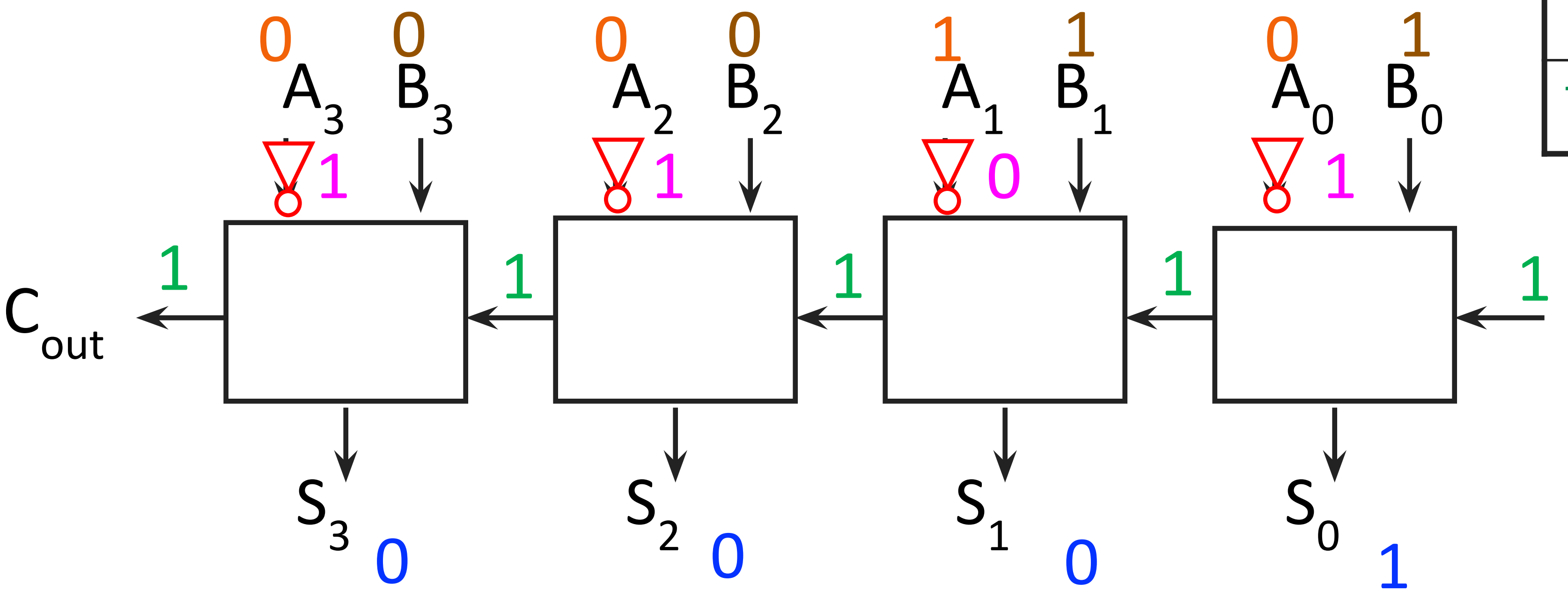
1 = subtract

0 = add

- Can we add and subtract with the same circuit?
- How can we disable the inverter?

4-bit Adder to 4-bit Subtractor

sub?	in	out
0	0	0
0	1	1
1	0	1
1	1	0

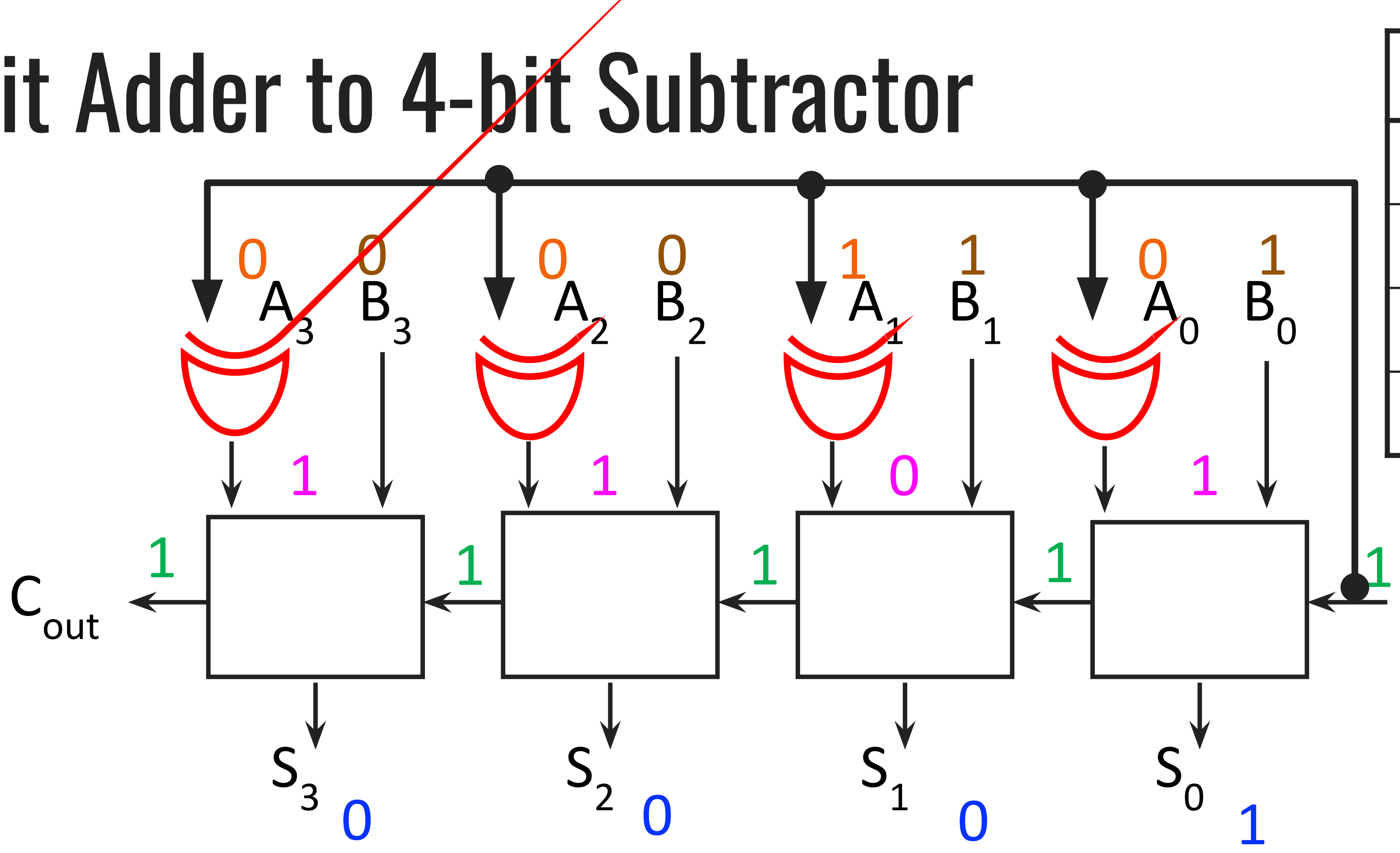


1 = subtract

0 = add

- Can we add and subtract with the same circuit?
- How can we disable the inverter?

4-bit Adder to 4-bit Subtractor



sub?	in	out
0	0	0
0	1	1
1	0	1
1	1	0

1 = subtract

0 = add

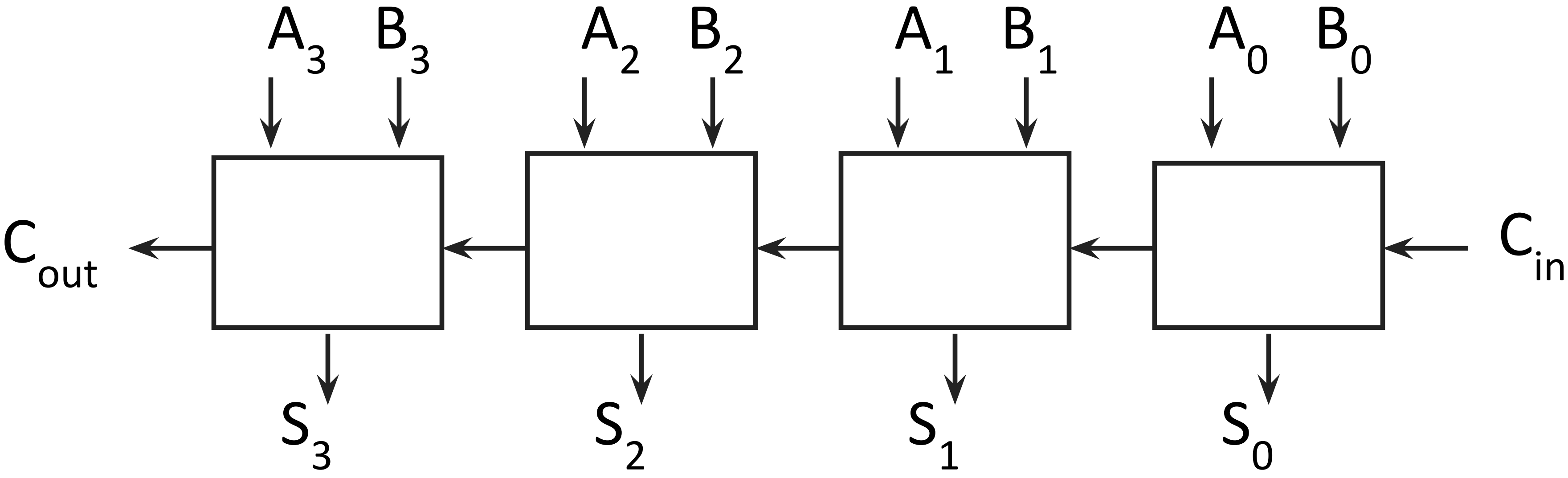
- Can we add and subtract with the same circuit?
- How can we disable the inverter?

Cornell Bowers CIS
Computer Science

State

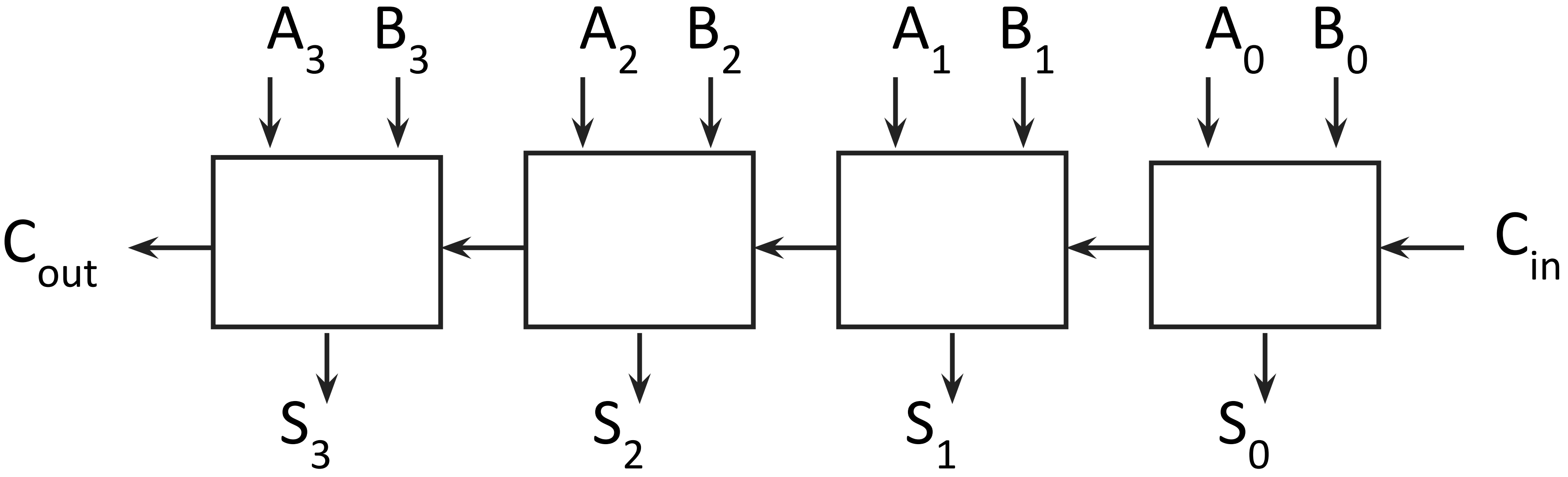
CS 3410: Computer System Organization and Programming

4-bit Adder: Delay Model



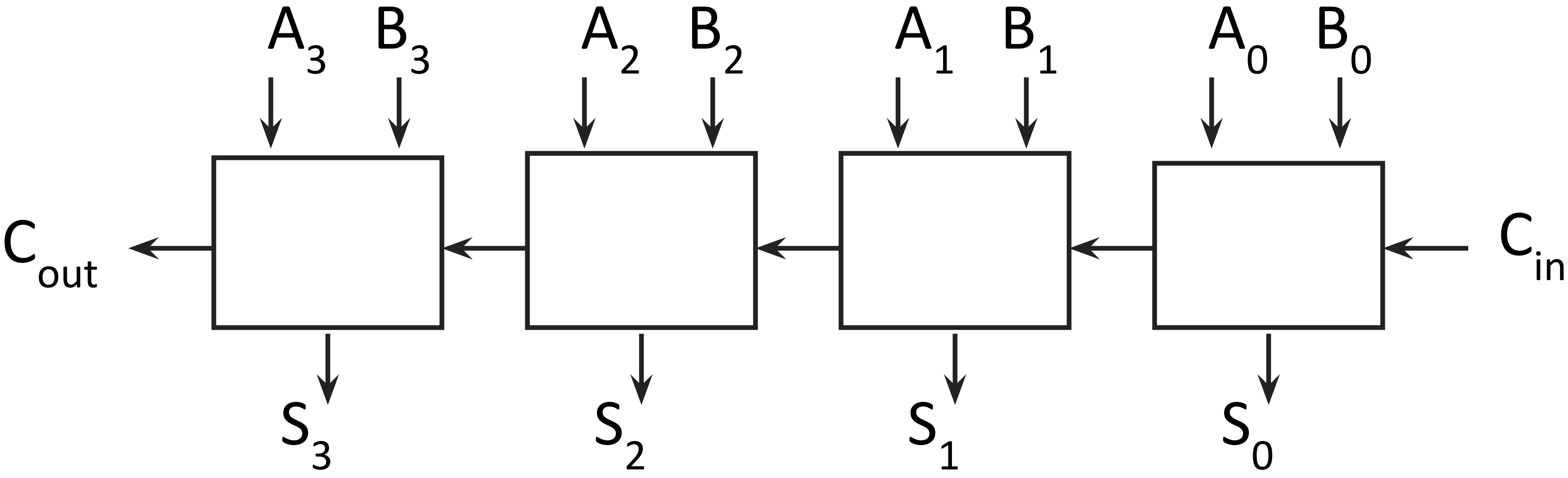
- So far, gates compute instantaneous

4-bit Adder: Delay Model



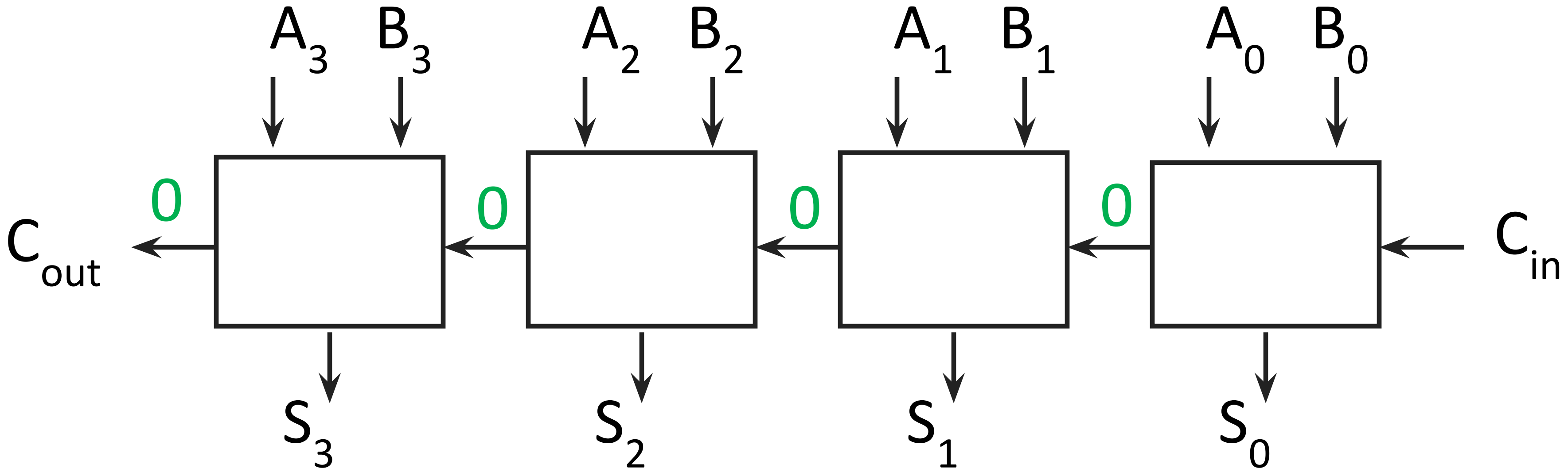
- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.

4-bit Adder: Delay Model



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

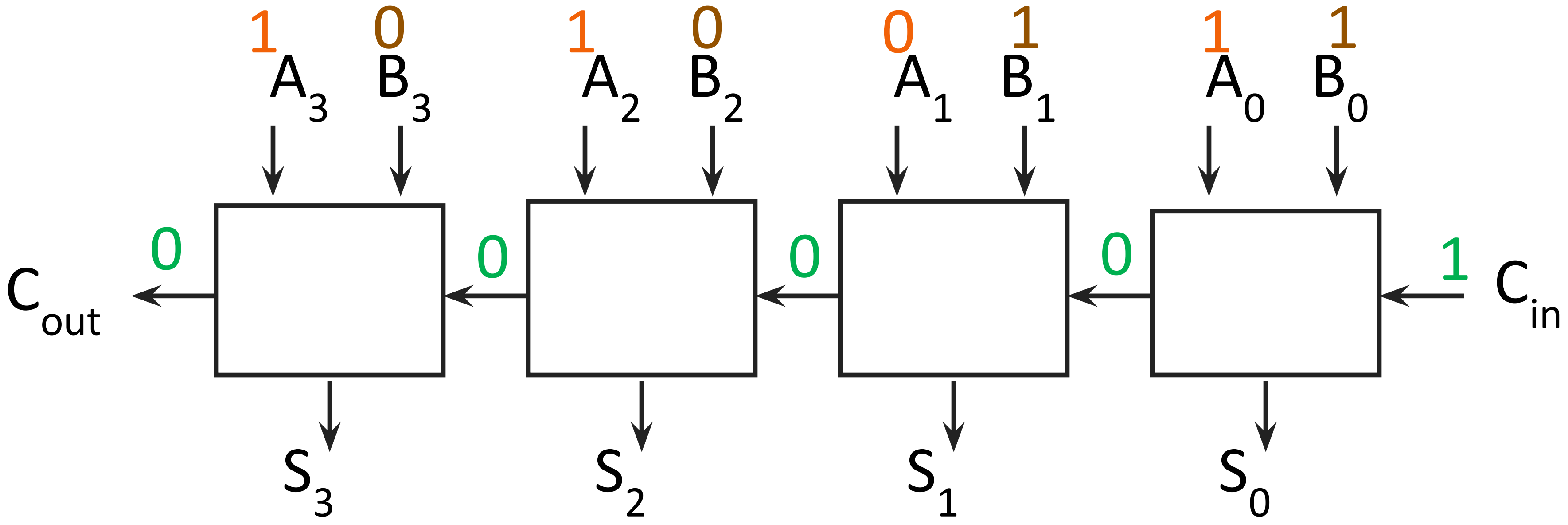
4-bit Adder: Delay Model



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

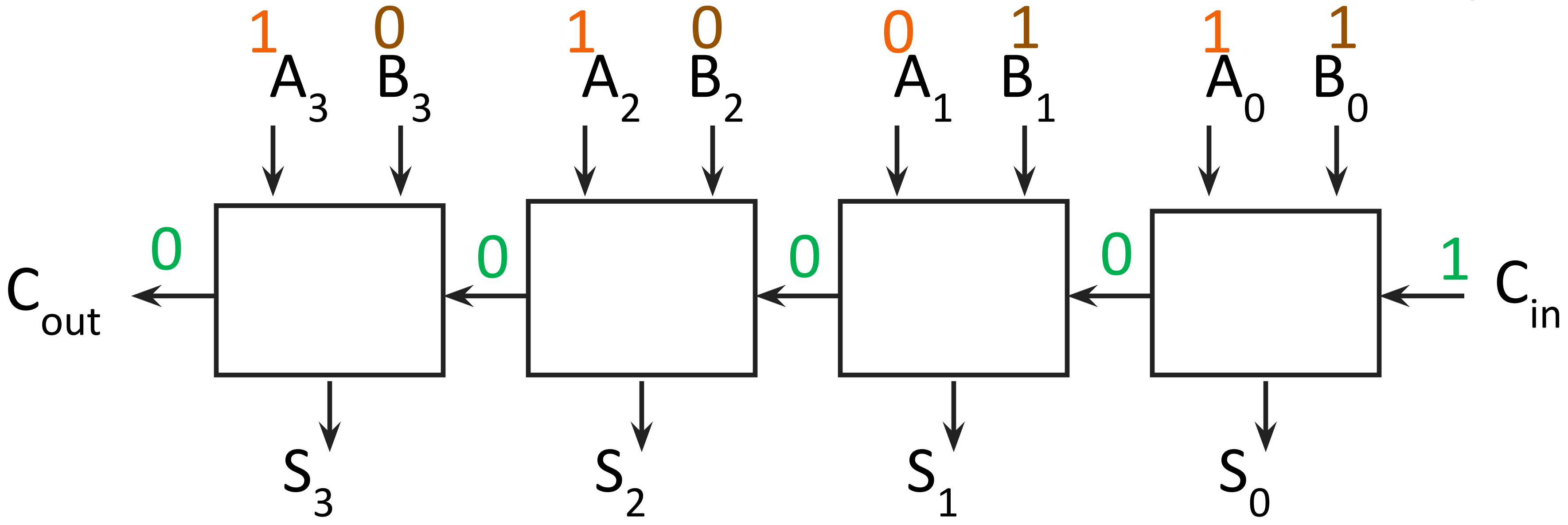
t = 0



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

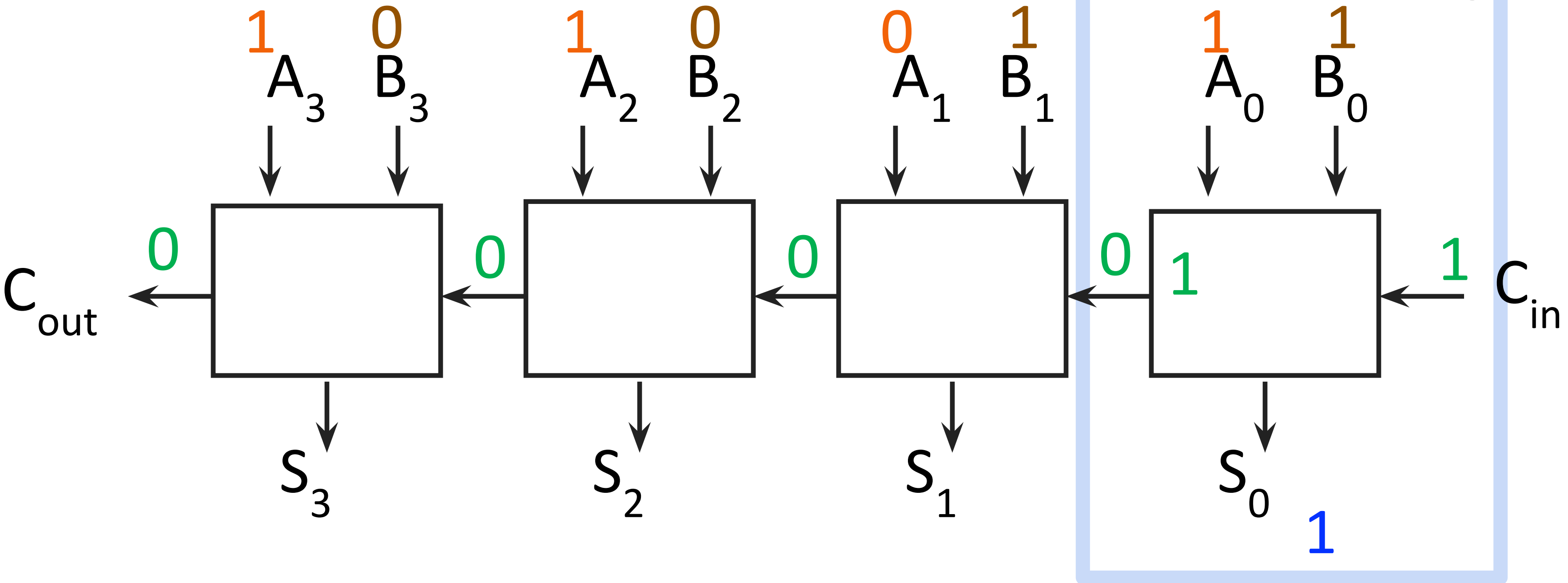
t = 0 → t = 1



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

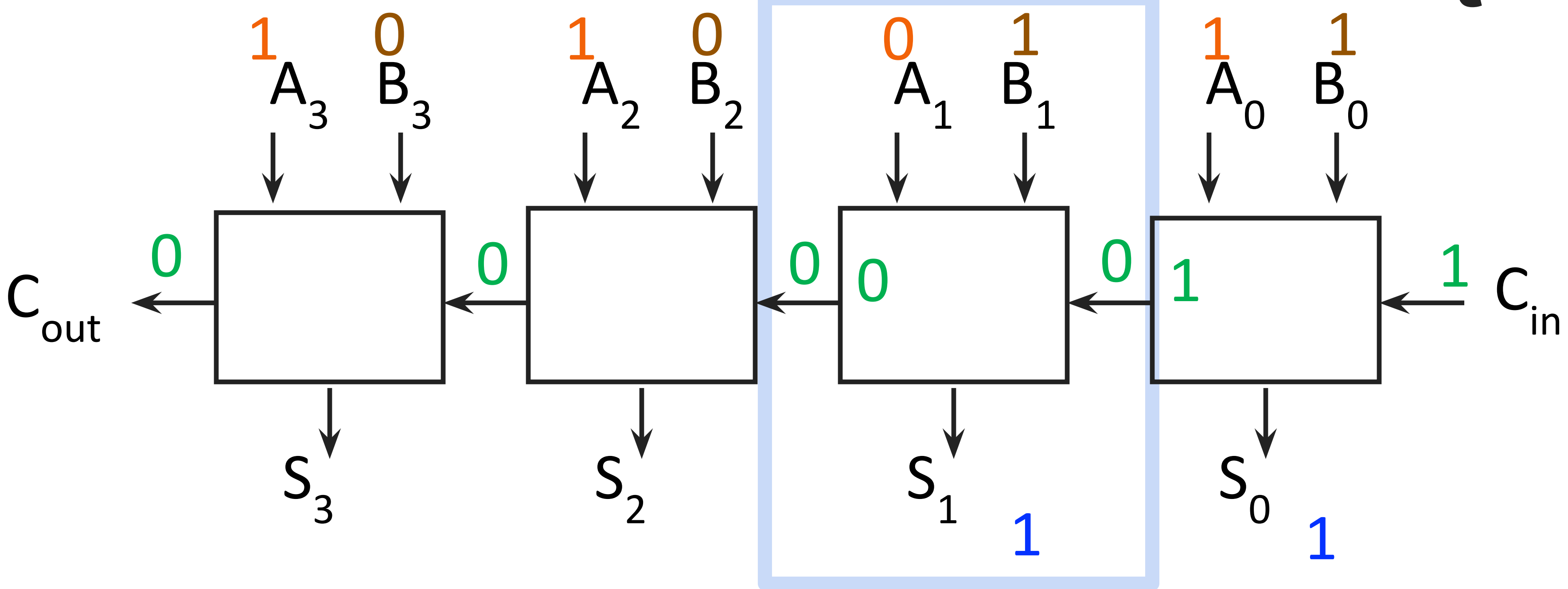
t = 0 → t = 1



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

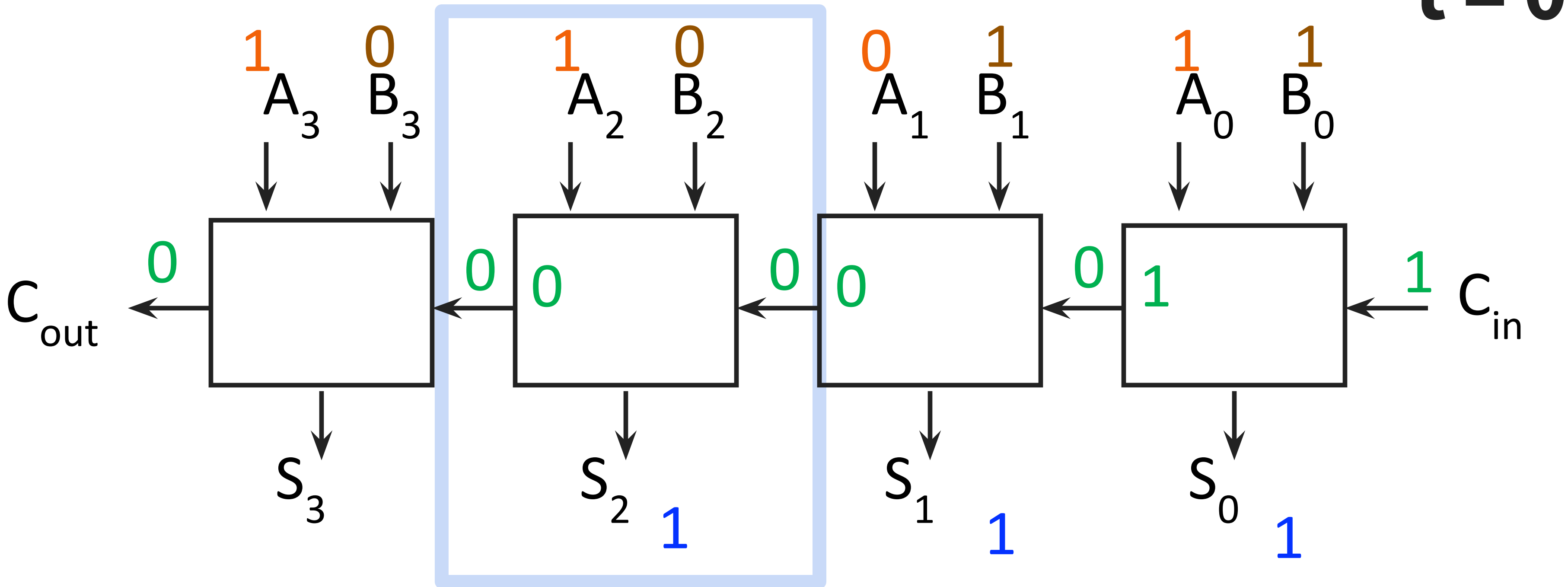
t = 0 → t = 1



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

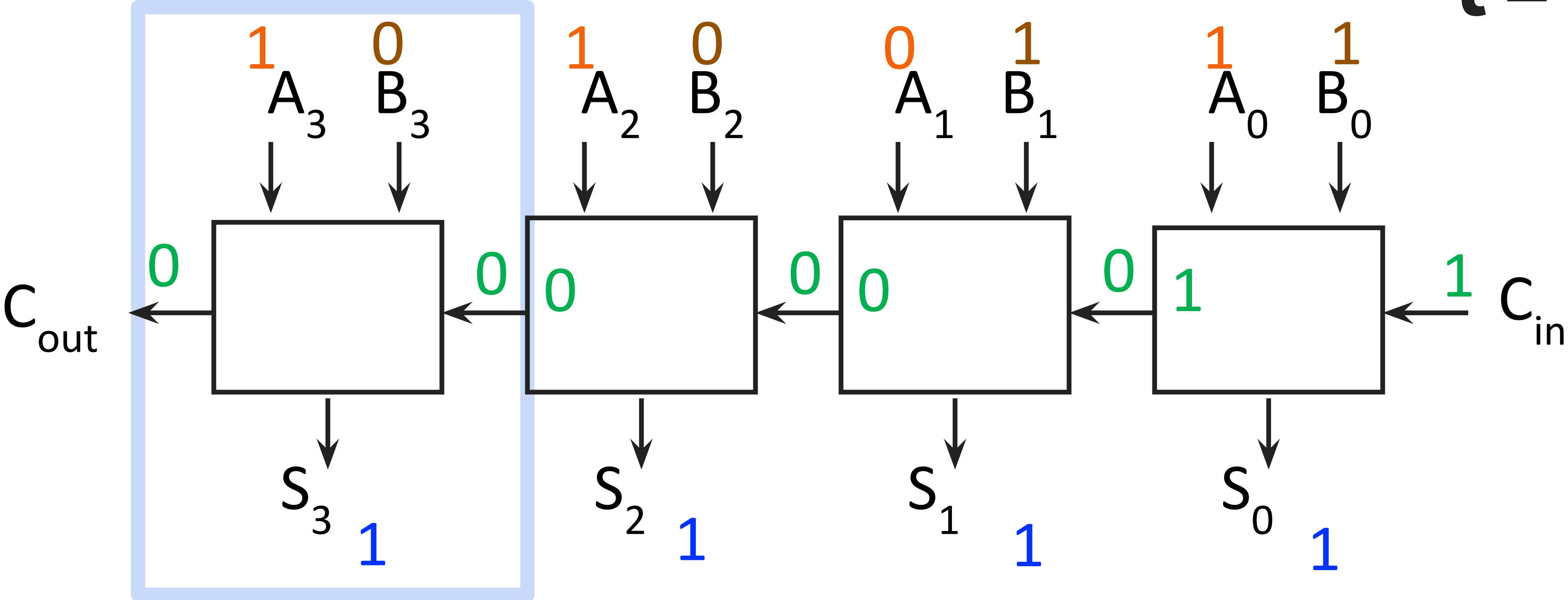
t = 0 → t = 1



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

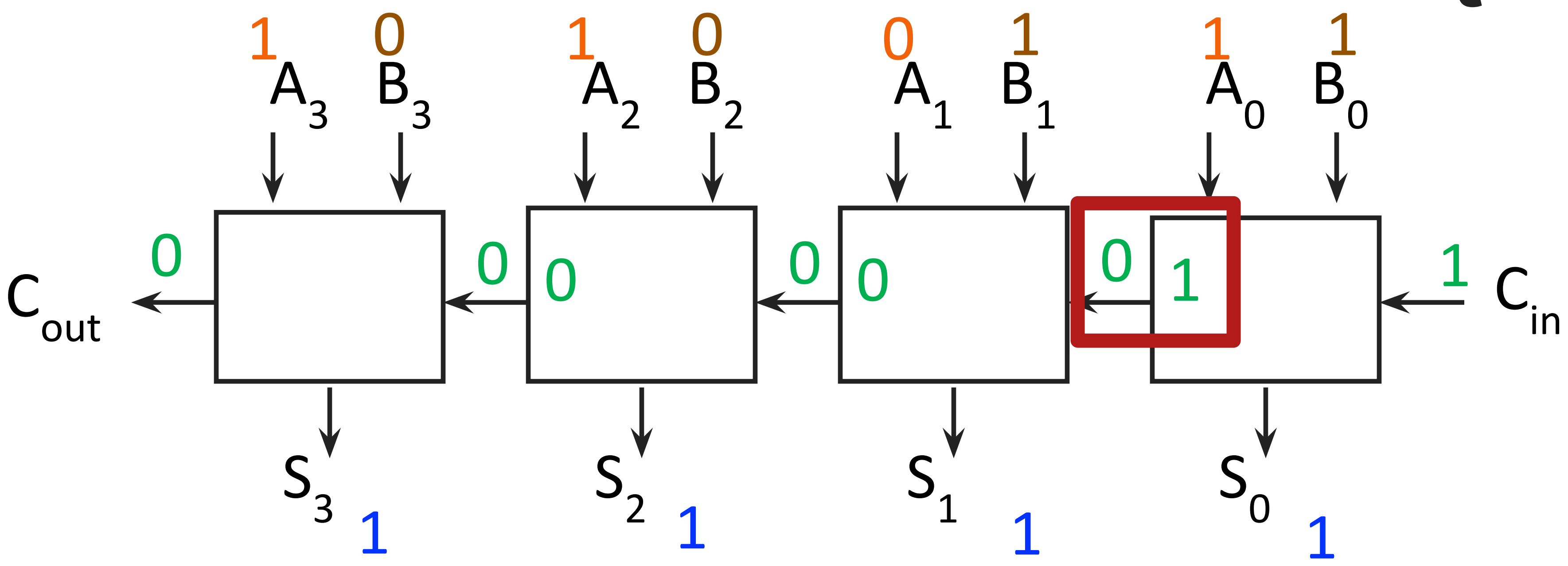
t = 0 → t = 1



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

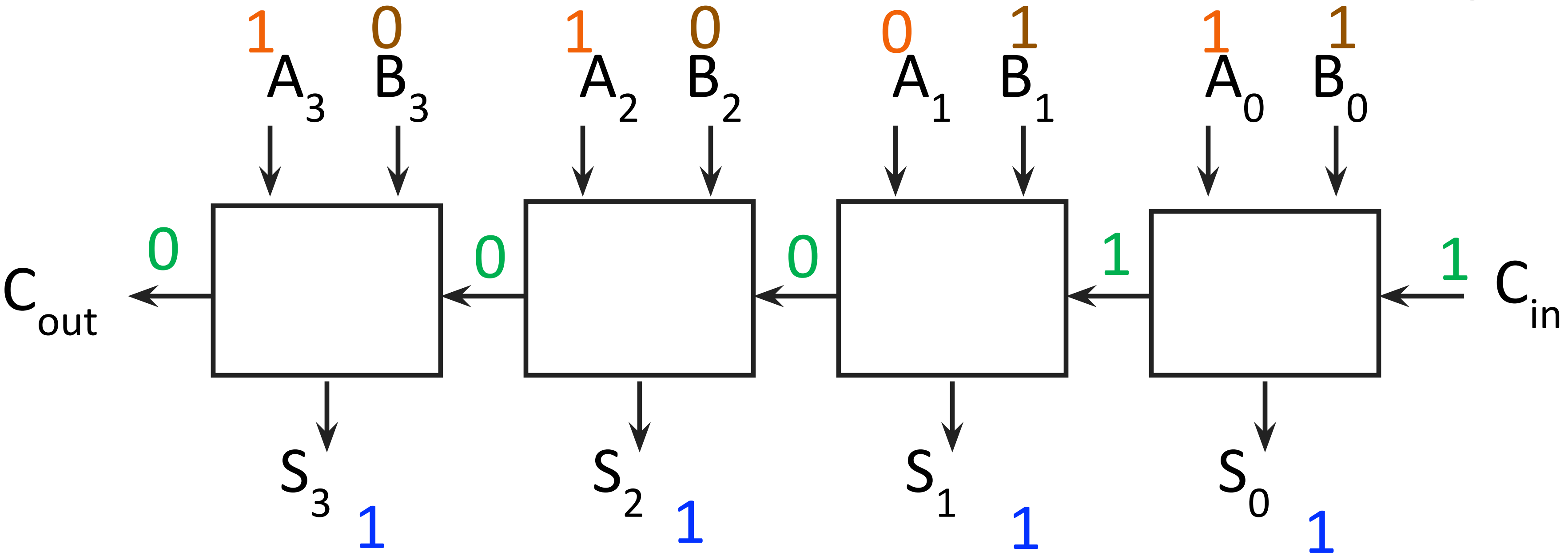
t = 0 → t = 1



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

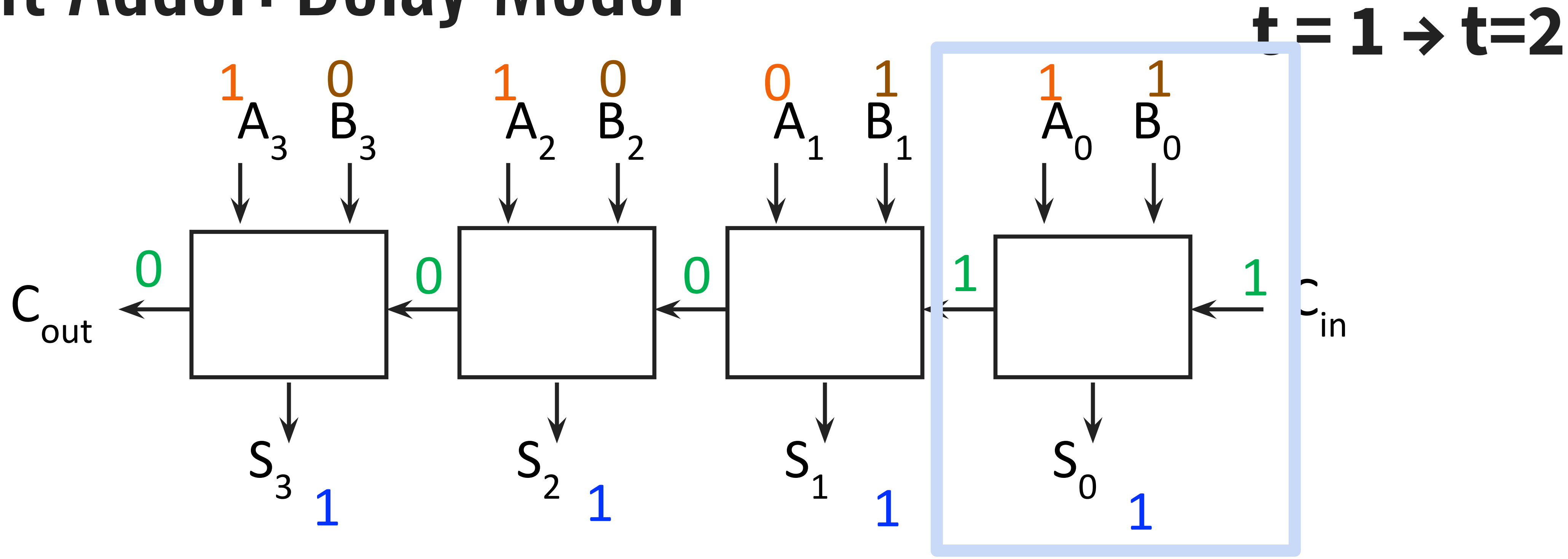
4-bit Adder: Delay Model

t = 1



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

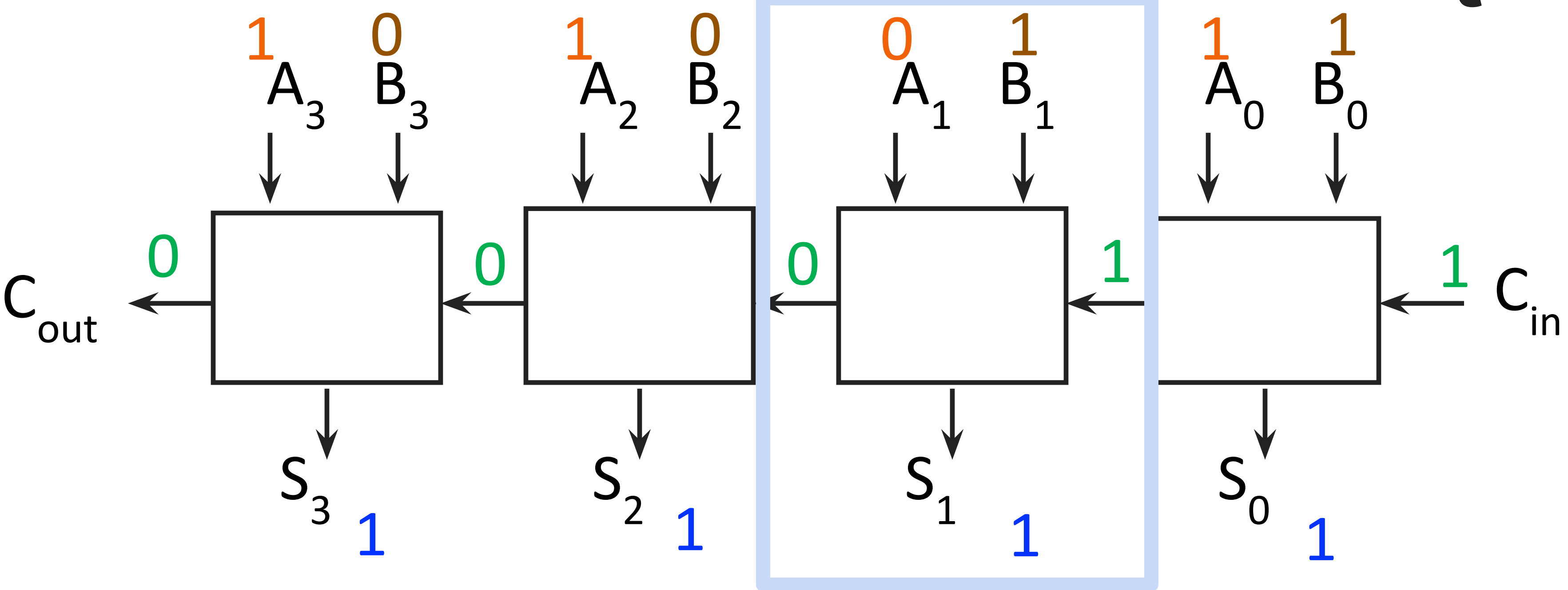


stays the same!

- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

t = 1 → t=2

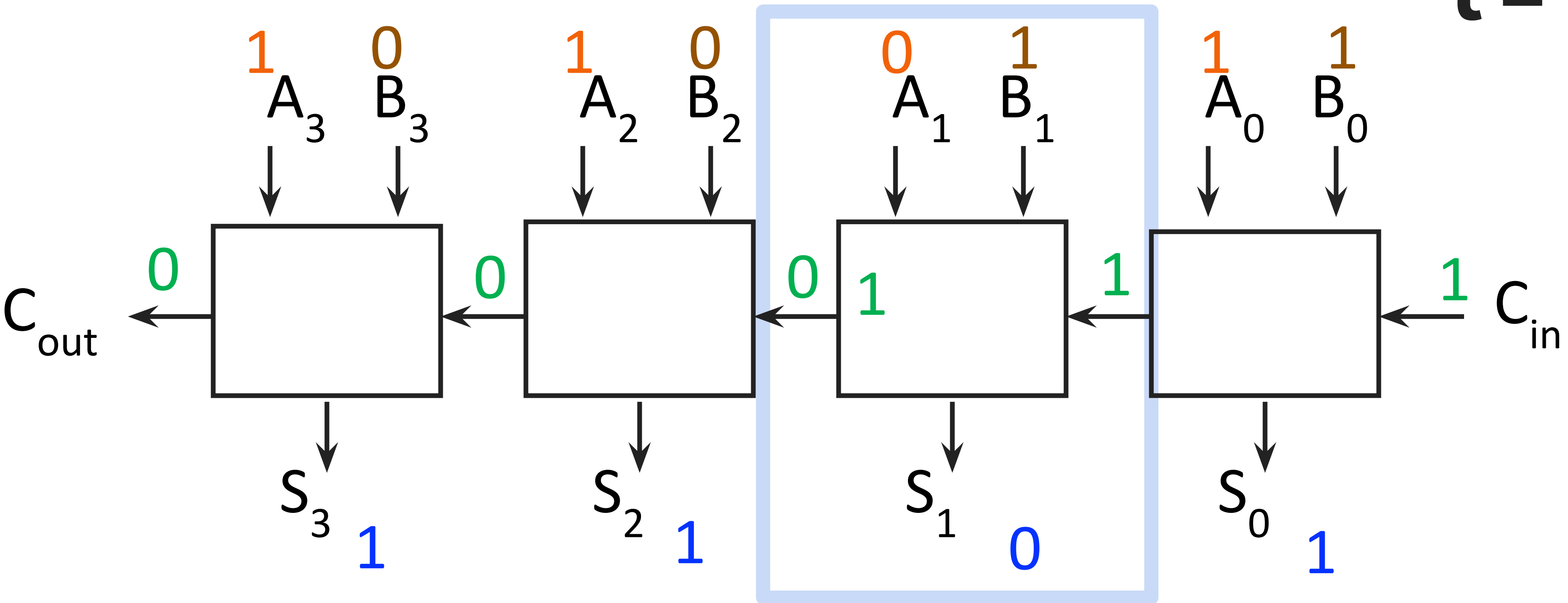


needs to be recomputed

- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

t = 1 → t=2

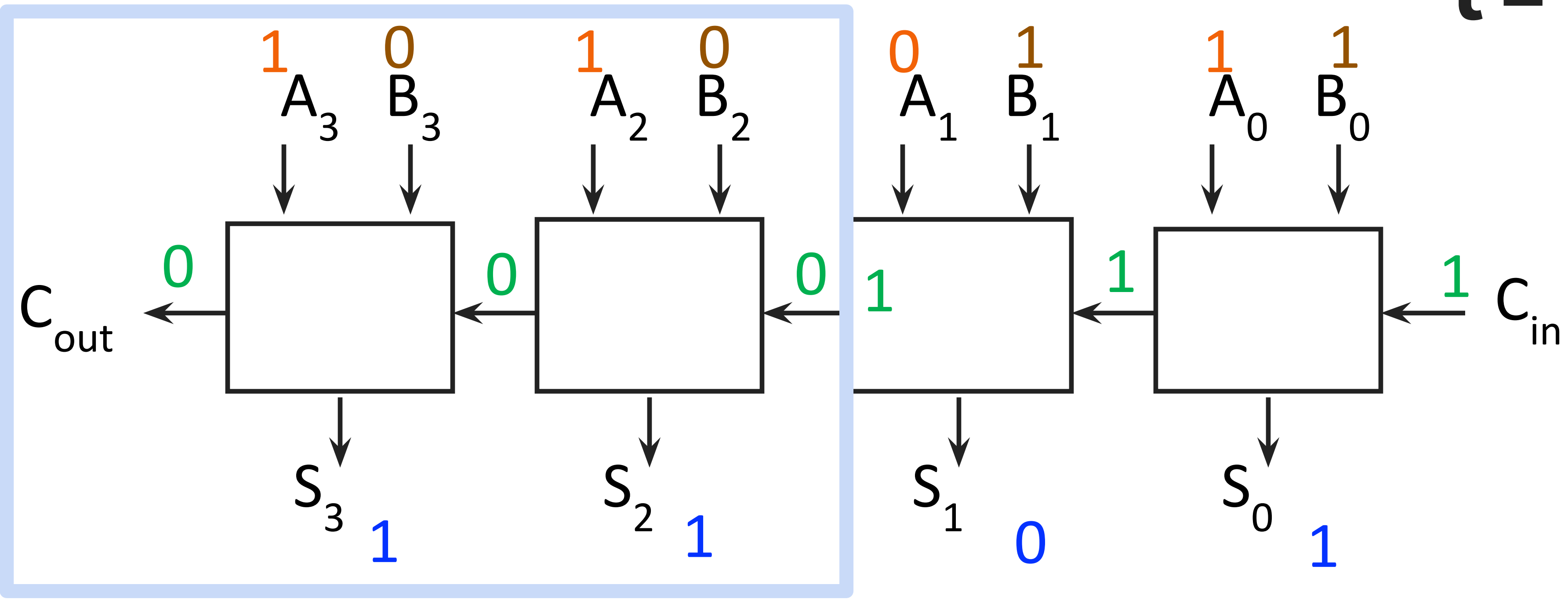


needs to be recomputed

- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

t = 1 → t=2

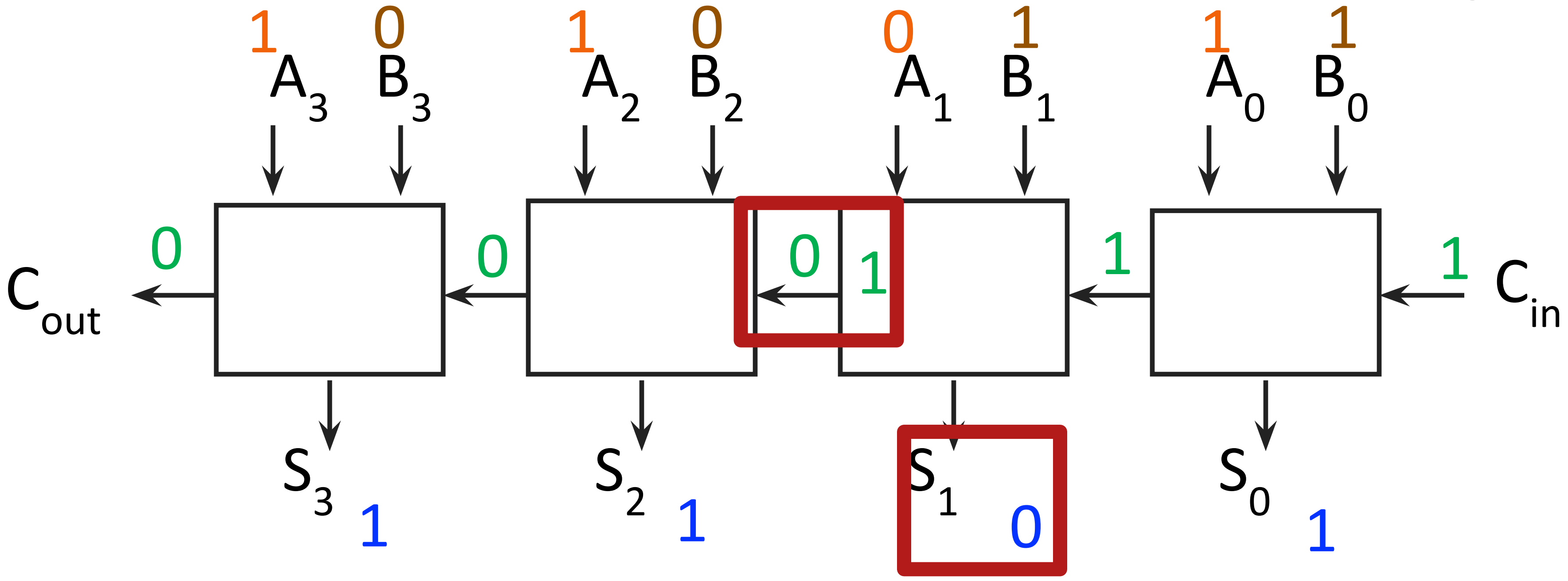


stays the same!

- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

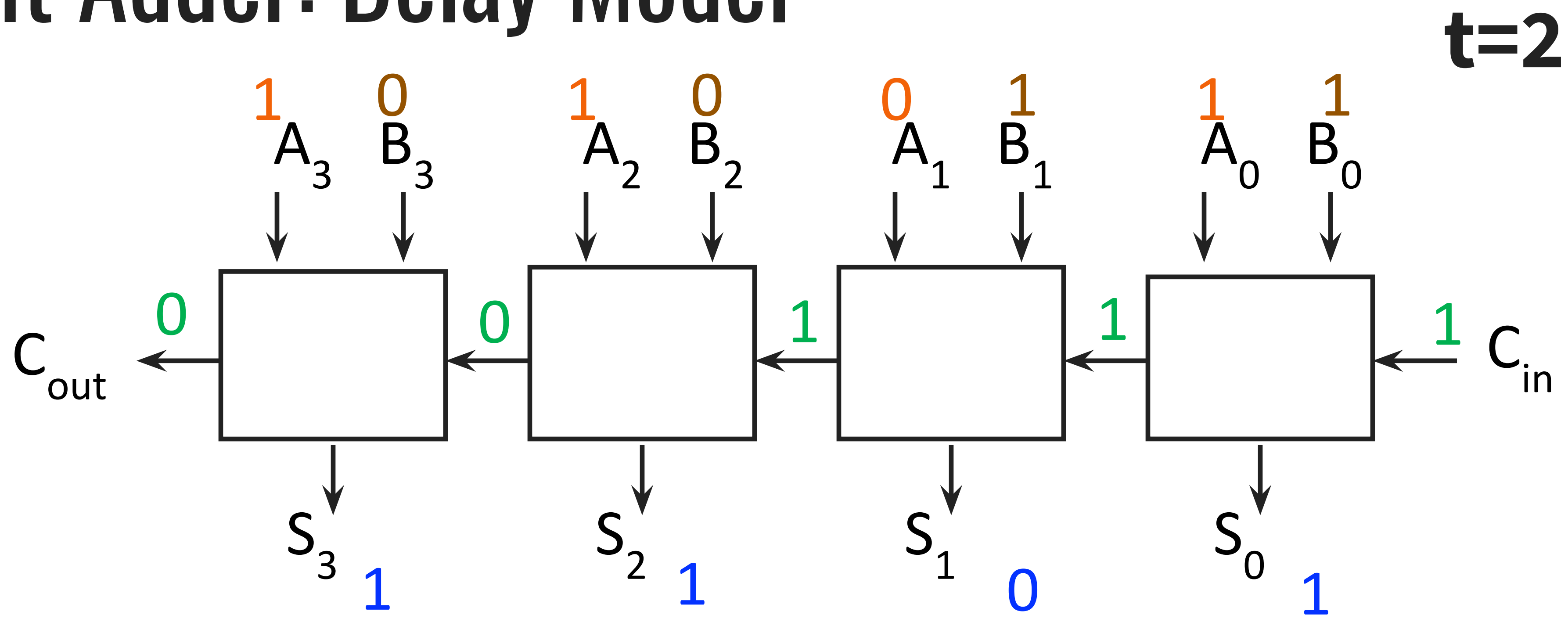
4-bit Adder: Delay Model

t = 1 → t=2



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

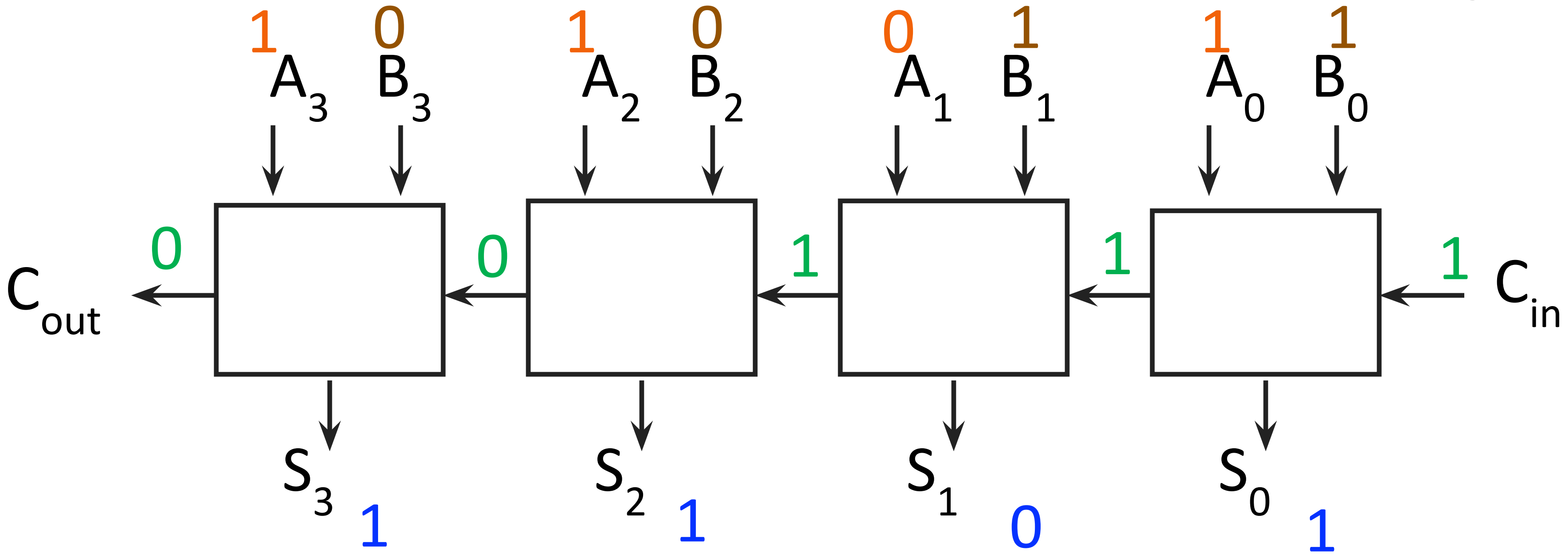
4-bit Adder: Delay Model



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

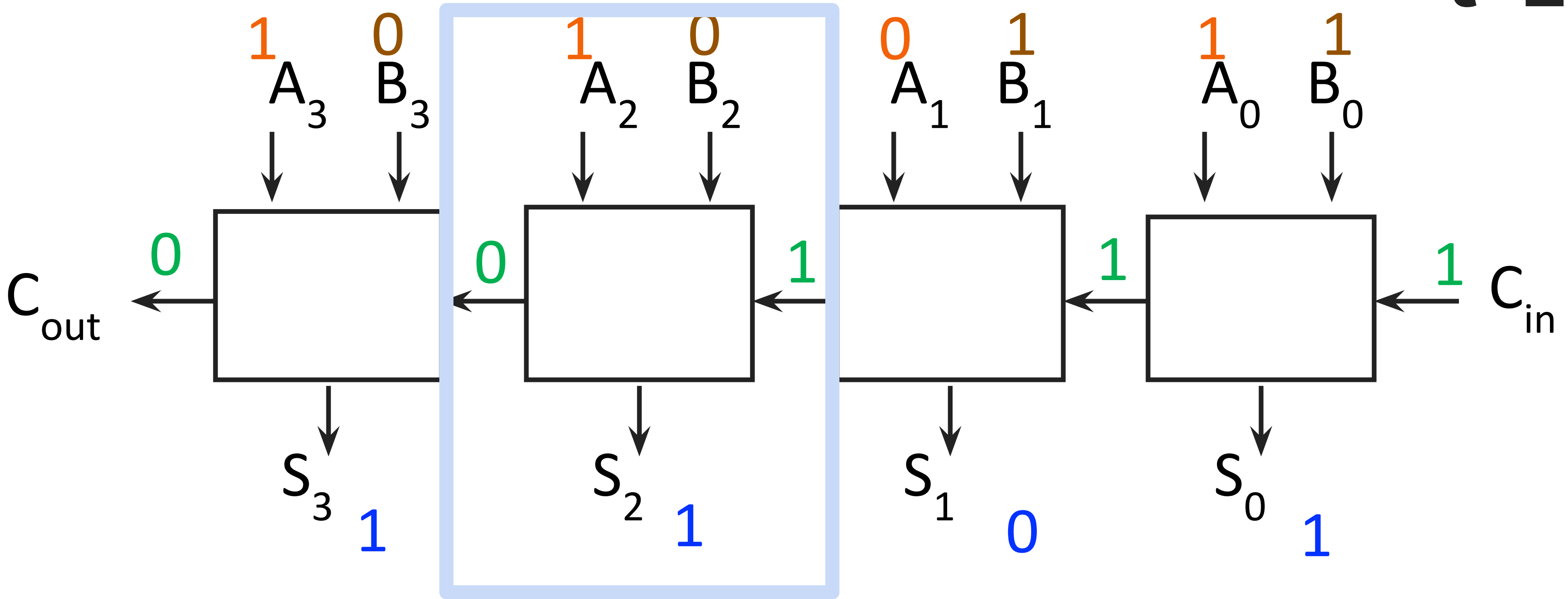
t=2 → t=3



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

t=2 → t=3

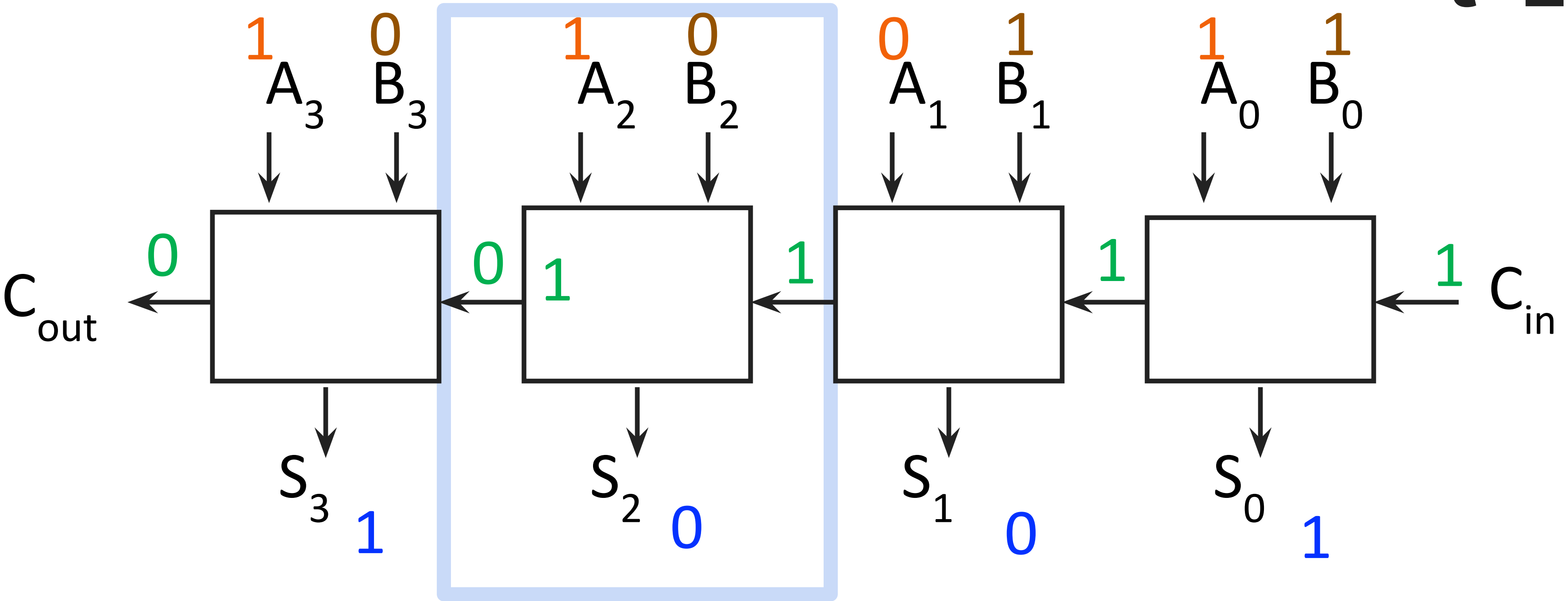


needs to be recomputed

- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

t=2 → t=3

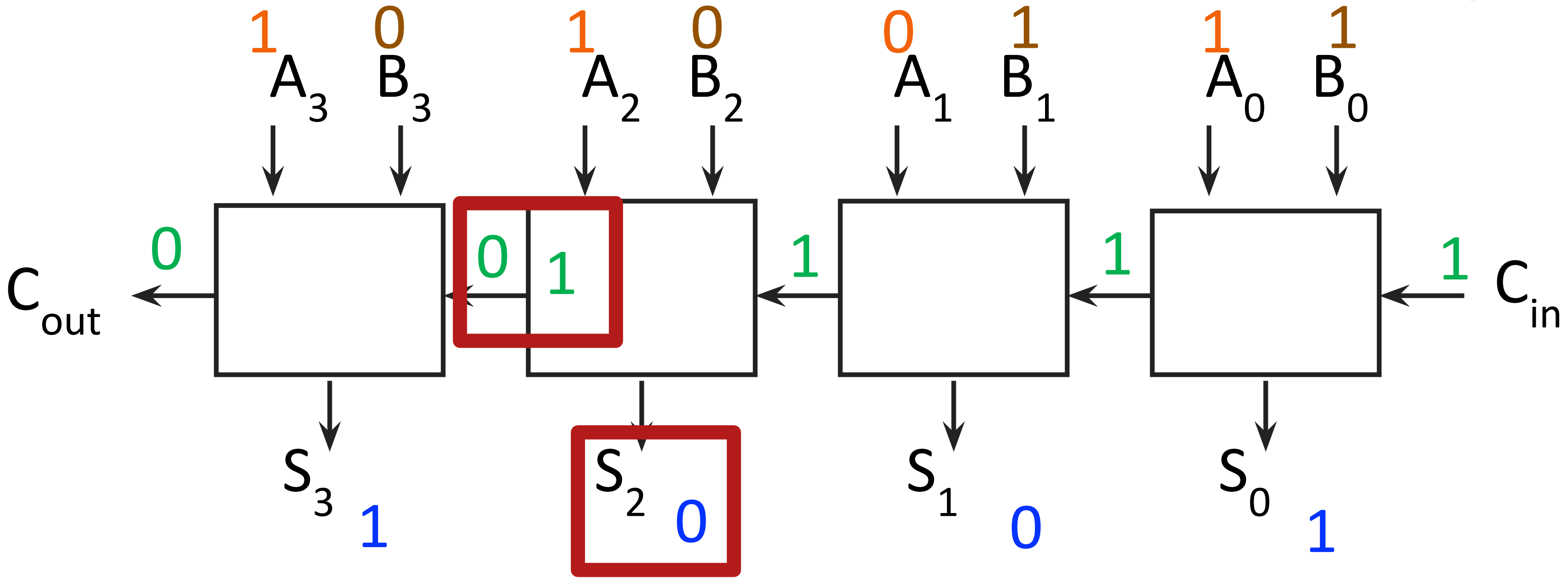


needs to be recomputed

- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

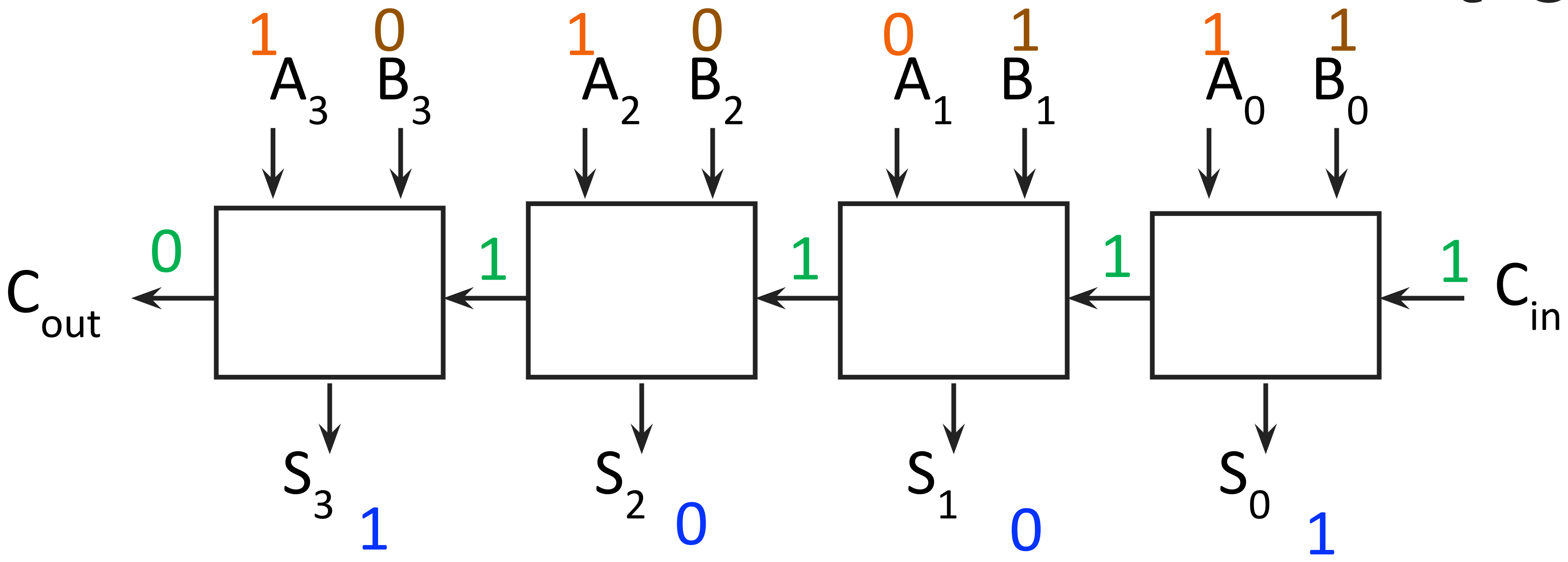
t=2 → t=3



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

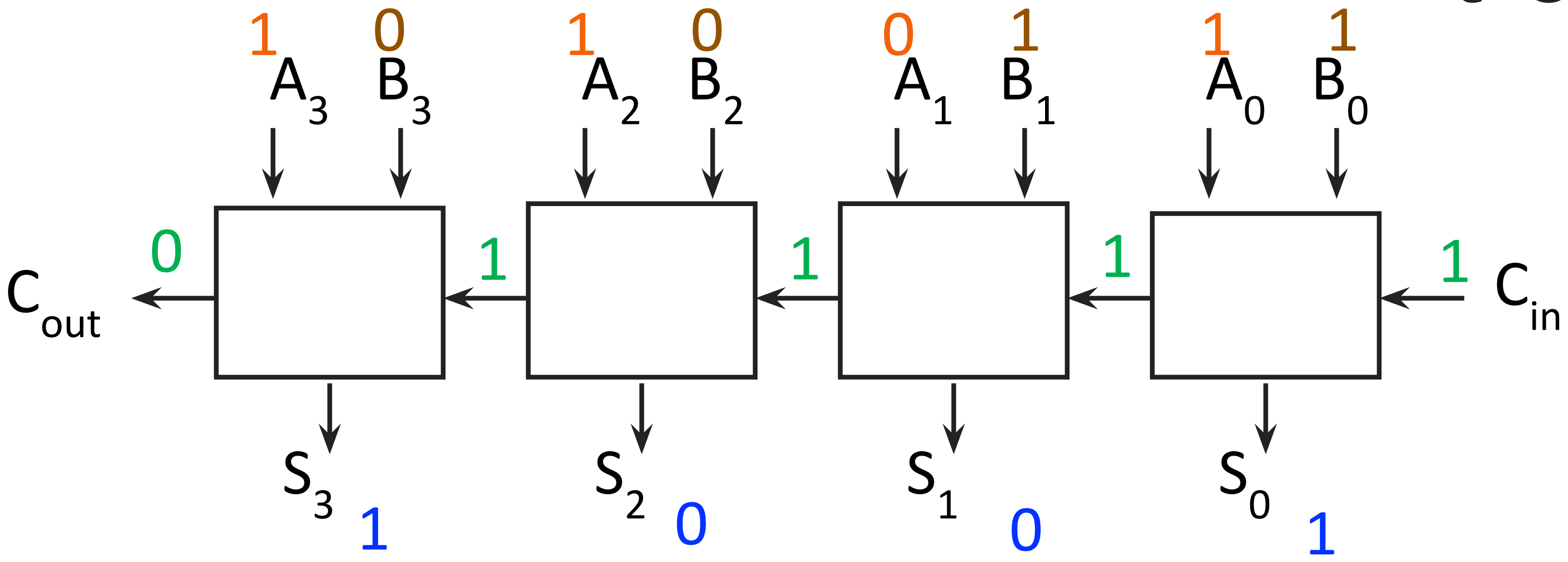
t=3



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

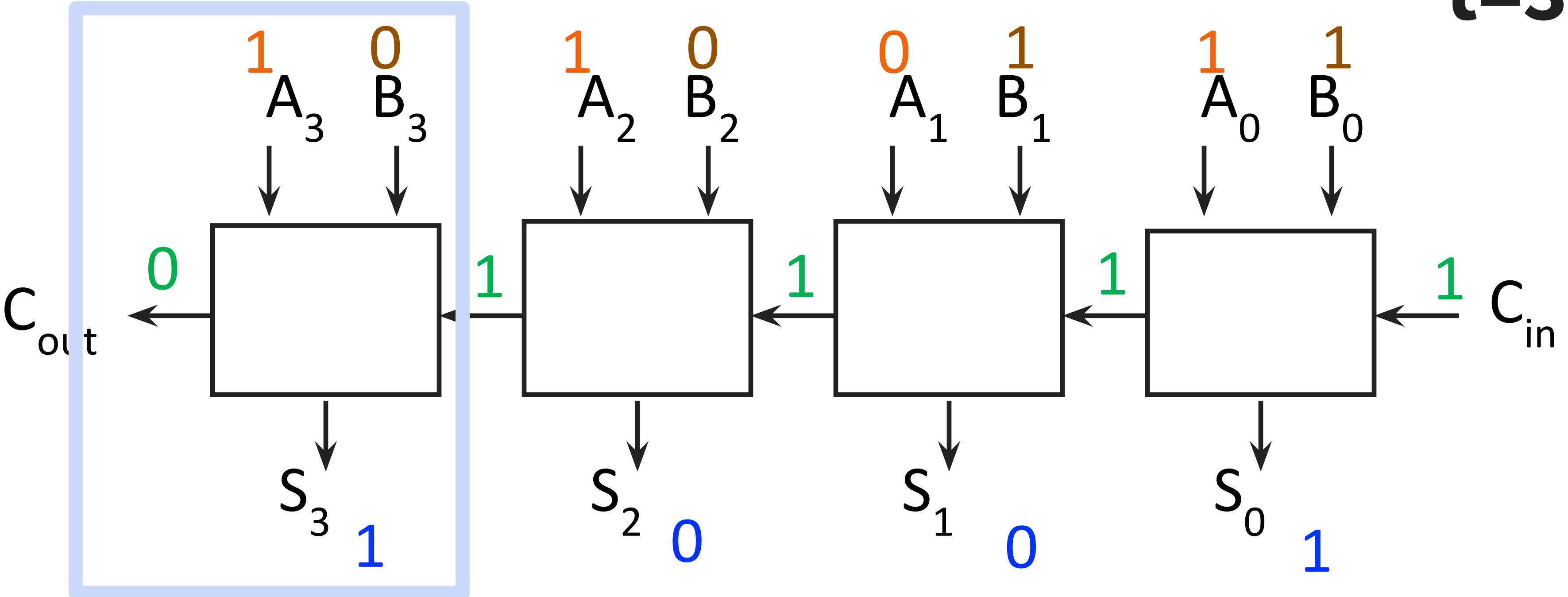
t=3 → t= 4



- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

t=3 → t= 4

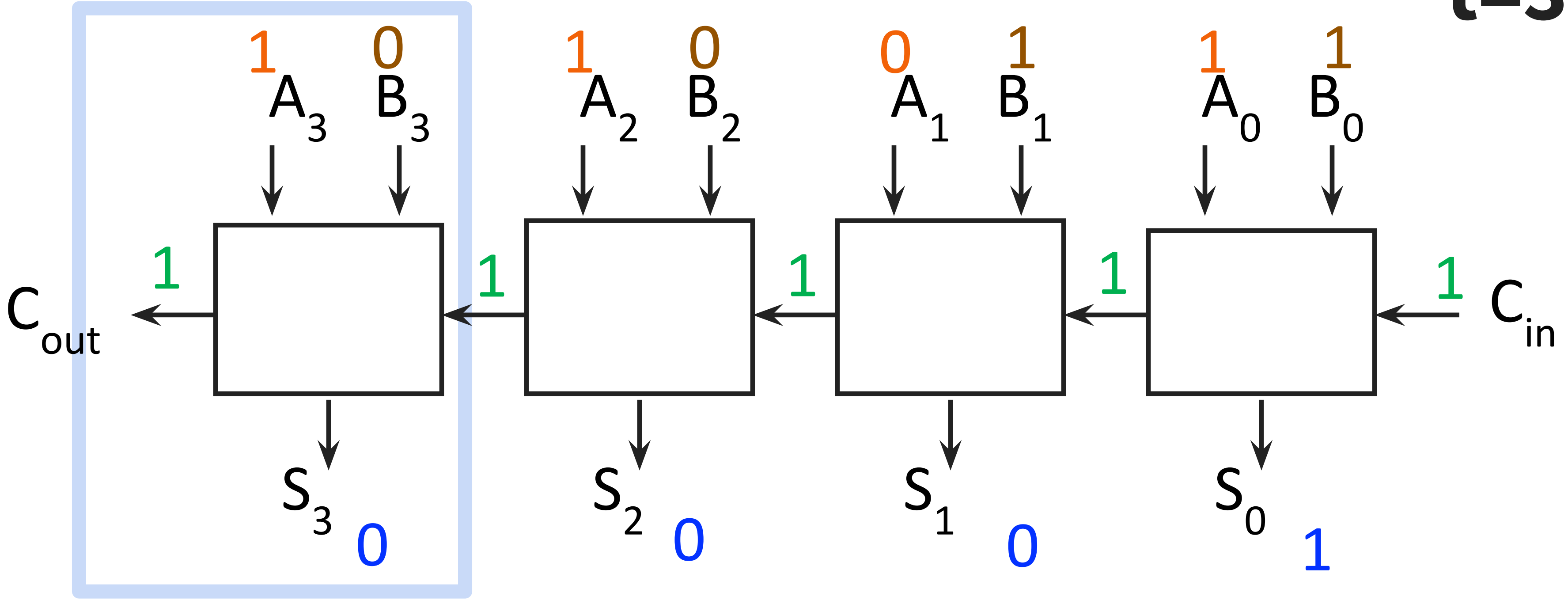


needs to be recomputed

- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

t=3 → t= 4

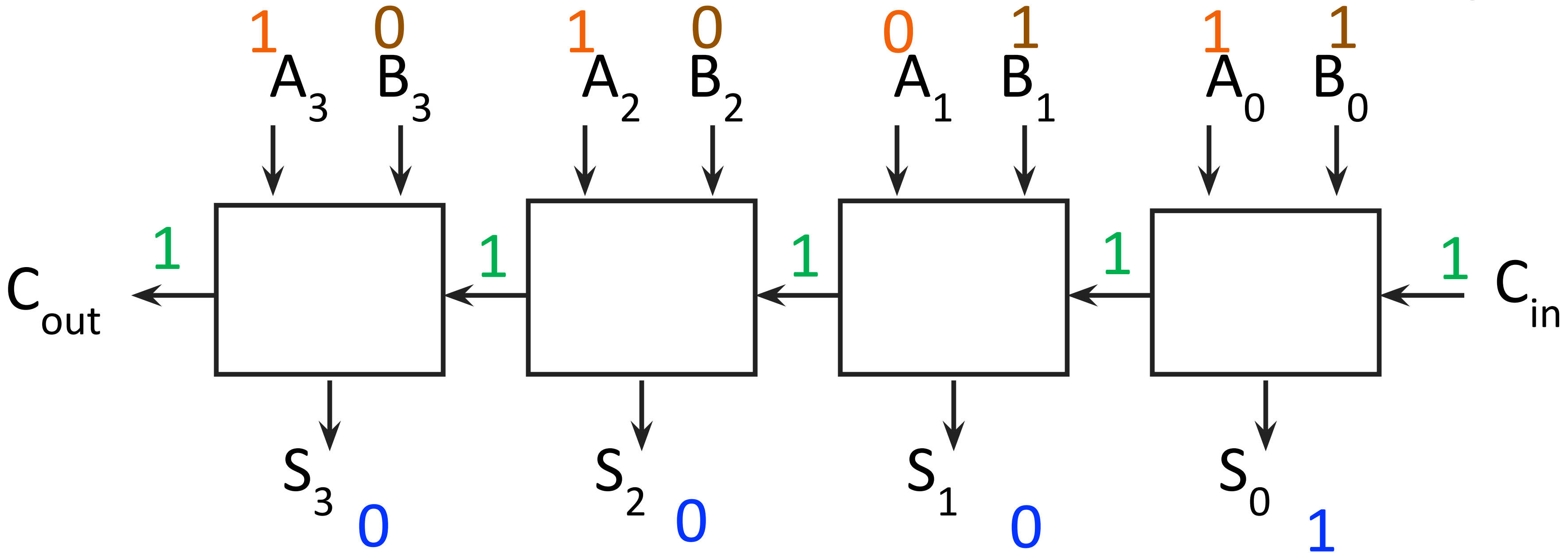


needs to be recomputed

- So far, gates compute instantaneous
- In reality, there is a delay, because it takes time to compute.
- Simple model: it takes 1 unit of time to propagate results through a 1-bit adder

4-bit Adder: Delay Model

t= 4



- No more changes to the adder inputs → we reached a fixed point.

4-bit Adder: Outputs over time

	C_{out}	S
t = 0	?	?
t = 1	0	1111_2
t = 2	0	1101_2
t = 3	0	1001_2
t = 4	1	0001_2
t = 5	1	0001_2

4-bit Adder: Outputs over time

	C_{out}	S
t = 0	?	?
t = 1	0	111**1**$_2$
t = 2	0	11**01**$_2$
t = 3	0	1**001**$_2$
t = 4	**1**	**0001**$_2$
t = 5	**1**	**0001**$_2$

- Lower bits are ready first.
- The correct value is only available after 4 time units.

4-bit Adder: Outputs over time

	C_{out}	S
t = 0	?	?
t = 1	0	$111\mathbf{1}_2$
t = 2	0	$11\mathbf{01}_2$
t = 3	0	$1\mathbf{001}_2$
t = 4	**1**	$\mathbf{0001}_2$
t = 5	**1**	$\mathbf{0001}_2$

- Lower bits are ready first.
- The correct value is only available after 4 time units.
- In reality, the delays are more complicated than our model.

4-bit Adder: Outputs over time

	C_{out}	S
t = 0	?	?
t = 1	0	$111\mathbf{1}_2$
t = 2	0	$11\mathbf{01}_2$
t = 3	0	$1\mathbf{001}_2$
t = 4	**1**	$\mathbf{0001}_2$
t = 5	**1**	$\mathbf{0001}_2$

- Lower bits are ready first.
- The correct value is only available after 4 time units.
- In reality, the delays are more complicated than our model.
- We assumed that all inputs to the 4-bit adder arrive at t = 0.

4-bit Adder: Outputs over time

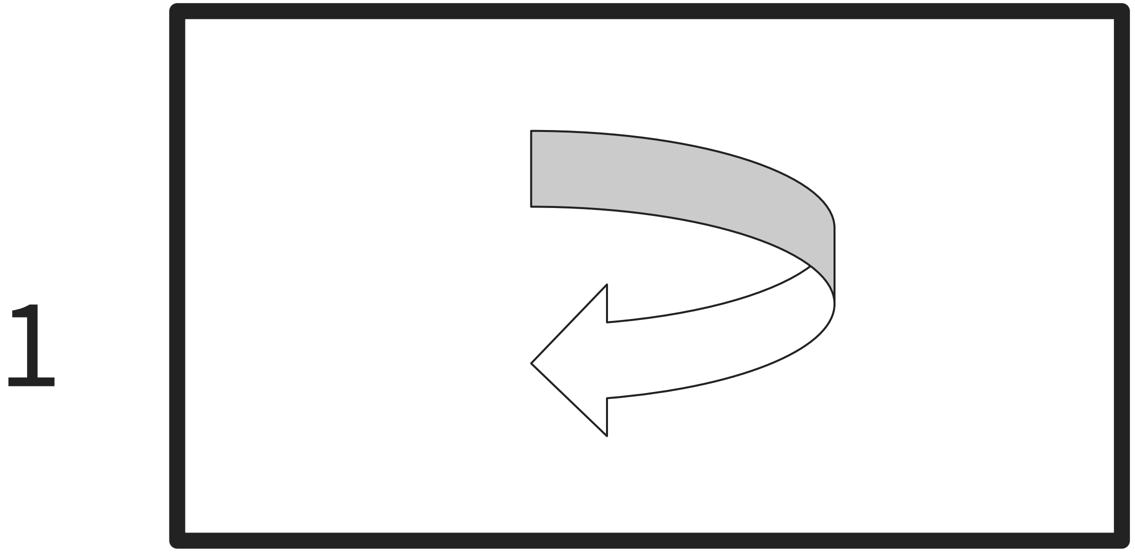
	C_{out}	S
t = 0	?	?
t = 1	0	$111\mathbf{1}_2$
t = 2	0	$11\mathbf{01}_2$
t = 3	0	$1\mathbf{001}_2$
t = 4	**1**	$\mathbf{0001}_2$
t = 5	**1**	$\mathbf{0001}_2$

- Lower bits are ready first.
- The correct value is only available after 4 time units.
- In reality, the delays are more complicated than our model.
- We assumed that all inputs to the 4-bit adder arrive at t = 0.
- Need state to synchronize.

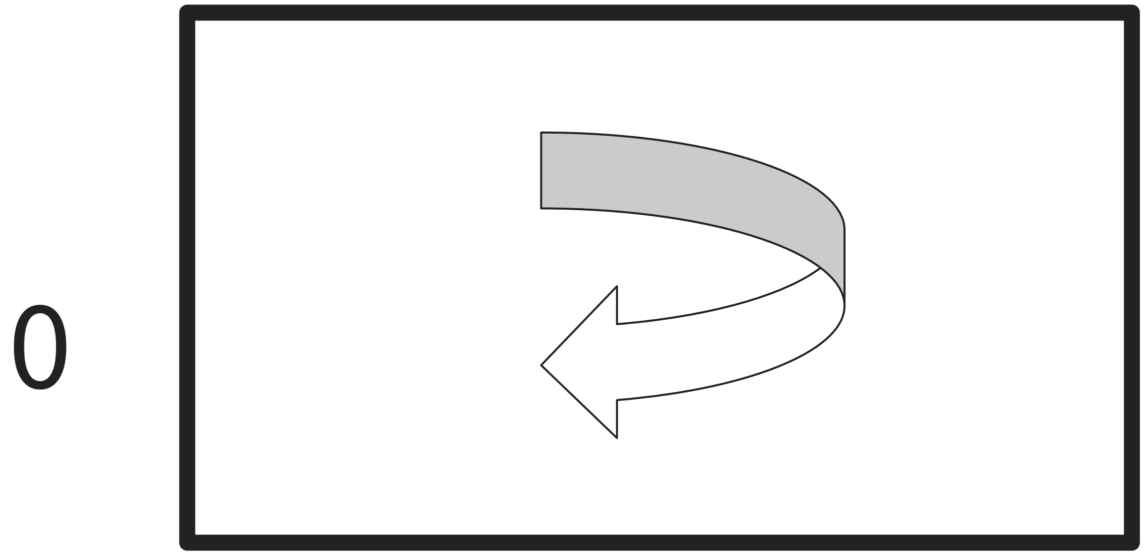
How can we store information in a binary circuit?

Idea: use feedback

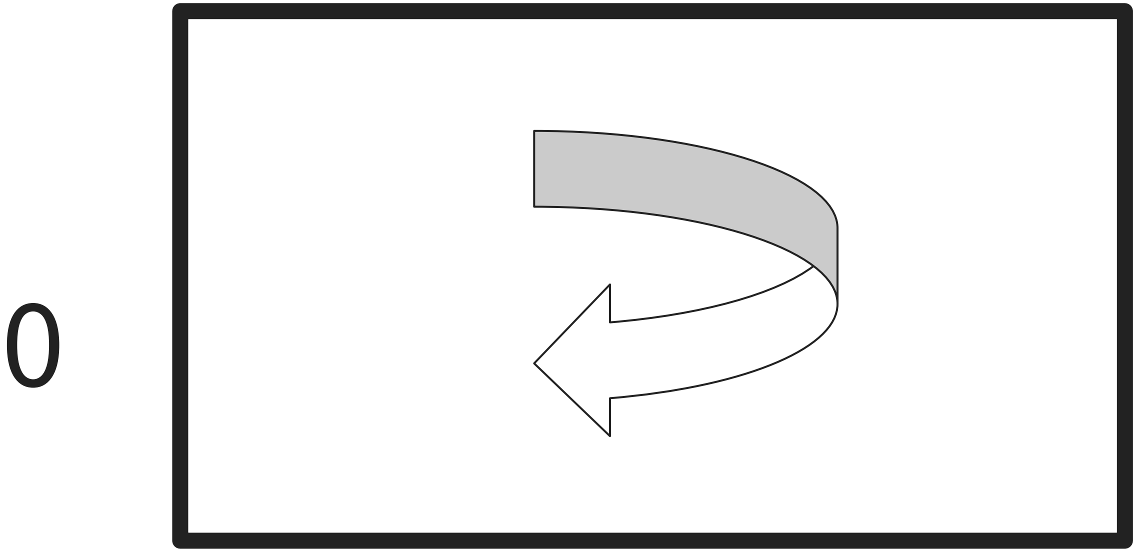
Idea: use feedback



Idea: use feedback

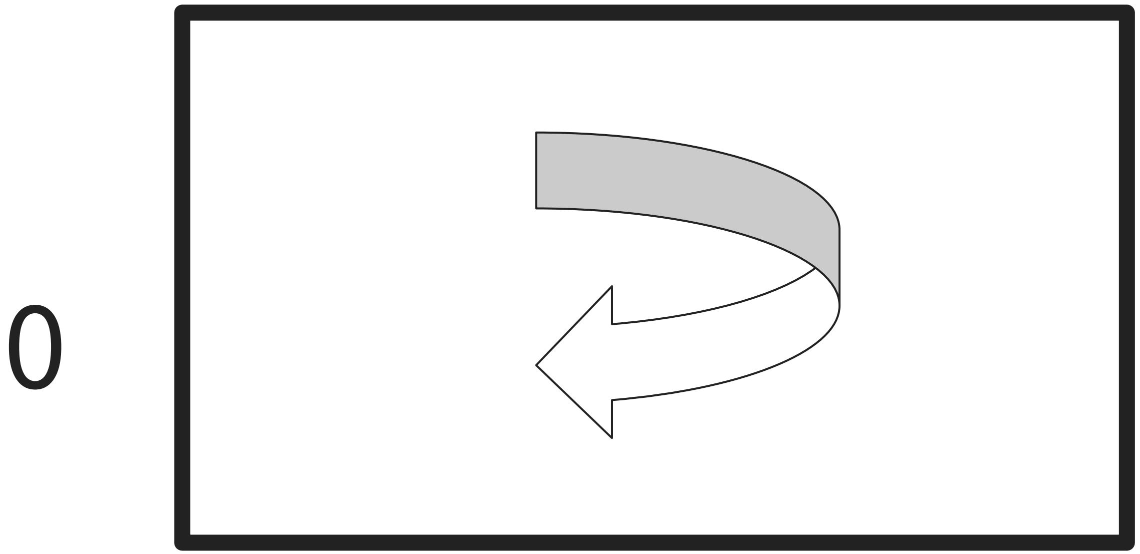


Idea: use feedback



We are storing a charge. This is how your DRAM main memory works.

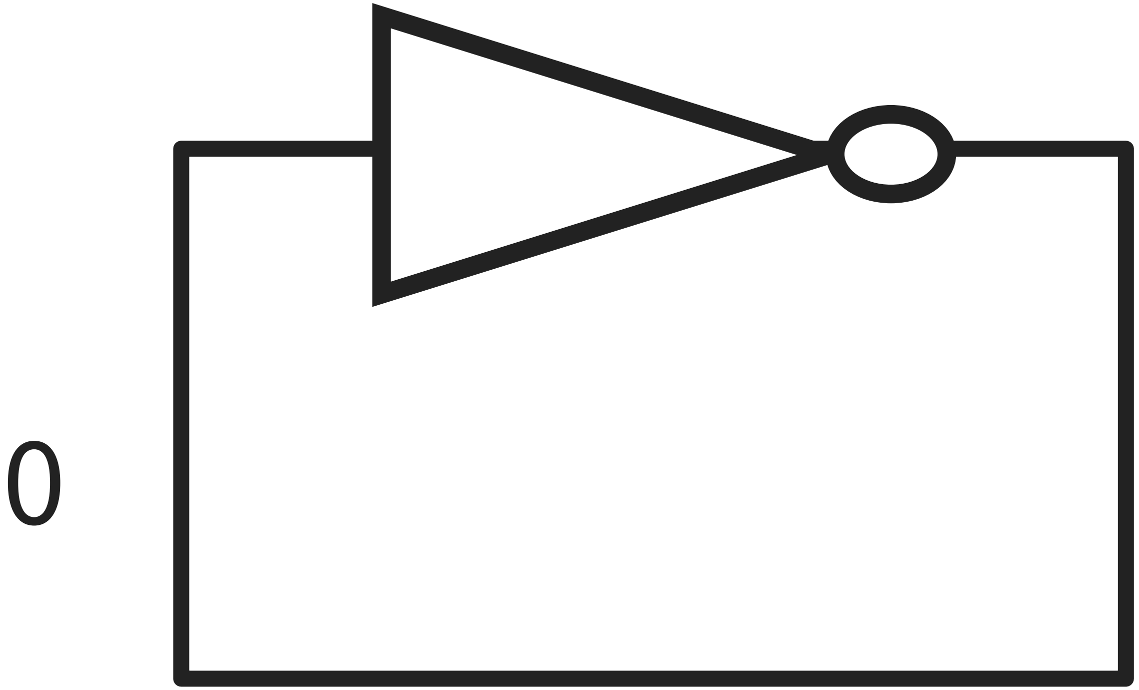
Idea: use feedback



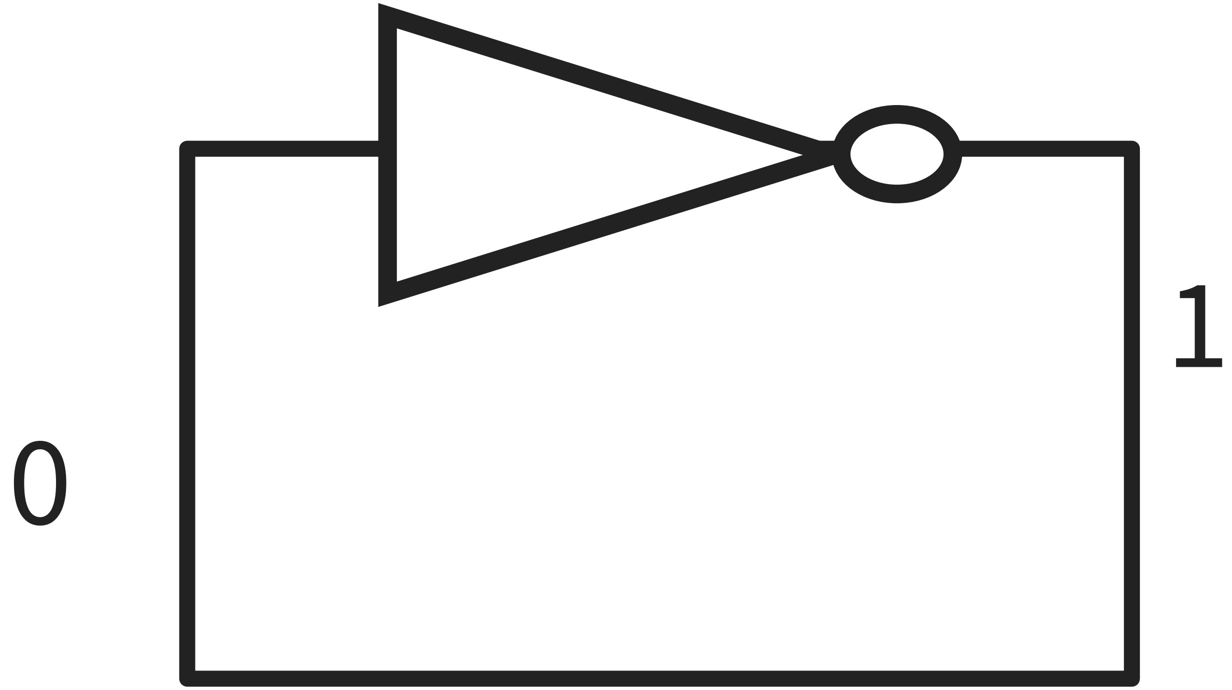
We are storing a charge. This is how your DRAM main memory works.

Problem: Charge disappears over time.

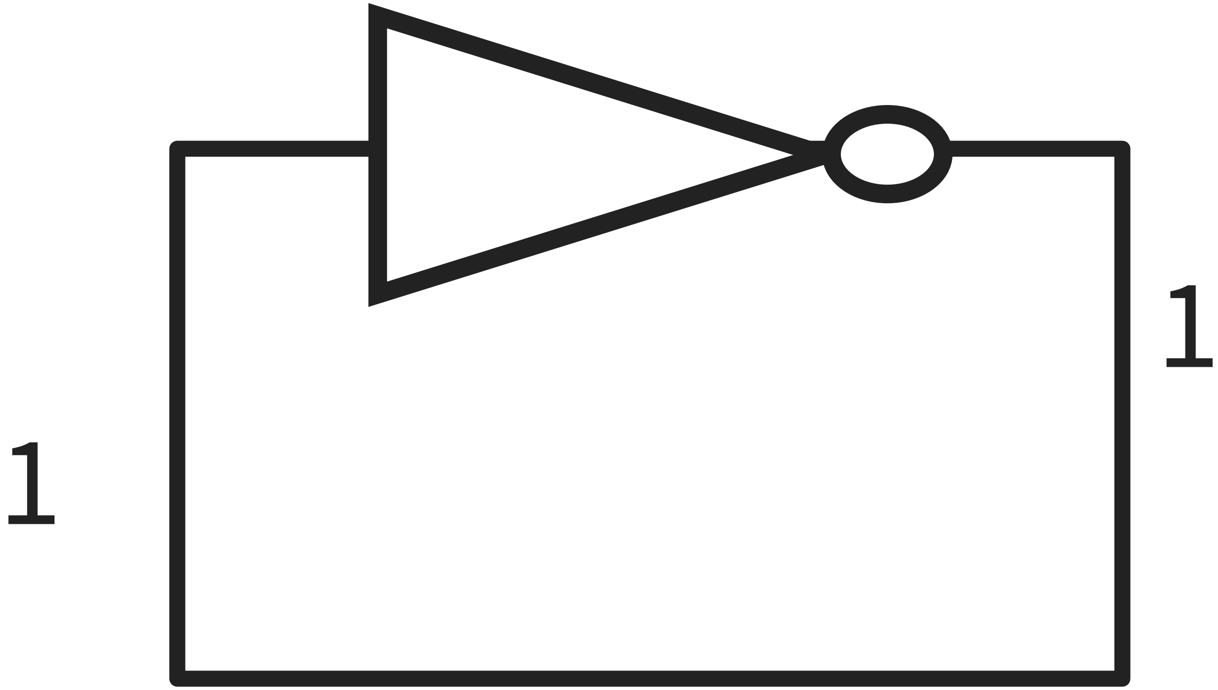
Use Active Element to Maintain Charge?



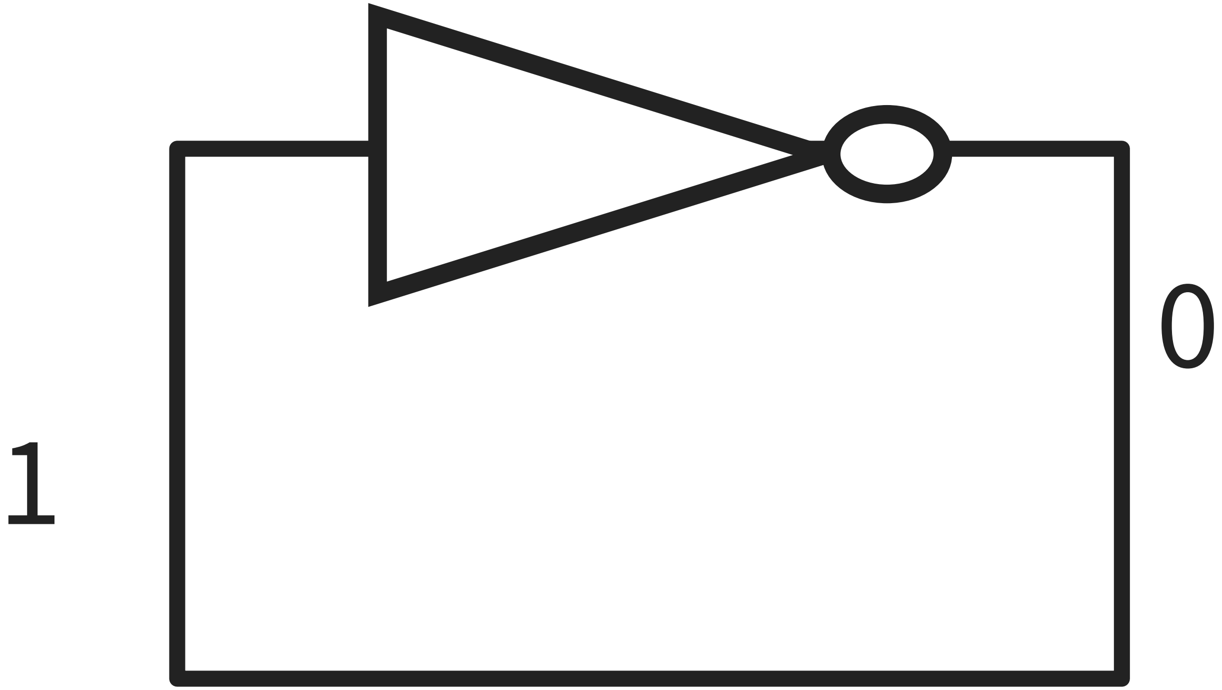
Use Active Element to Maintain Charge?



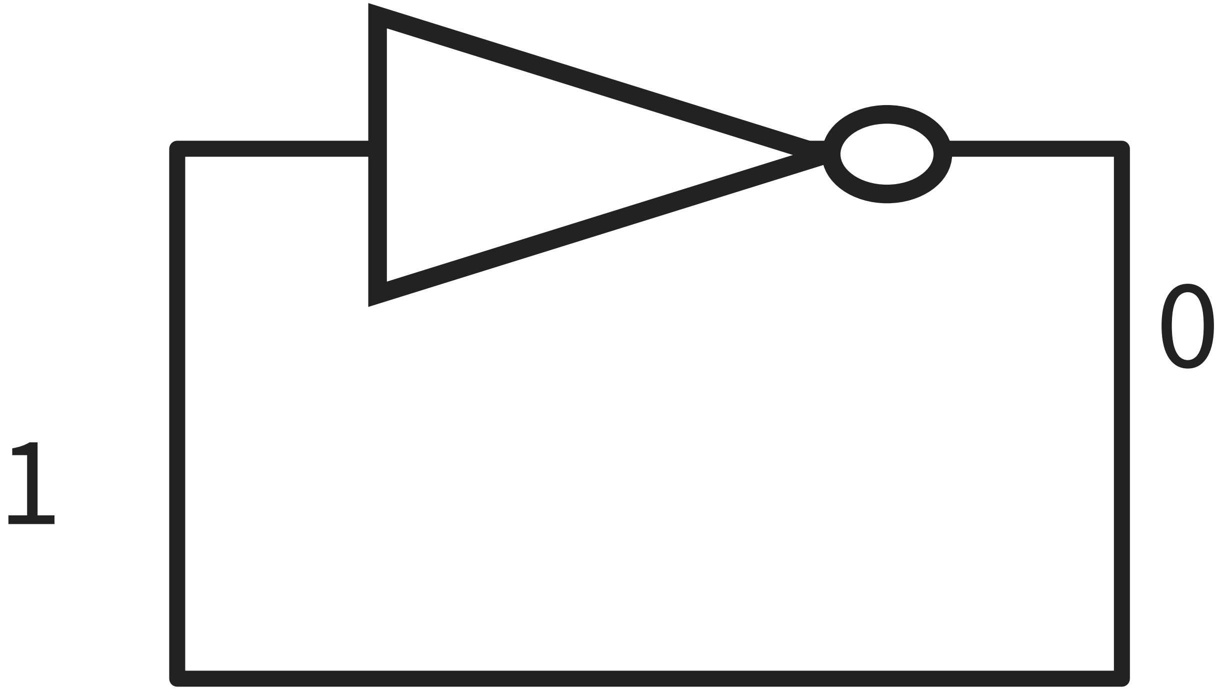
Use Active Element to Maintain Charge?



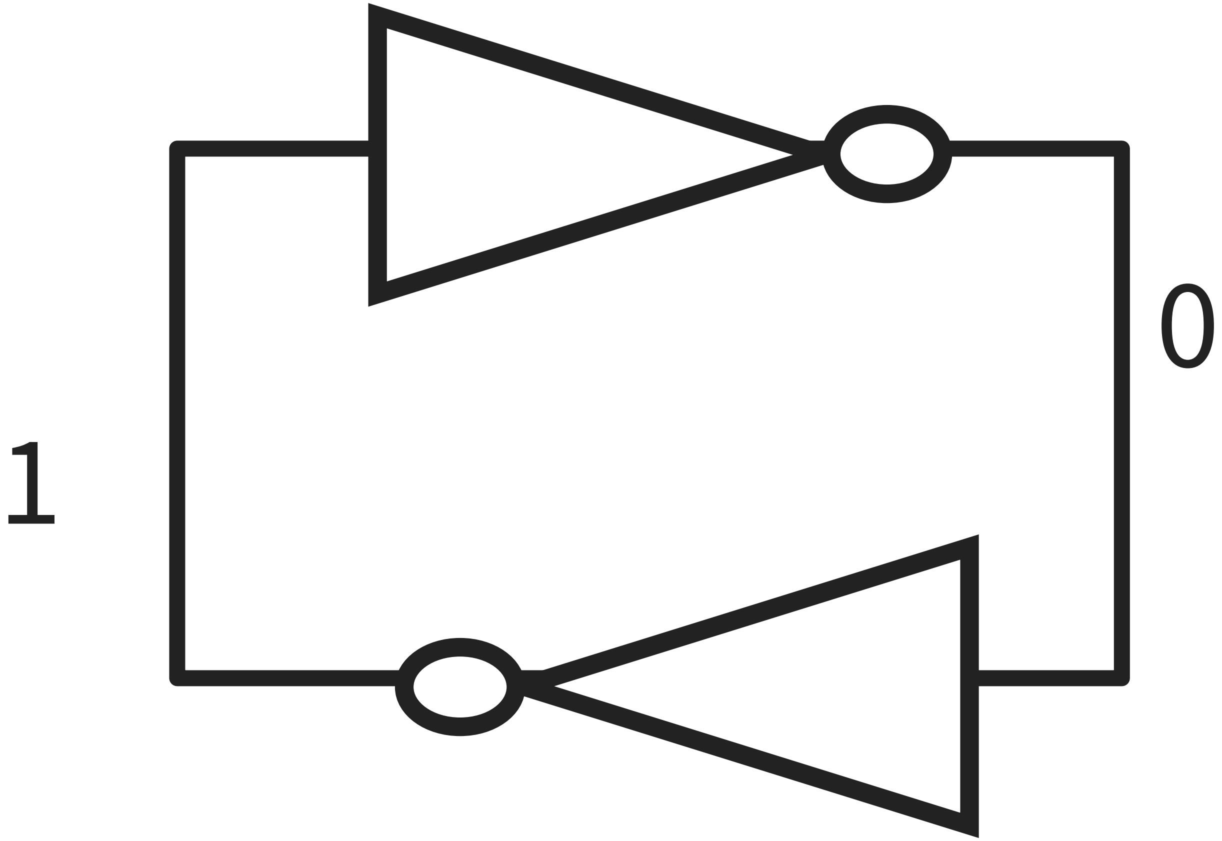
Use Active Element to Maintain Charge?

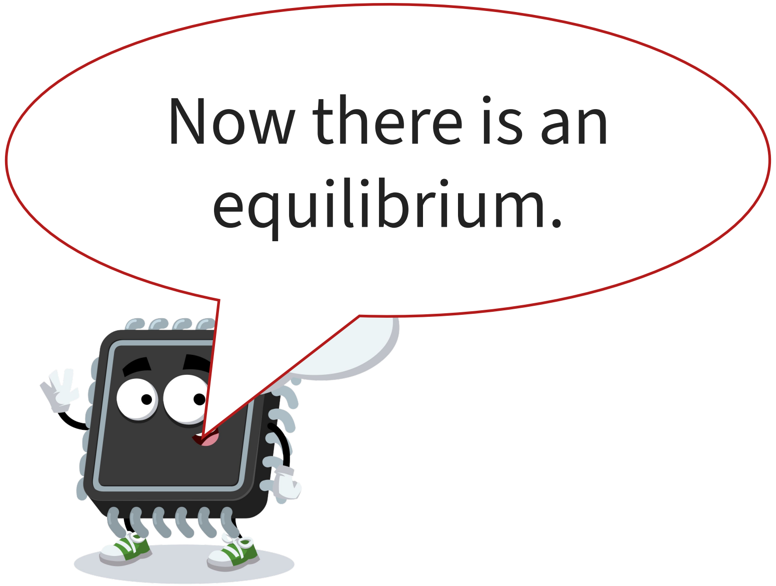


Use Active Element to Maintain Charge?

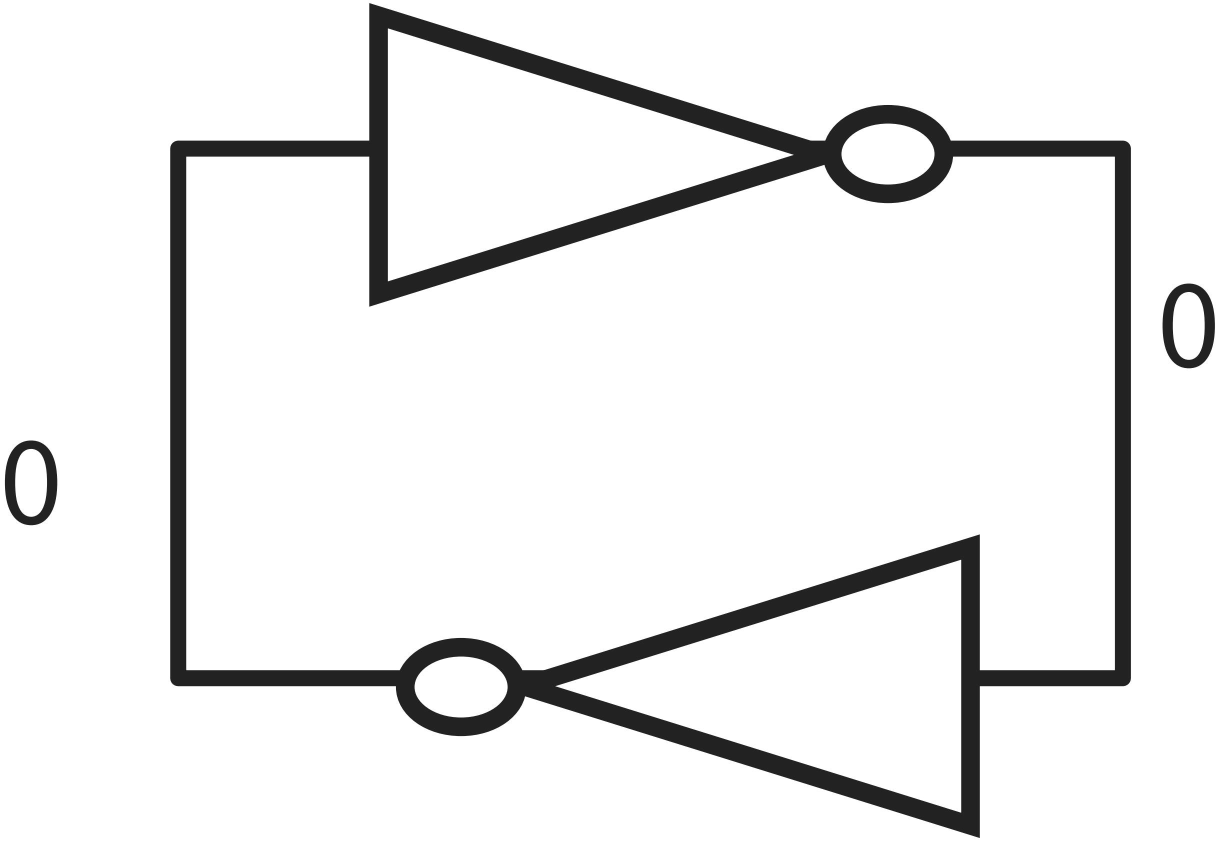


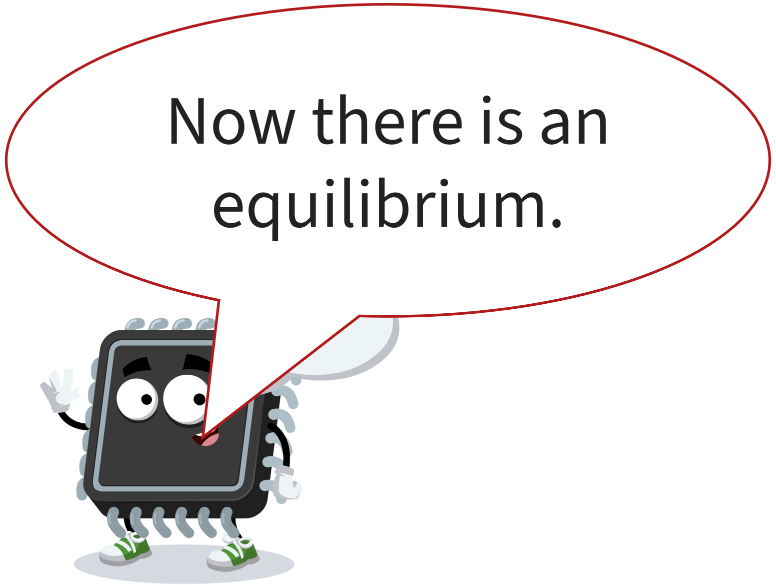
Bi-Stable Device



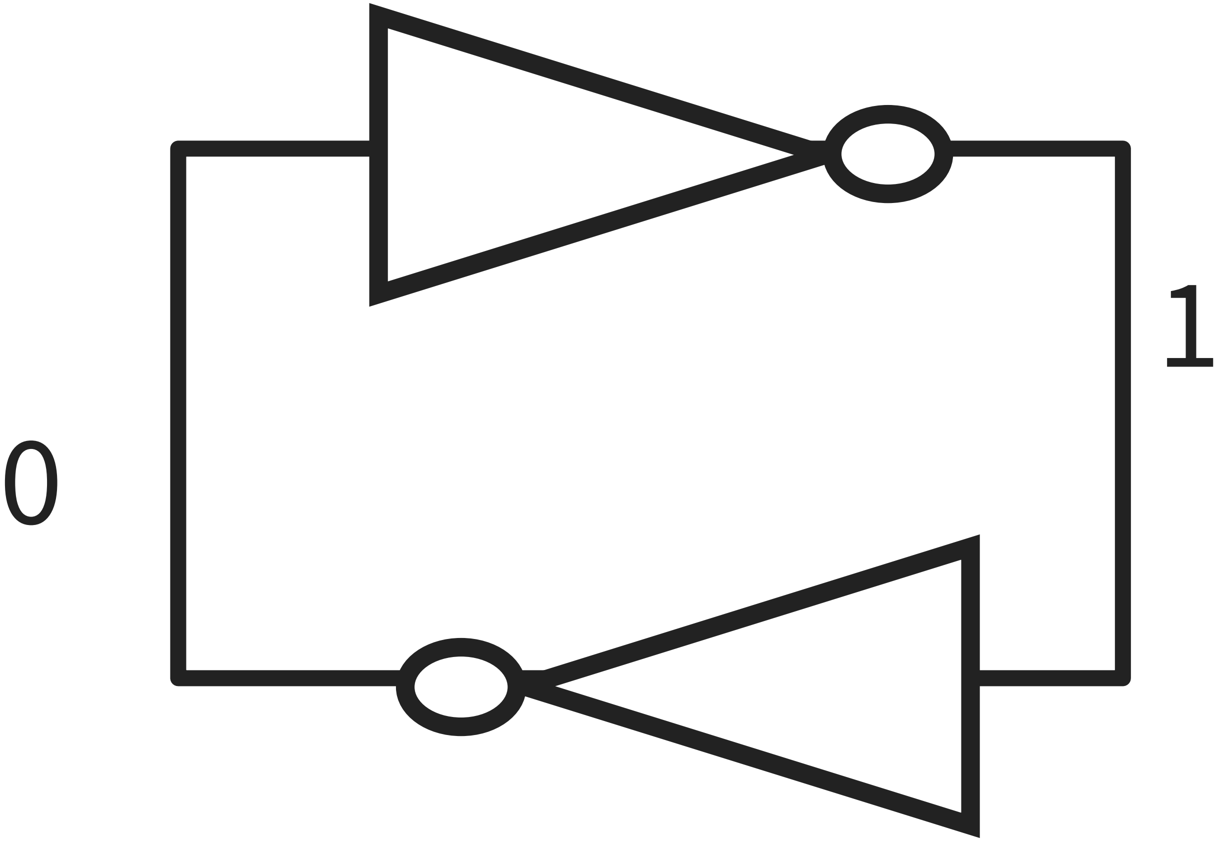


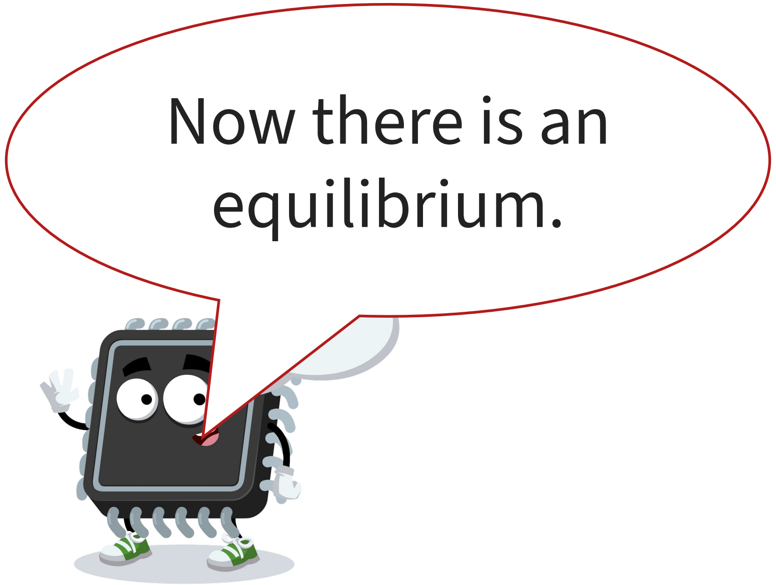
Bi-Stable Device



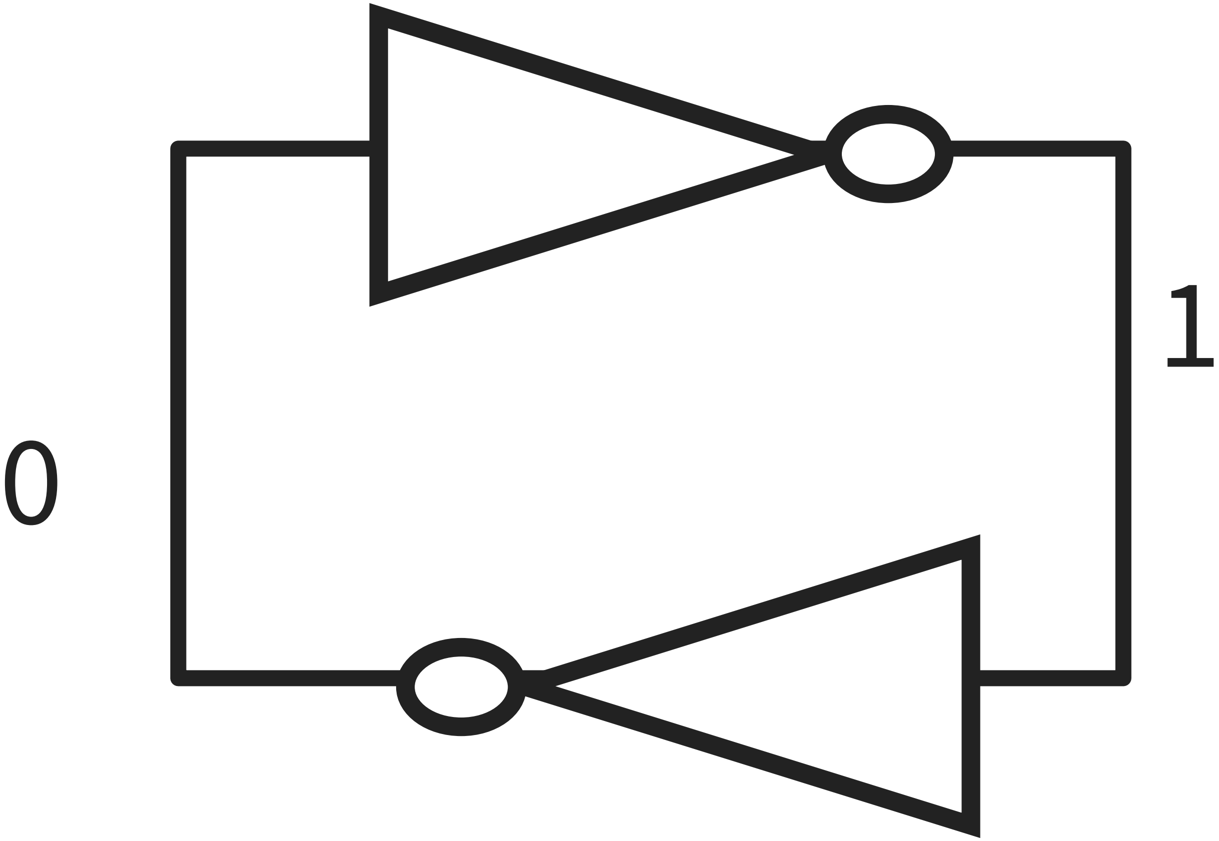


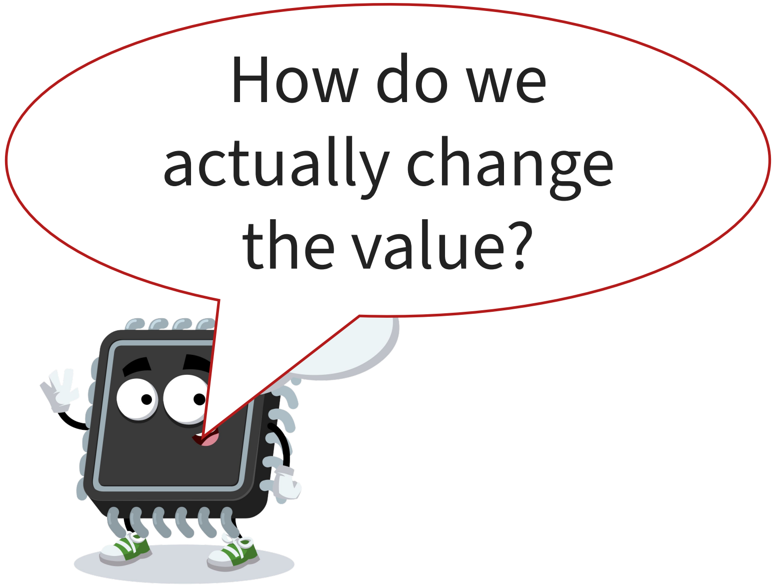
Bi-Stable Device



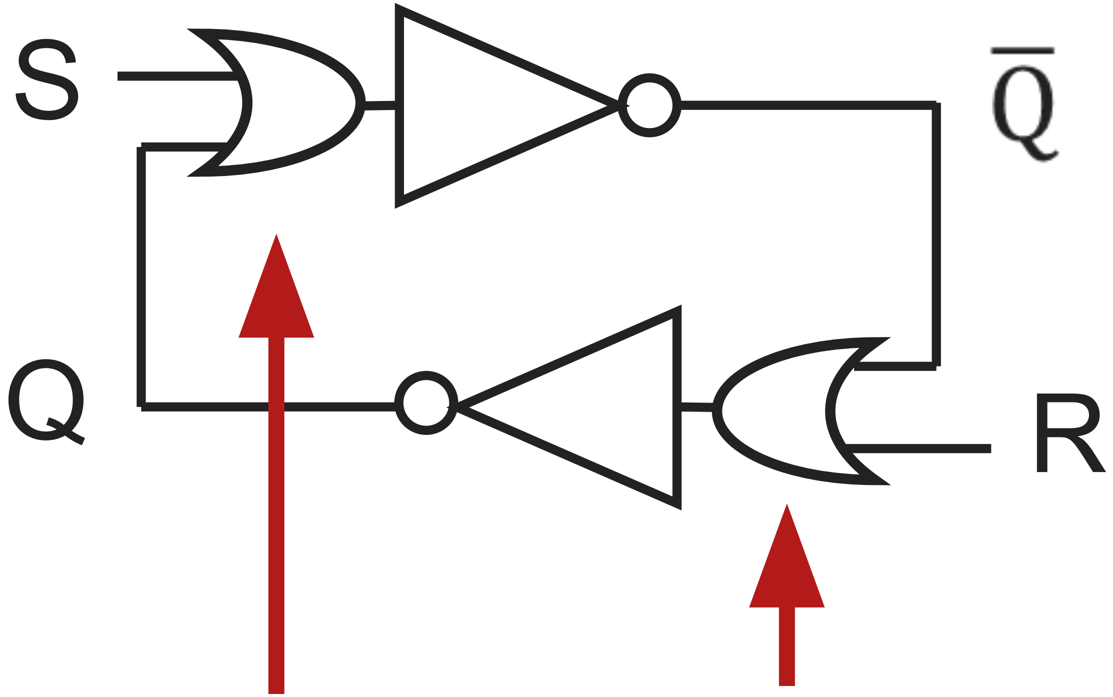


Bi-Stable Device



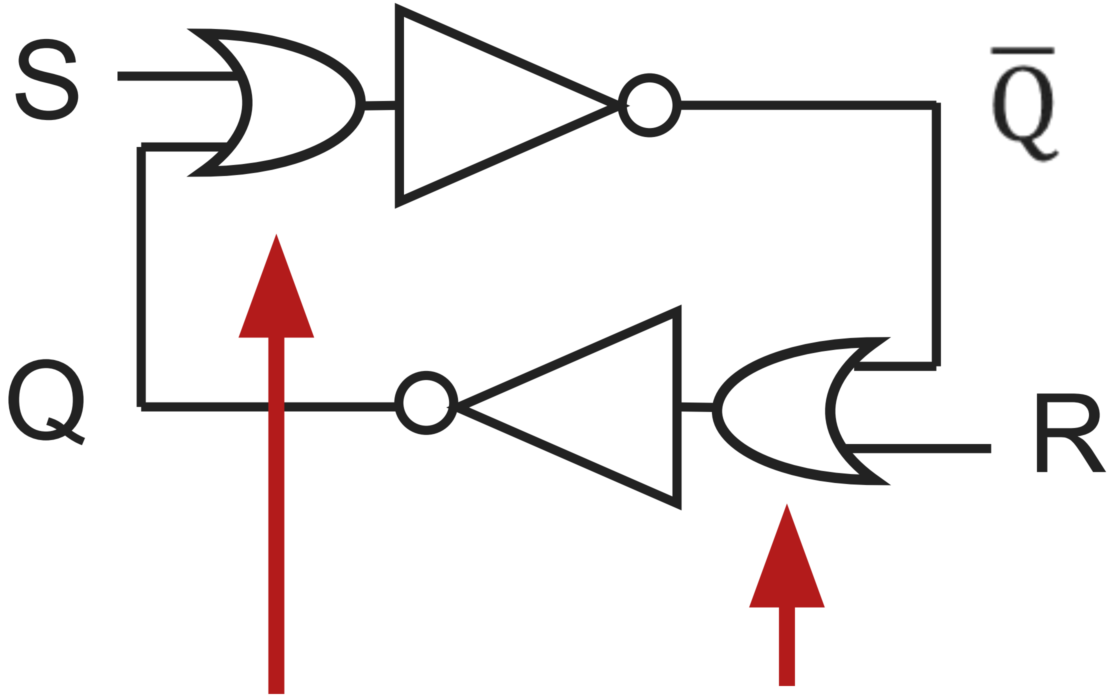


Set-Reset Latch



Add two OR gates.

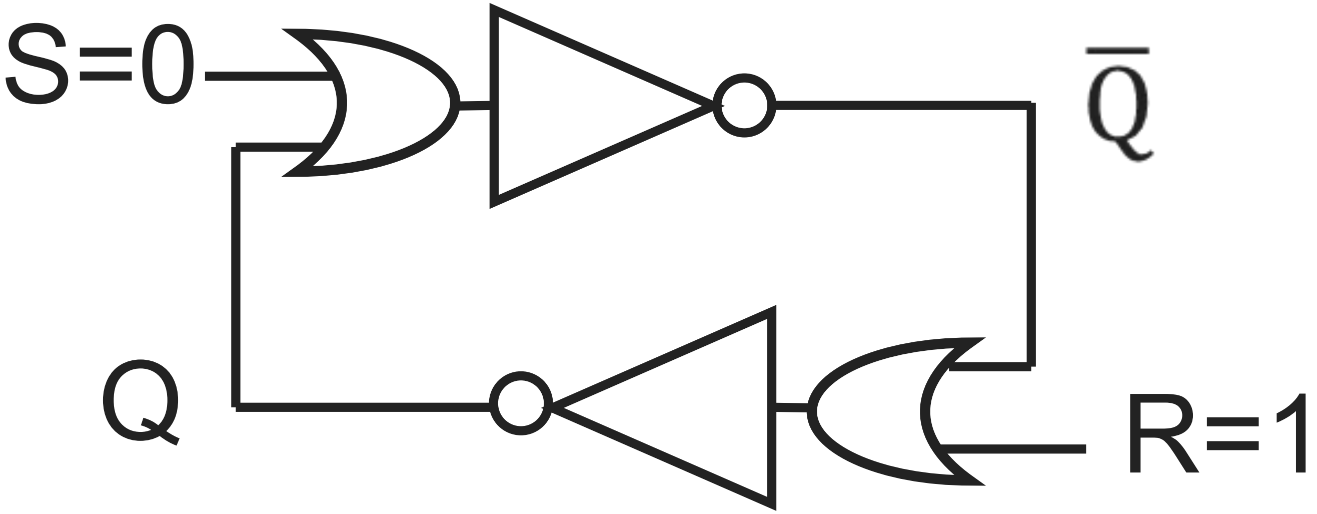
Set-Reset Latch



Add two OR gates.

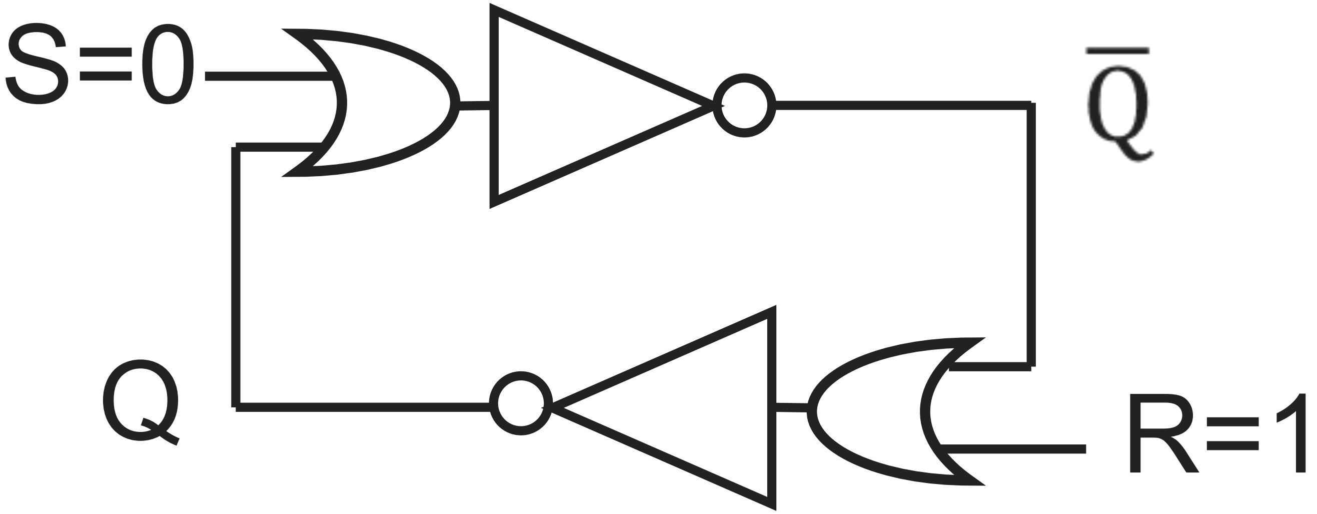
S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

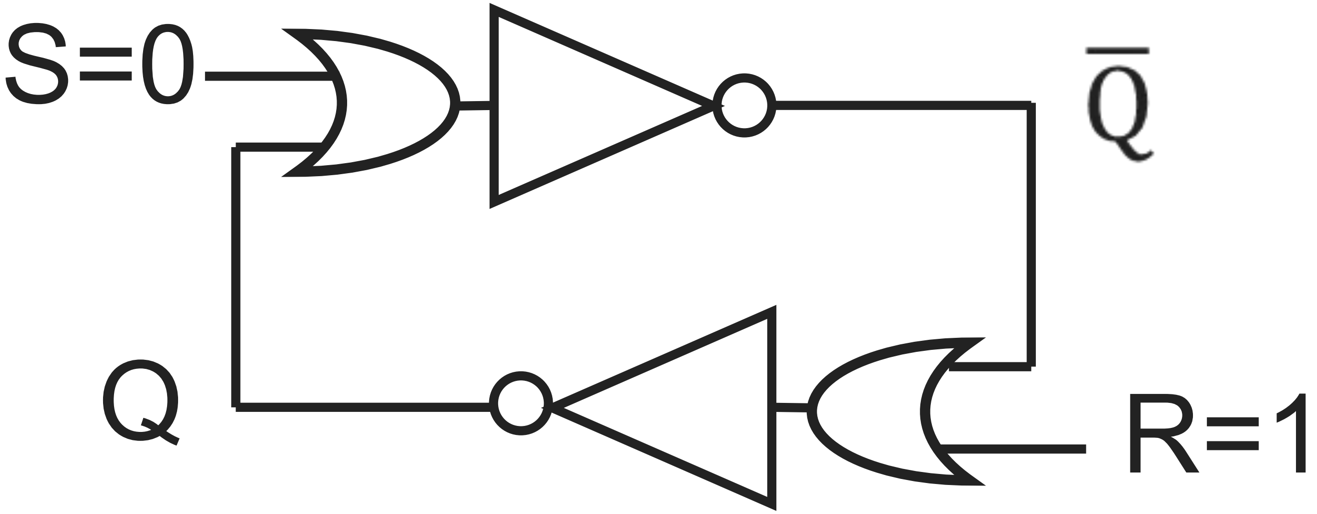
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Where do we start our analysis?

Set-Reset Latch

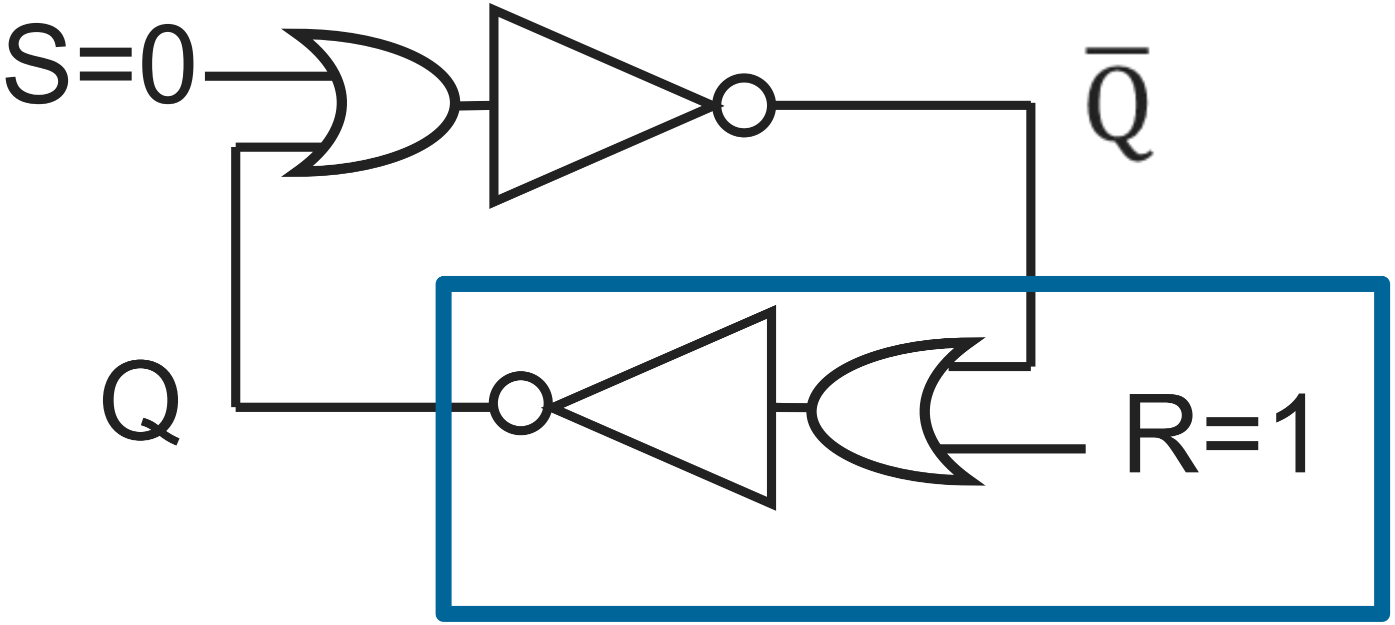


S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Where do we start our analysis?

Remember: 1 ∨ a = 1

Set-Reset Latch

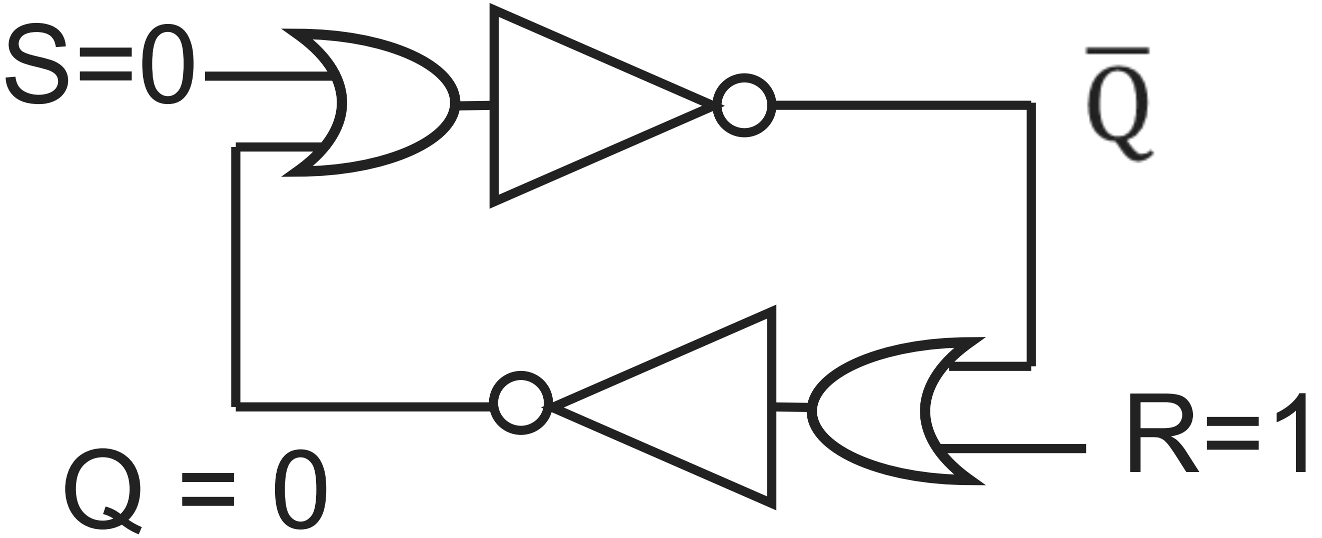


S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Where do we start our analysis?

Remember: $1 \vee a = 1$

Set-Reset Latch

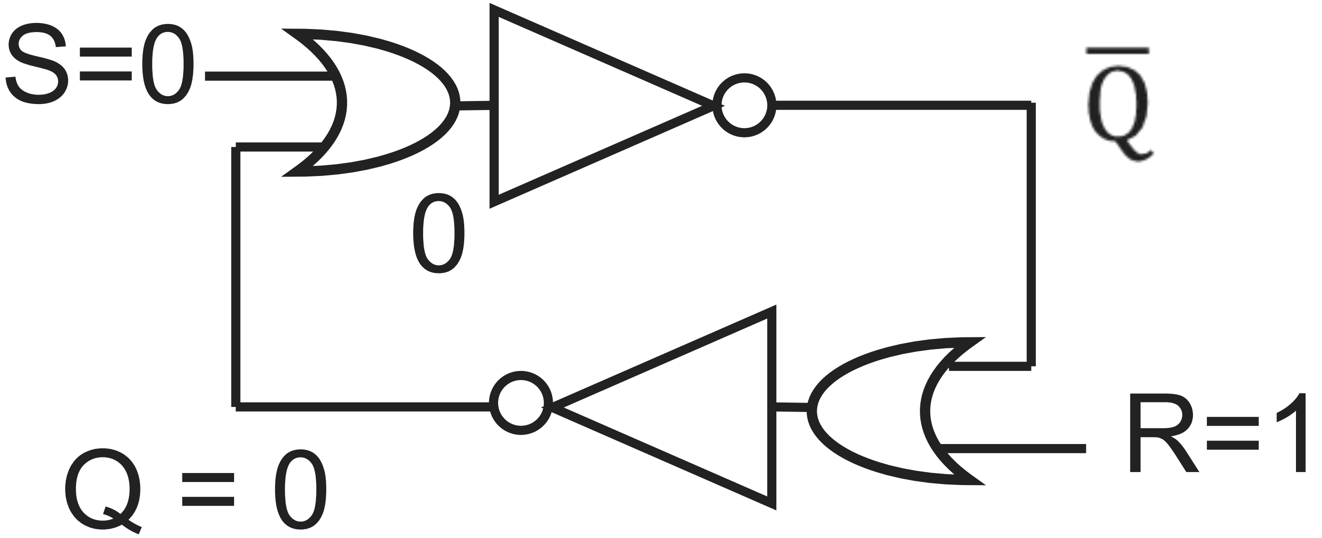


S	R	Q	¬Q
0	0		
0	1	0	
1	0		
1	1		

Where do we start our analysis?

Remember: 1 ∨ a = 1

Set-Reset Latch

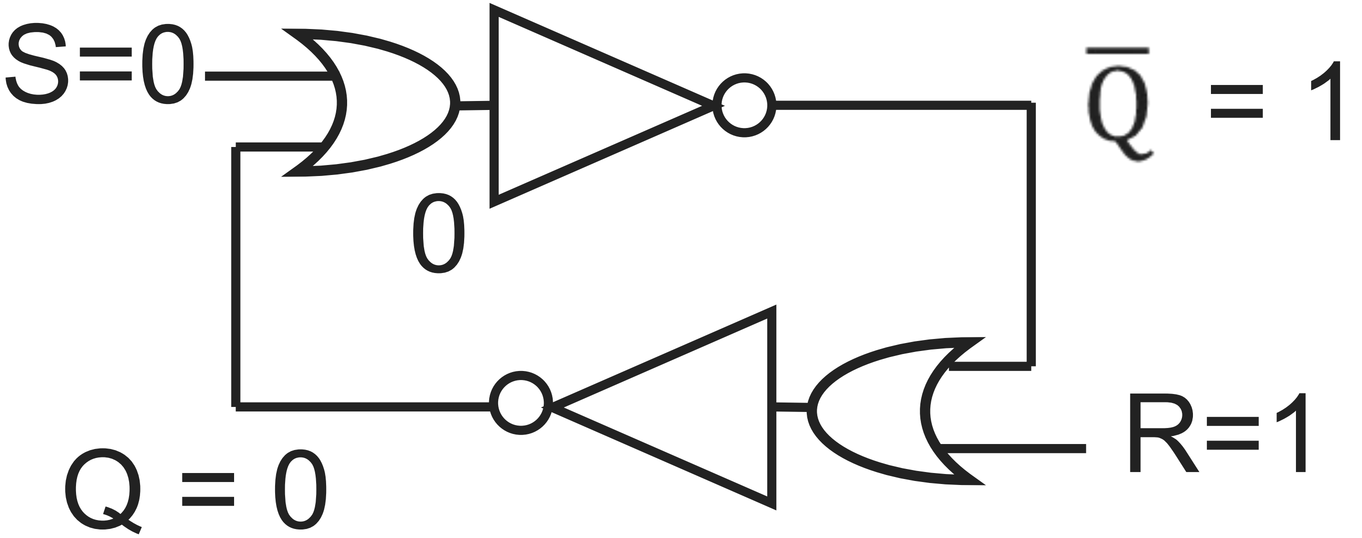


S	R	Q	¬Q
0	0		
0	1	0	
1	0		
1	1		

Where do we start our analysis?

Remember: 1 ∨ a = 1

Set-Reset Latch

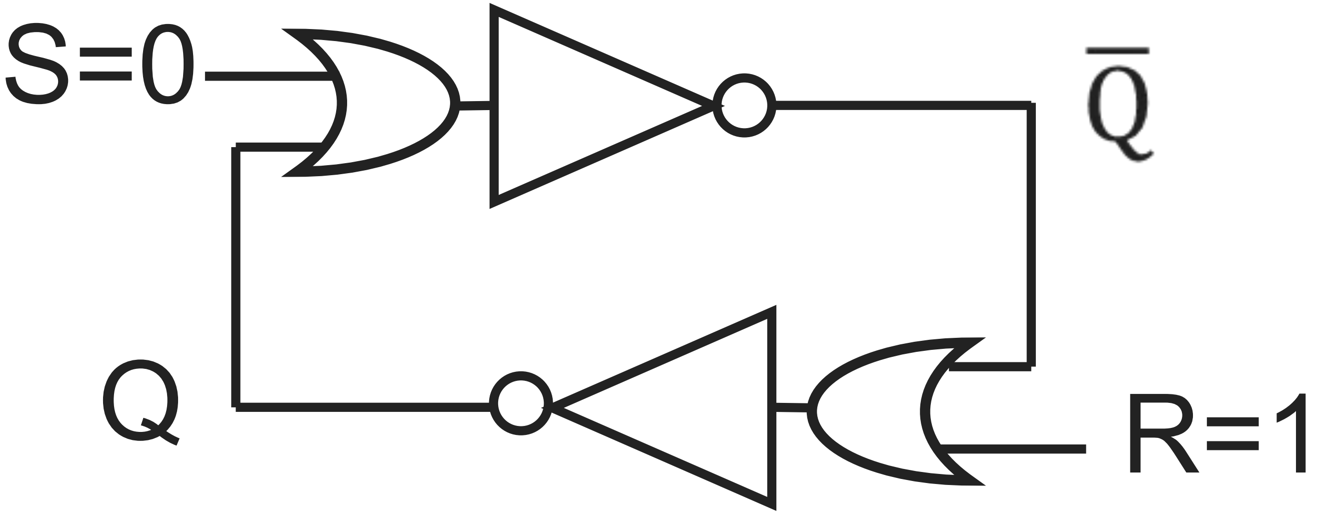


S	R	Q	¬Q
0	0		
0	1	0	1
1	0		
1	1		

Where do we start our analysis?

Remember: 1 ∨ a = 1

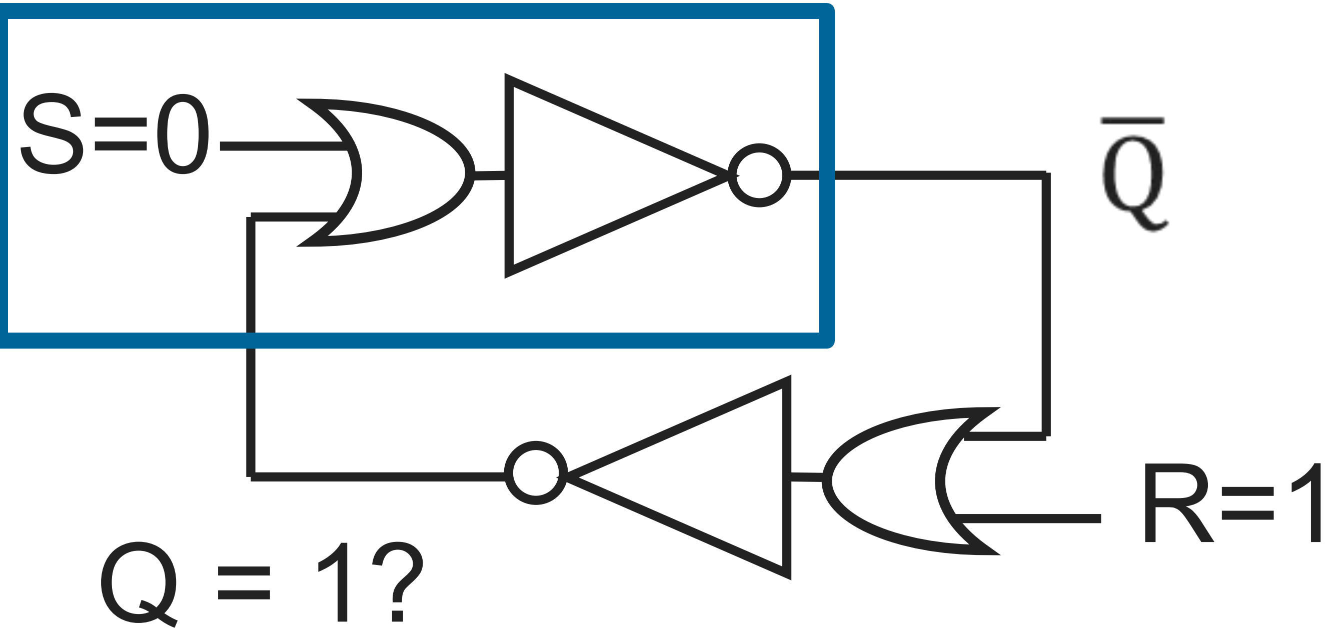
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Alternative: Just guess!

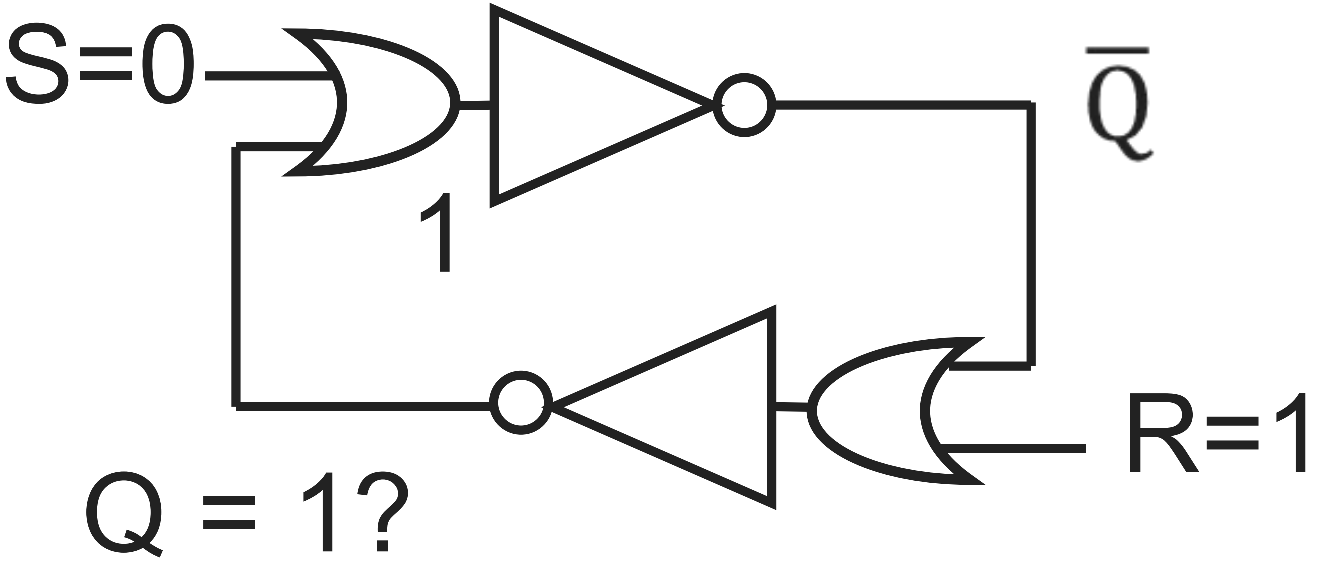
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Alternative: Just guess!

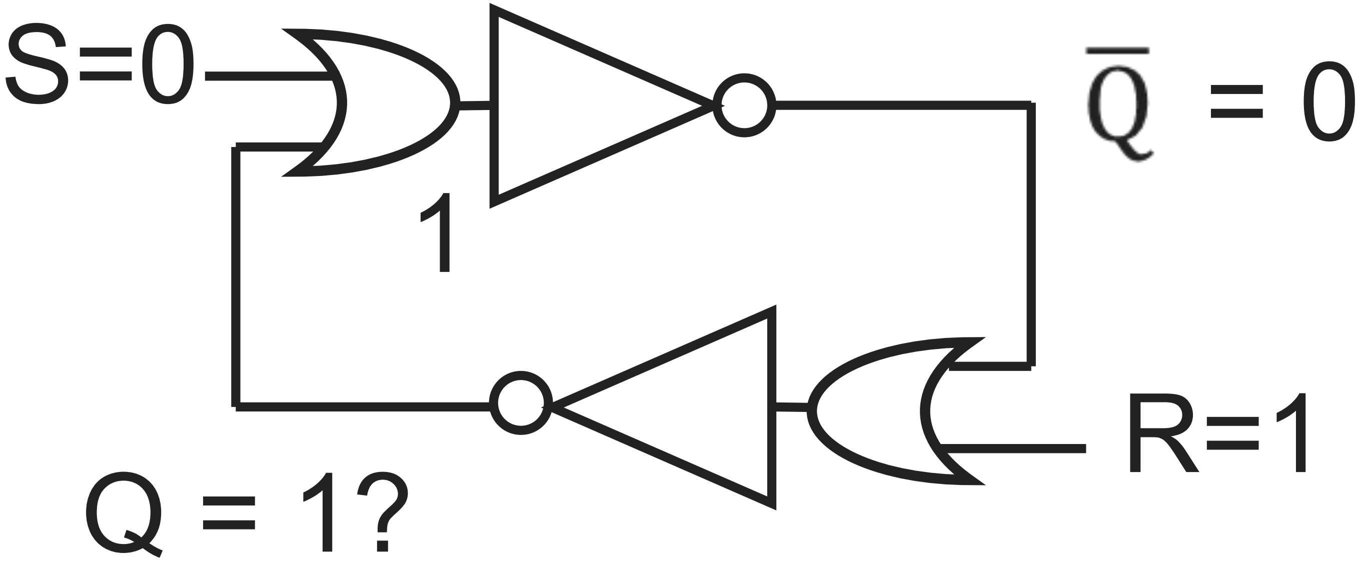
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Alternative: Just guess!

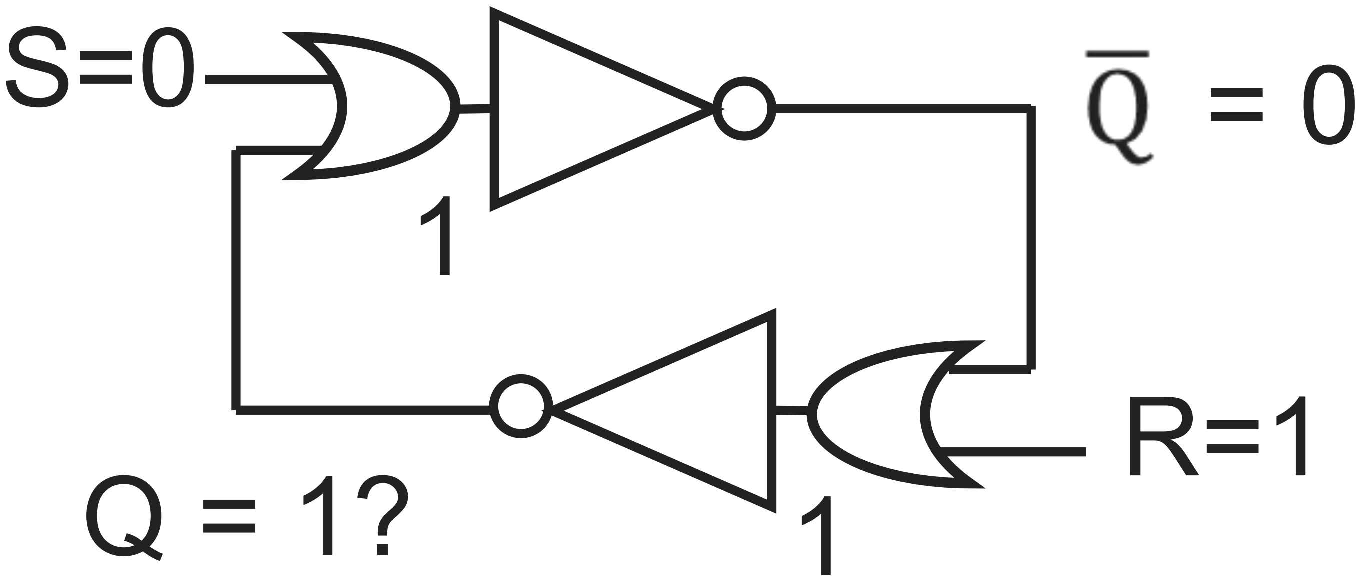
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Alternative: Just guess!

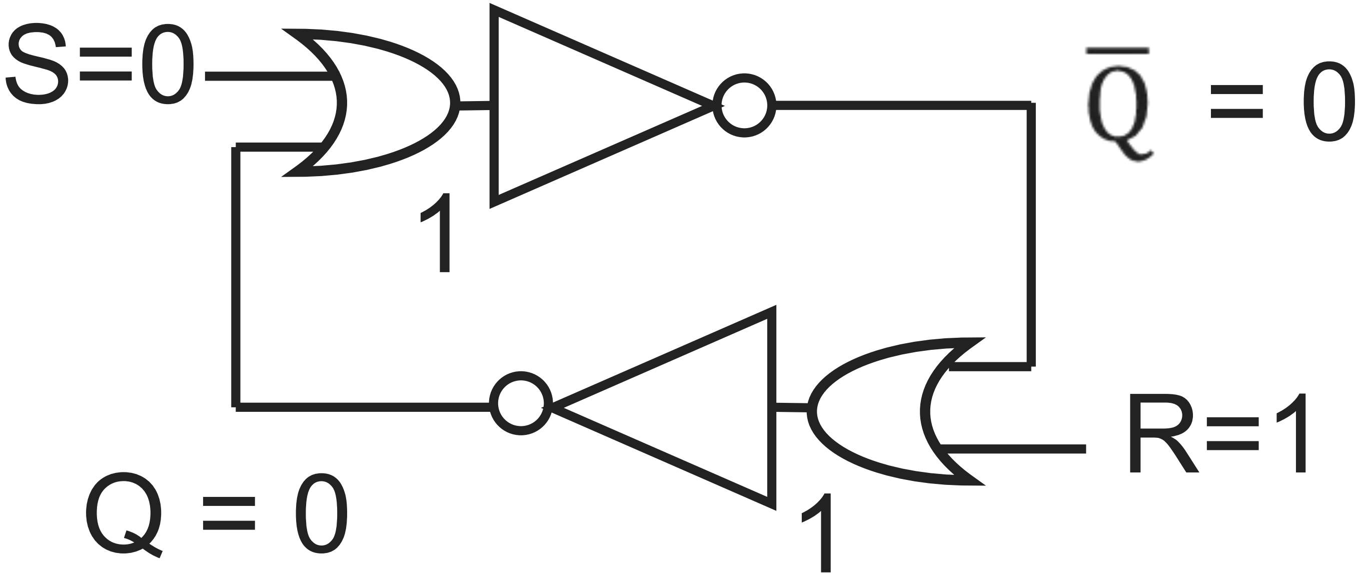
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Alternative: Just guess!

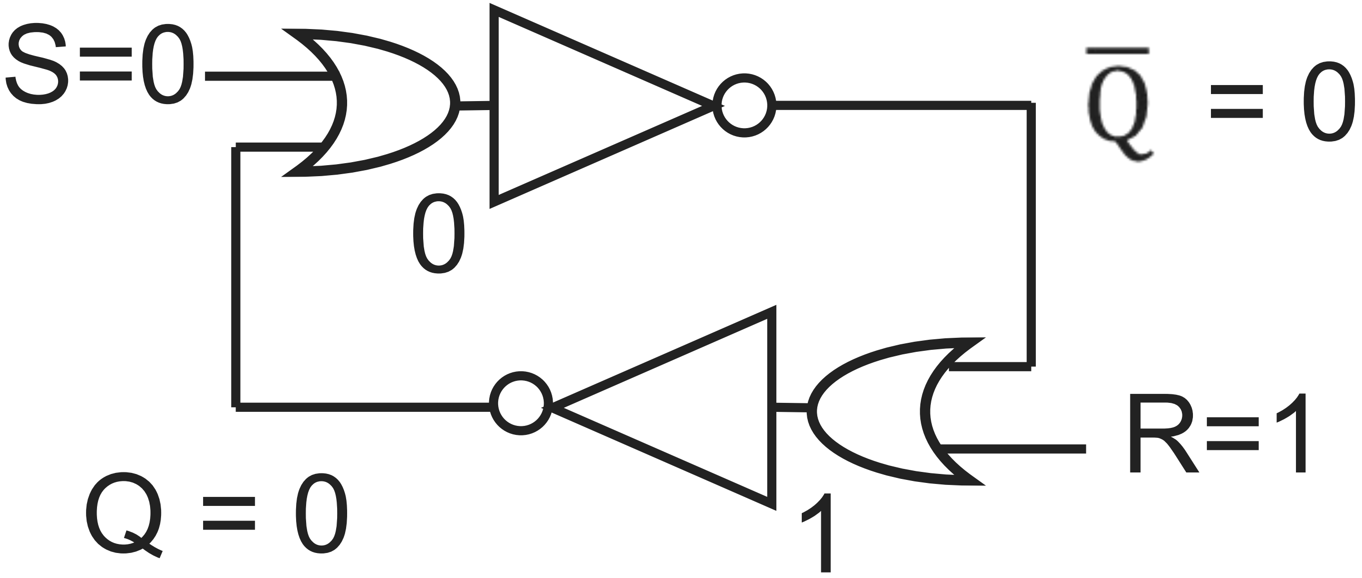
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Alternative: Just guess!

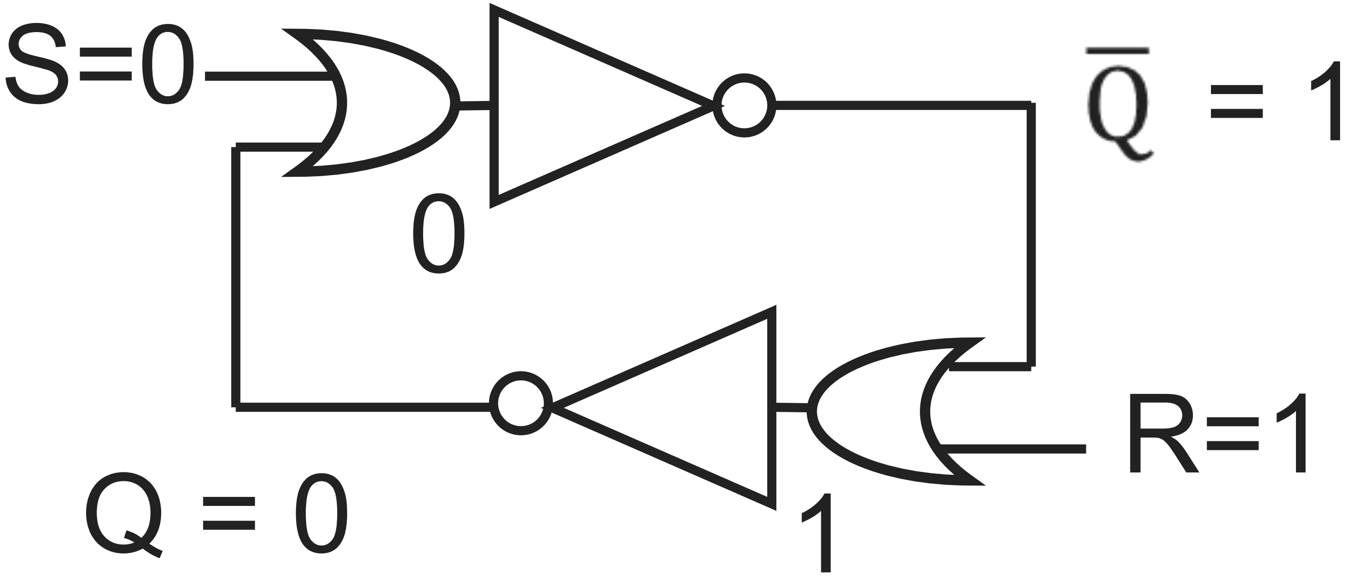
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Alternative: Just guess!

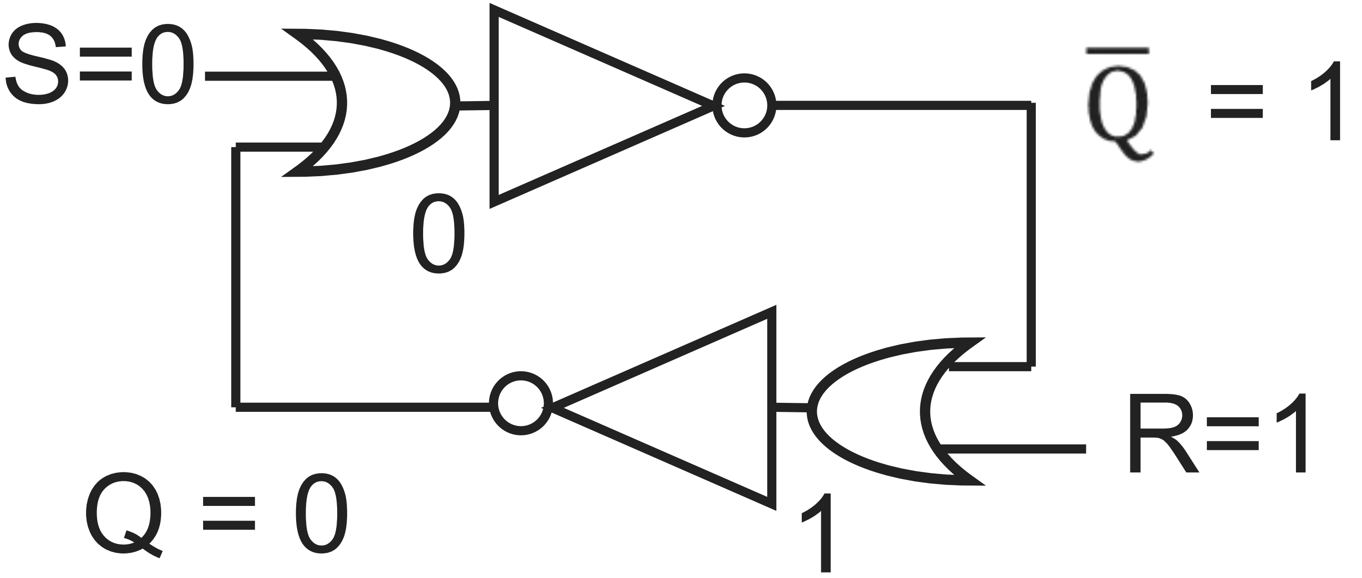
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1		
1	0		
1	1		

Alternative: Just guess!

Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1	0	1
1	0		
1	1		

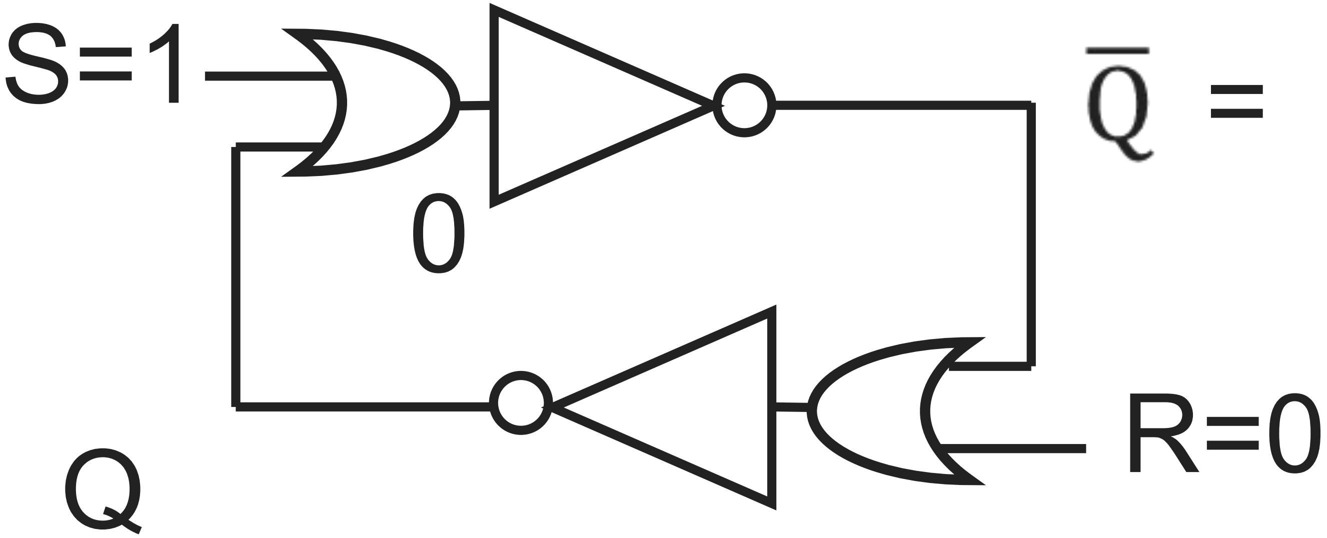
Alternative: Just guess!

S = 0, R = 1 is a **stable** state.

We always converge to the same state.

Cornell Bowers CIS
Computer Science

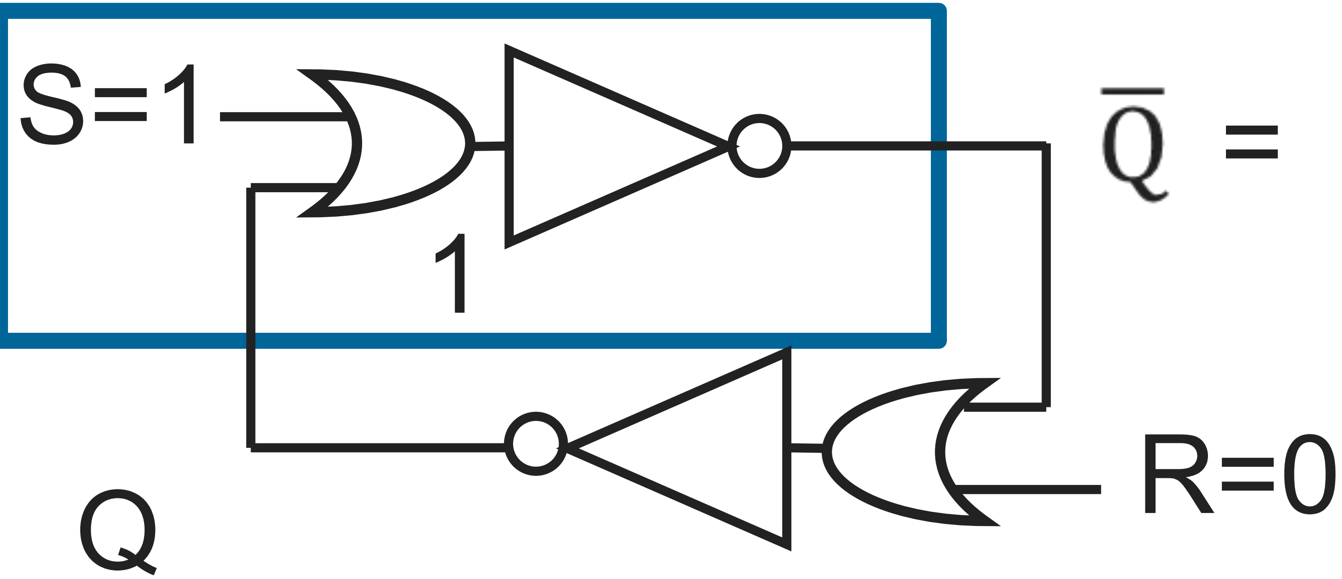
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1	0	1
1	0		
1	1		

PollEv.com/cs3410

Set-Reset Latch

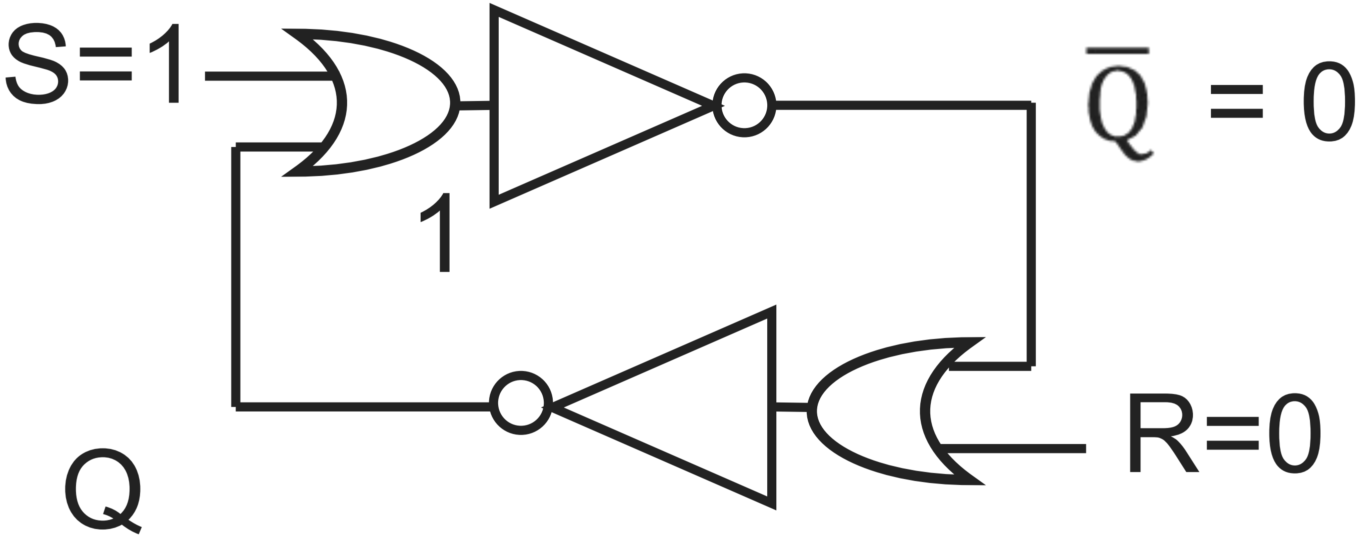


S	R	Q	¬Q
0	0		
0	1	0	1
1	0		
1	1		

Where do we start our analysis?

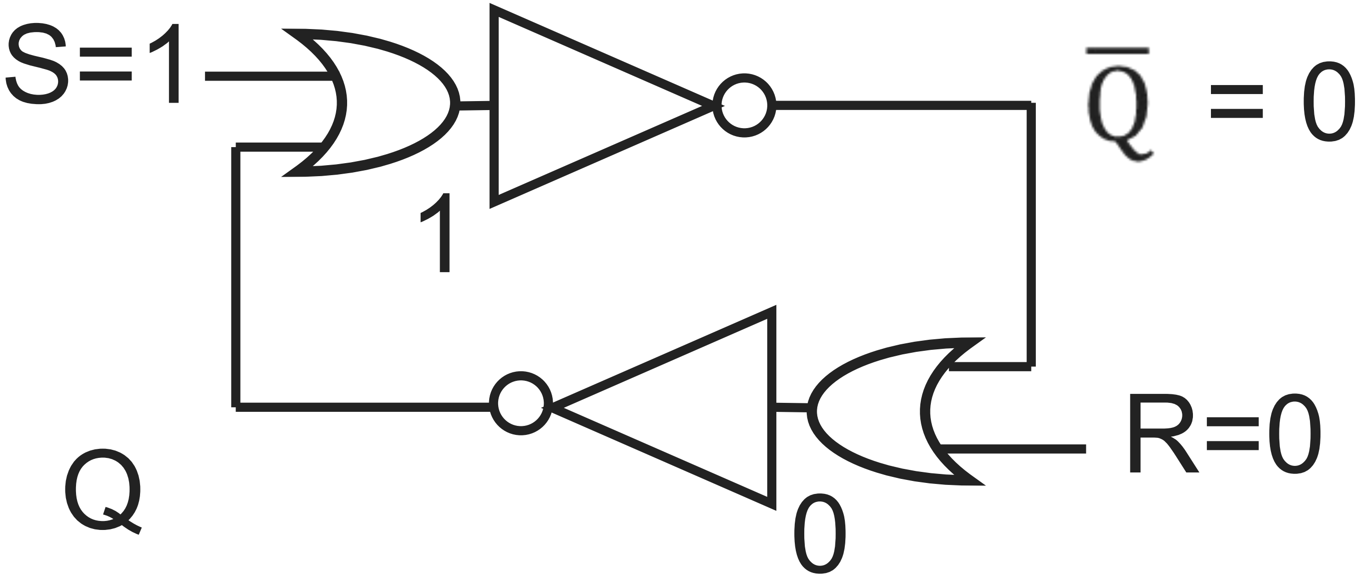
Remember: 1 ∨ a = 1

Set-Reset Latch



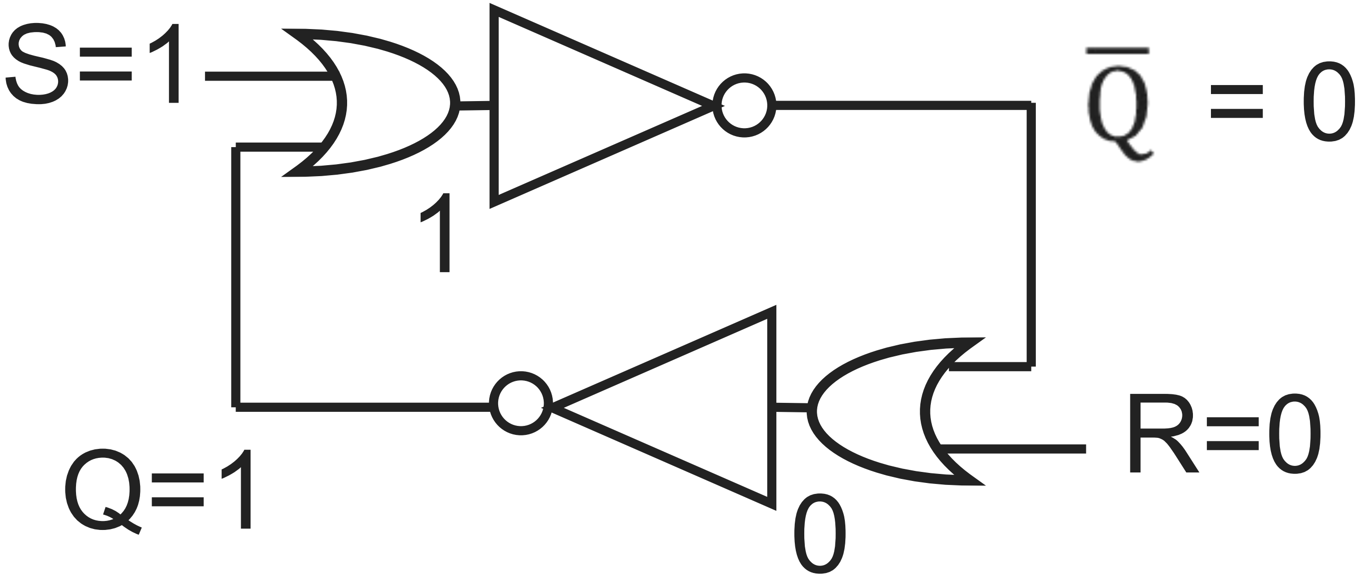
S	R	Q	¬Q
0	0		
0	1	0	1
1	0		
1	1		

Set-Reset Latch



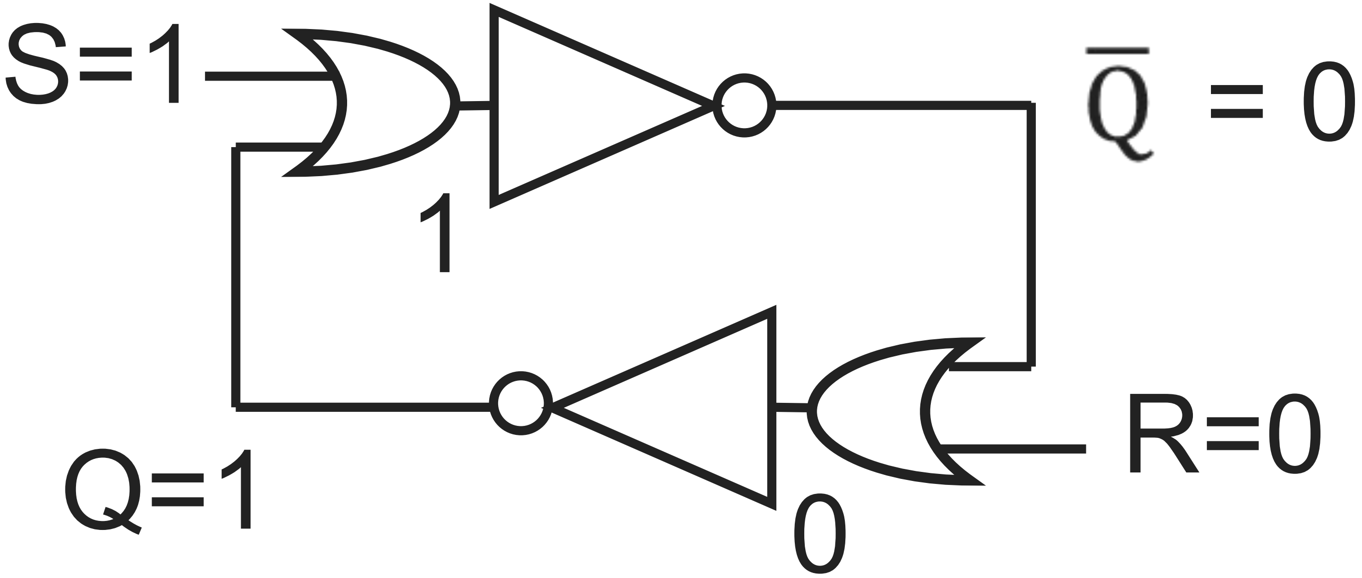
S	R	Q	¬Q
0	0		
0	1	0	1
1	0		
1	1		

Set-Reset Latch



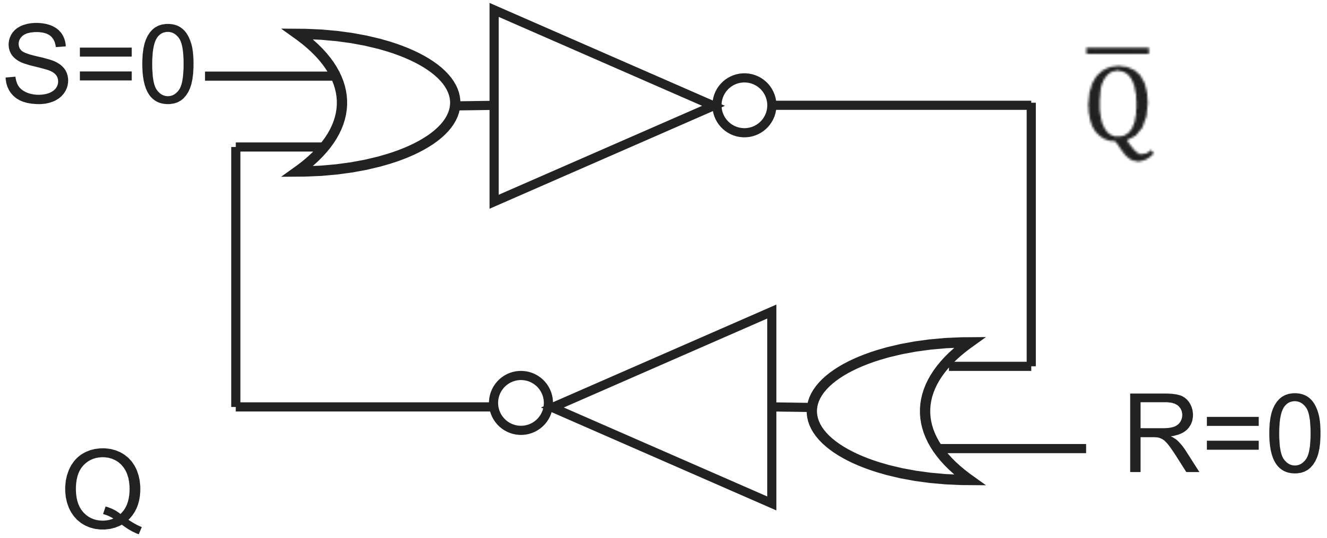
S	R	Q	¬Q
0	0		
0	1	0	1
1	0		
1	1		

Set-Reset Latch



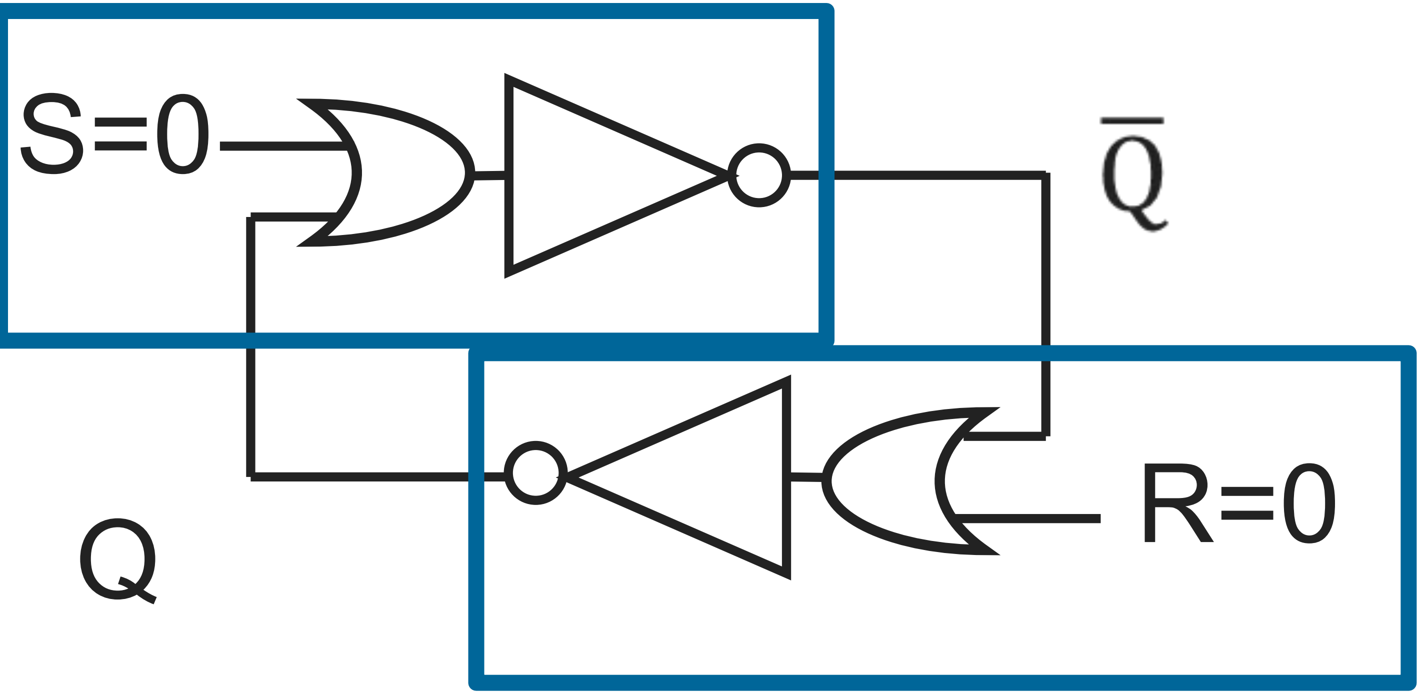
S	R	Q	¬Q
0	0		
0	1	0	1
1	0	1	0
1	1		

Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1	0	1
1	0	1	0
1	1		

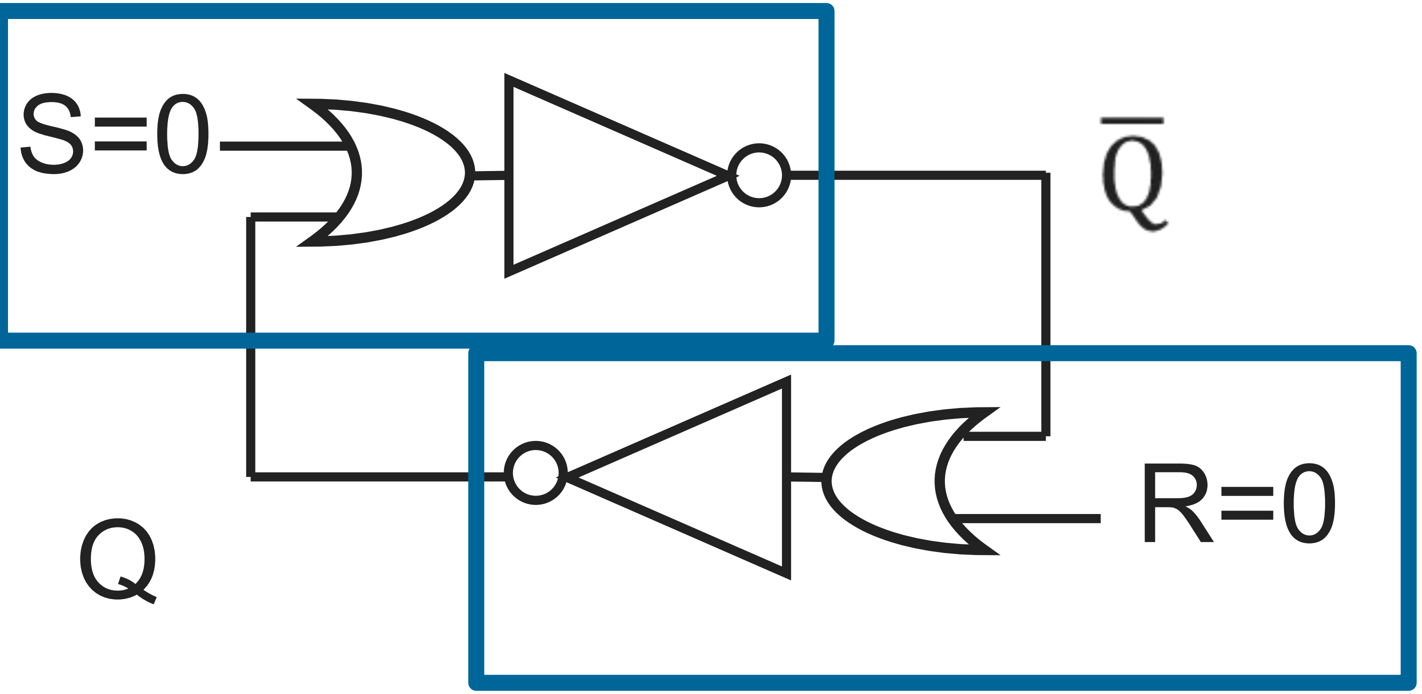
Set-Reset Latch



S	R	Q	¬Q
0	0		
0	1	0	1
1	0	1	0
1	1		

Our “OR trick” no longer works.

Set-Reset Latch

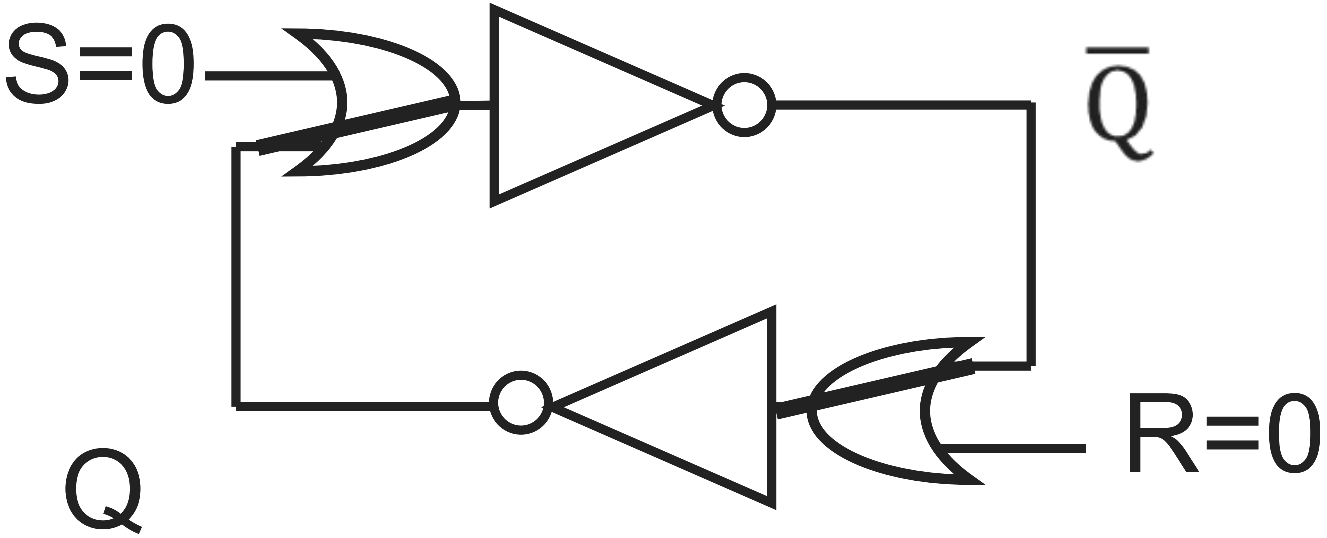


S	R	Q	¬Q
0	0		
0	1	0	1
1	0	1	0
1	1		

Our “OR trick” no longer works.

Remember: $0 \vee a = a$

Set-Reset Latch

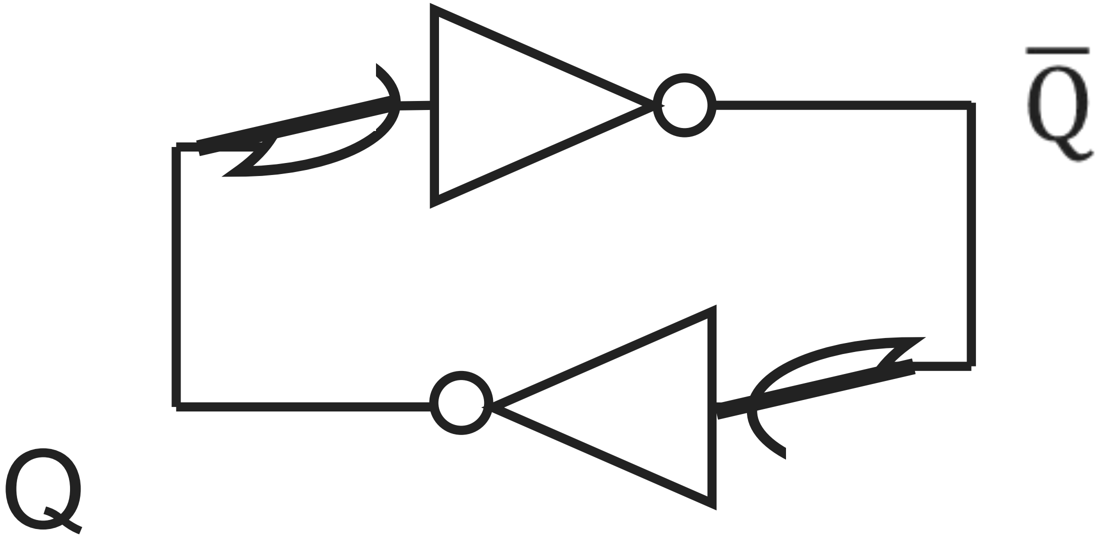


S	R	Q	¬Q
0	0		
0	1	0	1
1	0	1	0
1	1		

Our “OR trick” no longer works.

Remember: 0 ⋁ a = a

Set-Reset Latch

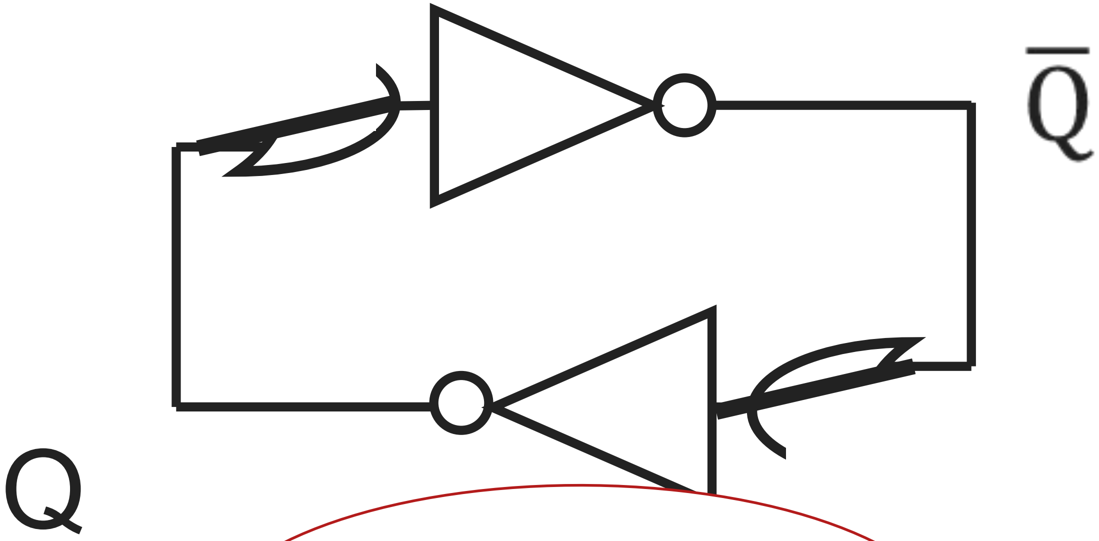


S	R	Q	¬Q
0	0		
0	1	0	1
1	0	1	0
1	1		

Our “OR trick” no longer works.
Remember: 0 ∨ a = a

Cornell Bowers CIS
Computer Science

Set-Reset Latch

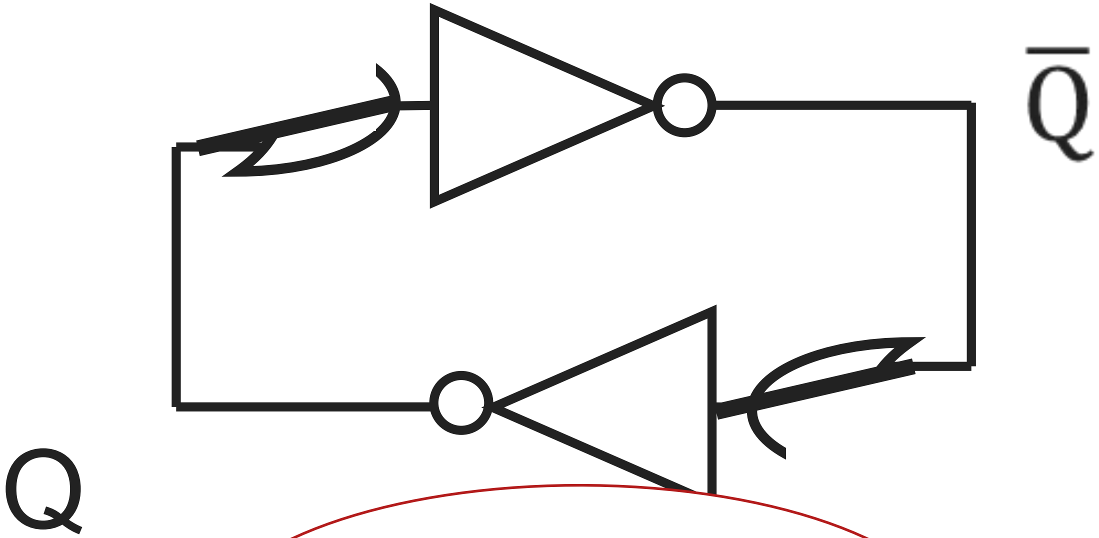


S	R	Q	¬Q
0	0		
0	1	0	1
1	0	1	0
1	1		

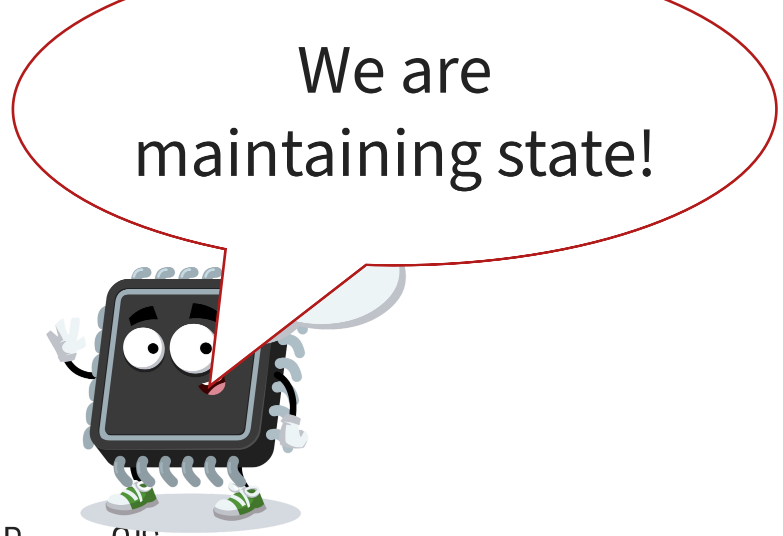
Our “OR trick” no longer works.

Remember: $0 \lor a = a$

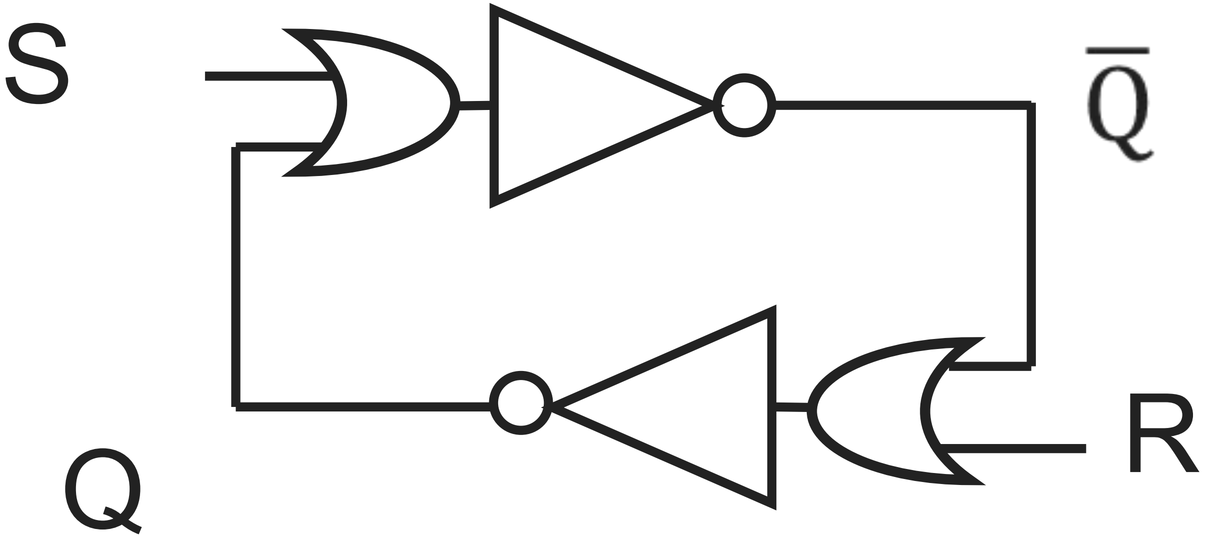
Set-Reset Latch



S	R	Q	¬Q
0	0	Q_{t-1}	¬Q_{t-}
			1
0	1	0	1
1	0	1	0
1	1		

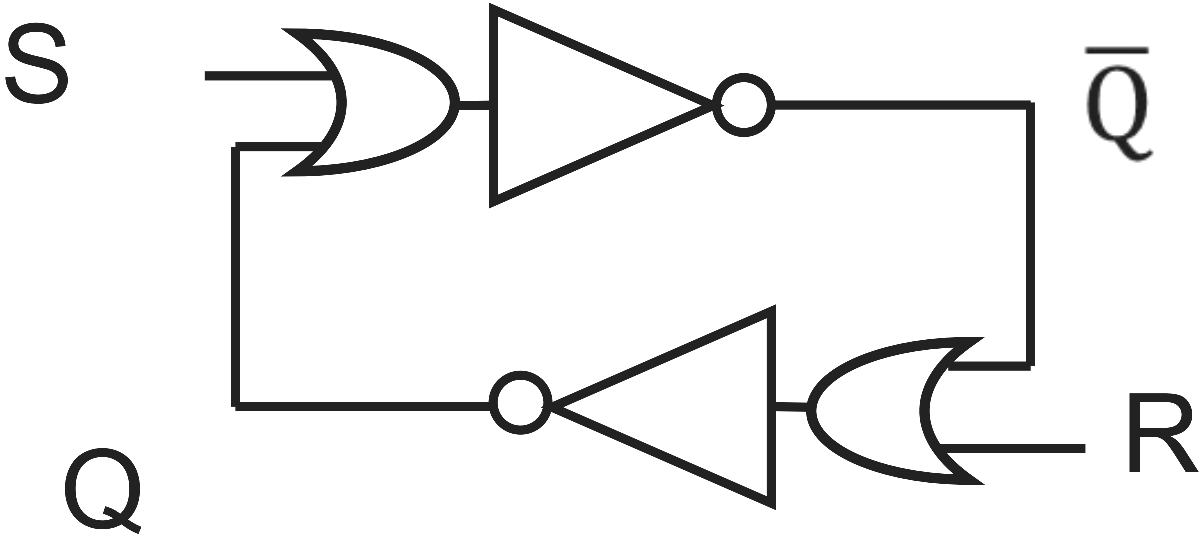


Set-Reset Latch



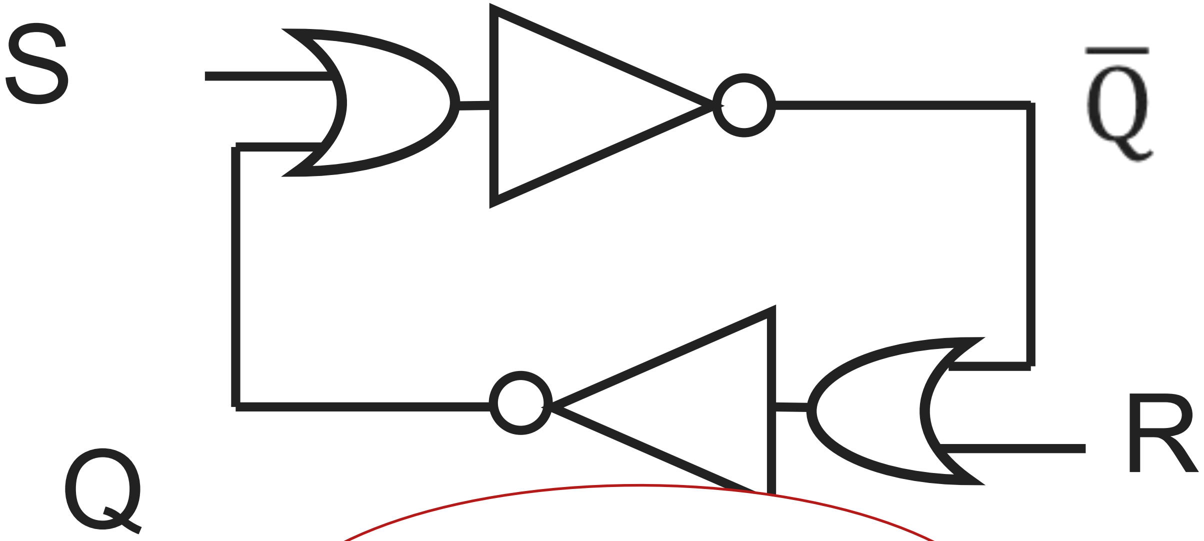
S	R	Q	¬Q
0	0	Q_{t-1}	$\neg Q_{t-1}$
0	1	0	1
1	0	1	0
1	1		

Set-Reset Latch



S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			

Set-Reset Latch

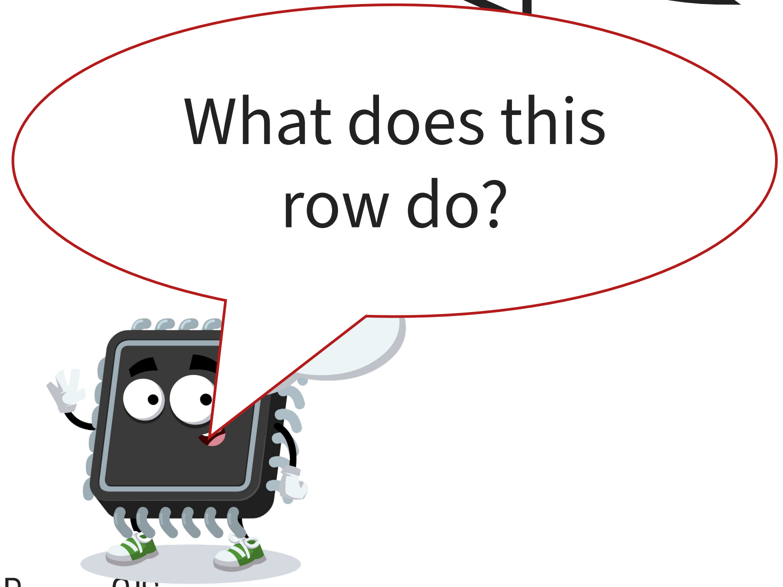


S	R	Q	¬Q
0	0	Q_{t-1}	$\neg Q_{t-}$ 1
0	1	0	1
1	0	1	0
1	1		

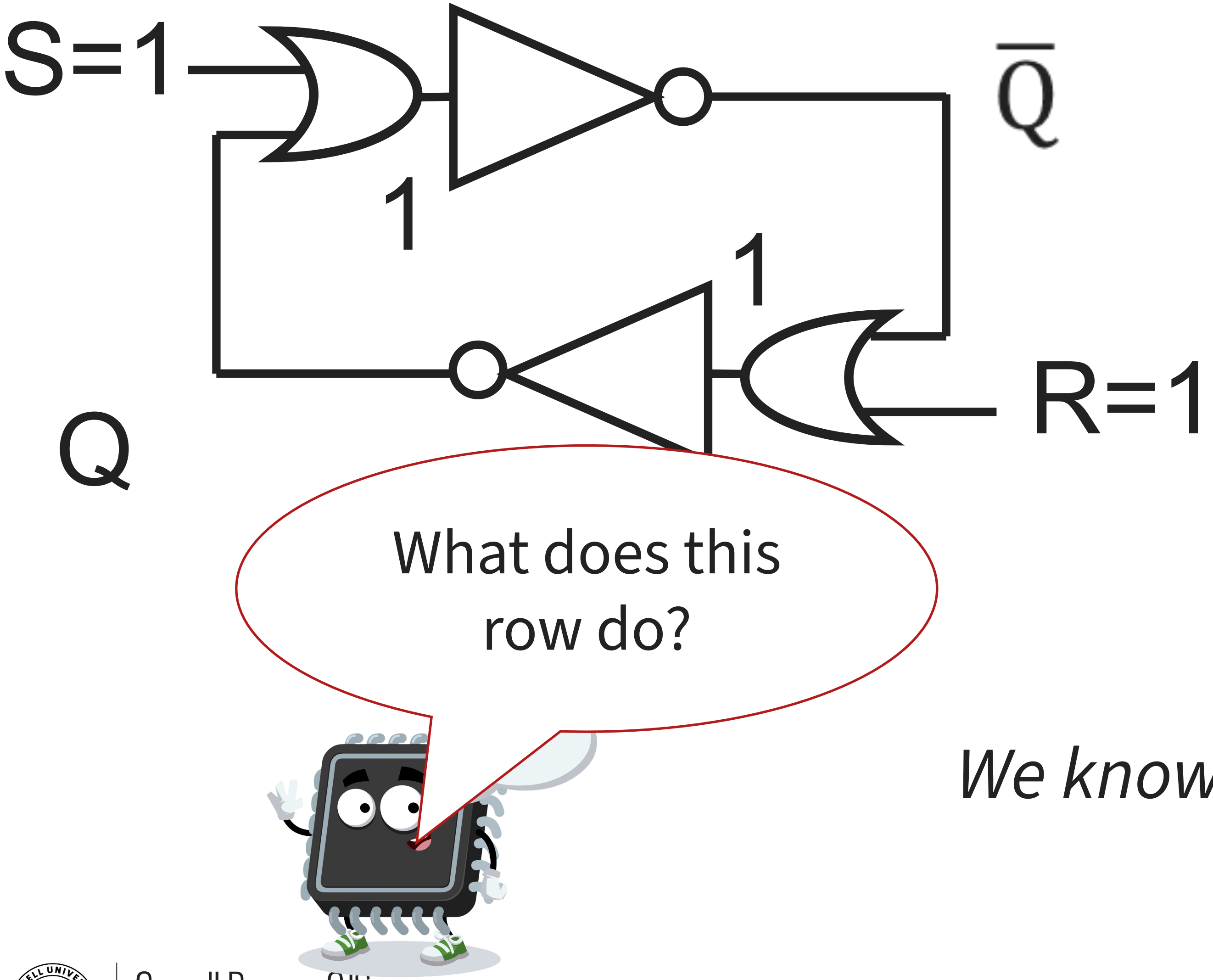
Retain

Reset

Set



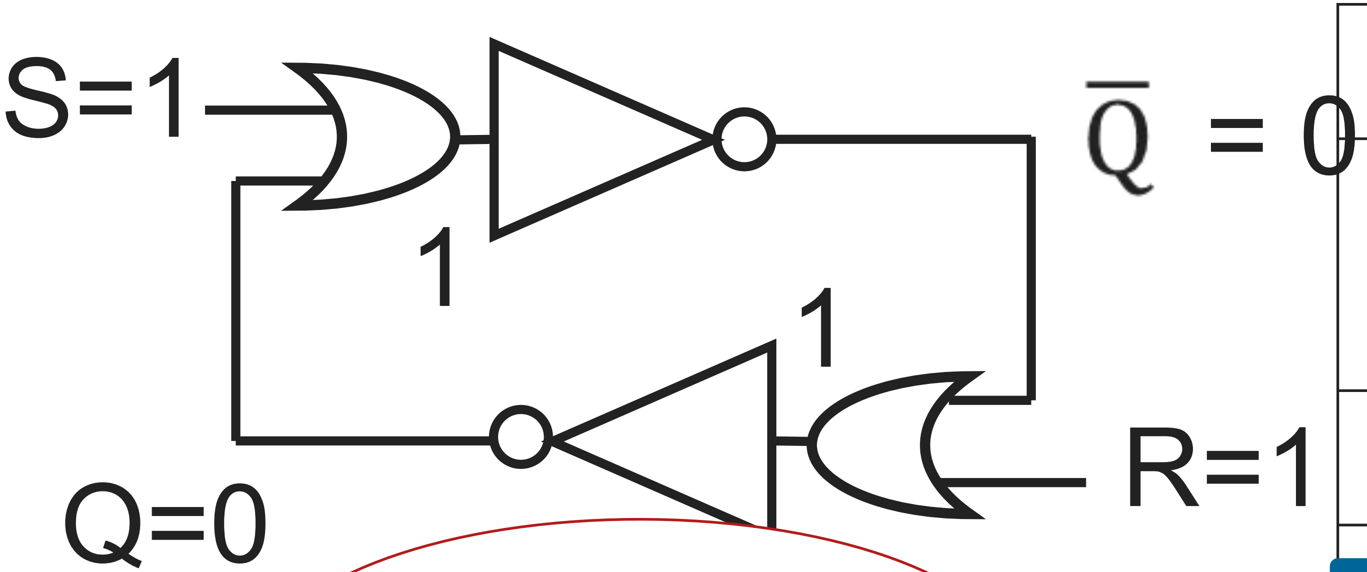
Set-Reset Latch



S	R	Q	¬Q	
0	0	Q_{t-1}	¬Q_{t-} 1	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			

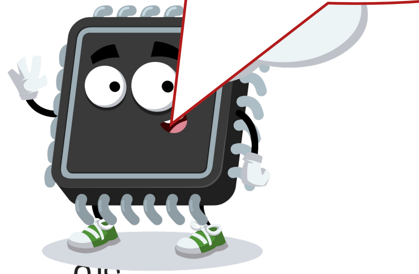
We know: 1 ∨ a = 1

Set-Reset Latch



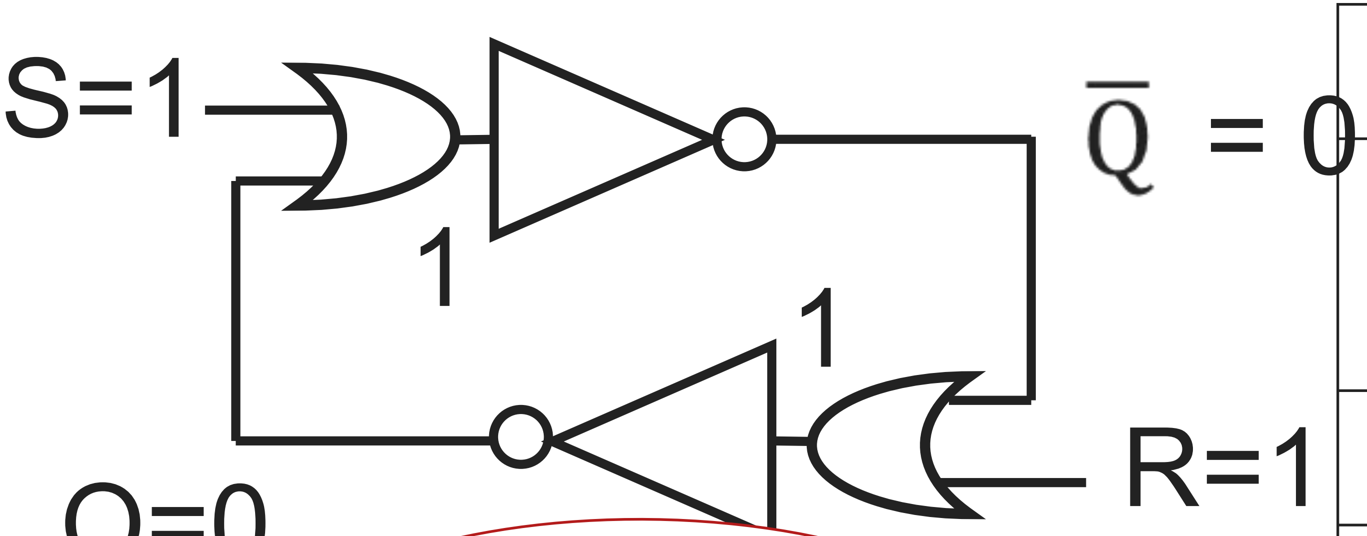
S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			

What does this row do?

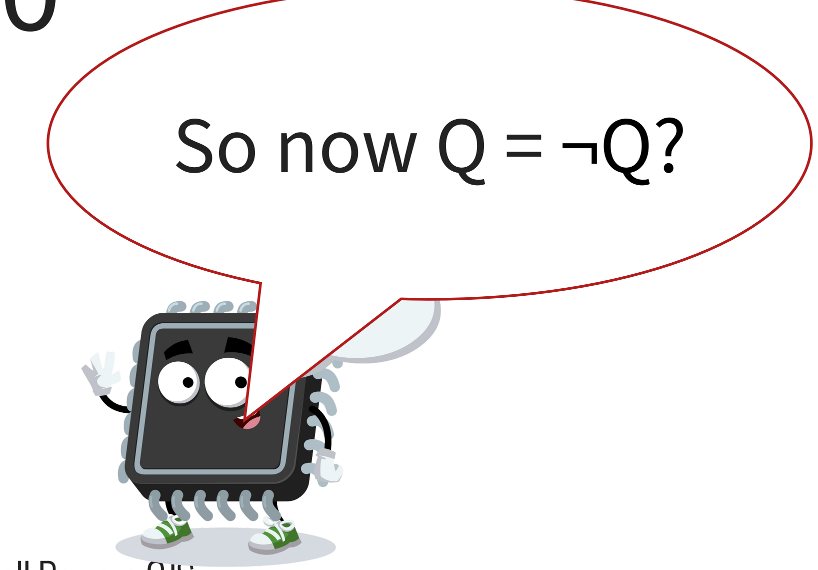


We know: 1 ∨ a = 1

Set-Reset Latch

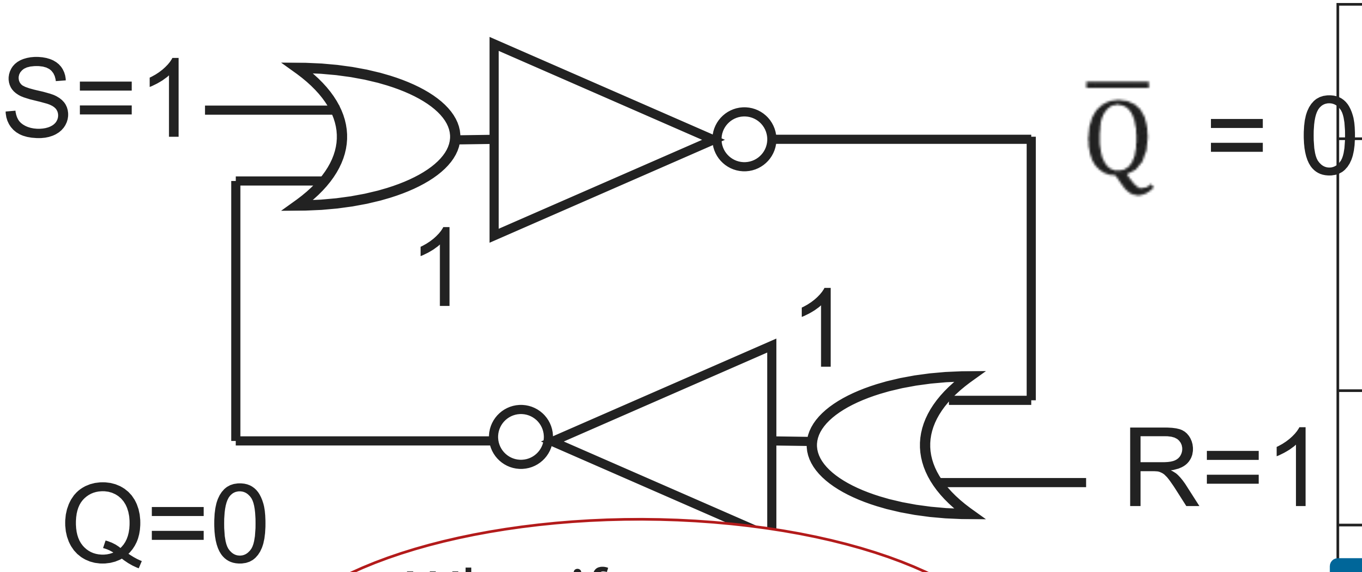


S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	Retain Reset
0	1	0	1	Set
1	0	1	0	
1	1			



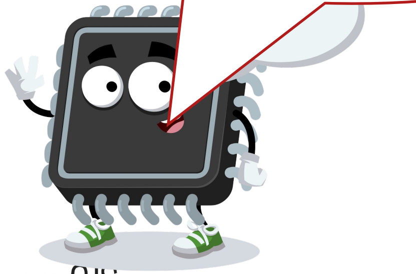
We know: 1 ∨ a = 1

Set-Reset Latch

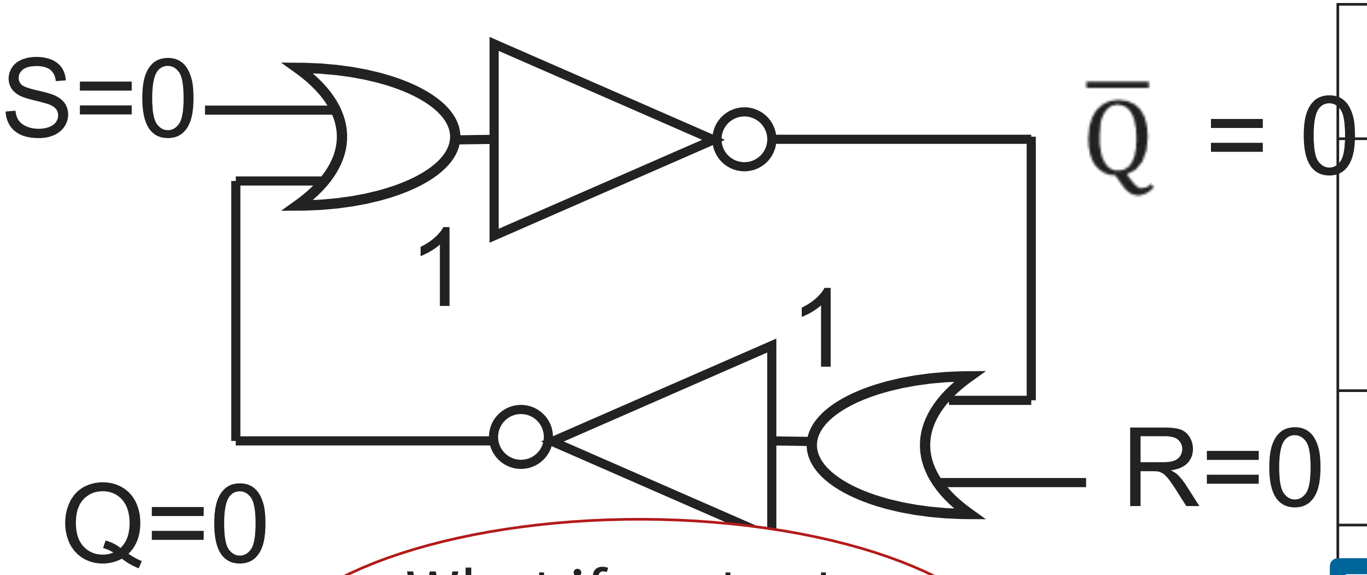


S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-}$ 1	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			

What if we try to go back to S = 0 and R = 0?

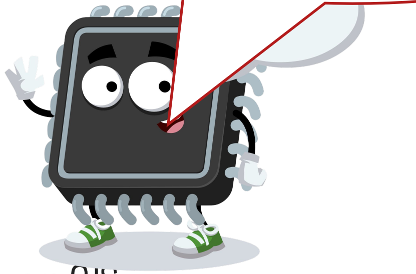


Set-Reset Latch

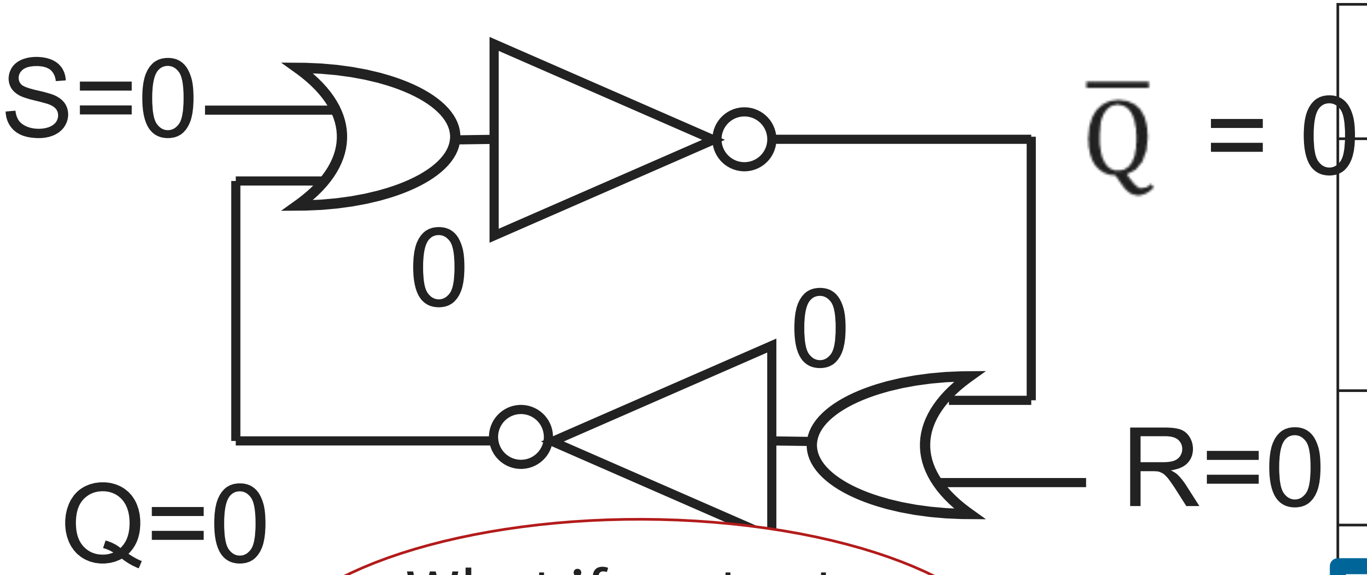


S	R	Q	¬Q	
0	0	Q_{t-1}	¬Q_{t-} 1	Retain Reset
0	1	0	1	Set
1	0	1	0	
1	1			

What if we try to go back to S = 0 and R = 0?

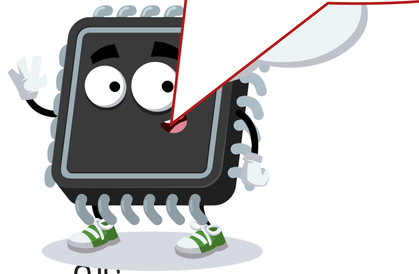


Set-Reset Latch

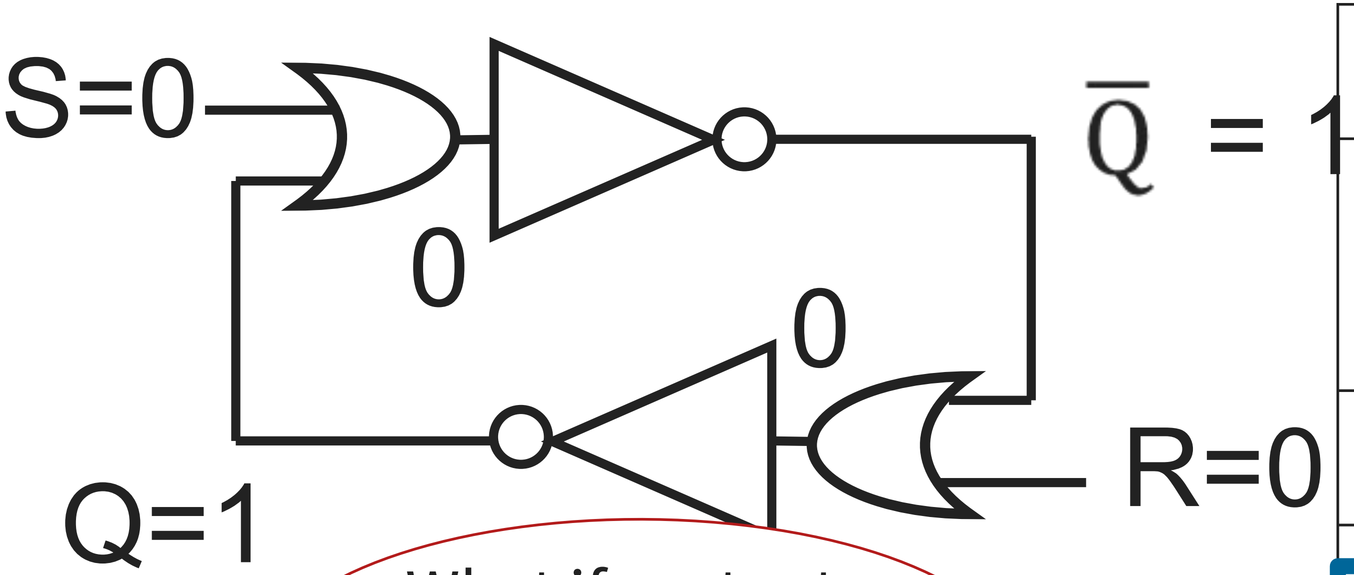


S	R	Q	¬Q	
0	0	Q_{t-1}	$¬Q_{t-1}$	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			

What if we try to go back to S = 0 and R = 0?

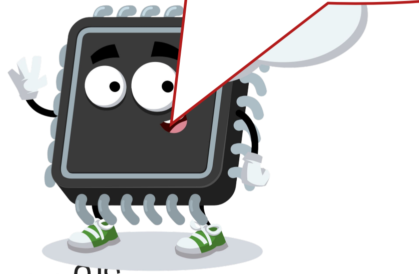


Set-Reset Latch

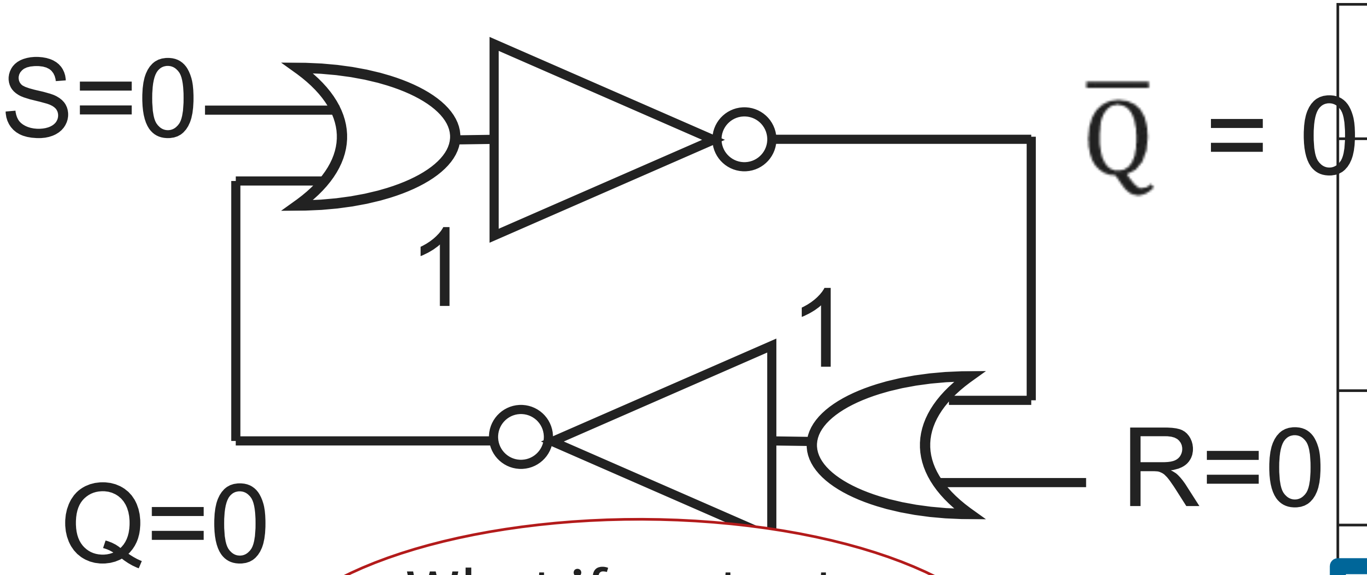


S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			

What if we try to go back to S = 0 and R = 0?

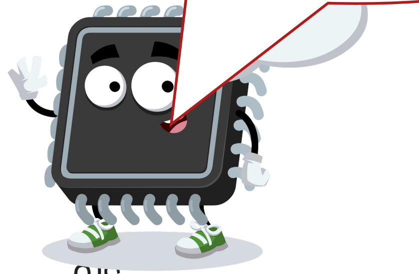


Set-Reset Latch

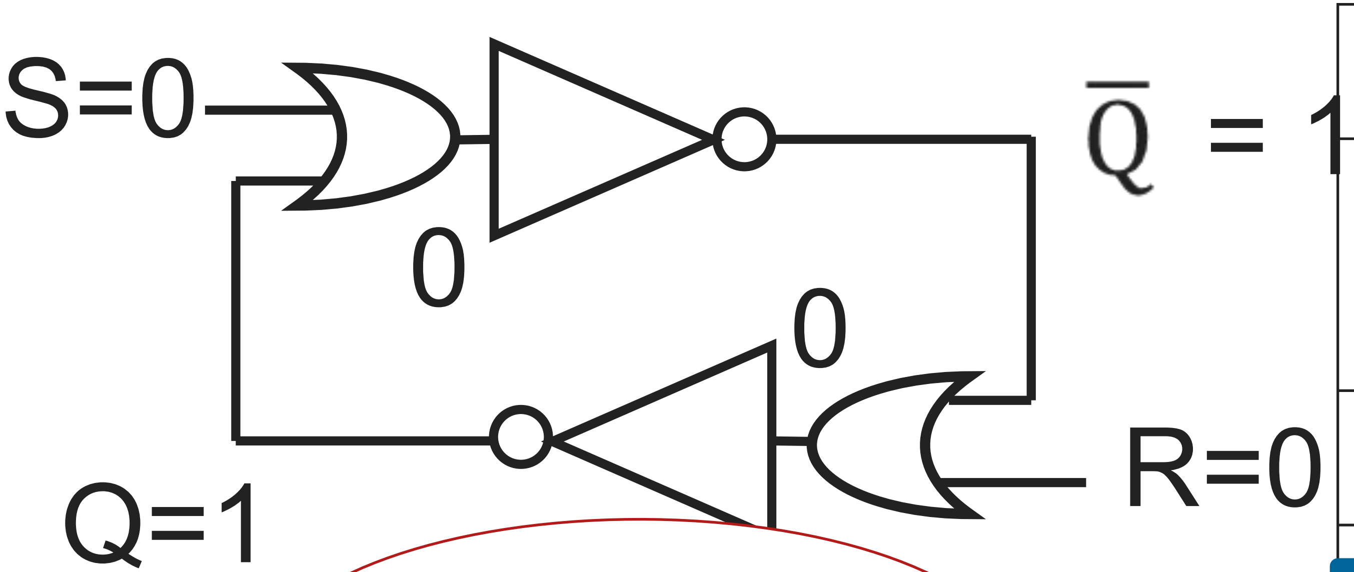


S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-}$ 1	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			

What if we try to go back to S = 0 and R = 0?

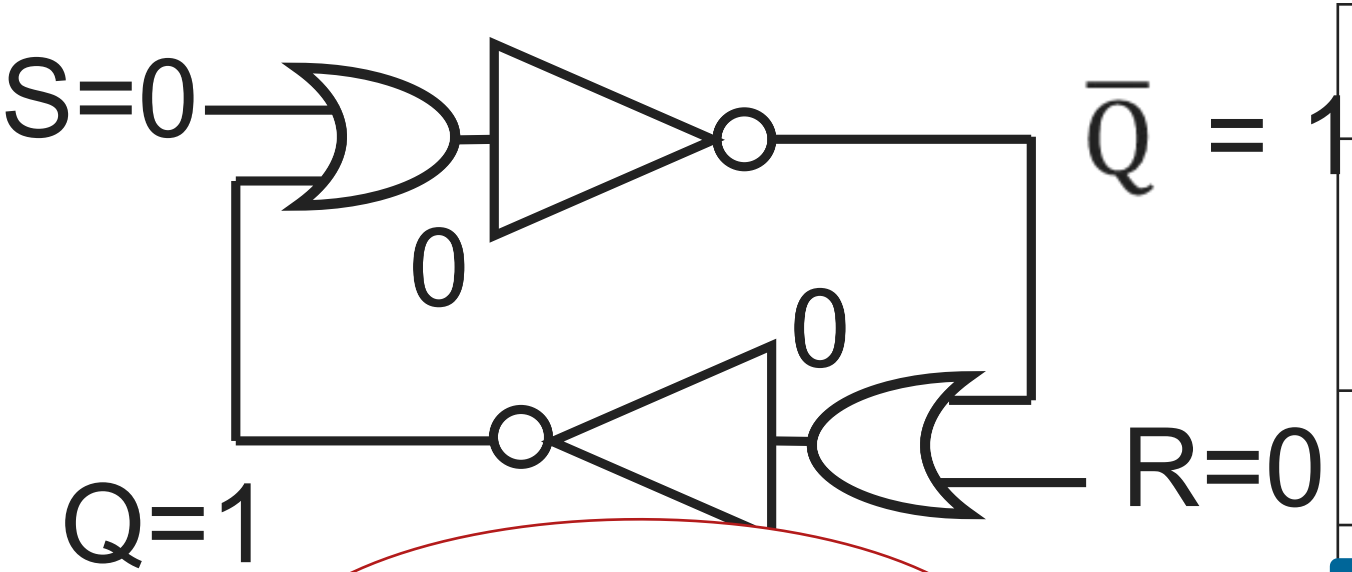


Set-Reset Latch

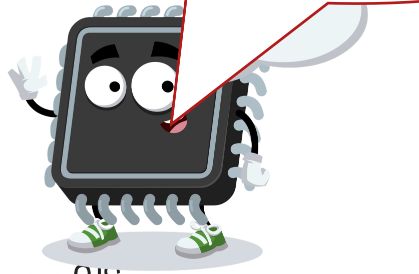


S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			

Set-Reset Latch



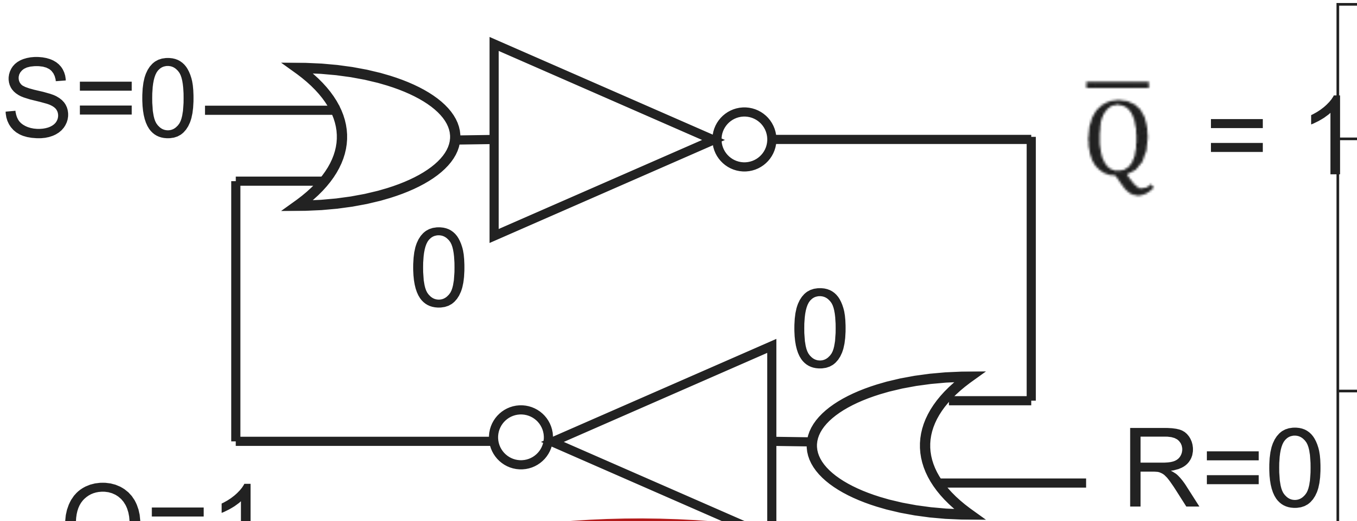
S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			



In theory: Yes.

In practice: we will end up with stable feedback, but unpredictable if Q = 0 or Q = 1.

Set-Reset Latch



S	R	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	Retain
0	1	0	1	Reset
1	0	1	0	Set
1	1			Forbidden

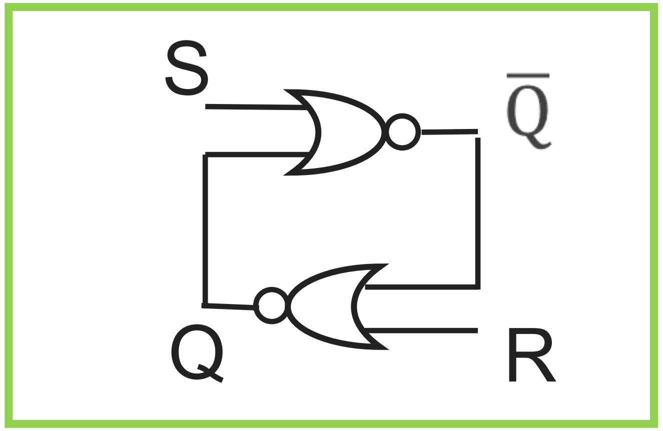
In theory: Yes.

In practice: we will end up with stable feedback, but unpredictable if Q = 0 or Q = 1.

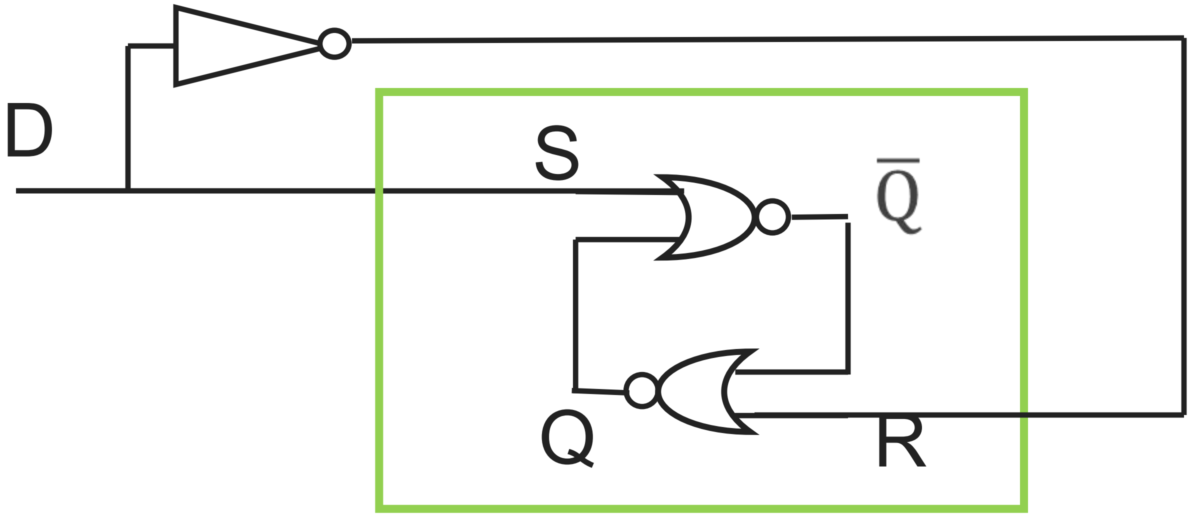
Next Goal

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

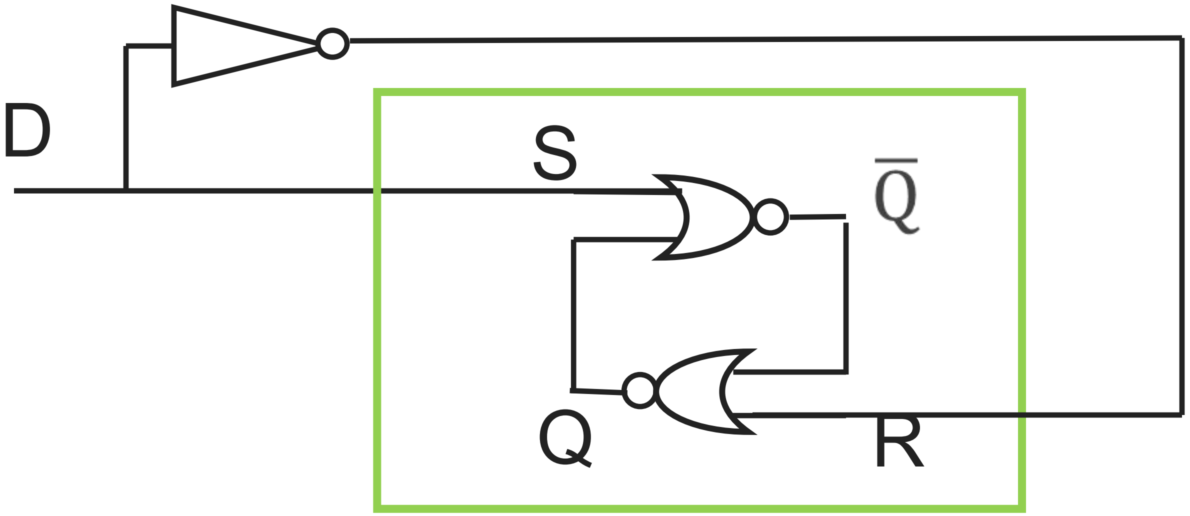
Fourth Attempt: (Unclocked) D Latch



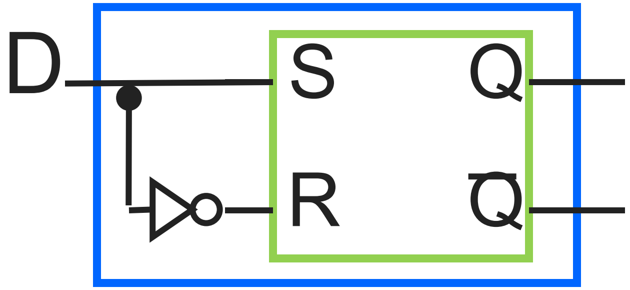
Fourth Attempt: (Unclocked) D Latch



Fourth Attempt: (Unclocked) D Latch

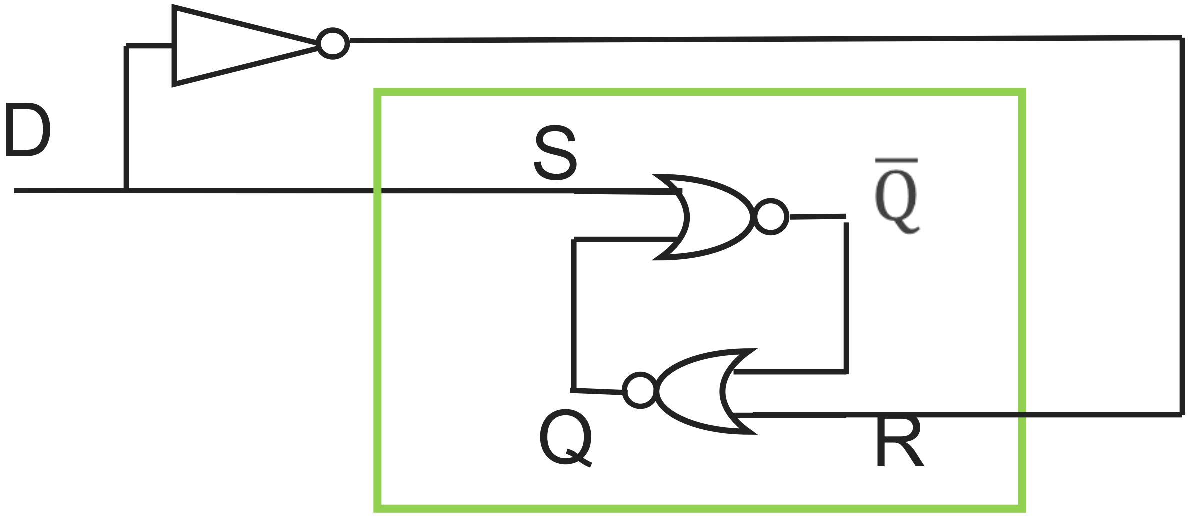


Fill in the truth table?

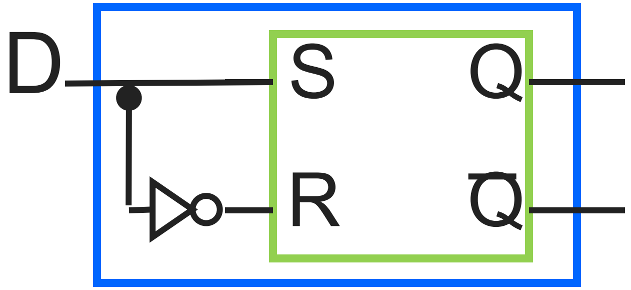


D	Q	
0		
1		

Fourth Attempt: (Unclocked) D Latch

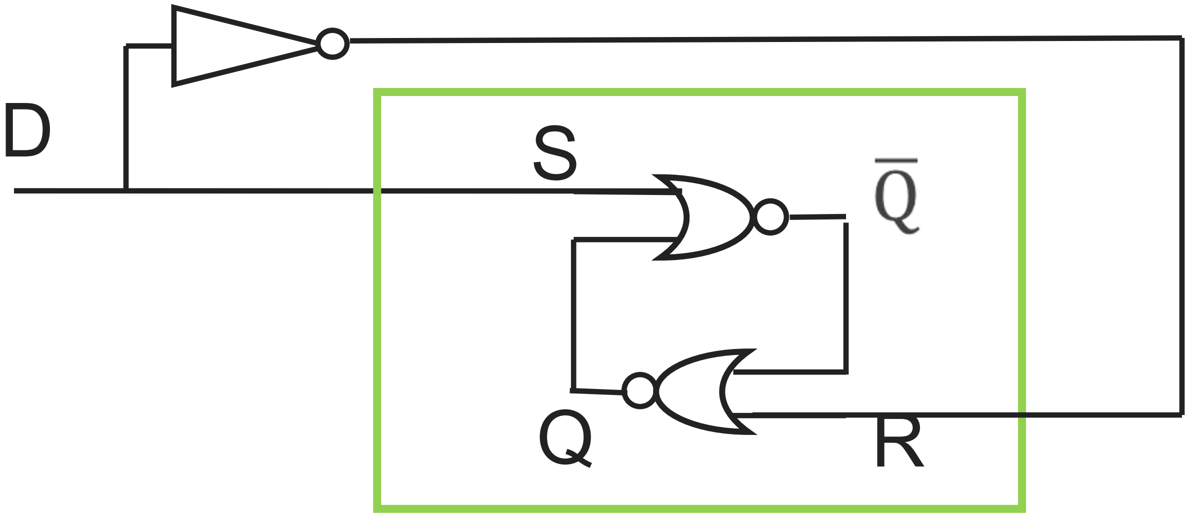


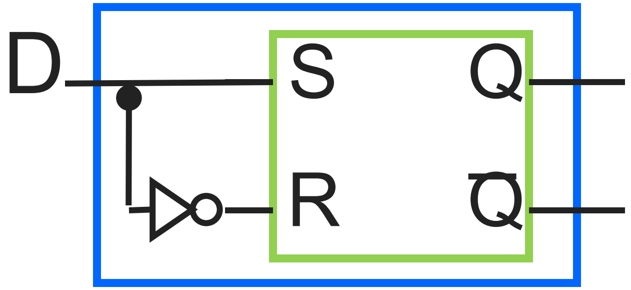
Fill in the truth table?



D	Q	¬Q
0	0	1
1	1	0

Fourth Attempt: (Unclocked) D Latch





D	Q	¬Q
0	0	1
1	1	0

Cannot enter an undefined state

When D changes, Q changes

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

We aren’t really storing anything anymore!

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.

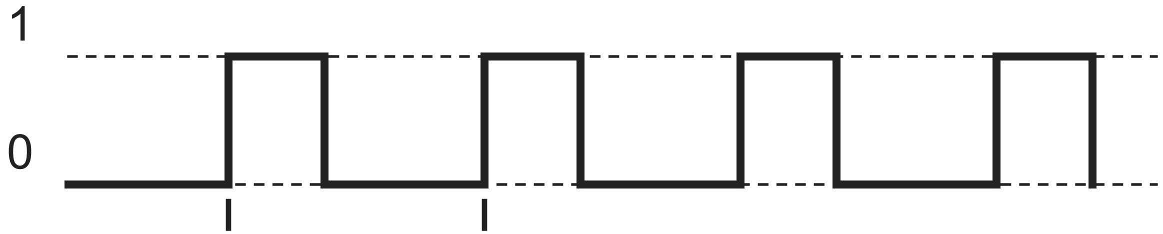
Next Goal

How do we coordinate state changes to a D Latch?

Aside: Clocks

Clock helps coordinate state changes

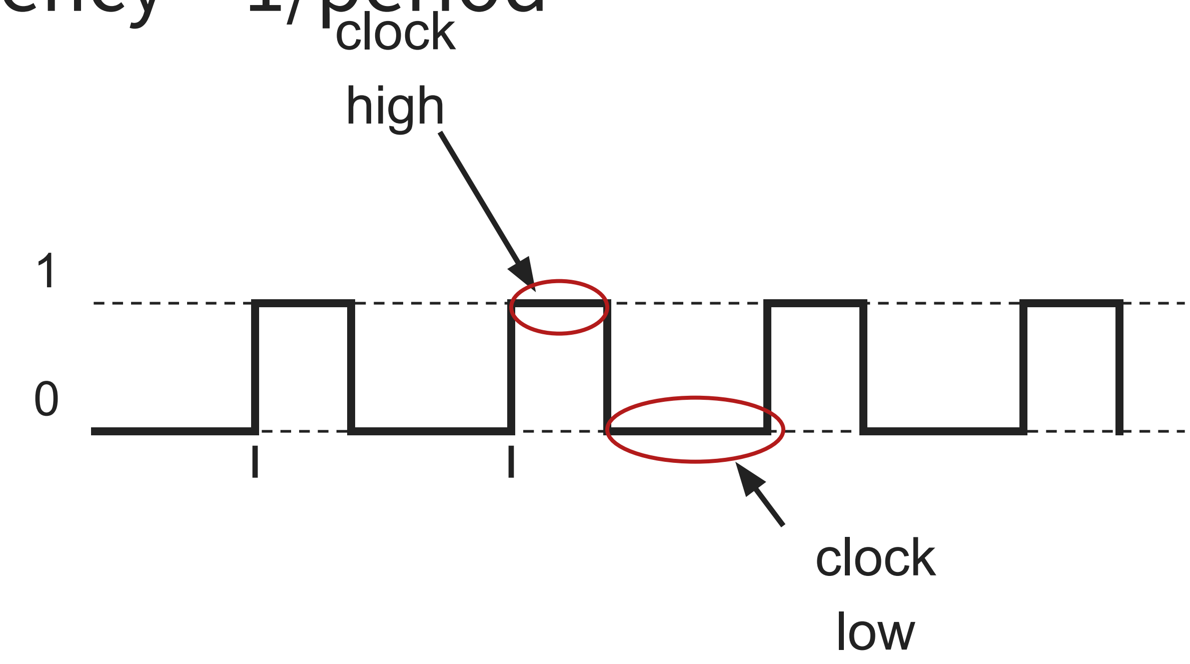
- Usually generated by an oscillating crystal
- Fixed period
- Frequency = 1/period



Aside: Clocks

Clock helps coordinate state changes

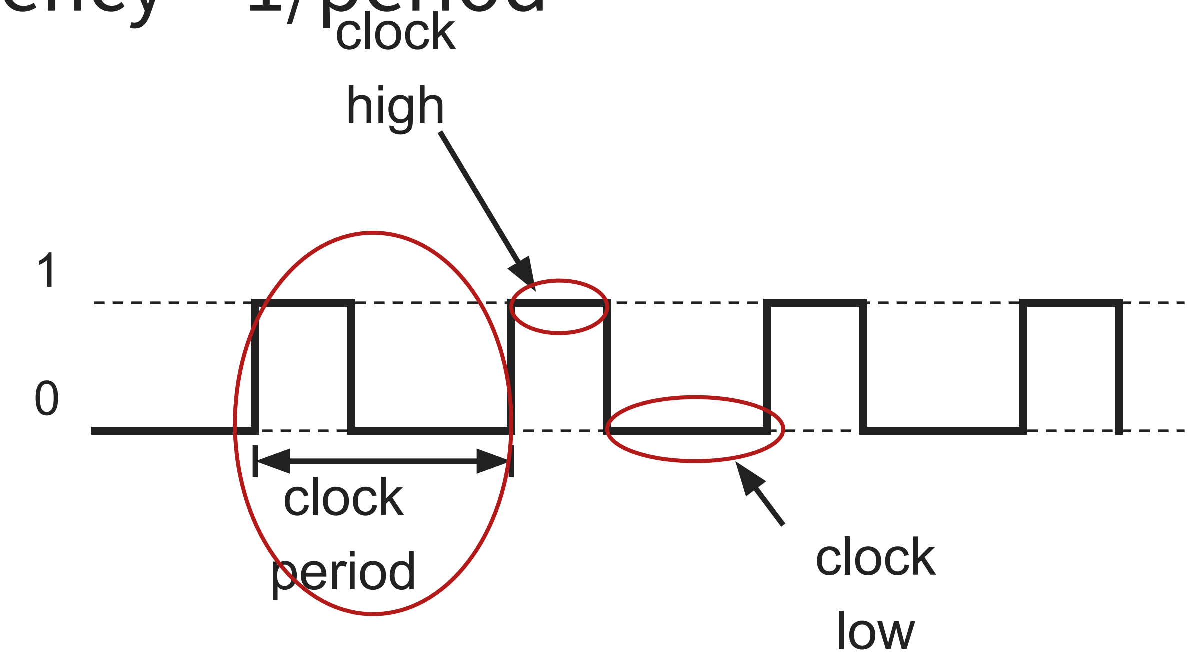
- Usually generated by an oscillating crystal
- Fixed period
- Frequency = 1/period



Aside: Clocks

Clock helps coordinate state changes

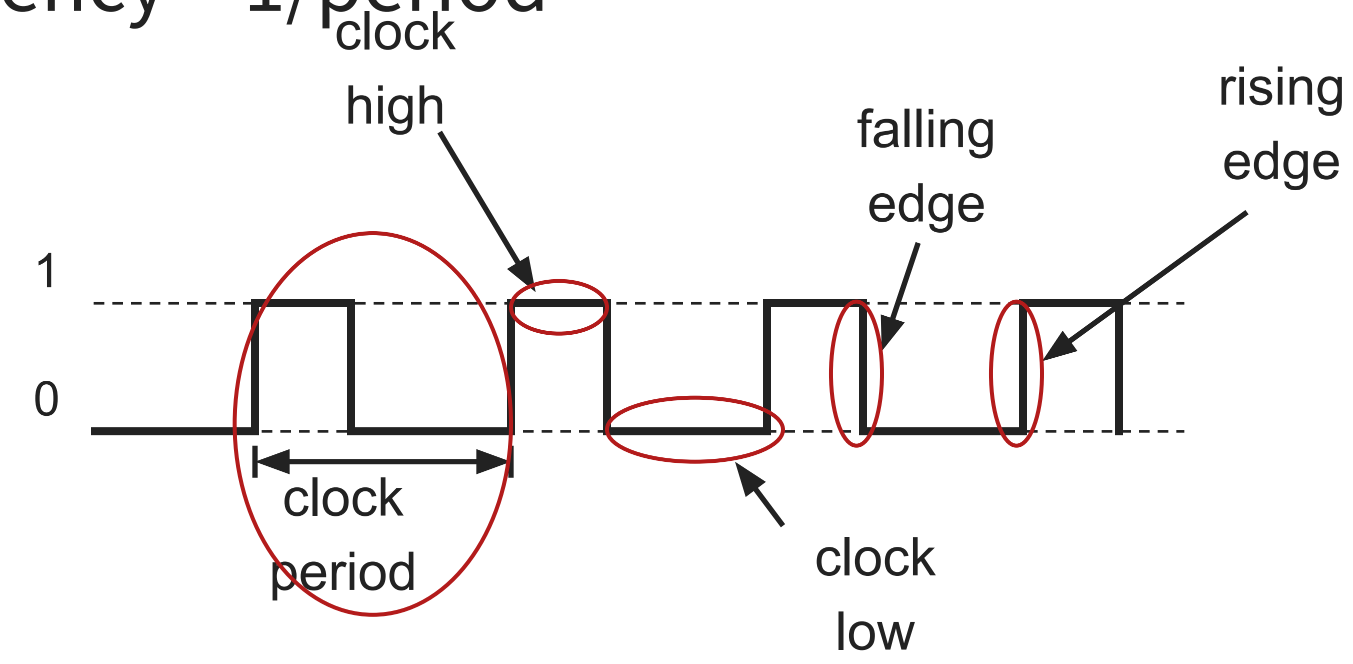
- Usually generated by an oscillating crystal
- Fixed period
- Frequency = 1/period



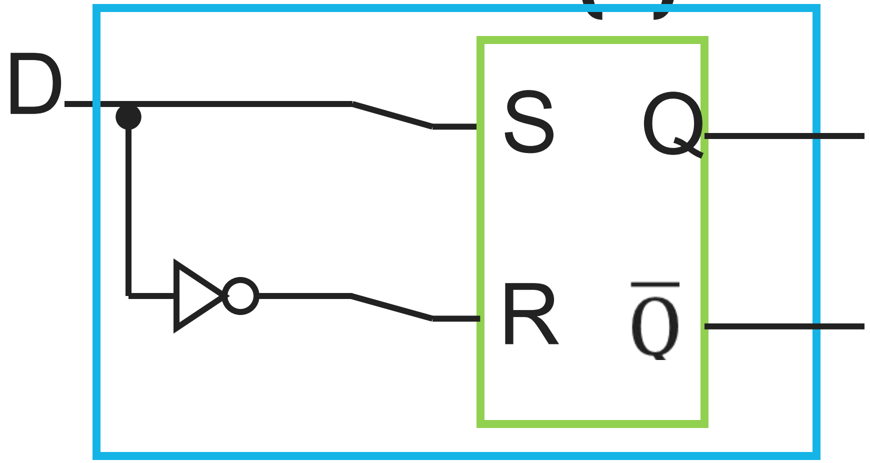
Aside: Clocks

Clock helps coordinate state changes

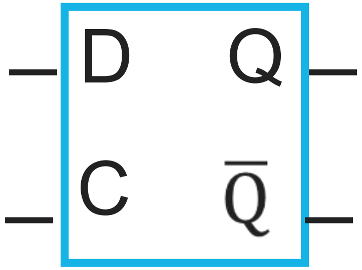
- Usually generated by an oscillating crystal
- Fixed period
- Frequency = 1/period



Round 2: D Latch (1)

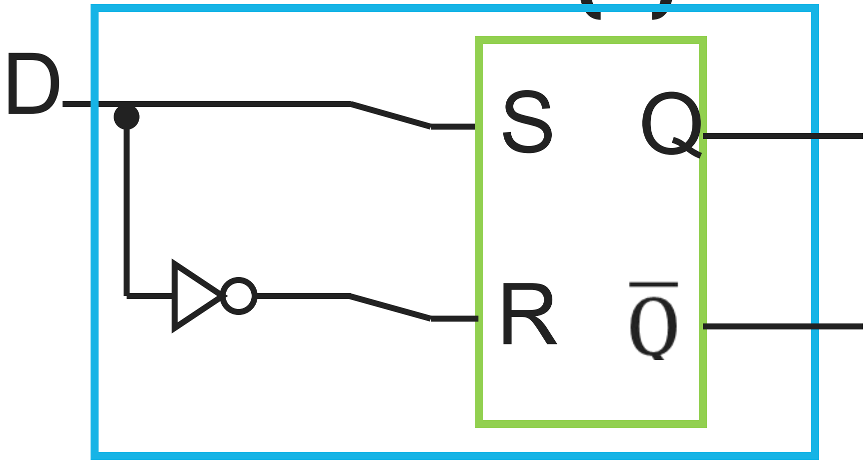


- Inverter prevents SR Latch from entering 1,1 state

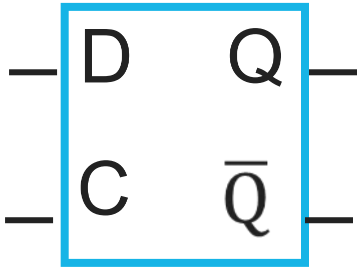


	D	Q	¬Q	
	0			*Reset*
	1			*Set*

Round 2: D Latch (1)



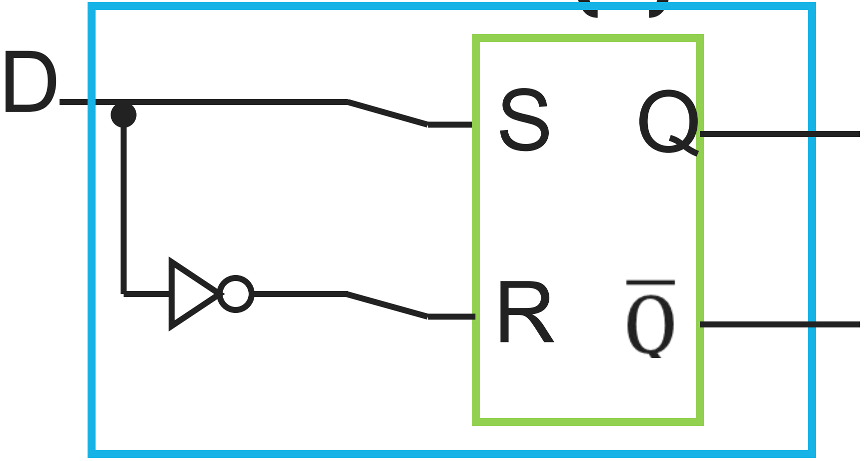
- Inverter prevents SR Latch from entering 1,1 state



	D	Q	¬Q	
	0	0	1	*Reset*
	1	1	0	*Set*

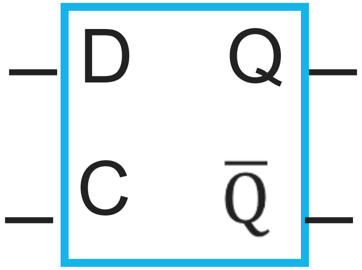
Cornell Bowers CIS
Computer Science

Round 2: D Latch (1)

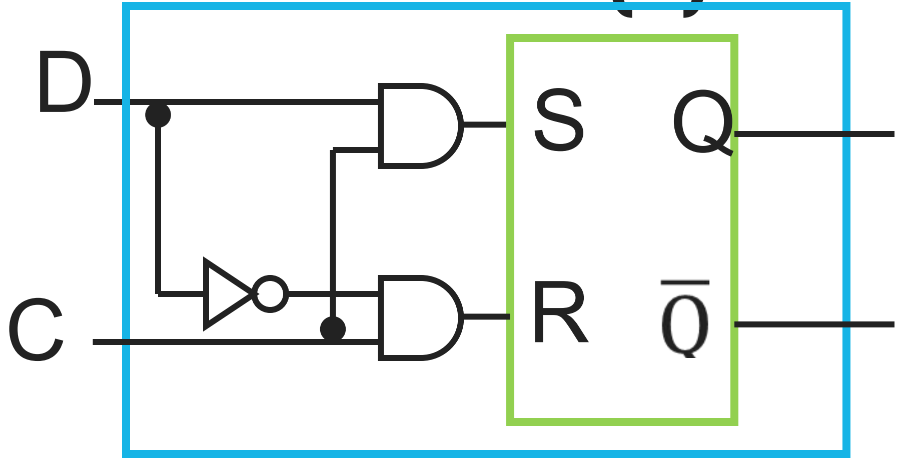


- Inverter prevents SR Latch from entering 1,1 state

	D	Q	¬Q	
	0	0	1	*Reset*
	1	1	0	*Set*

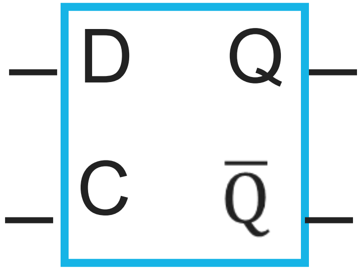


Round 2: D Latch (1)

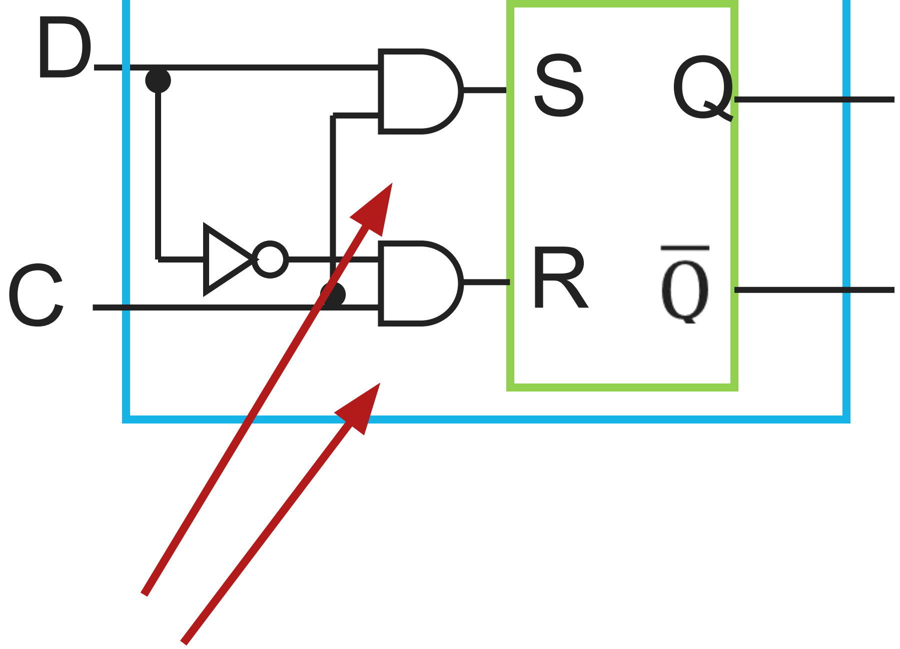


- Inverter prevents SR Latch from entering 1,1 state

	D	Q	¬Q	
	0	0	1	*Reset*
	1	1	0	*Set*



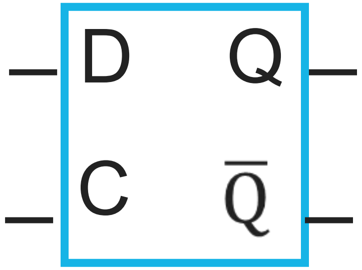
Round 2: D Latch (1)



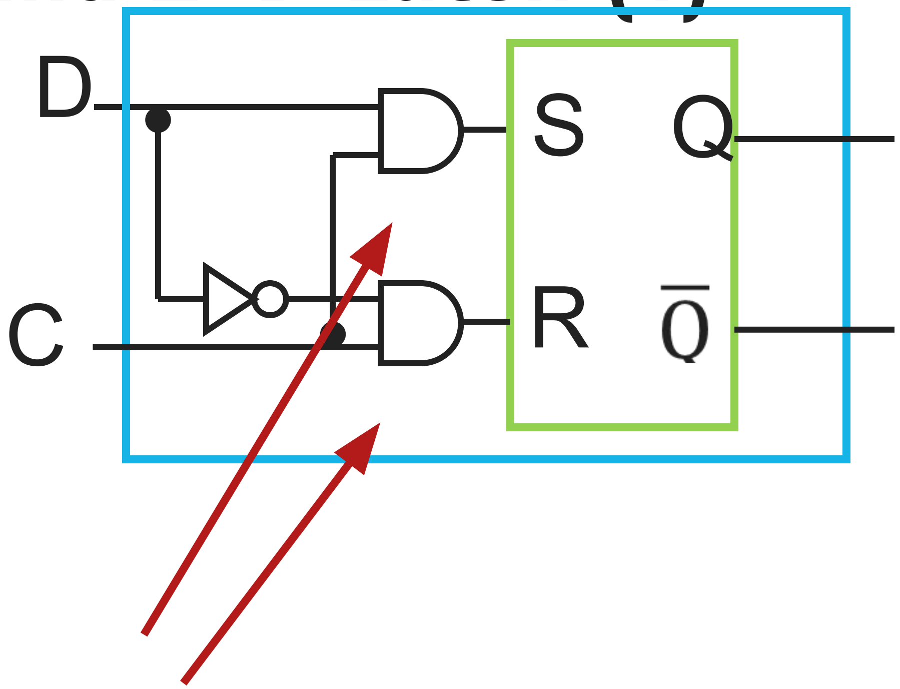
- Inverter prevents SR Latch from entering 1,1 state

AND gate **forces a 0** when C = 0.

	D	Q	¬Q	
	0	0	1	*Reset*
	1	1	0	*Set*



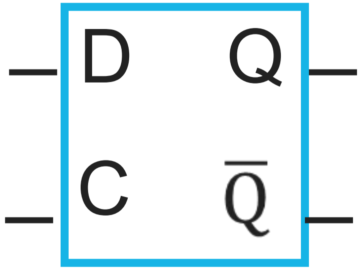
Round 2: D Latch (1)



- Inverter prevents SR Latch from entering 1,1 state

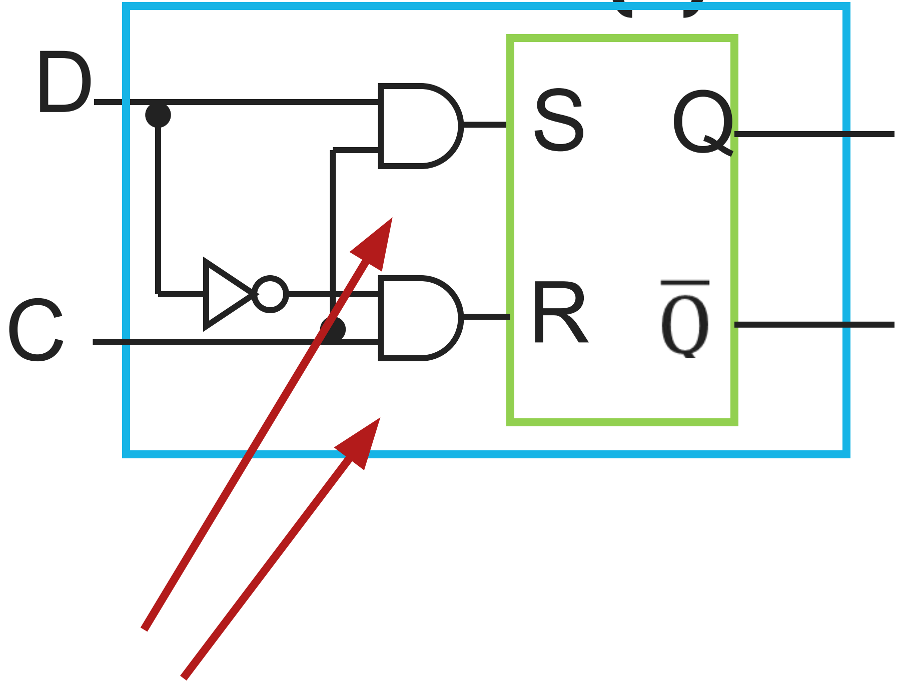
AND gate **forces a 0** when C = 0.

Remember: 0 ∧ a = 0



	D	Q	¬Q	
	0	0	1	*Reset*
	1	1	0	*Set*

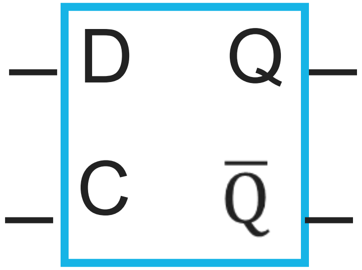
Round 2: D Latch (1)



- Inverter prevents SR Latch from entering 1,1 state

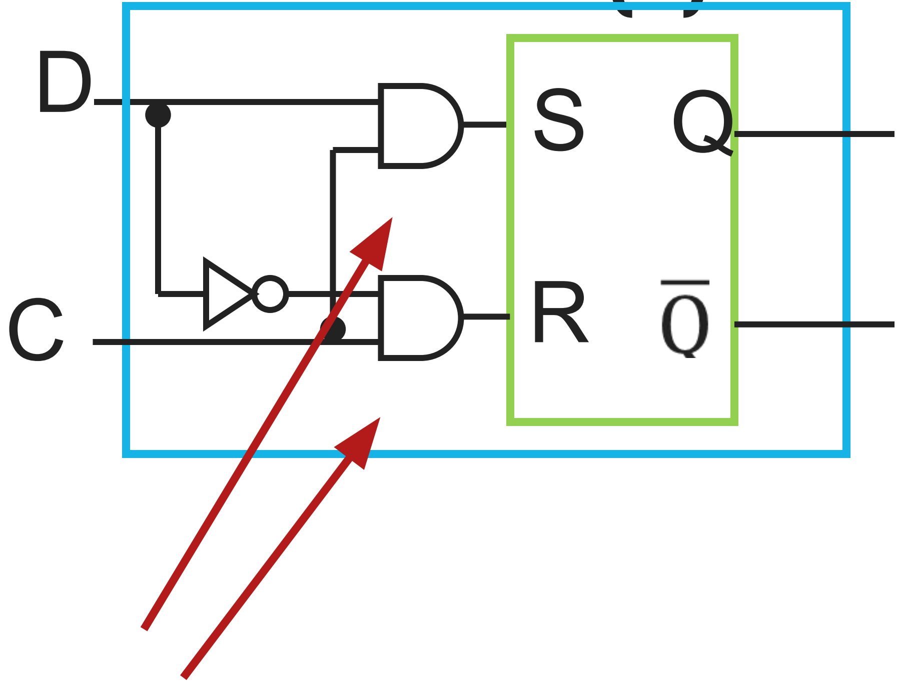
AND gate **forces a 0** when C = 0.

Remember: 0 ∧ a = 0



C	D	Q	¬Q	
0	0	Q_{t-1}	$¬Q_{t-1}$	*Hold*
0	1	Q_{t-1}	$¬Q_{t-1}$	*Hold*

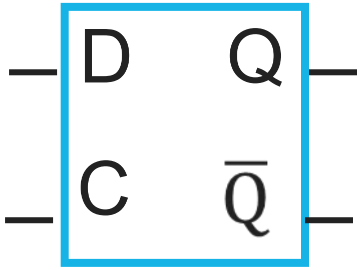
Round 2: D Latch (1)



- Inverter prevents SR Latch from entering 1,1 state

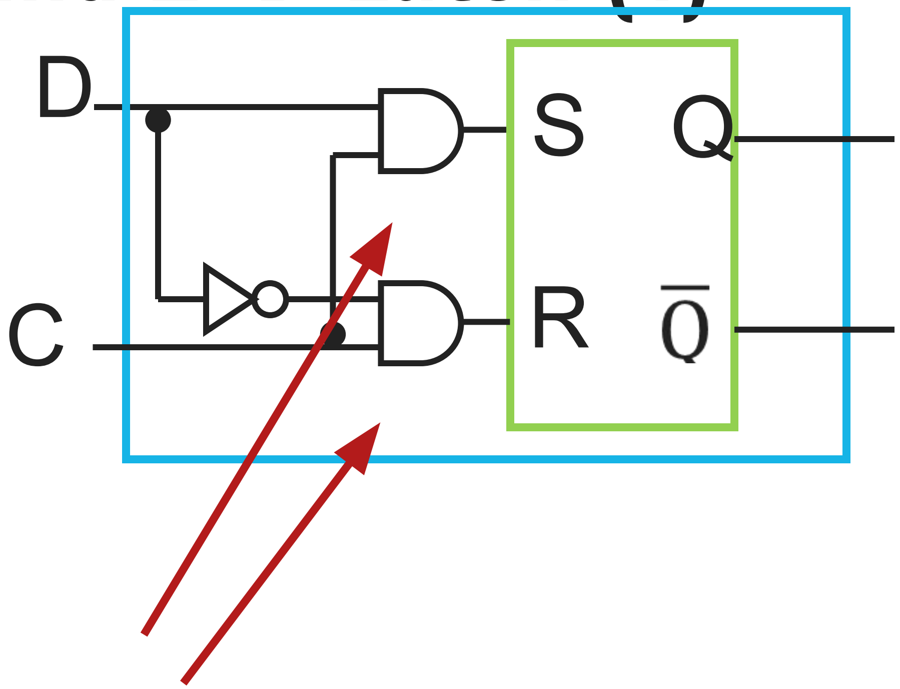
AND gate **lets D pass** when C = 1.

Remember: 1 ∧ a = a



C	D	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	*Hold*
0	1	Q_{t-1}	$\neg Q_{t-1}$	*Hold*

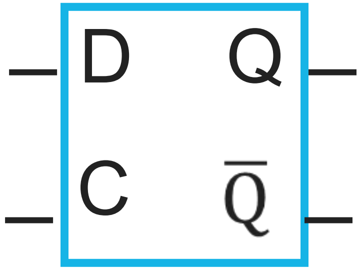
Round 2: D Latch (1)



- Inverter prevents SR Latch from entering 1,1 state

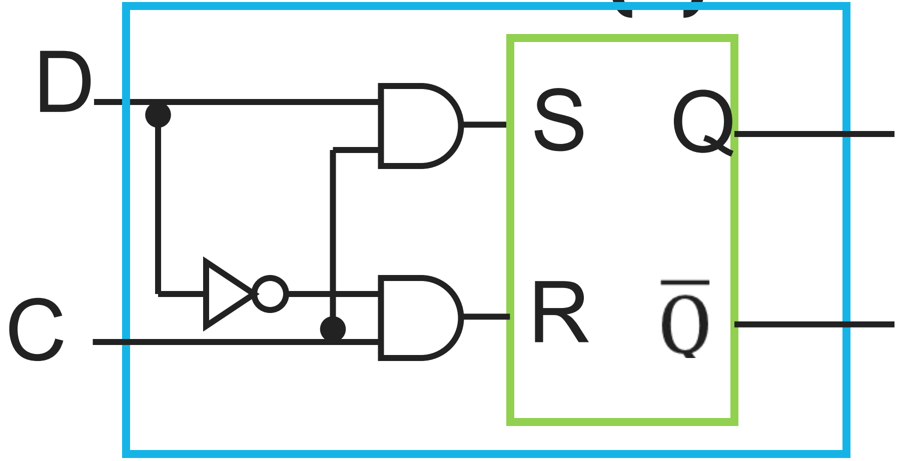
AND gate **lets D pass** when C = 1.

Remember: 1 ∧ a = a

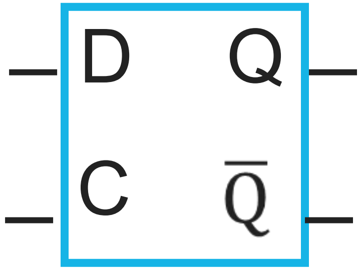


C	D	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	*Hold*
0	1	Q_{t-1}	$\neg Q_{t-1}$	*Hold*
1	0	0	1	*Reset*
1	1	1	0	*Set*

Round 2: D Latch (1)

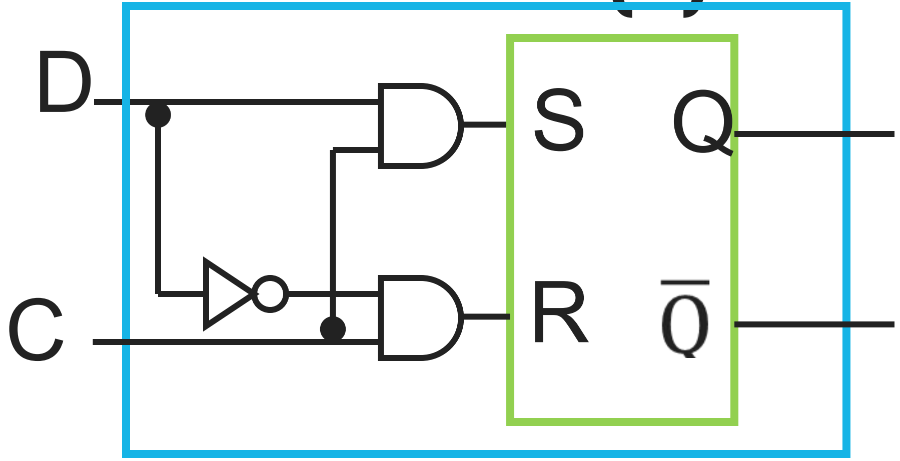


- <u>Level sensitive</u>
- Inverter prevents SR Latch from entering 1,1 state



C	D	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	*Hold*
0	1	Q_{t-1}	$\neg Q_{t-1}$	*Hold*
1	0	0	1	*Reset*
1	1	1	0	*Set*

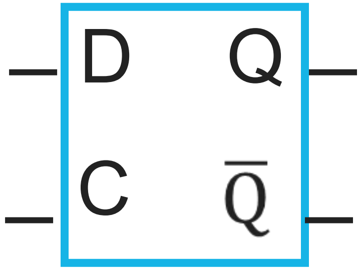
Round 2: D Latch (1)



- <u>Level sensitive</u>
- Inverter prevents SR Latch from entering 1,1 state

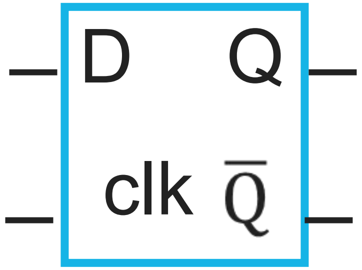
C = 1, D Latch *transparent*:
set/reset (according to D)

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



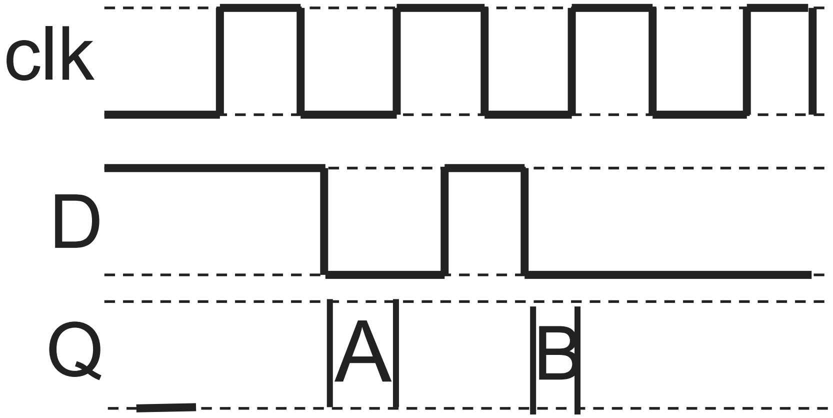
C	D	Q	¬Q	
0	0	Q_{t-1}	$\neg Q_{t-1}$	*Hold*
0	1	Q_{t-1}	$\neg Q_{t-1}$	*Hold*
1	0	0	1	*Reset*
1	1	1	0	*Set*

PollEV Question



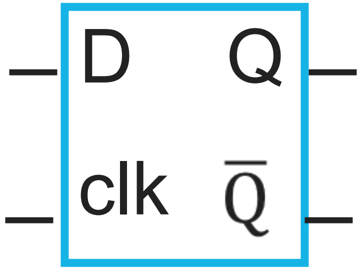
What is the value of Q at A & B?

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



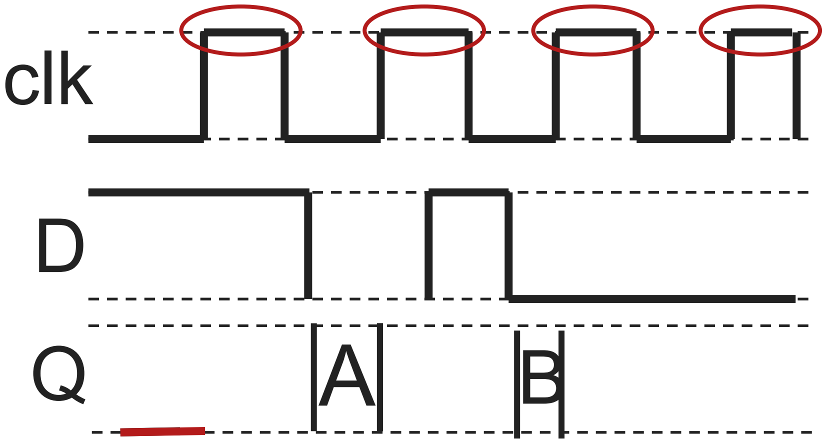
clk	D	Q	
0	0	Q	
0	1	Q	
1	0	0	1
1	1	1	0

PollEV Question



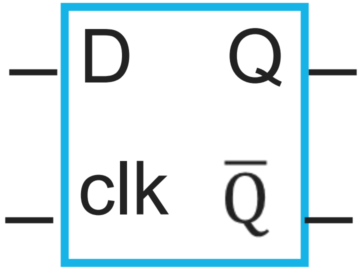
What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
[✓ d) A = 1, B = 1]



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

PollEV Question



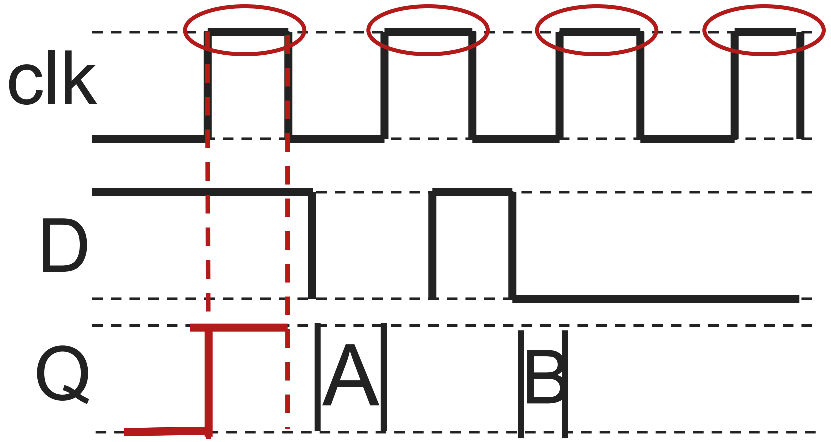
What is the value of Q at A & B?

a) A = 0, B = 0

b) A = 0, B = 1

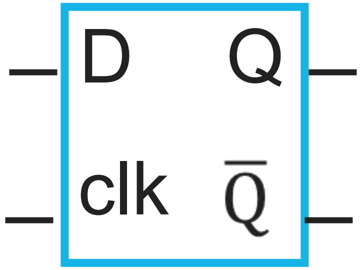
c) A = 1, B = 0

[✓ d) A = 1, B = 1]



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

PollEV Question



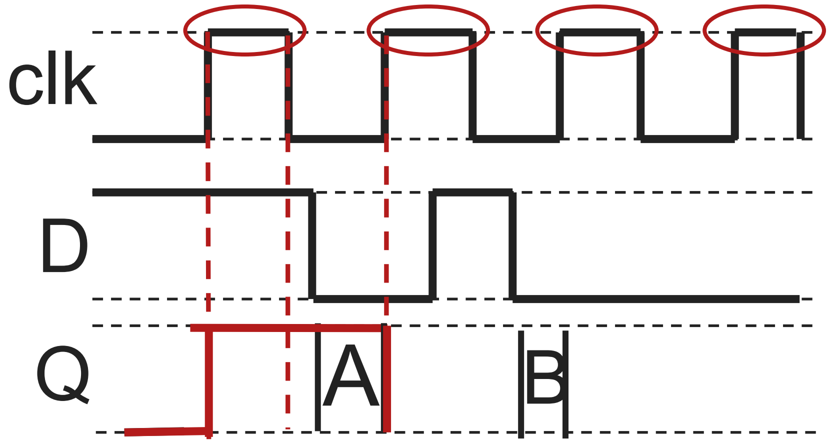
What is the value of Q at A & B?

a) A = 0, B = 0

b) A = 0, B = 1

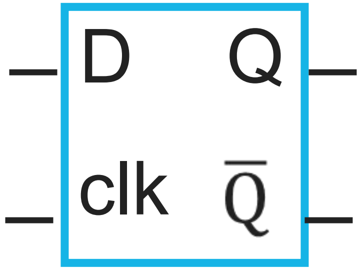
c) A = 1, B = 0

[✓ d) A = 1, B = 1]



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

PollEV Question



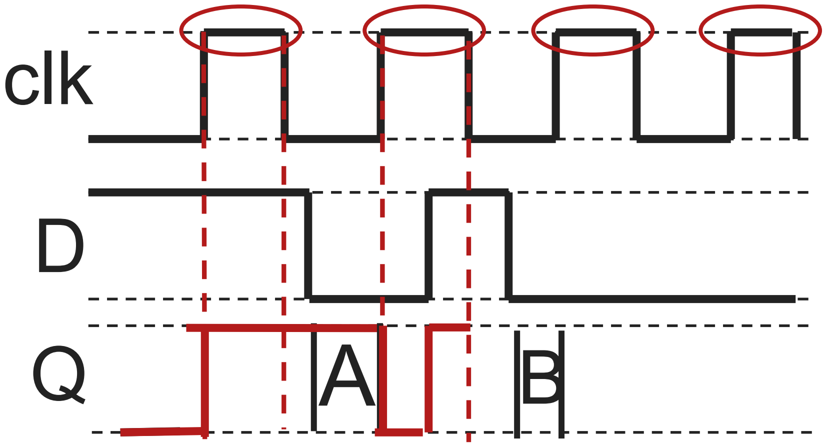
What is the value of Q at A & B?

a) A = 0, B = 0

b) A = 0, B = 1

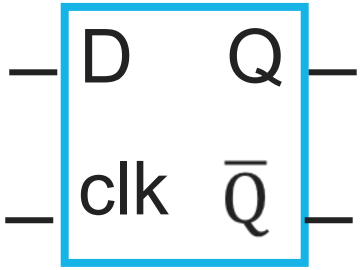
c) A = 1, B = 0

[✓ d) A = 1, B = 1]



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

PollEV Question



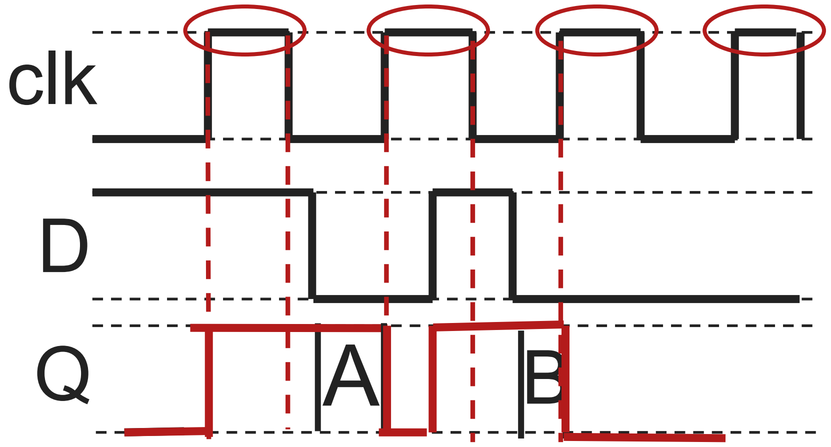
What is the value of Q at A & B?

a) A = 0, B = 0

b) A = 0, B = 1

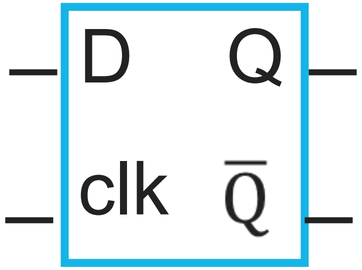
c) A = 1, B = 0

[✓ d) A = 1, B = 1]



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

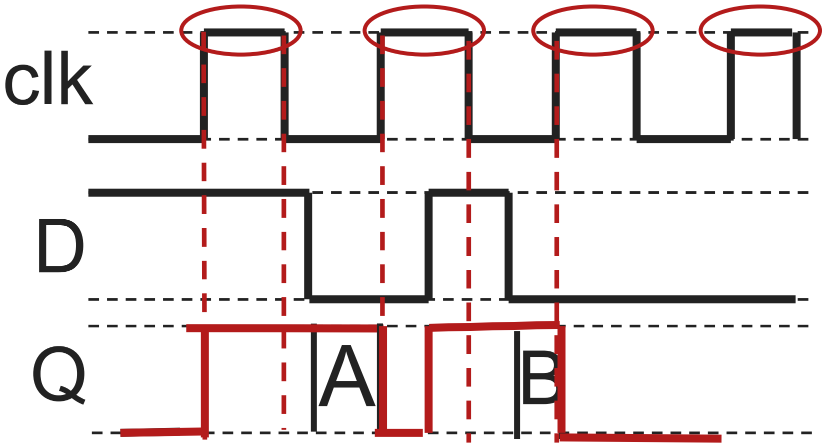
PollEV Question



Level Sensitive D Latch

Clock high:
  set/reset (according to D)

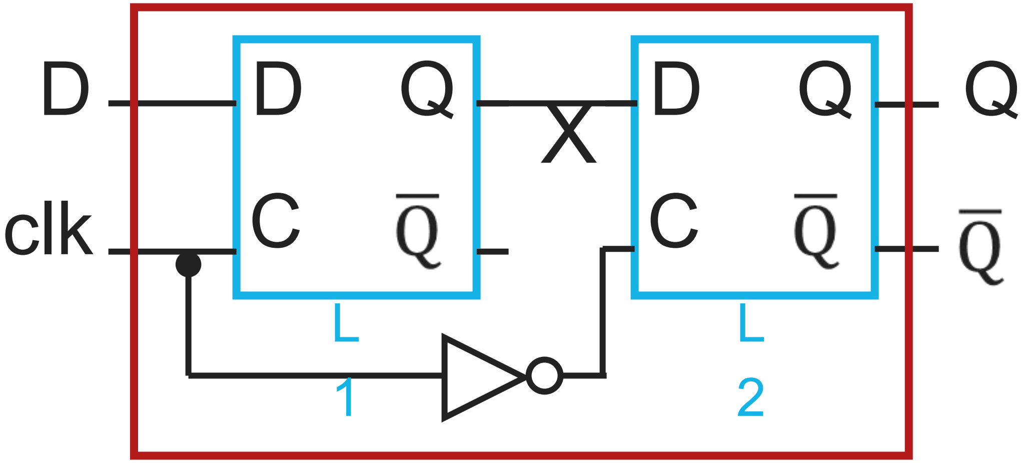
Clock low:
  keep state (ignore D)



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

Cornell Bowers CIS
Computer Science

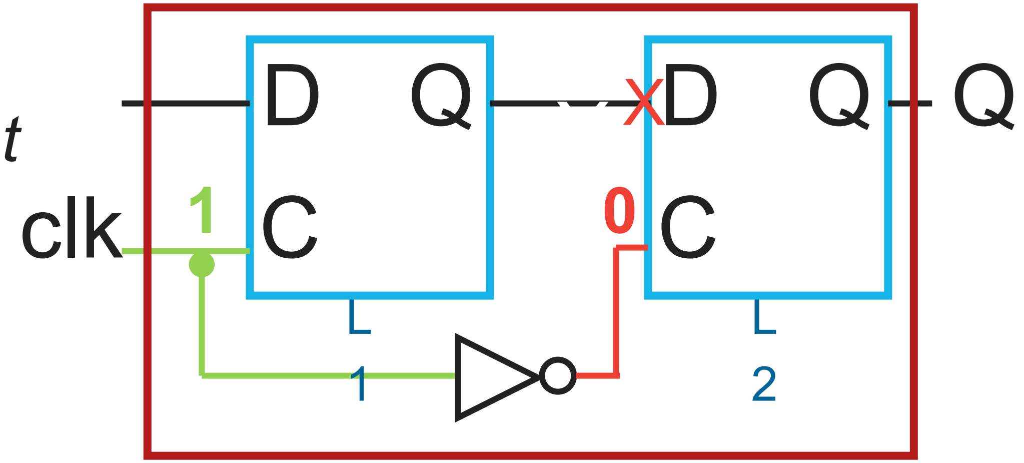
Round 3: D Flip-Flop



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

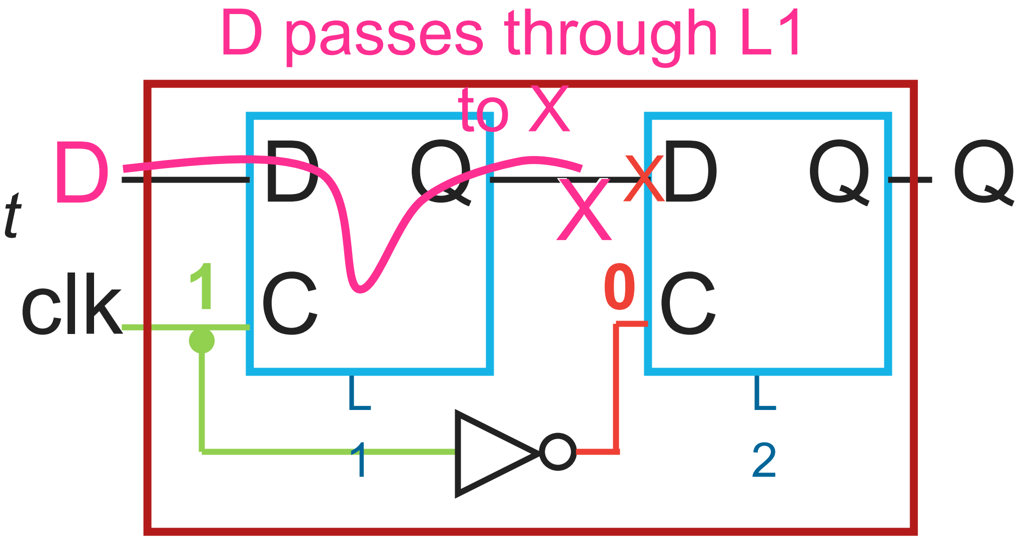
Round 3: D Flip-Flop

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



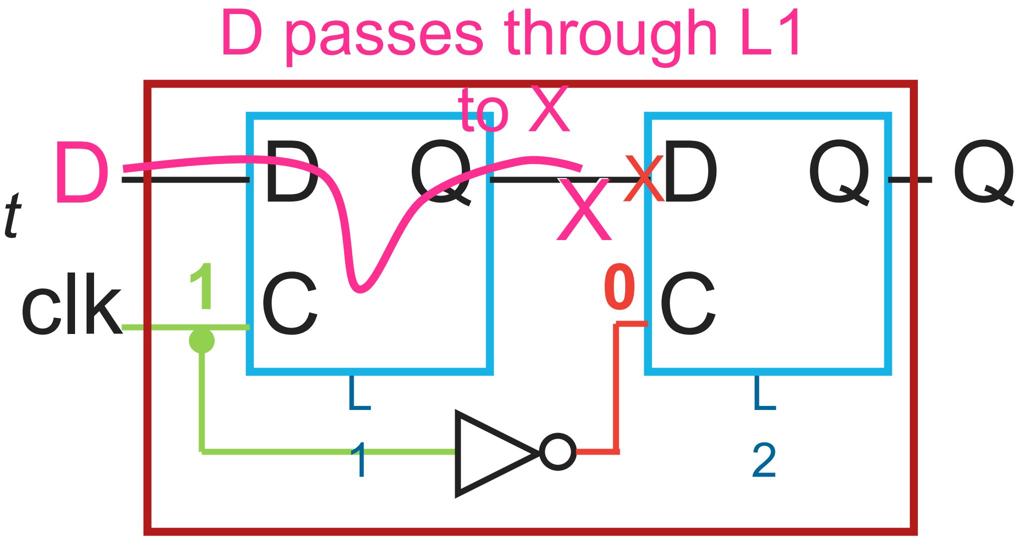
Round 3: D Flip-Flop

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



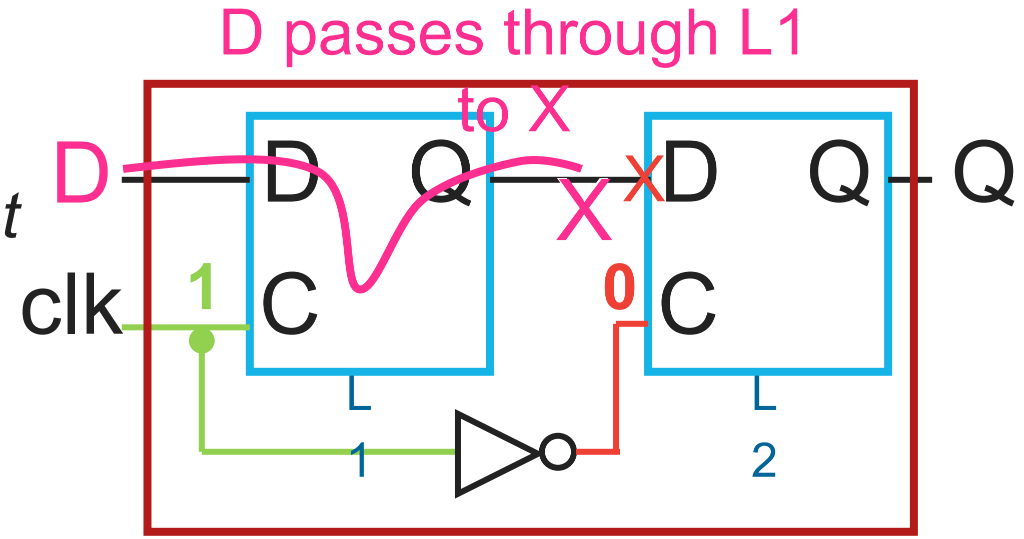
Round 3: D Flip-Flop

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

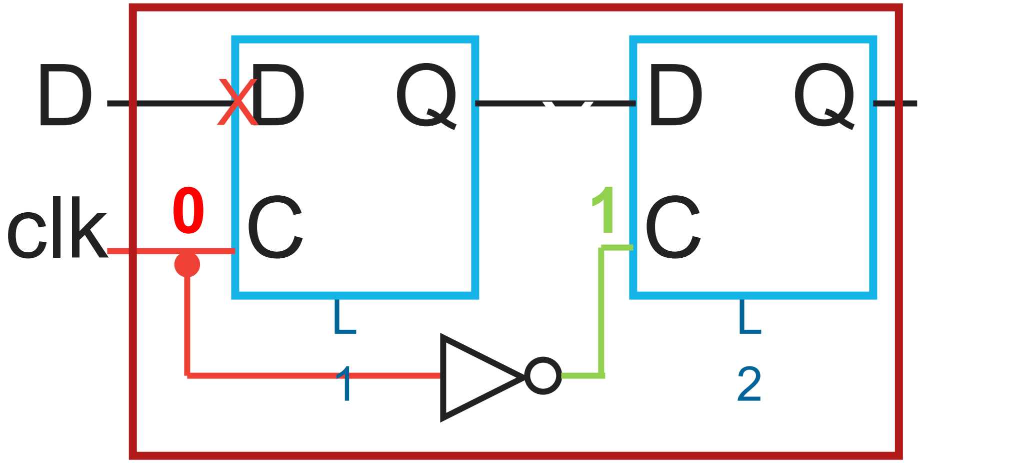


Round 3: D Flip-Flop

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



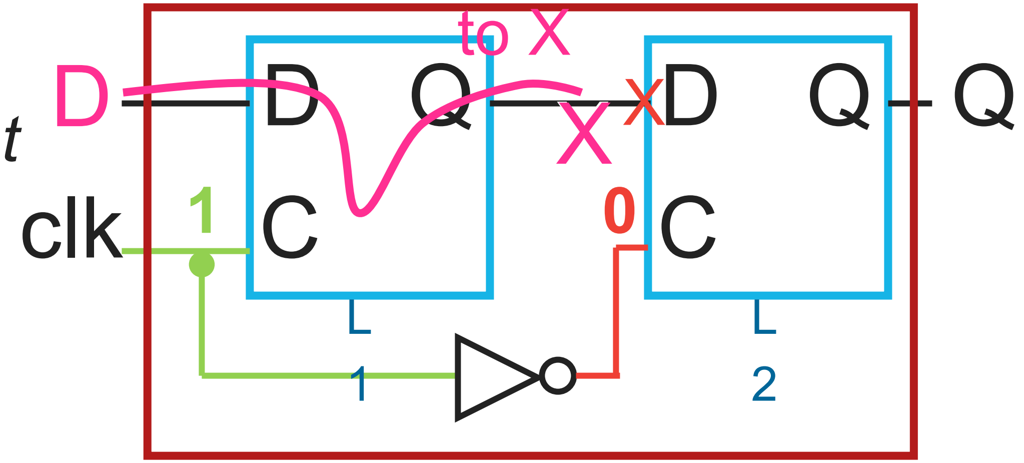
Clock = 0: L1 *opaque*
L2 *transparent*



Round 3: D Flip-Flop

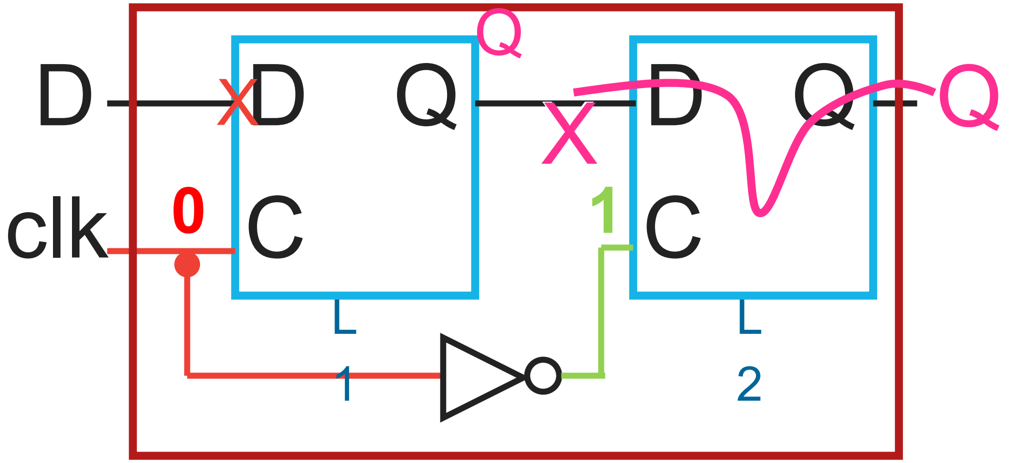
Clock = 1: L1 *transparent*
L2 *opaque*

D passes through L1 to X



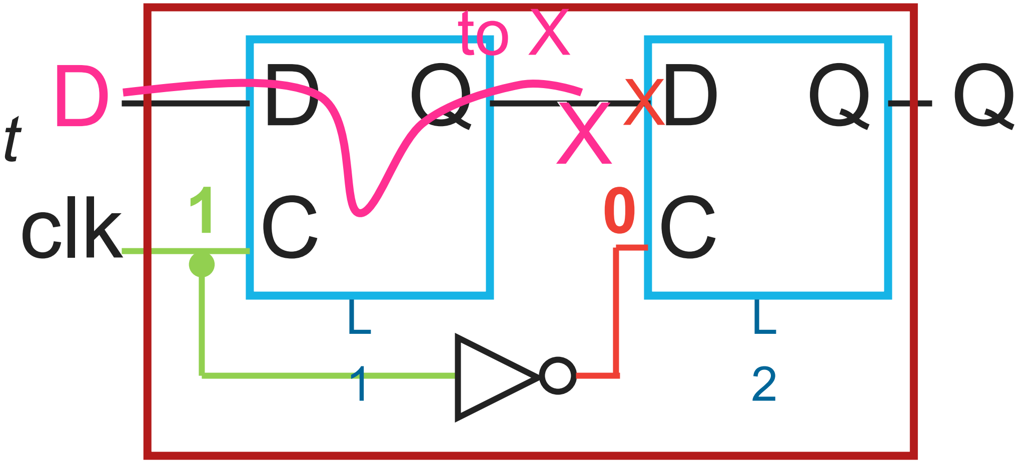
Clock = 0: L1 *opaque*
L2 *transparent*

X passes through L2 to Q



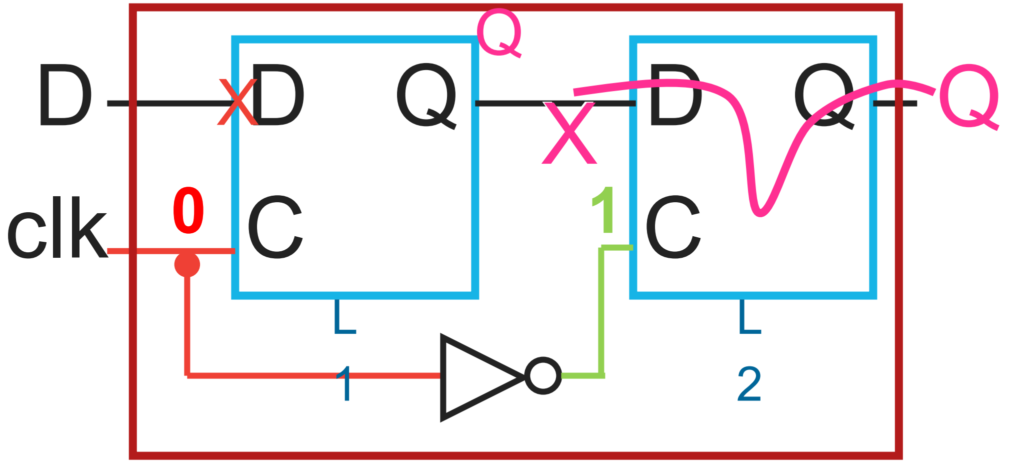
Round 3: D Flip-Flop

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



D passes through L1 to X

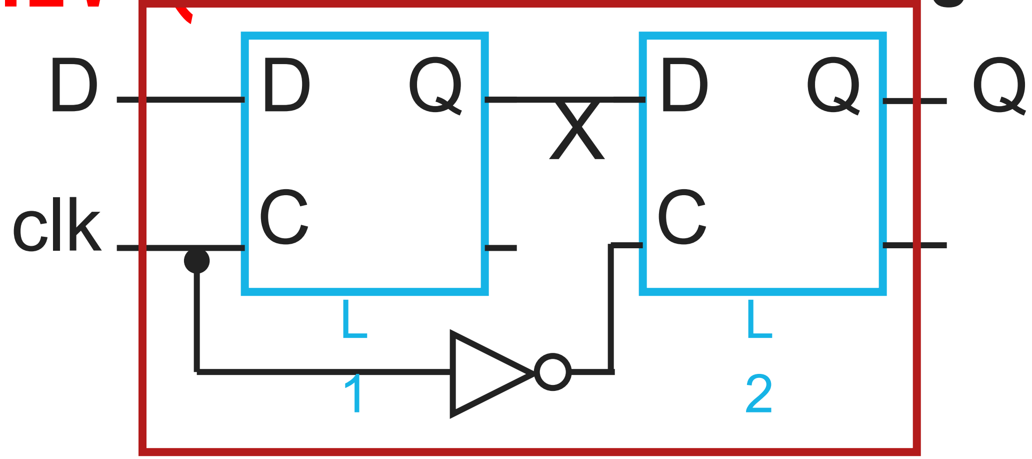
Clock = 0: L1 *opaque*
L2 *transparent*



X passes through L2 to Q

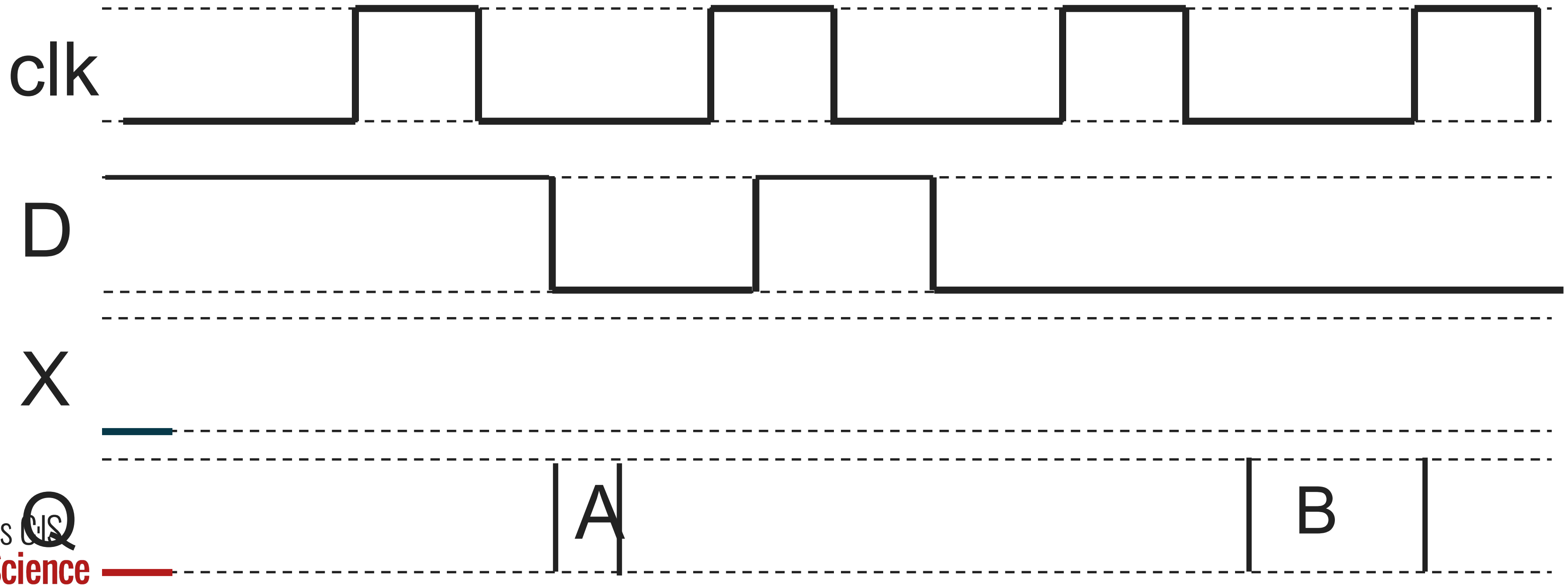
Sample data at the **falling *CLK* edge** (1□0)

PollEV Question – start here

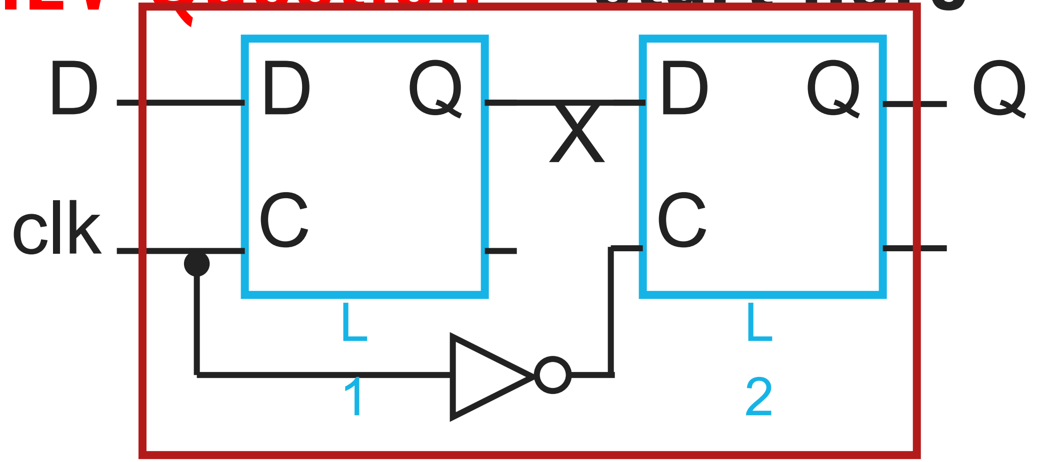


What is the value of Q at A & B?

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



PollEV Question – start here



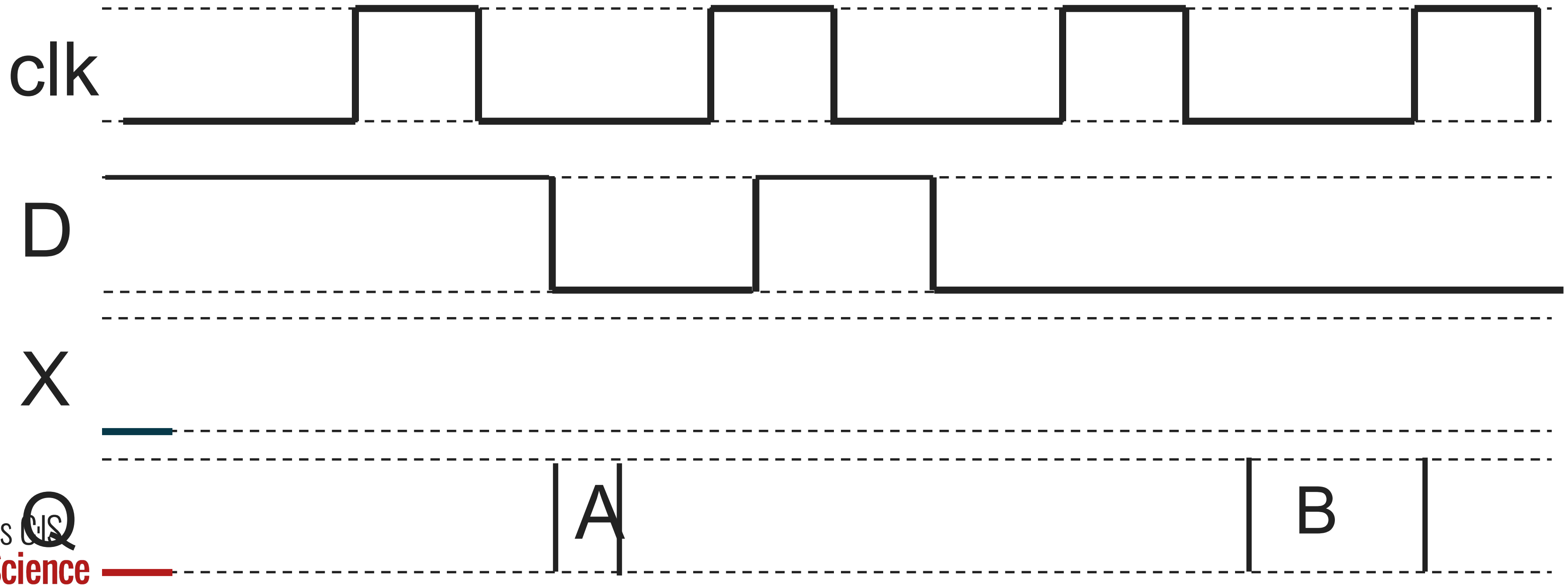
What is the value of Q at A & B?

a) A = 0, B = 0

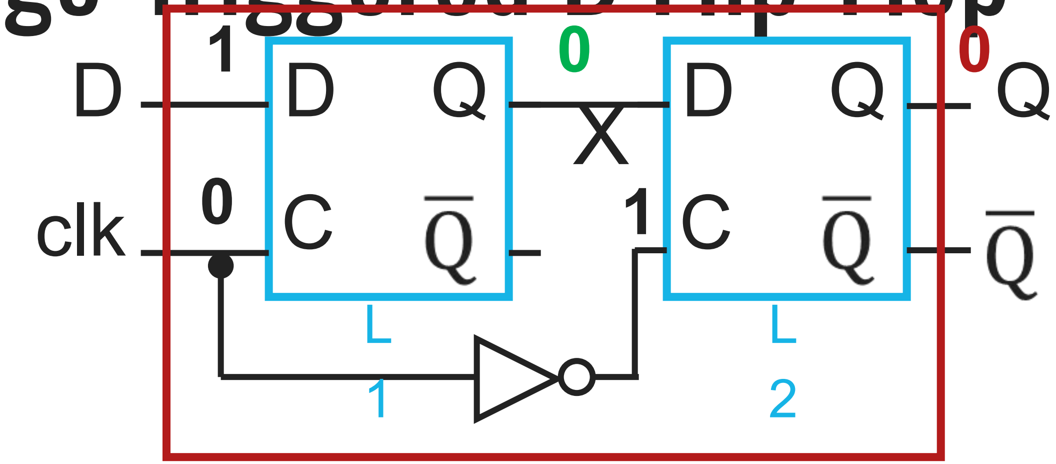
b) A = 0, B = 1

[✓ c) A = 1, B = 0]

d) A = 1, B = 1

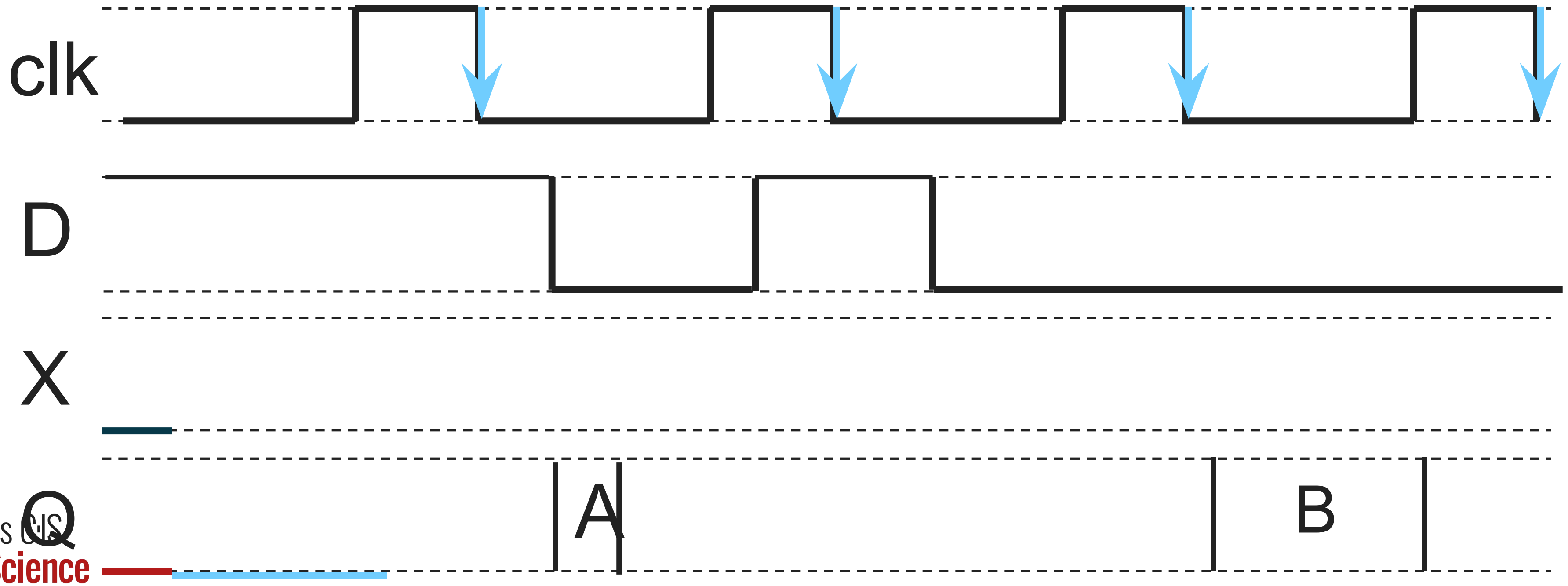


Edge-Triggered D Flip-Flop

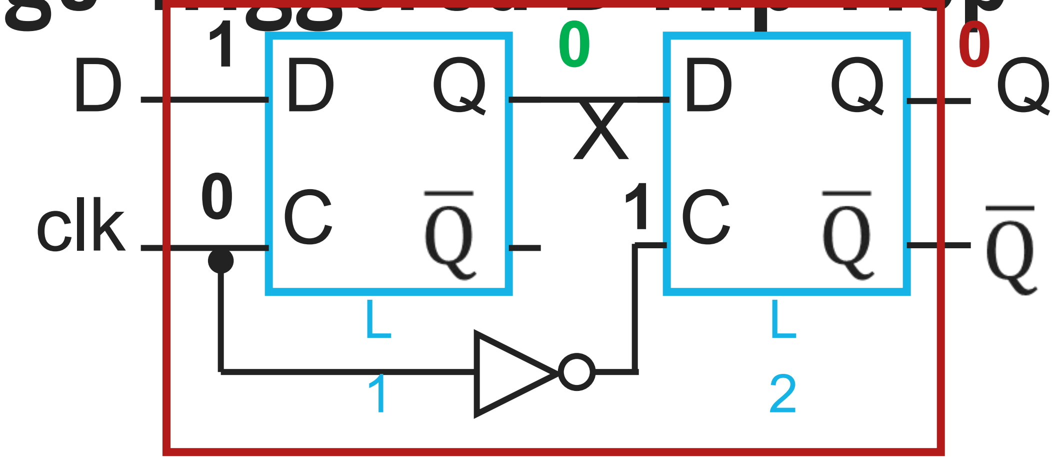


D Flip-Flop

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

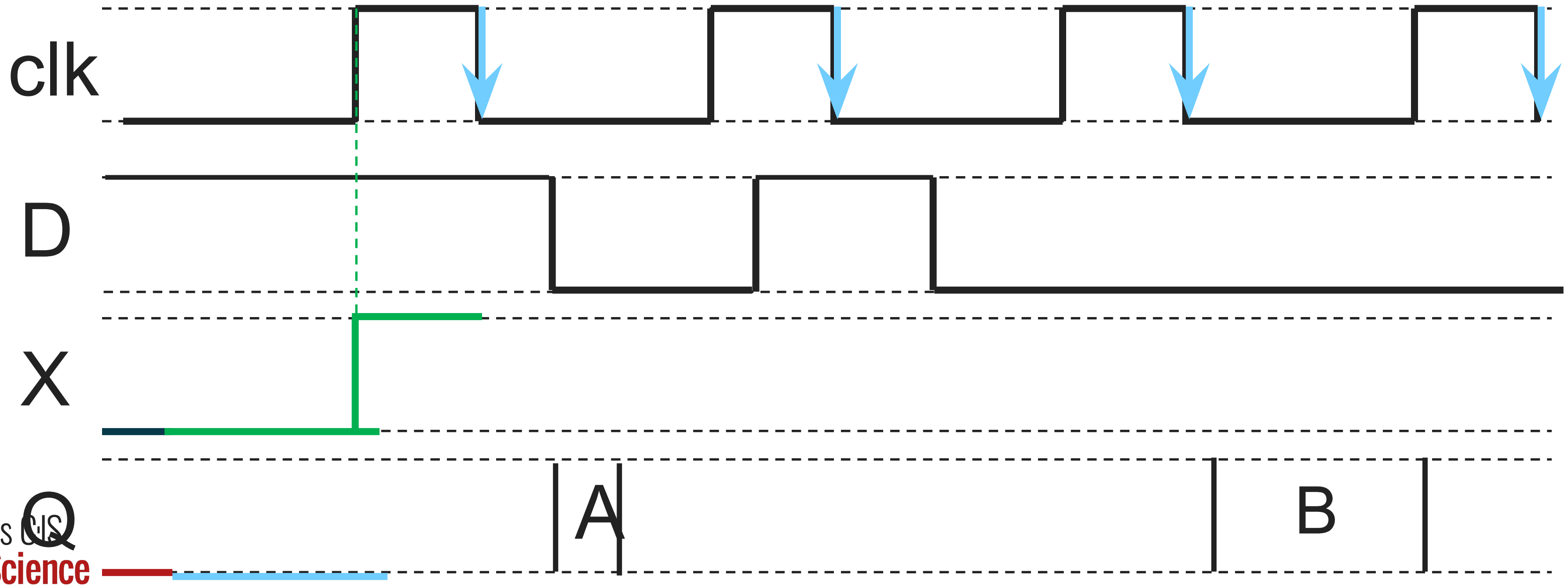


Edge-Triggered D Flip-Flop

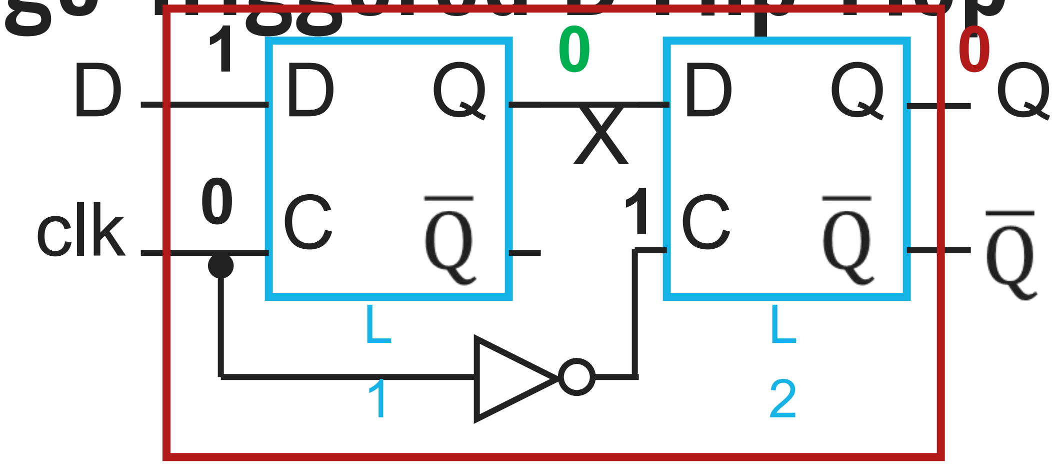


D Flip-Flop

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

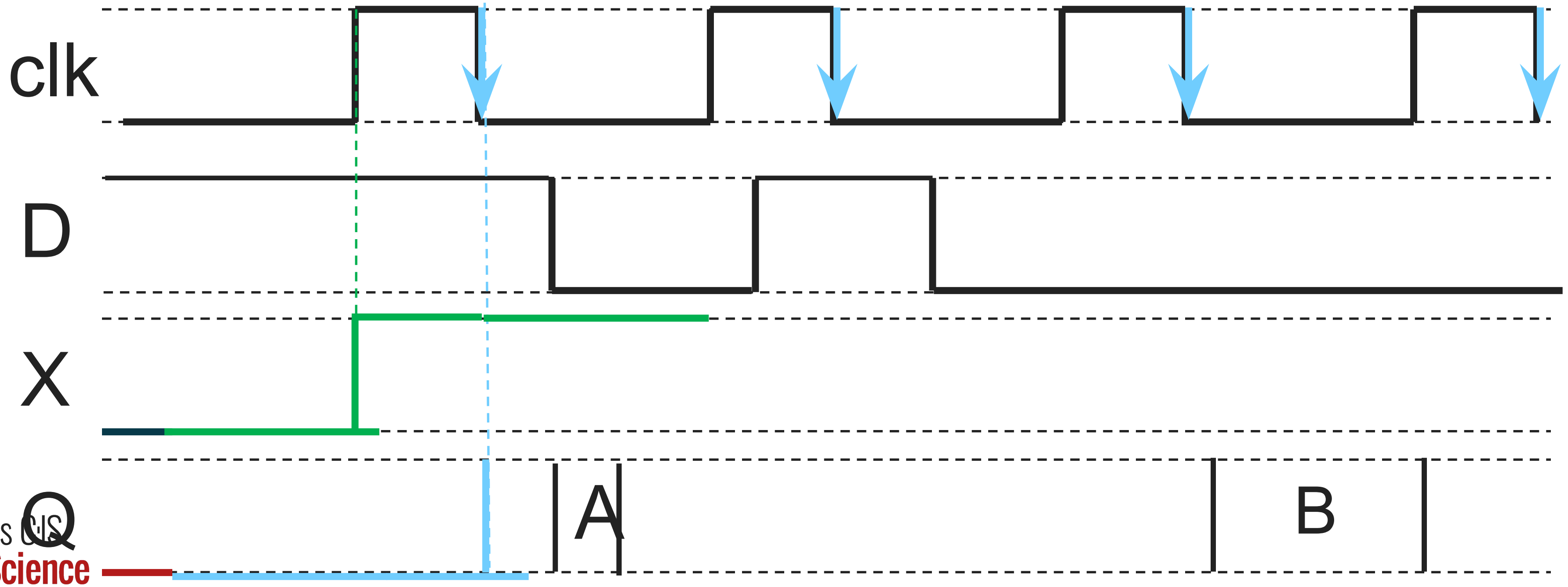


Edge-Triggered D Flip-Flop

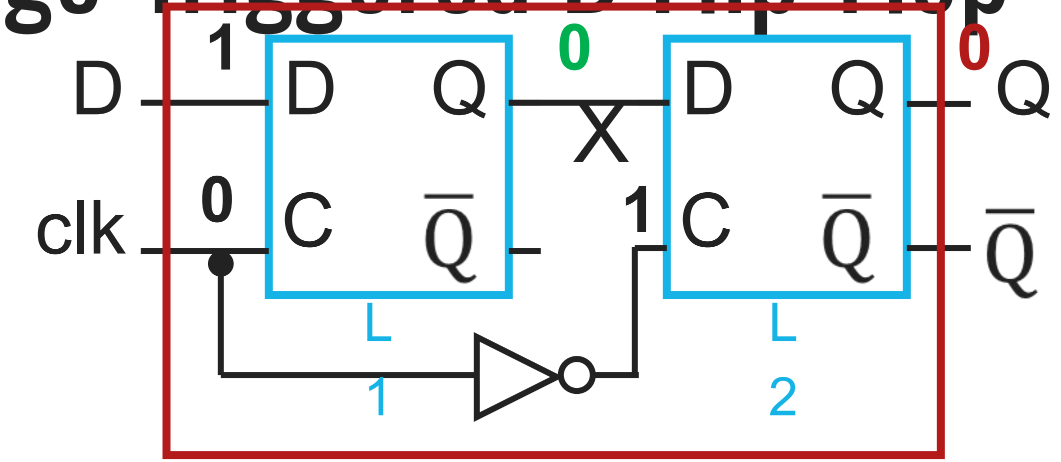


D Flip-Flop

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

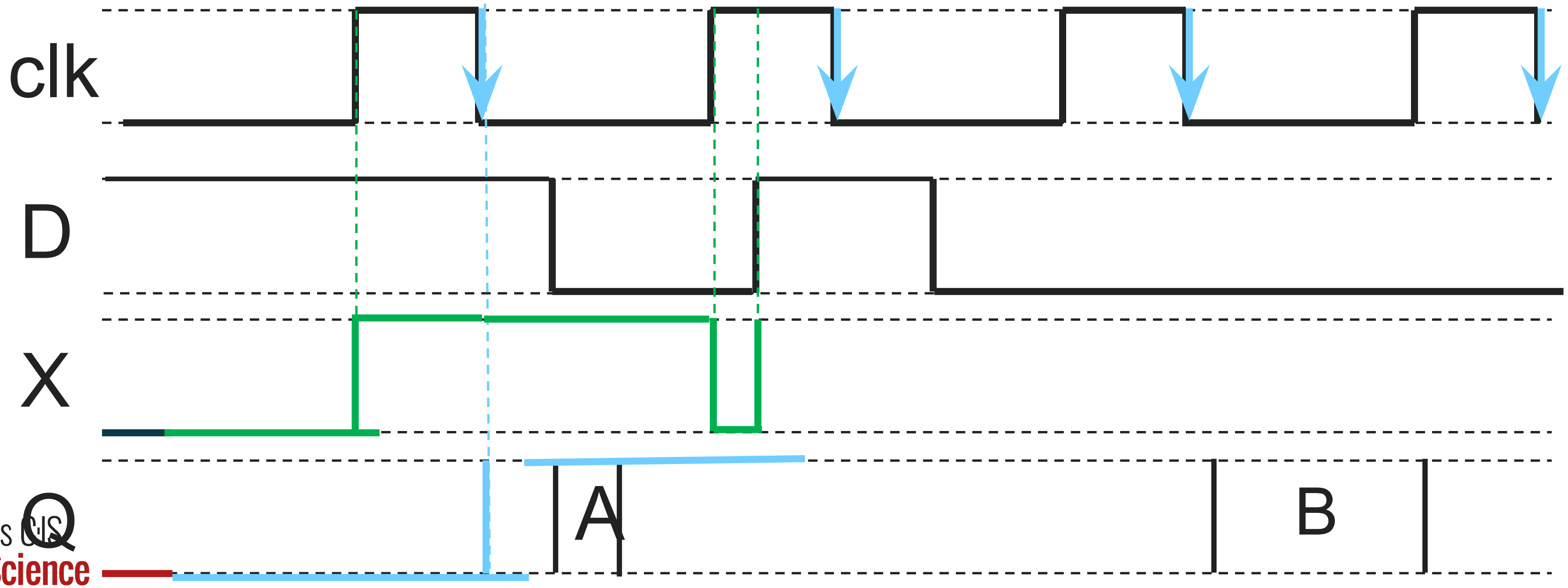


Edge-Triggered D Flip-Flop

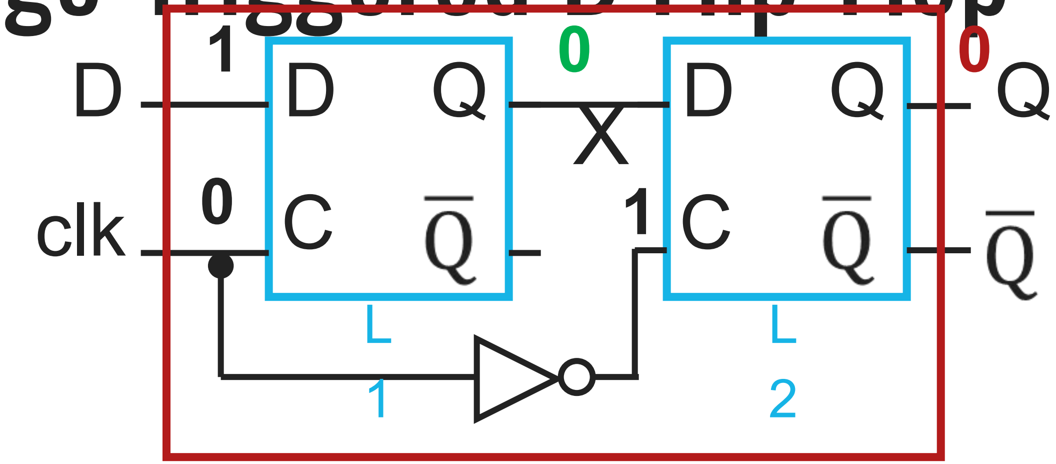


D Flip-Flop

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

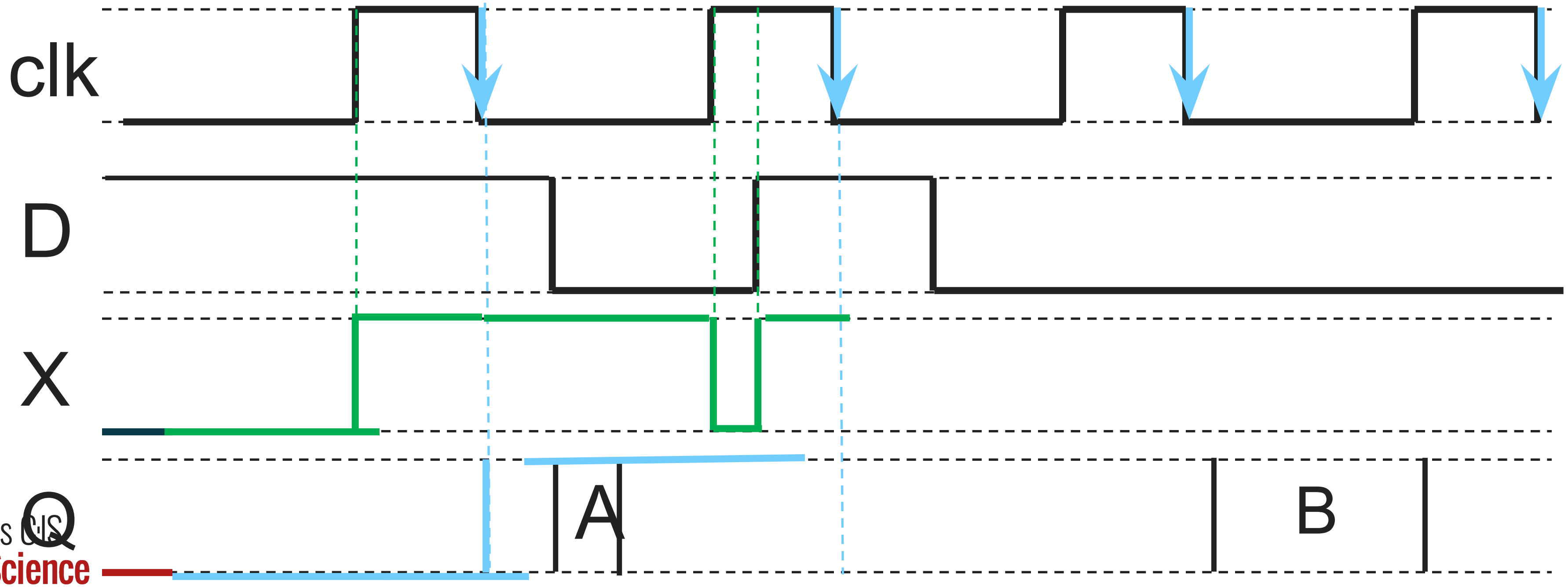


Edge-Triggered D Flip-Flop



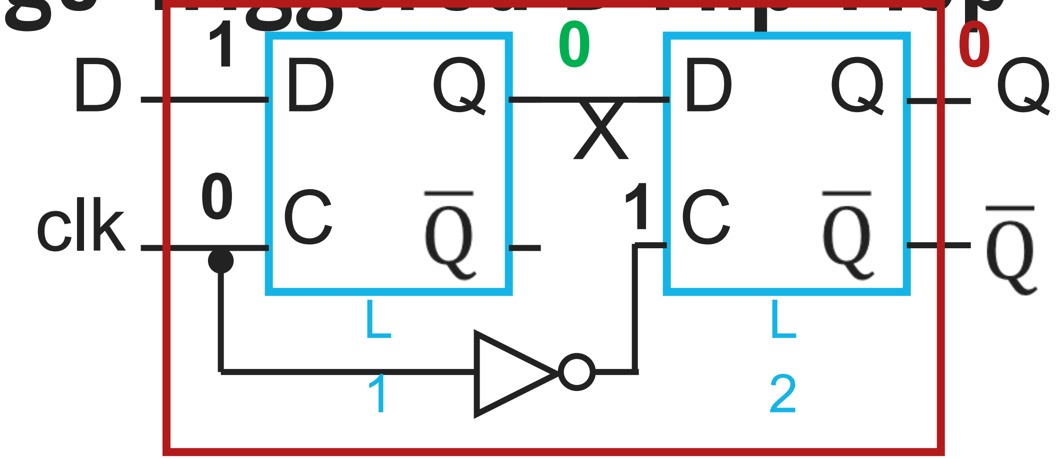
D Flip-Flop

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



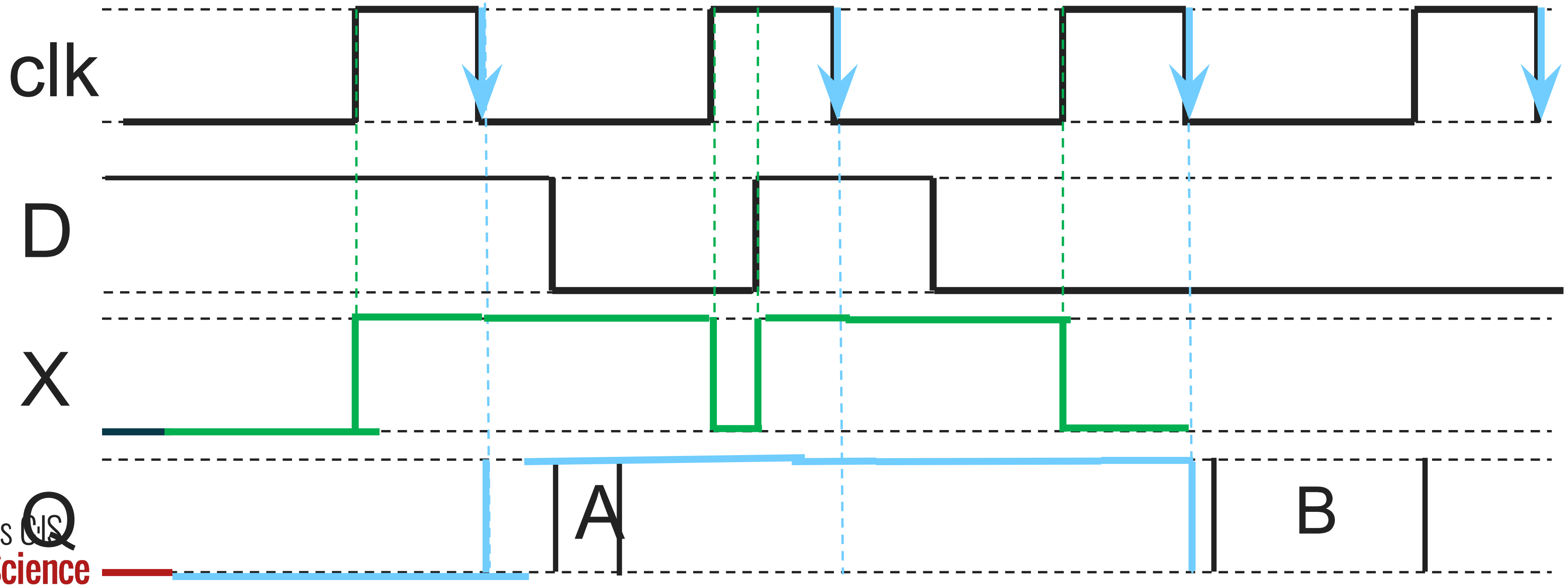
Cornell Bowers CIS
Computer Science

Edge-Triggered D Flip-Flop

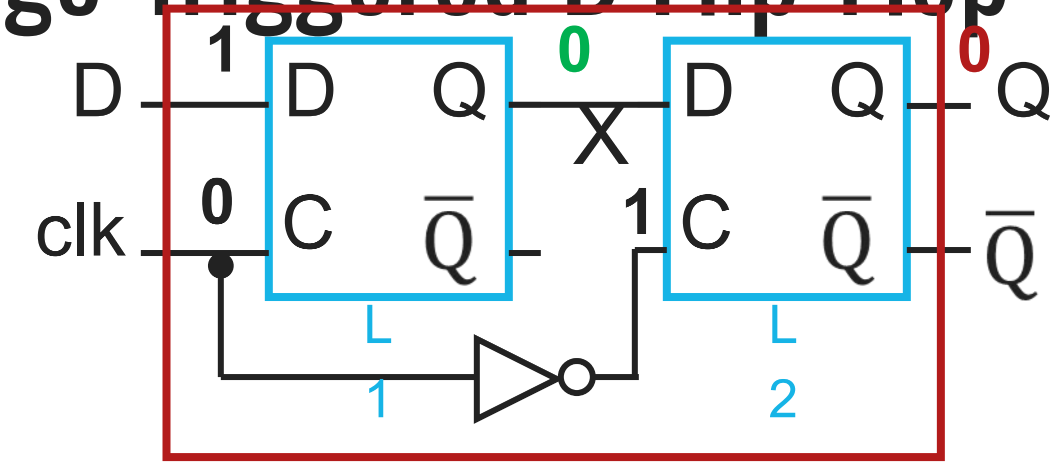


D Flip-Flop

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

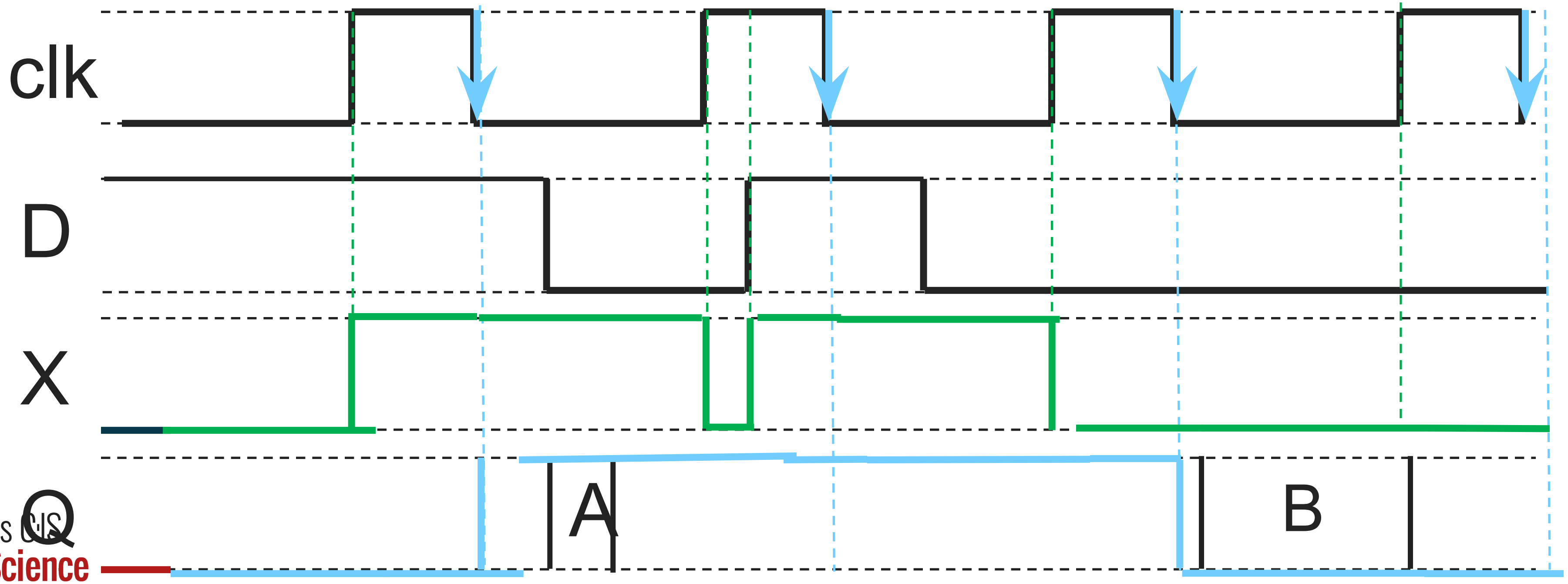


Edge-Triggered D Flip-Flop

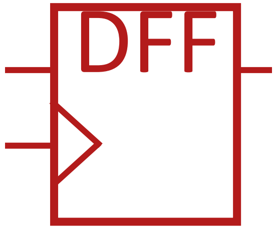


D Flip-Flop

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

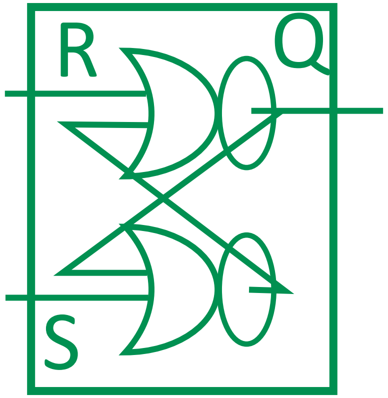


Building a D Flip Flop (DFF)

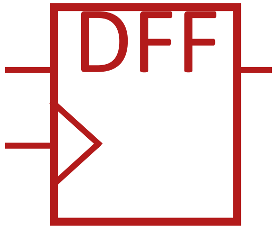


Step 1: Create an SR Latch

Set	Reset	Q
0	0	Q
0	1	0
1	0	1
1	1	?



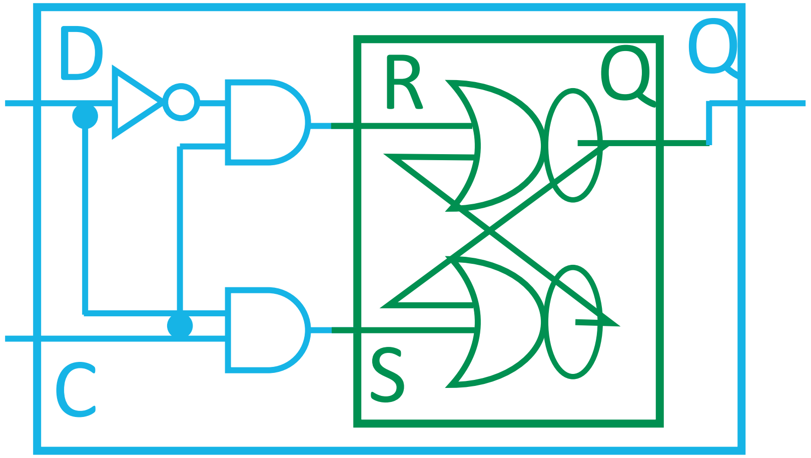
Building a D Flip Flop (DFF)



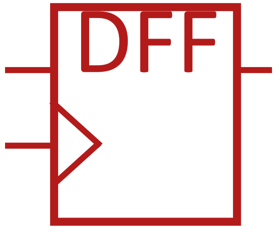
Step 1: Create an SR Latch

Step 2: Create a D Latch

Clk	Data	Q
0	0	Q
0	1	Q
1	0	0
1	1	1



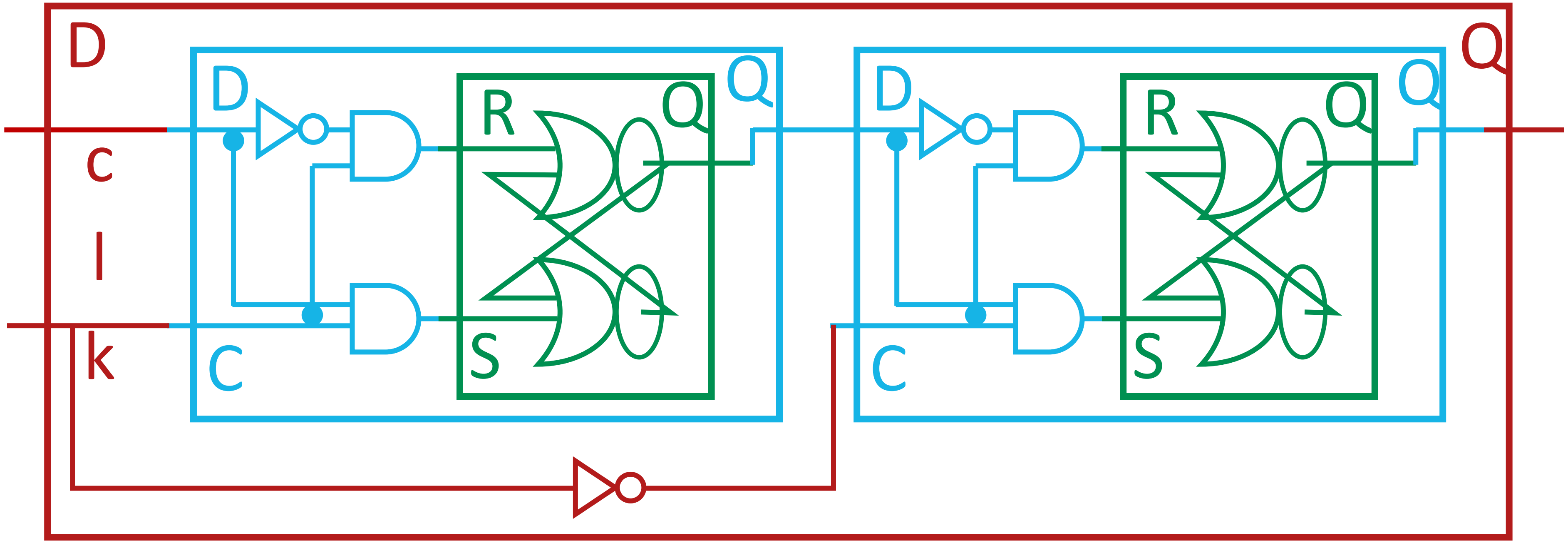
Building a D Flip Flop (DFF)



Step 1: Create an SR Latch

Step 2: Create a D Latch

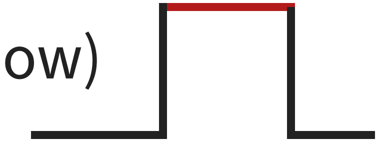
Step 3: Duplicate the D Latch, chain together



Clock Disciplines

Level sensitive

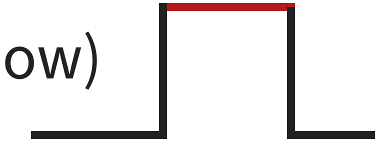
- State changes when clock is high (or low)



Clock Disciplines

Level sensitive

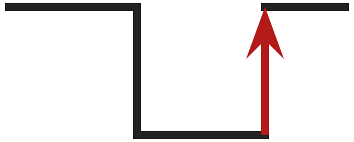
- State changes when clock is high (or low)



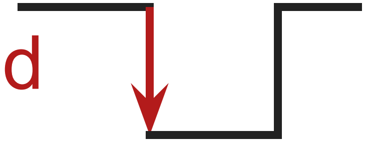
Edge triggered

- State changes at clock edge

positive edge-triggered



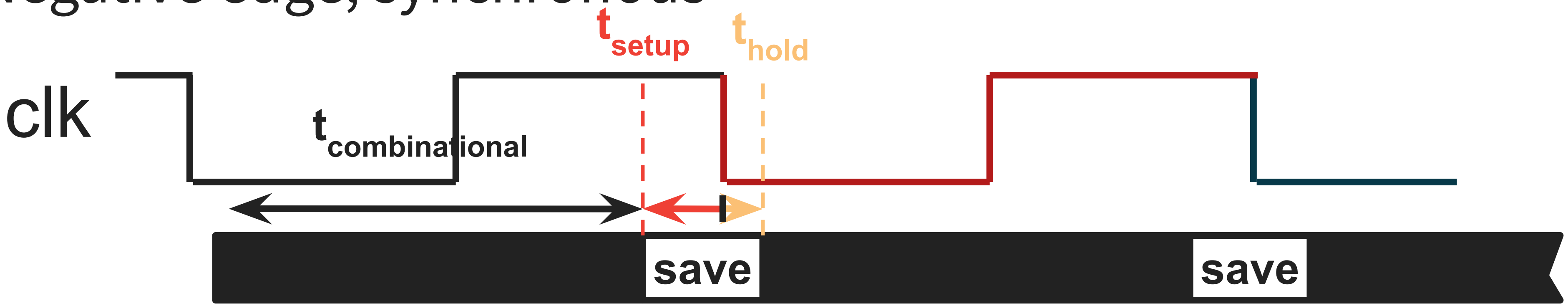
negative edge-triggered



Clock Methodology

Clock Methodology

- Negative edge, synchronous



Edge-Triggered □ signals must be stable near falling edge

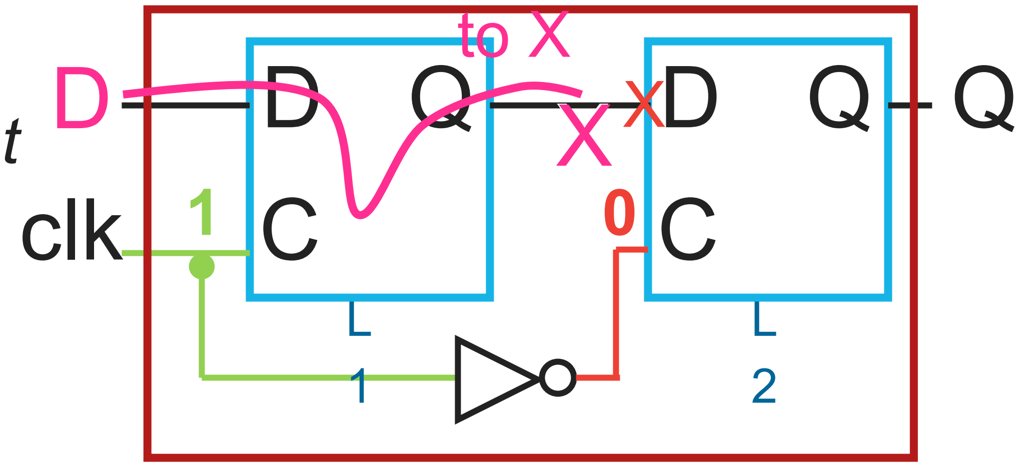
"near" = before and after

t_{setup} t_{hold}

Round 3: D Flip-Flop

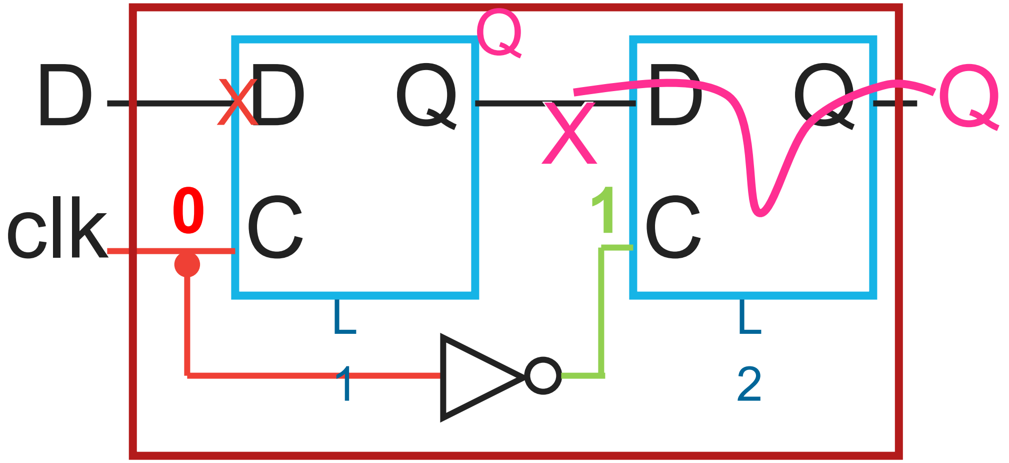
Clock = 1: L1 *transparent*
L2 *opaque*

D passes through L1 to X



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

X passes through L2 to Q

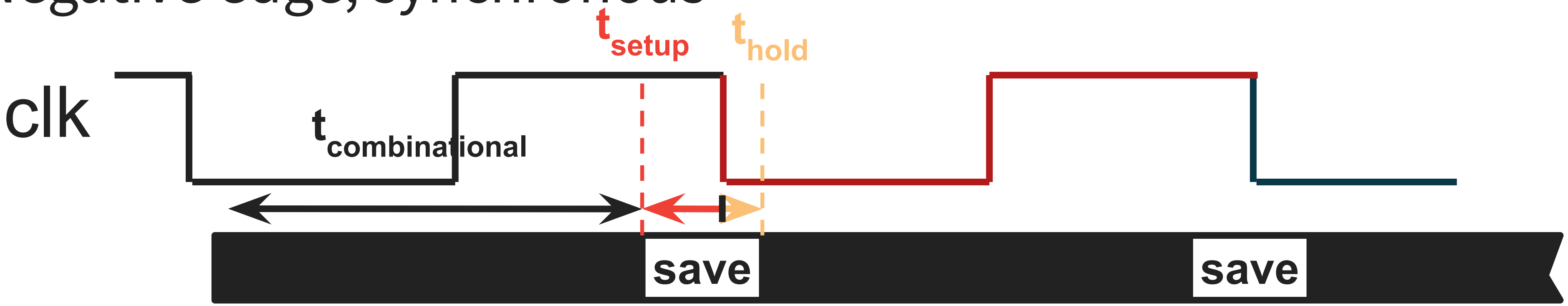


Sample data at the **falling** ***CLK*** **edge** (1□0)

Clock Methodology

Clock Methodology

- Negative edge, synchronous



Edge-Triggered □ signals must be stable near falling edge

“near” = before and after

t_{setup} t_{hold}

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.

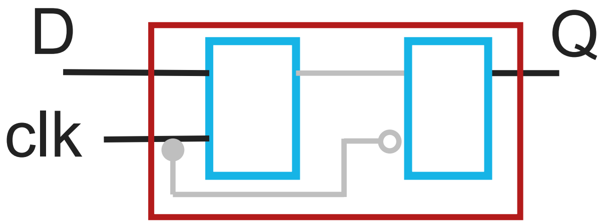
(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

An Edge-Triggered D Flip-Flip 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?

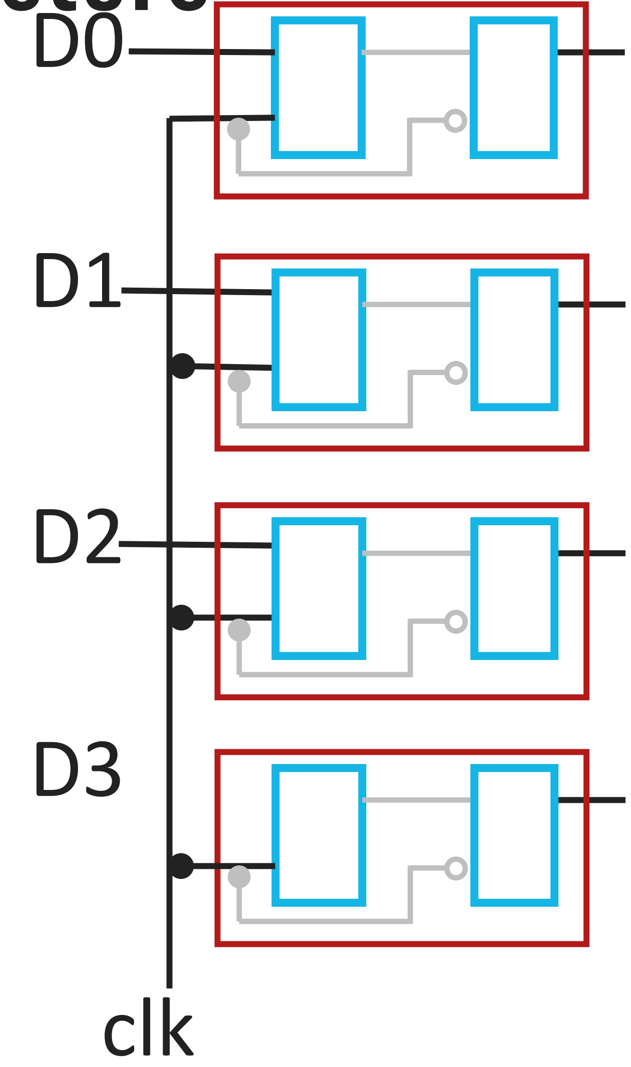
Registers



Register

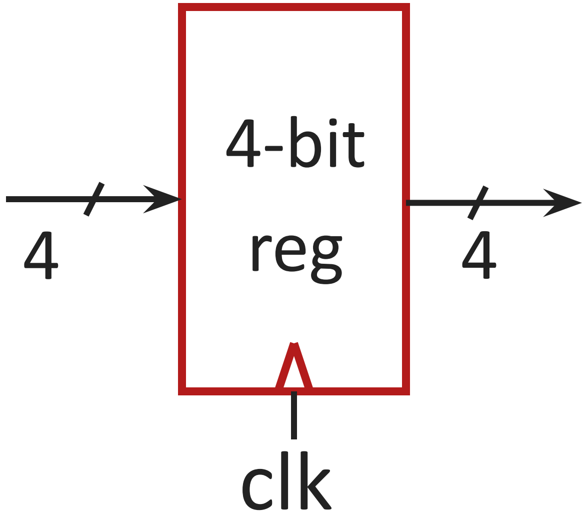
- D flip-flops in parallel

Registers

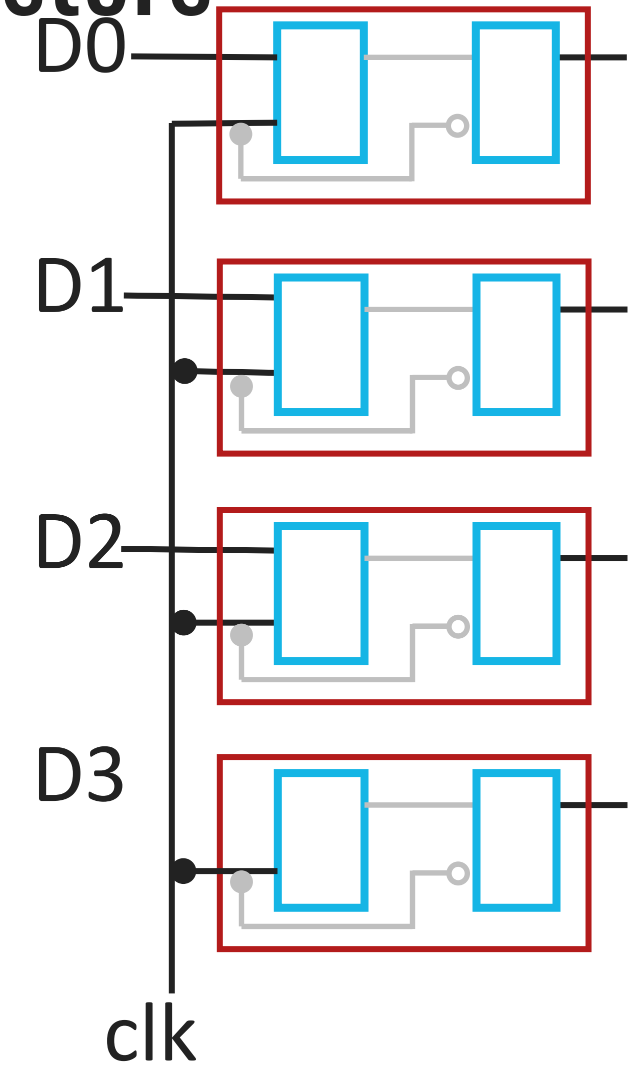


Register

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

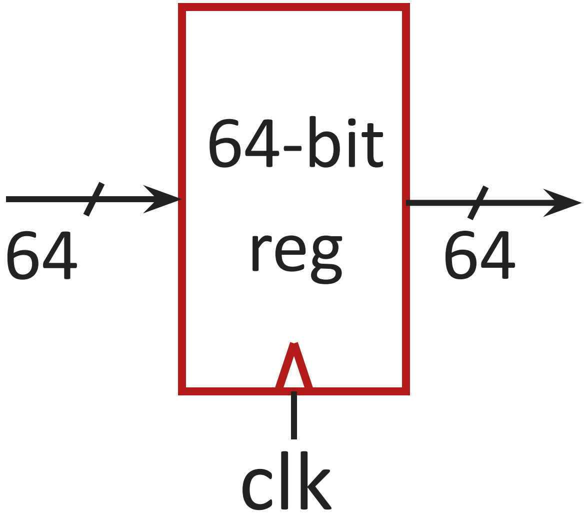


Registers



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-Flip 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.