State

CS 3410: Computer System Organization and Programming

[K. Bala, A. Bracy, E. Sirer, and H. Weatherspoon]

Big Picture: How to Design a Processor

C

```
int x = 10;
x = 2 * x + 15;
```

compiler

RISC-V assembly language

```
addi x5, x0, 10
muli x5, x5, 2
addi x5, x5, 15
```

x0 = 0

How does add work?

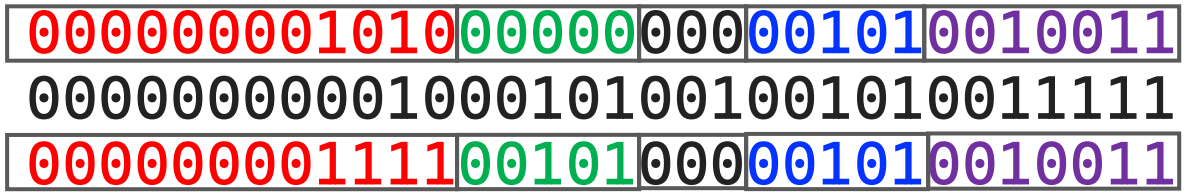
x5 = x0 + 10

x5 = x5 * 2

x5 = x5 + 15

assembler

RISC-V machine language



EVERYTHING IS A NUMBER!

Big Picture: How to Design a Processor

C

```
int x = 10;
x = 2 * x + 15;
```

compiler

RISC-V assembly language

```
addi x5, x0, 10
slli x5, x5, 1
addi x5, x5, 15
```

x0 = 0

How does add work?

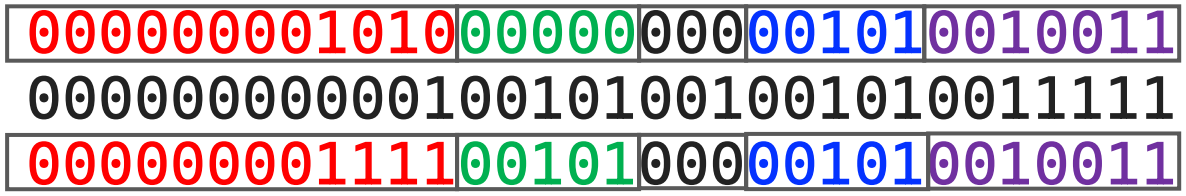
x5 = x0 + 10

x5 = x5 * 2

x5 = x5 + 15

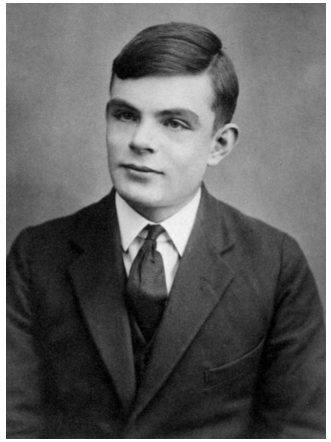
assembler

RISC-V machine language

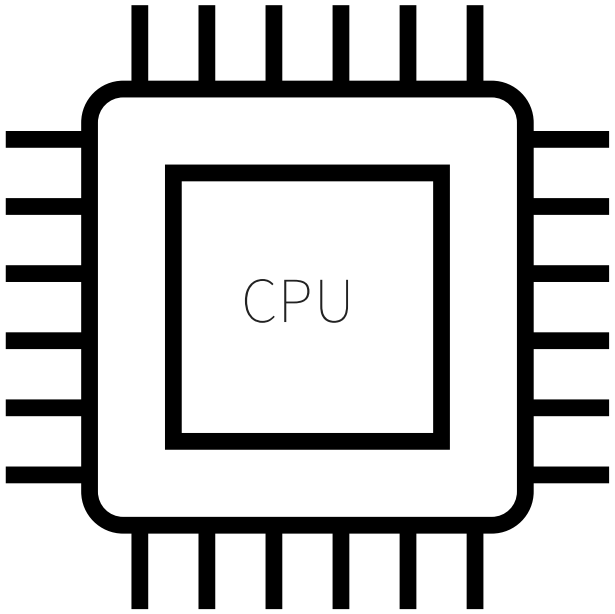


EVERYTHING IS A NUMBER!

Big Picture: How to Design a Processor

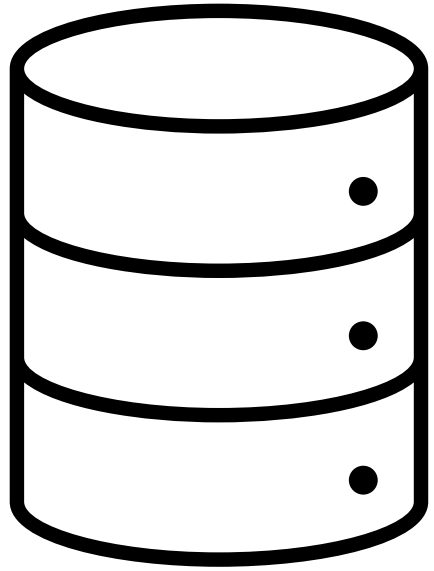


Processor



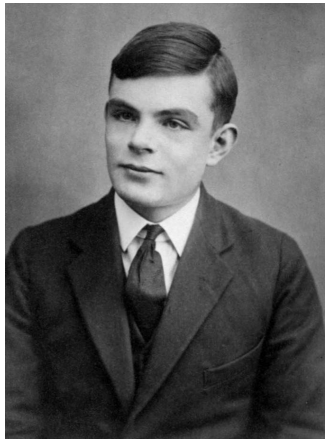
- ✓ Runs code; does computations
- ✗ Doesn't remember anything

Memory

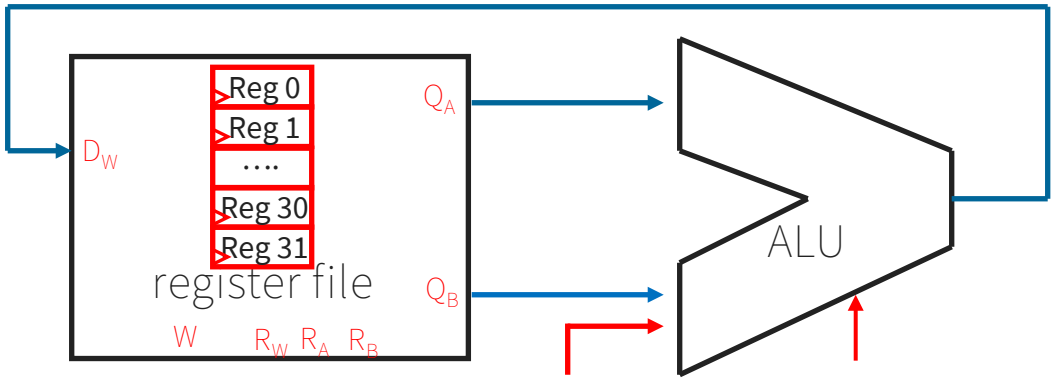


- ✗ Can't compute anything
- ✓ Stores data

Big Picture: How to Design a Processor

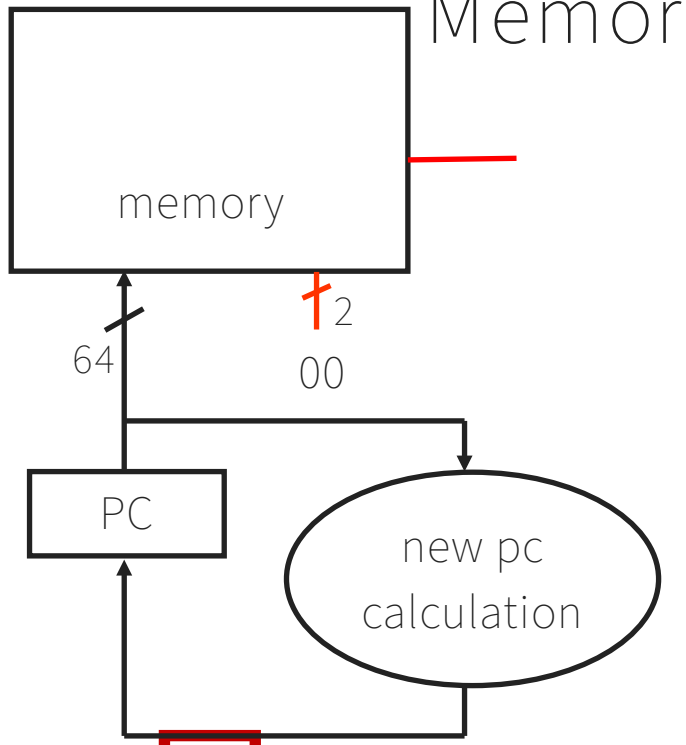


Processor



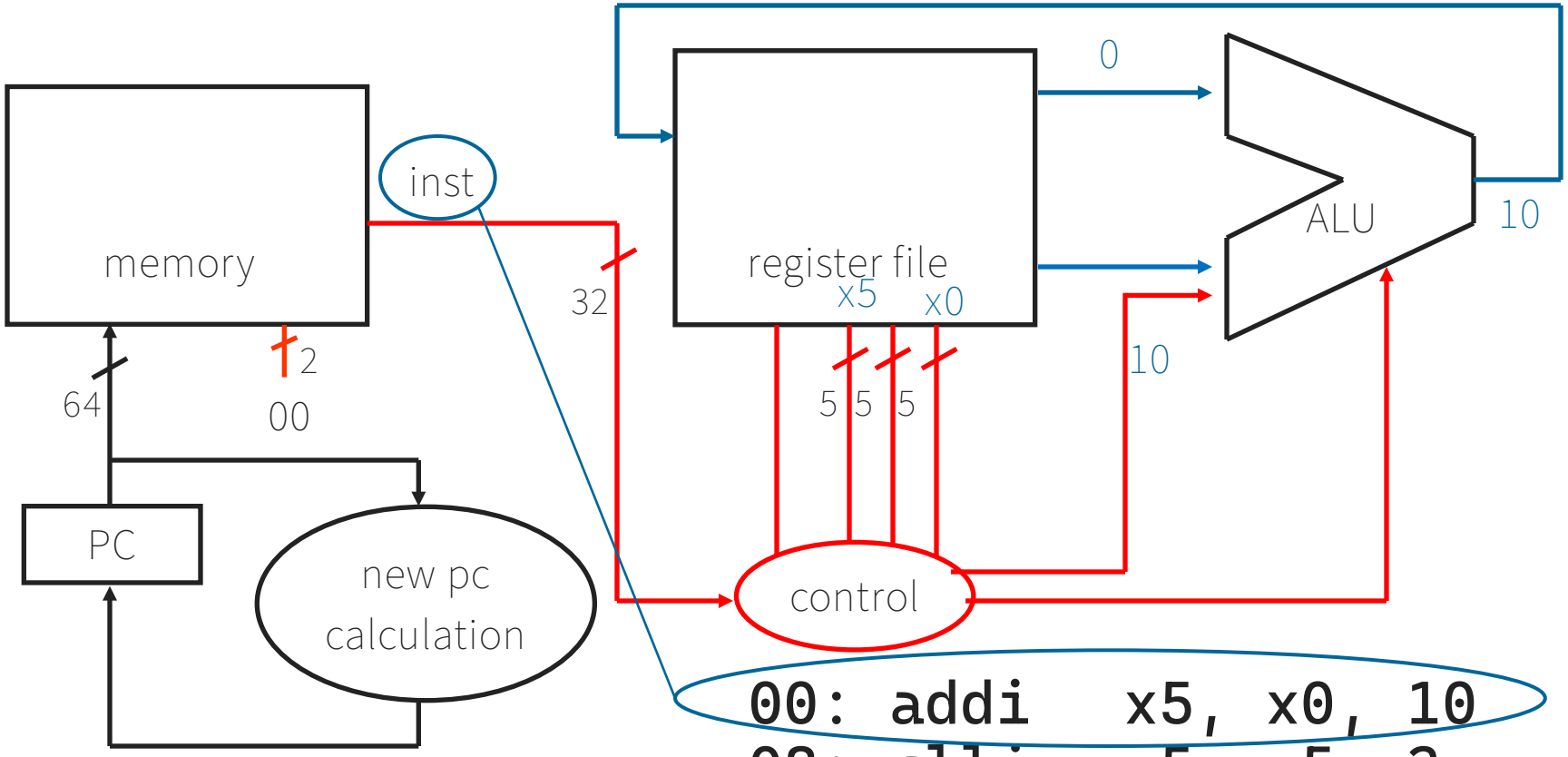
- ☑ Runs code; does computations
- ☒ Doesn't remember anything

Memory



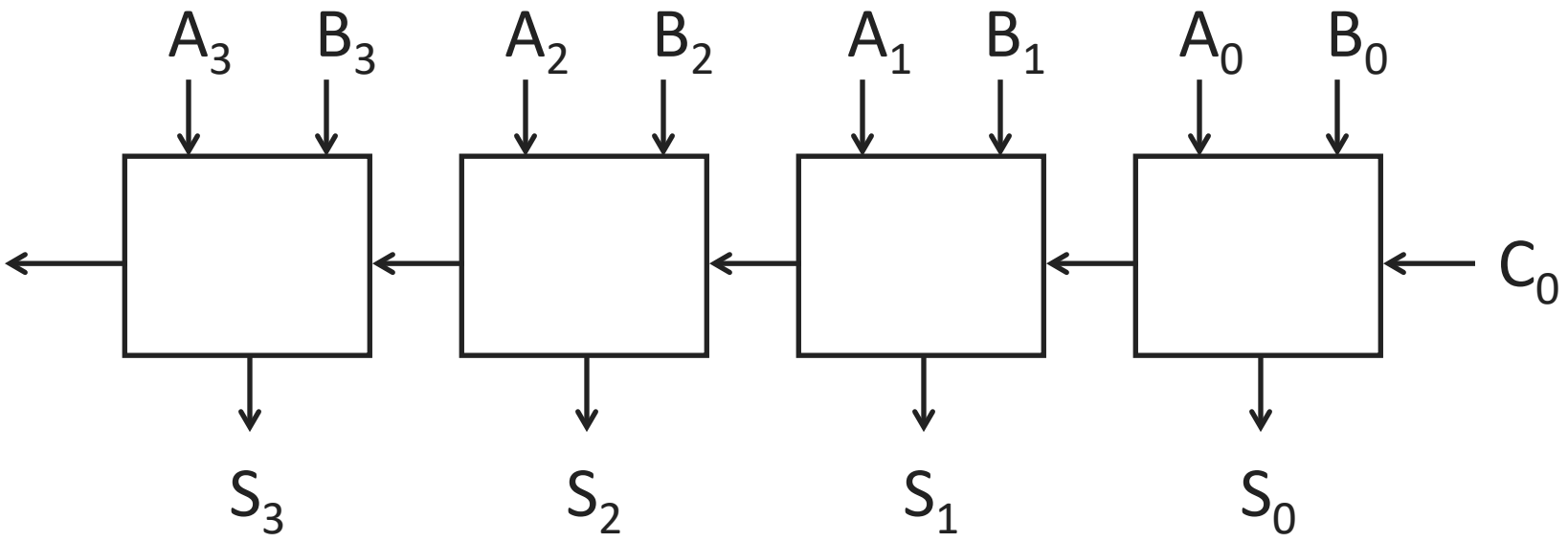
- ☒ Can't compute anything
- ☑ Stores data

Big Picture: How to Design a Processor



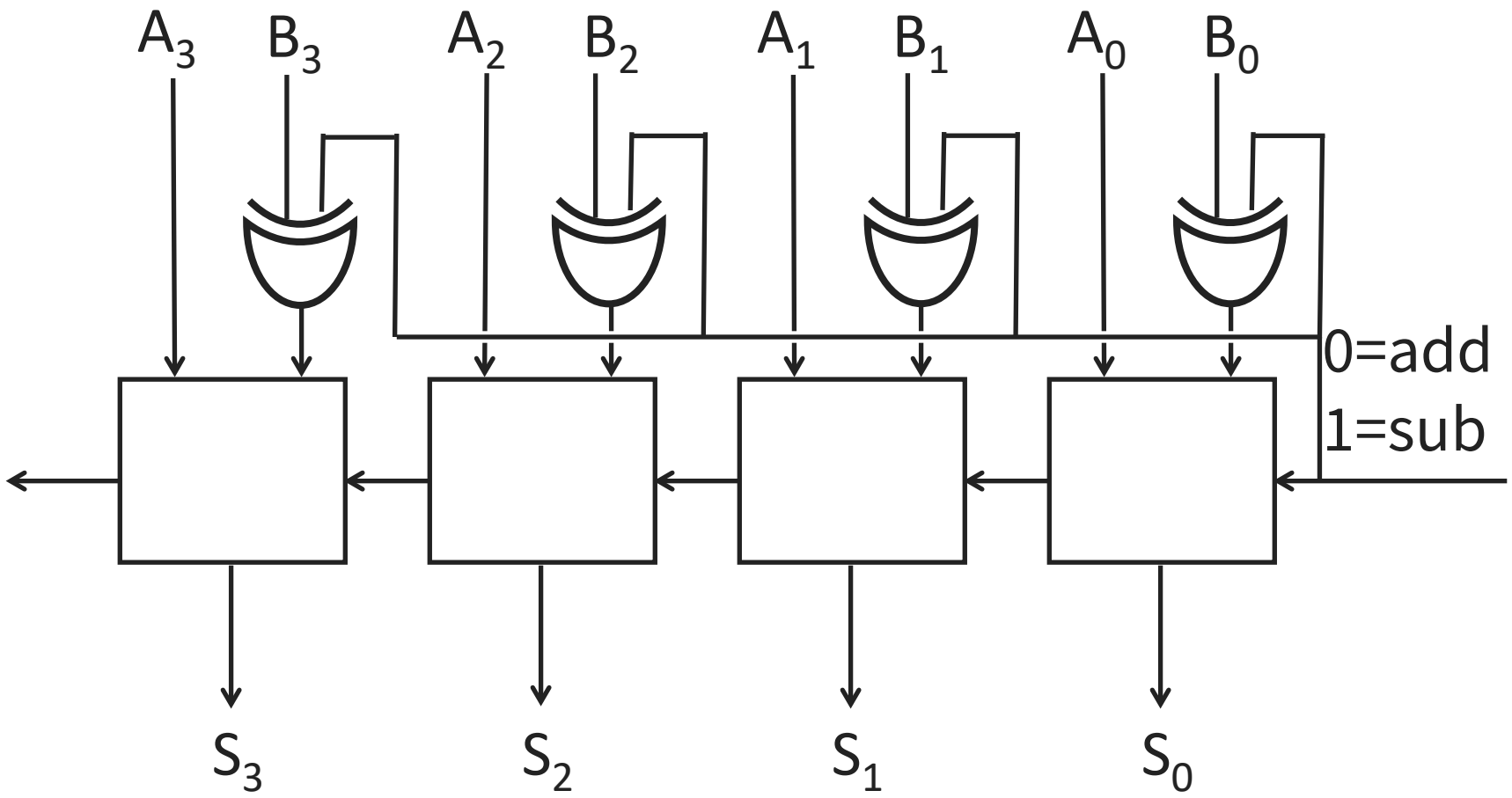
Review

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



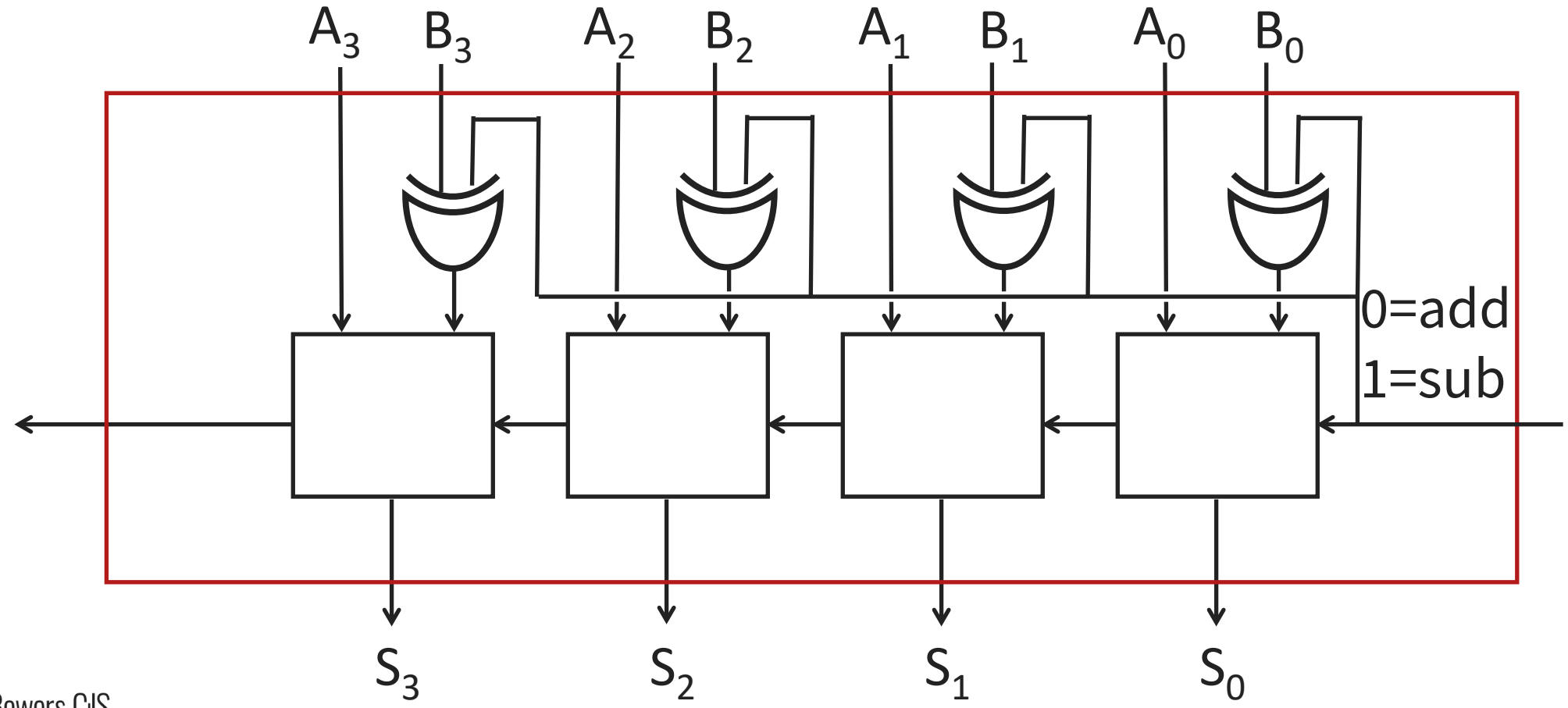
Review

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



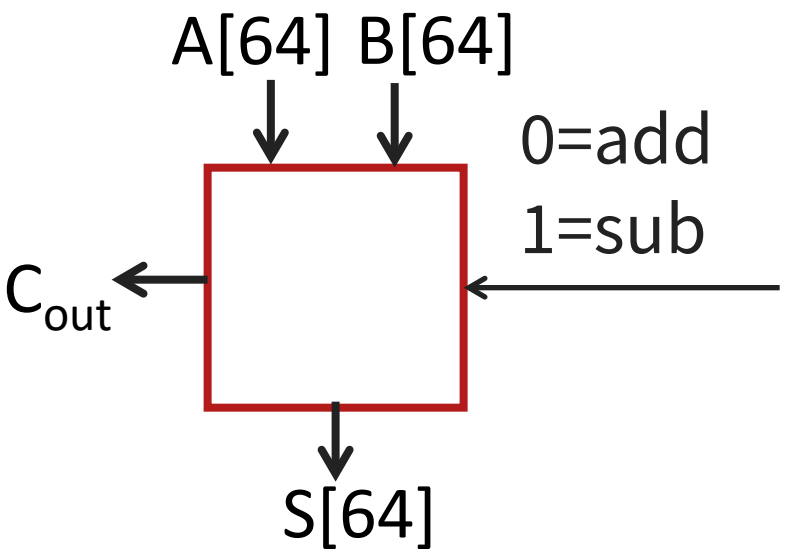
Review

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



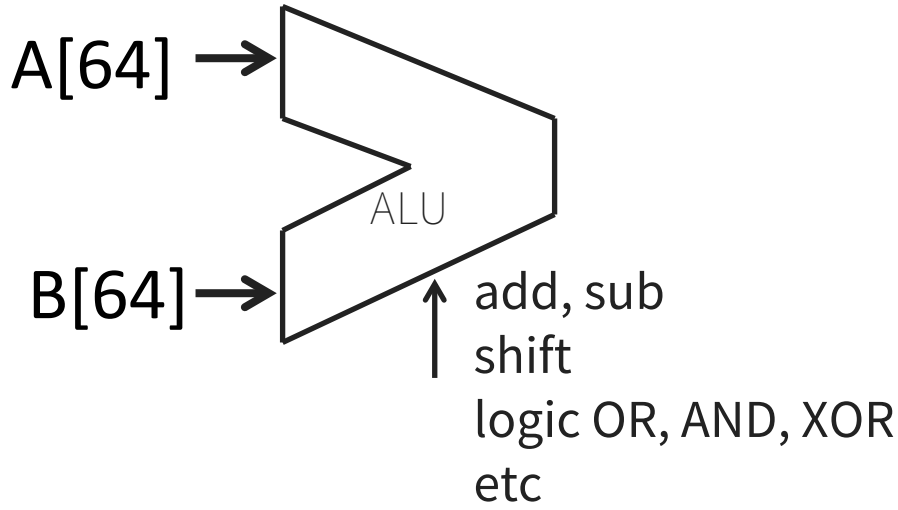
Review

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



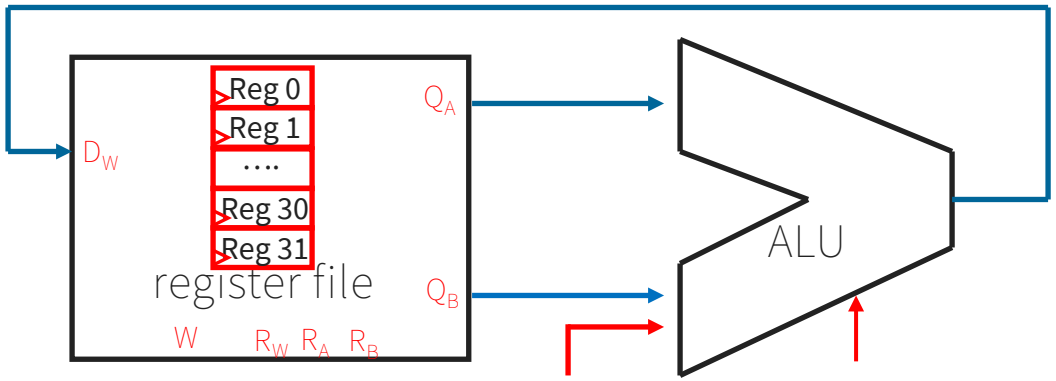
Review

- We can generalize 1-bit Full Adders to 32 bits, 64 bits …
- Arithmetic Logic Unit (ALU) adds, subtracts, shifts, logic OR, AND, XOR, etc



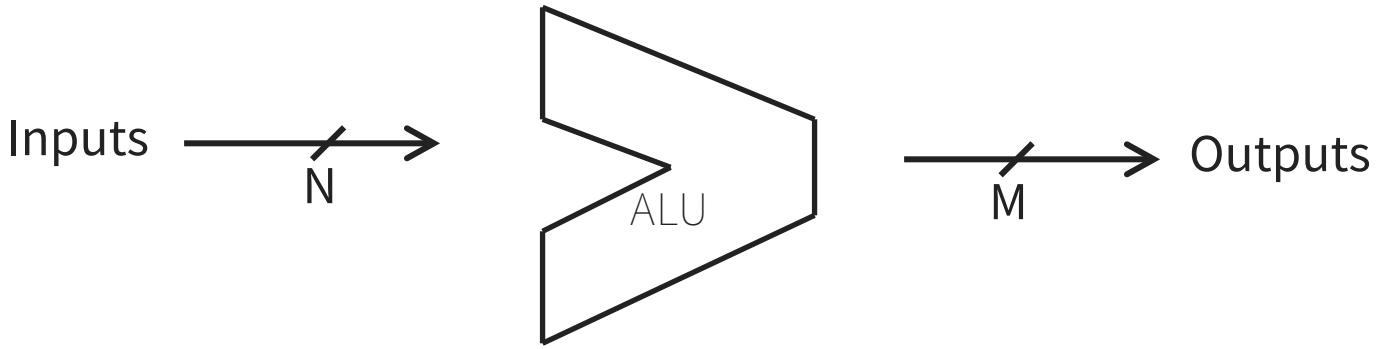
Goal for today

- How do we store results



Stateful Components

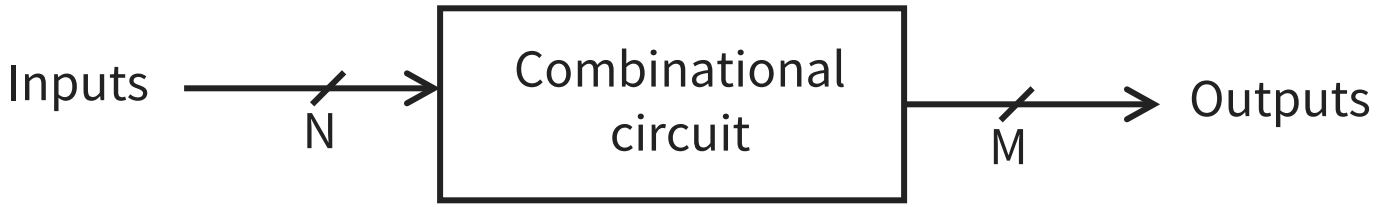
- Until now is combinational logic
  - Output is computed when inputs are present
  - System has no internal state
  - Nothing computed in the present can depend on what happened in the past!



- Need a way
  - to record data
  - to build stateful circuits
  - state-holding device

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
  - to build stateful circuits
  - state-holding device

Goals for Today

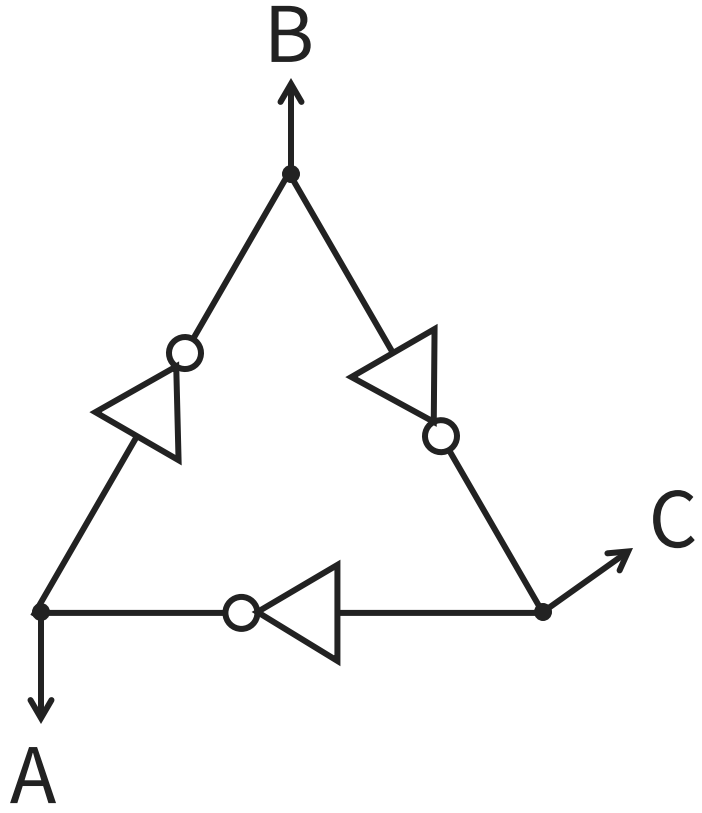
State

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

Goal

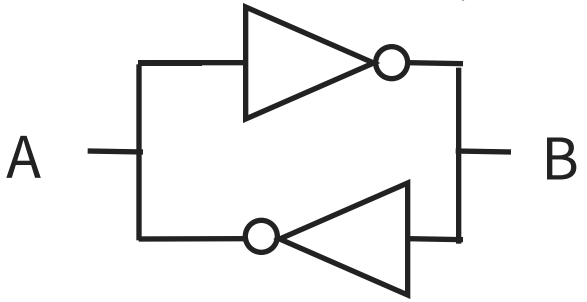
How do we store store one bit?

First Attempt: Unstable Devices



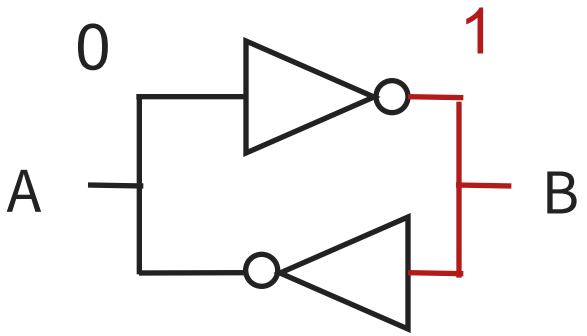
Second Attempt: Bistable Devices

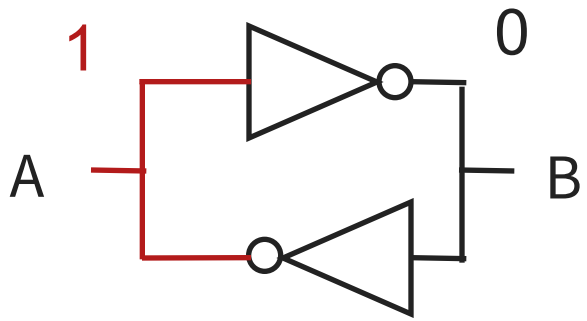
- Stable and unstable equilibria?



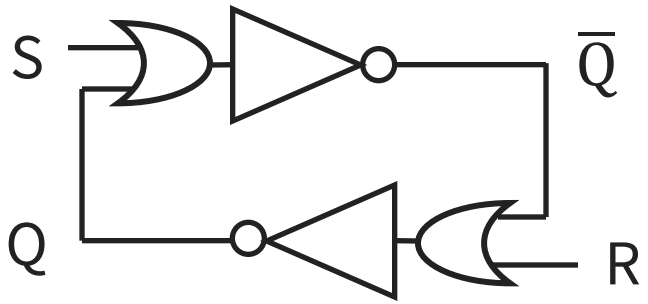
A Simple Device

In stable state, A != B

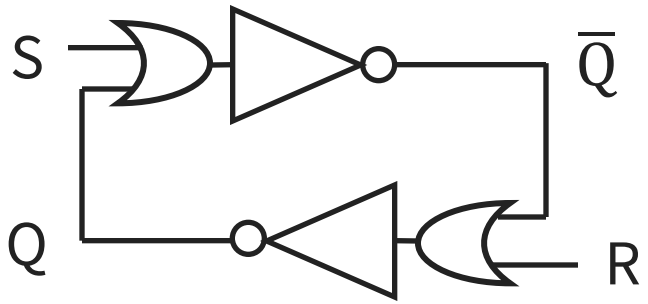




Third Attempt: Set-Reset Latch



Third Attempt: Set-Reset Latch



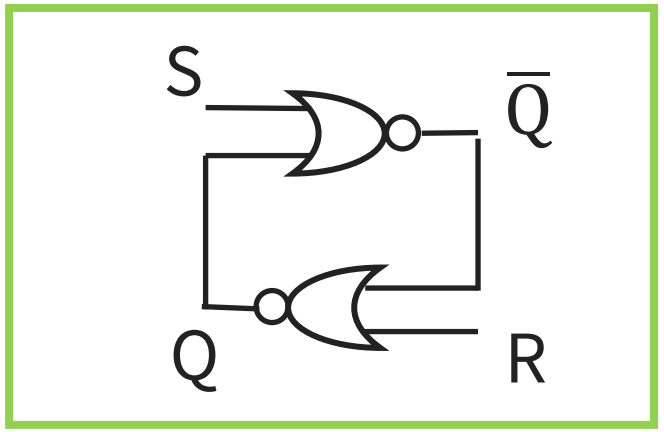
A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

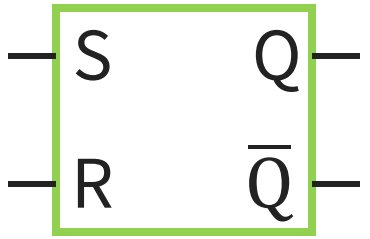
S	R	Q	$\overline{Q}$
0	0		
0	1		
1	0		
1	1		

Set-Reset (S-R) Latch

Stores a value Q and its complement

Third Attempt: Set-Reset Latch





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

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

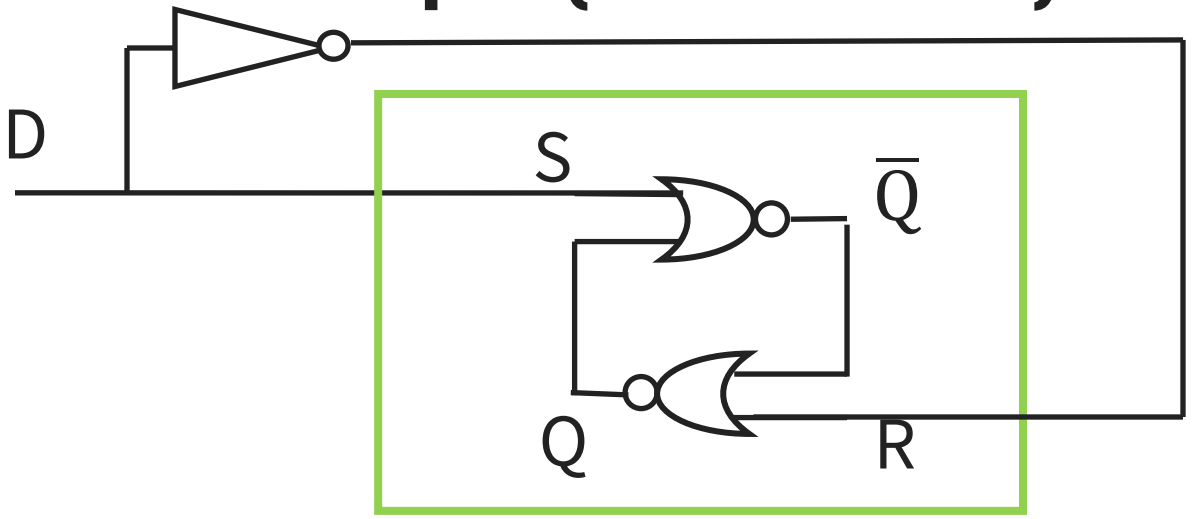
Takeaway

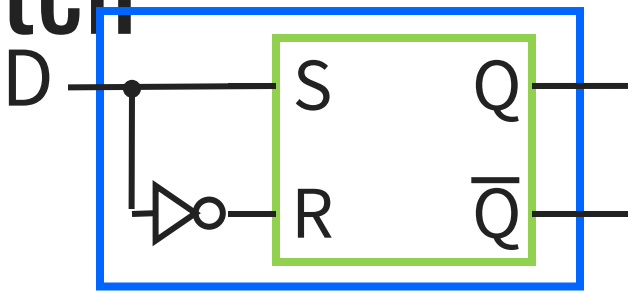
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

Next Goal

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

Fourth Attempt: (Unclocked) D Latch





Fill in the truth table?

D	Q	$\overline{Q}$
0		
1		

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.

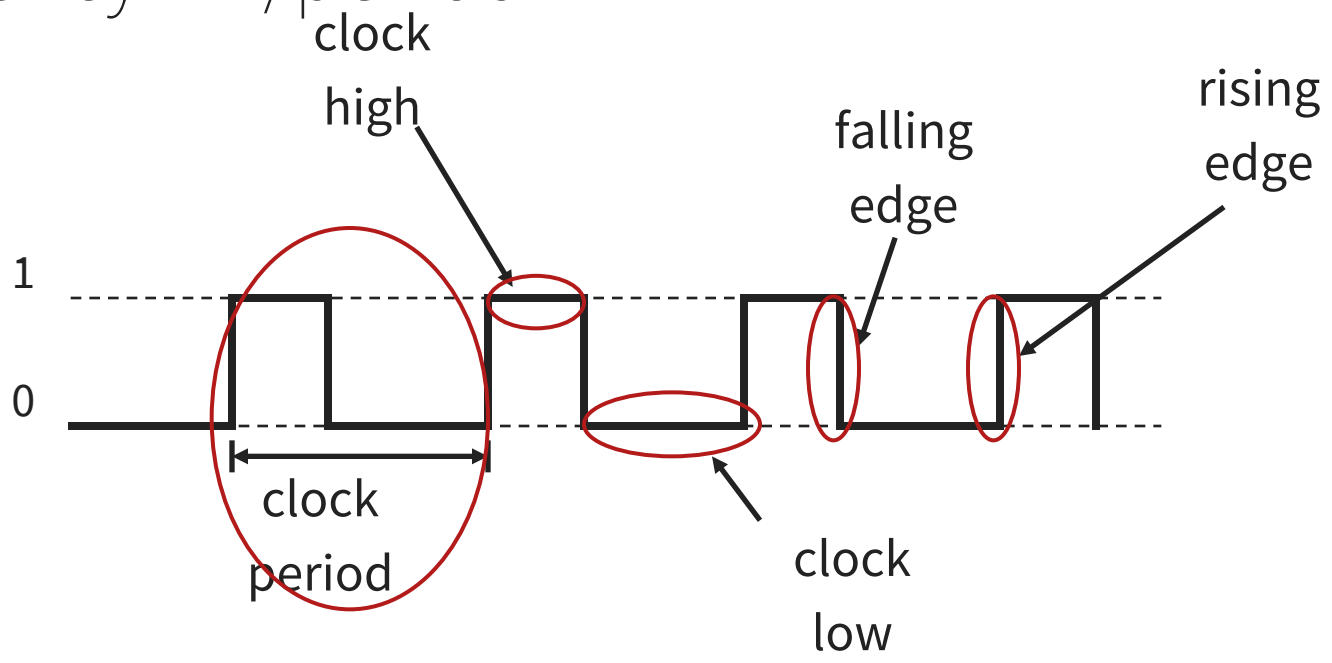
Next Goal

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/period



Clock Disciplines

Level sensitive

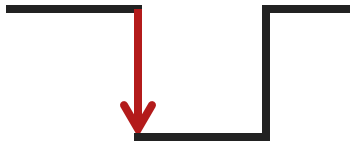
- State changes when clock is high (or low)

Edge triggered

- State changes at clock edge

positive edge-triggered

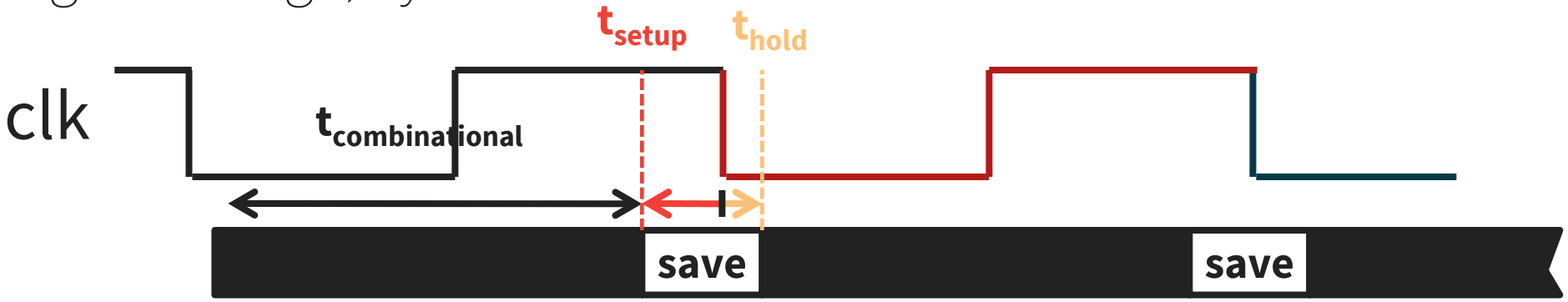
negative edge-triggered



Clock Methodology

Clock Methodology

- Negative edge, synchronous

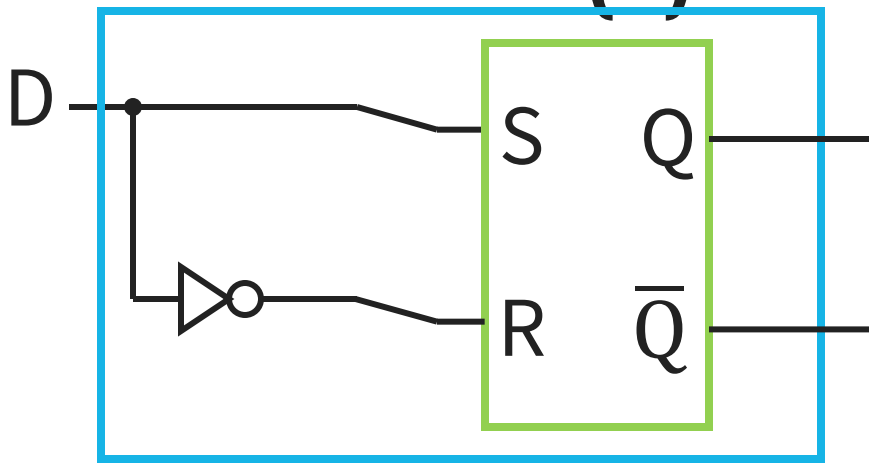


Edge-Triggered → signals must be stable near falling edge

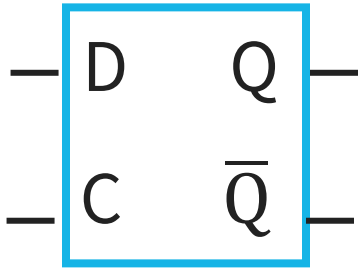
"near" = before and after

t_{setup} t_{hold}

Round 2: D Latch (1)

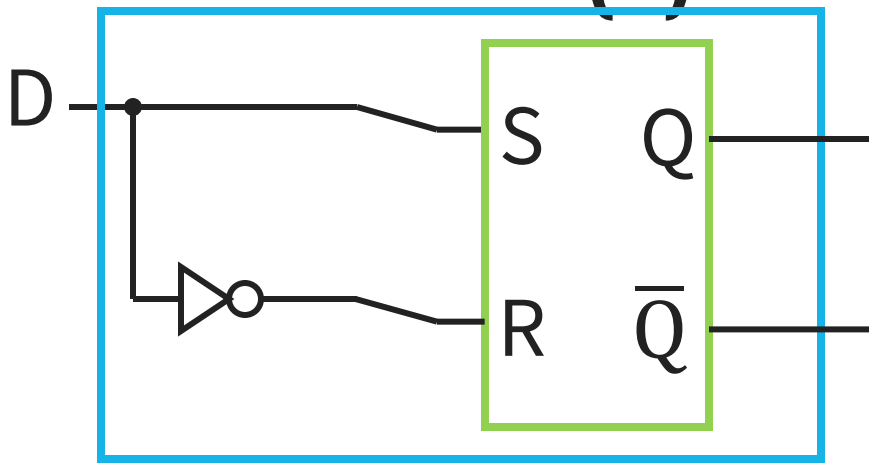


- Inverter prevents SR Latch from entering 1,1 state

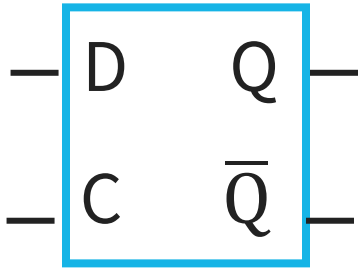


	D	Q	$\overline{Q}$	
	0			*Reset*
	1			*Set*

Round 2: D Latch (1)

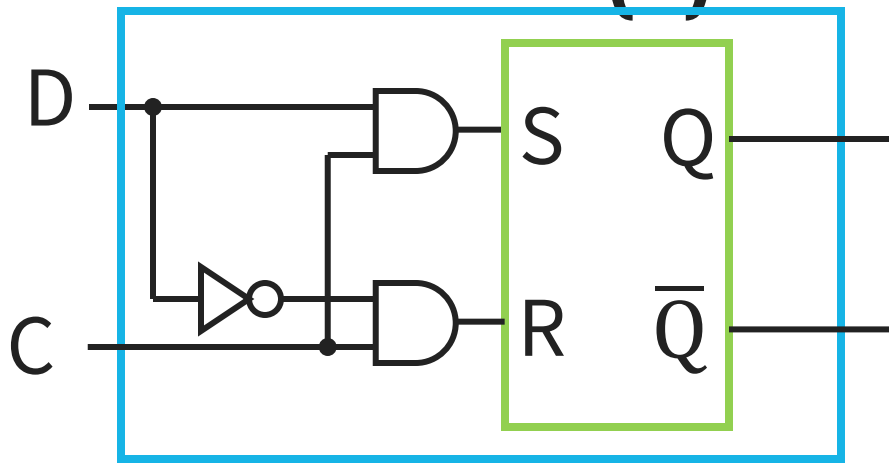


- Inverter prevents SR Latch from entering 1,1 state



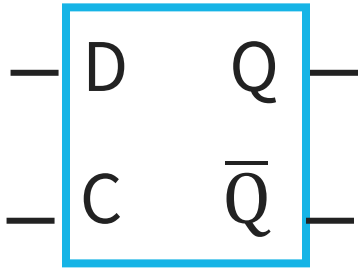
	D	Q	$\overline{Q}$	
	0	0	1	*Reset*
	1	1	0	*Set*

Round 2: D Latch (1)



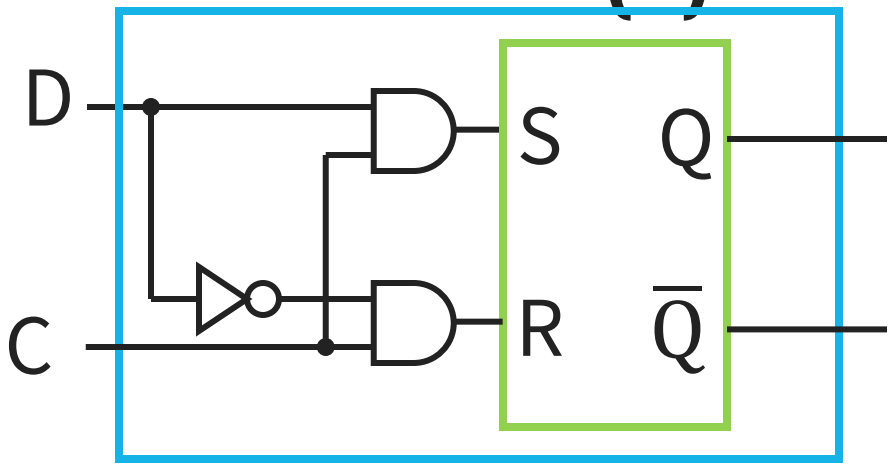
- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- C enables changes

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



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

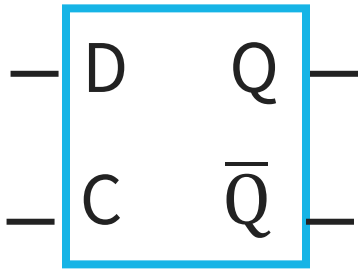
Round 2: D Latch (1)



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

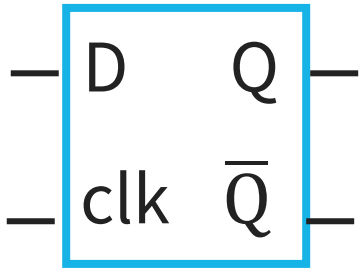
C = 1, D Latch *transparent*:
set/reset (according to D)
C = 0, D Latch *opaque*:
keep state (ignore D)

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



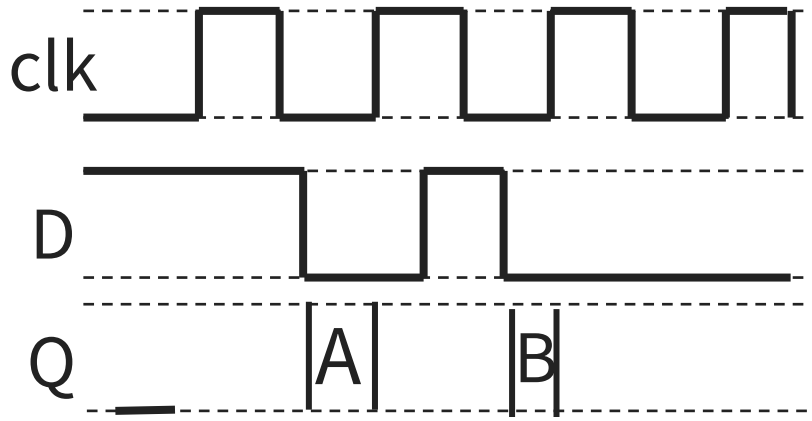
C	D	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	*No Change*
0	1	Q	$\overline{Q}$	
1	0	0	1	*Reset*
1	1	1	0	*Set*

PollEV Question



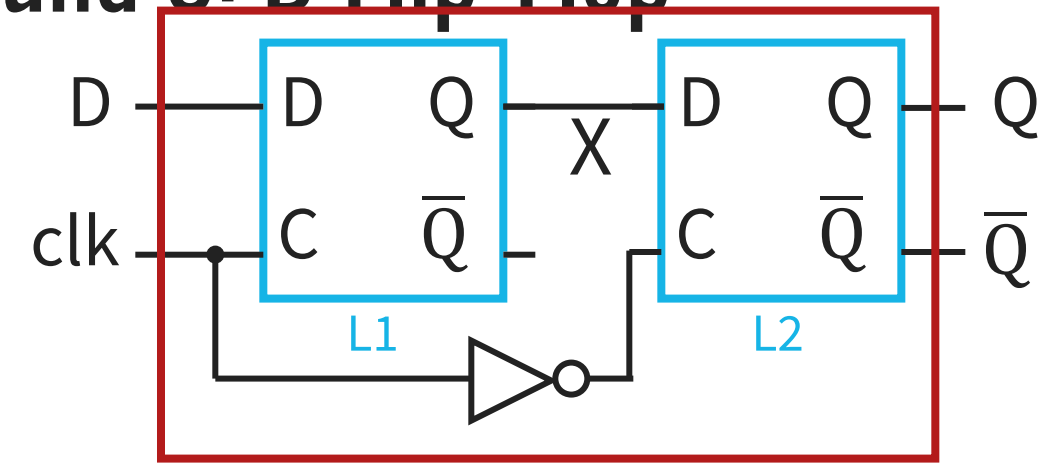
What is the value of Q at A & B?

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



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

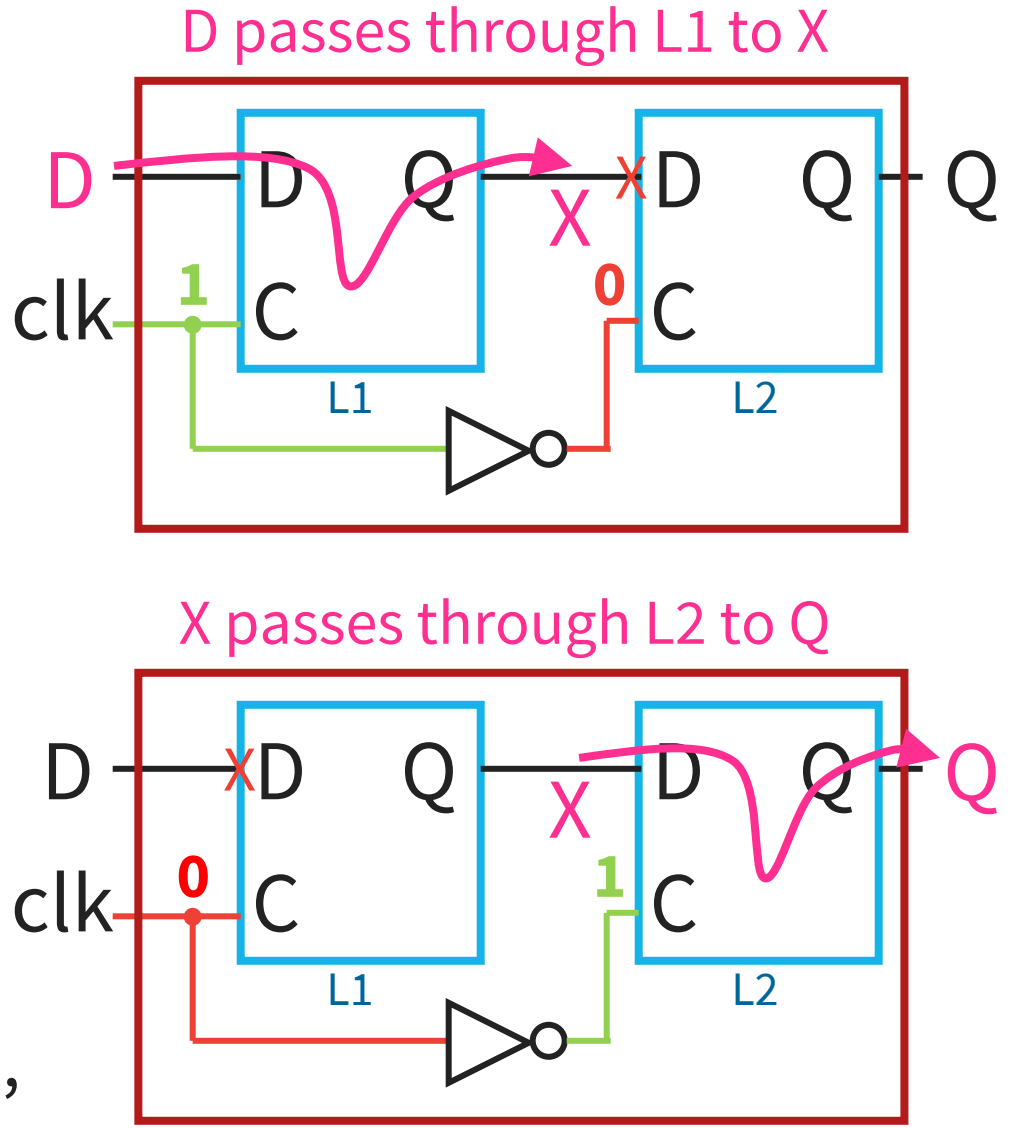
L2 *opaque*

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

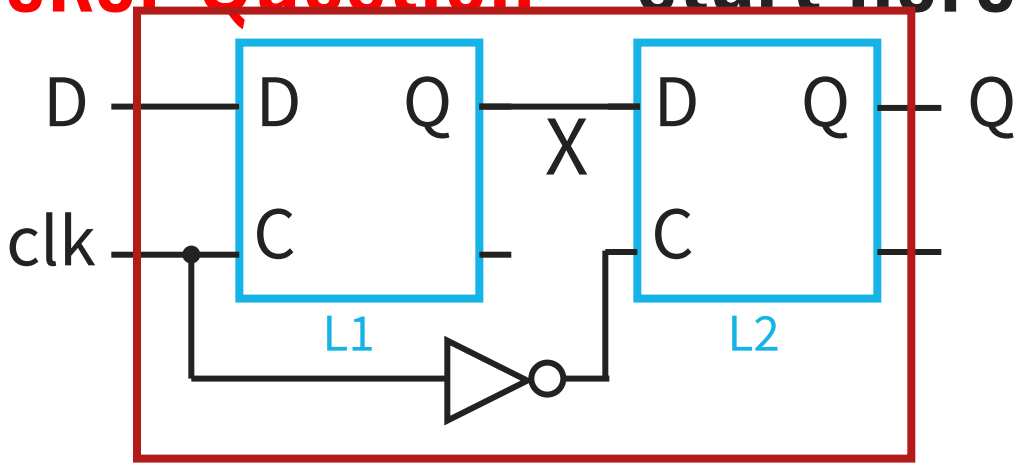
Clock = 0: L1 *opaque*

L2 *transparent*

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

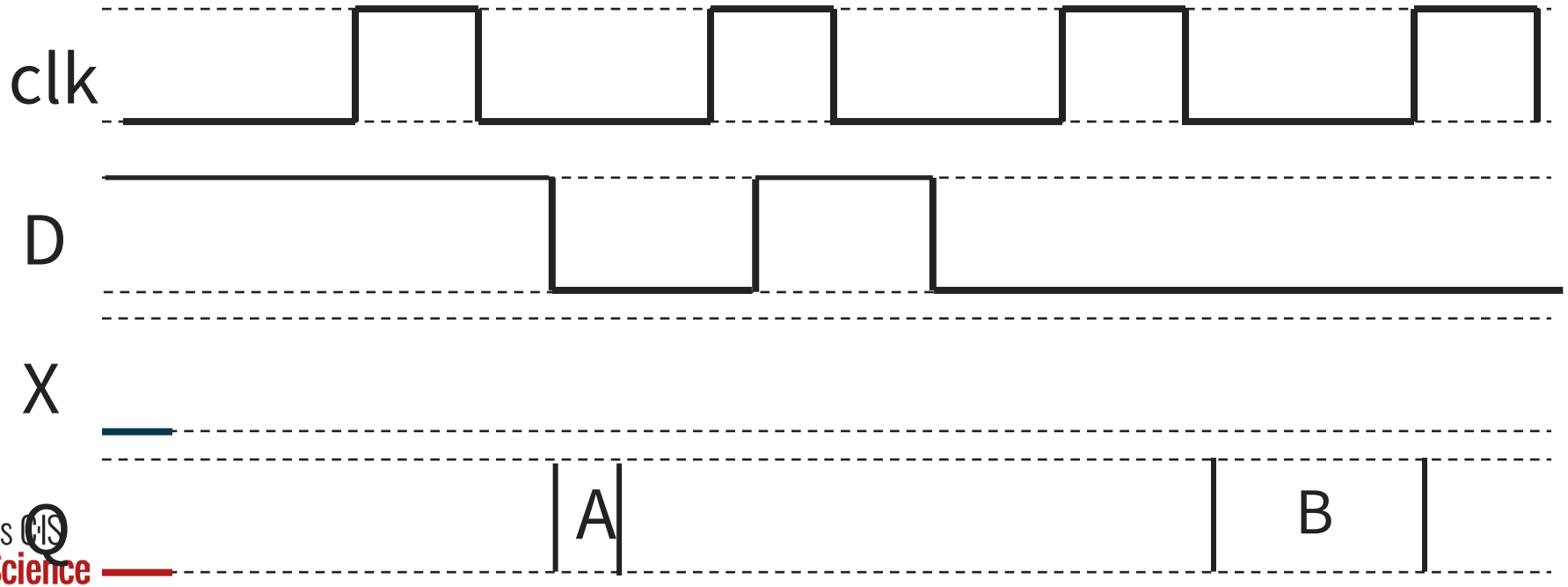


iClicker Question – start here

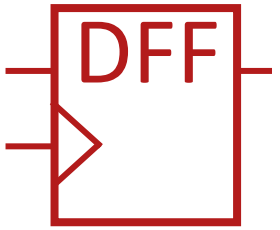


What is the value of Q at A & B?

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

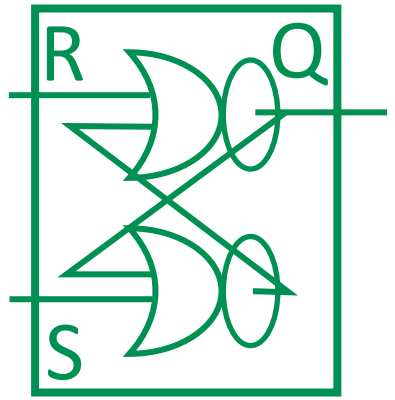


Building a D Flip Flop (DFF)

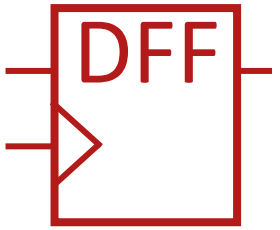


Step 1: Create an SR Latch

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



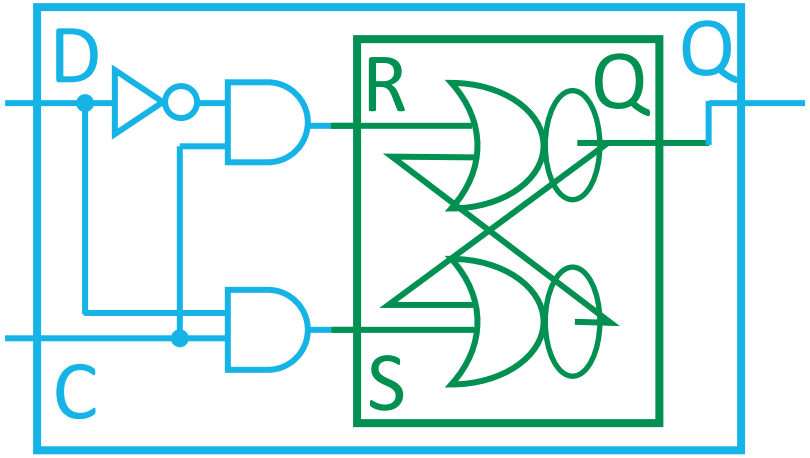
Building a D Flip Flop (DFF)



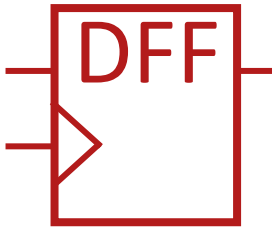
Step 1: Create an SR Latch

Step 2: Create a D Latch

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



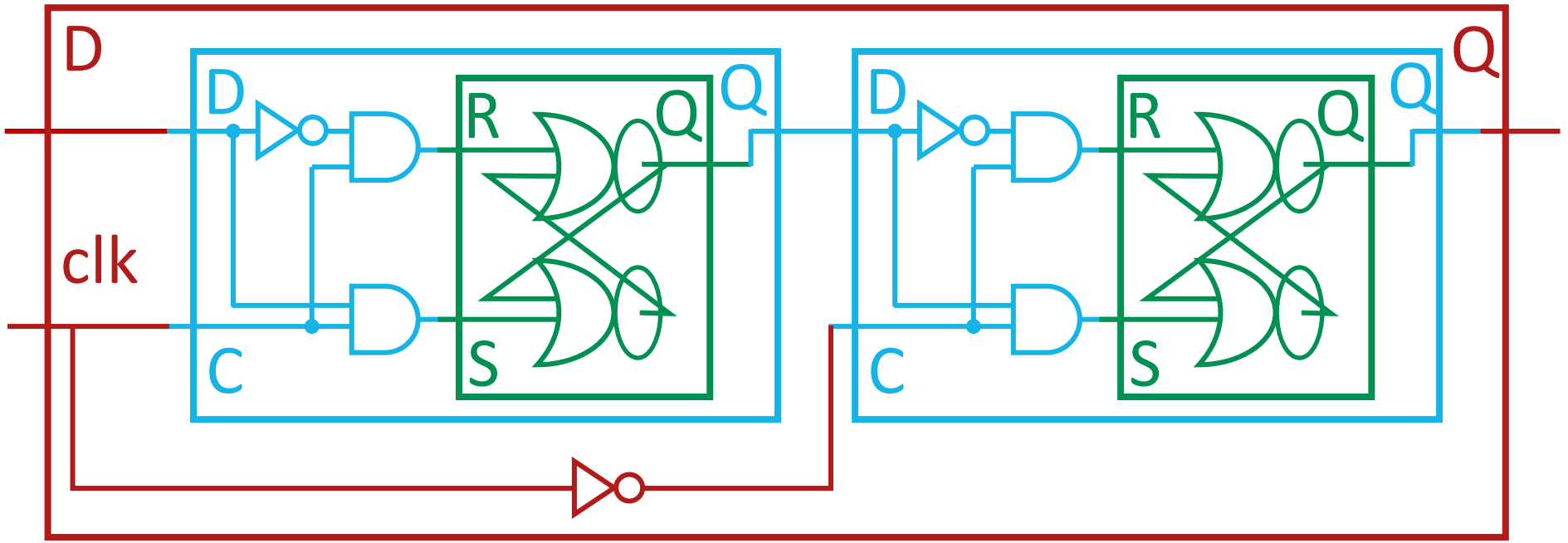
Building a D Flip Flop (DFF)



Step 1: Create an SR Latch

Step 2: Create a D Latch

Step 3: Duplicate the D Latch, chain together



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.

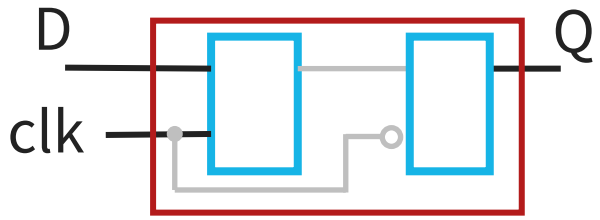
(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

An Edge-Triggered D Flip-Flip (aka Master-Slave 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?

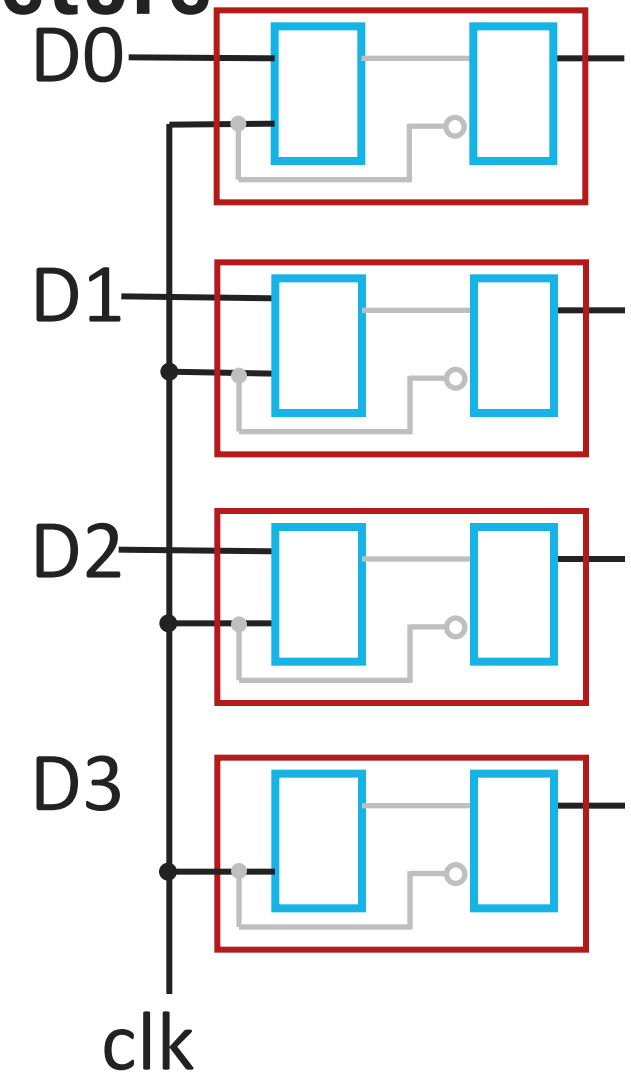
Registers



Register

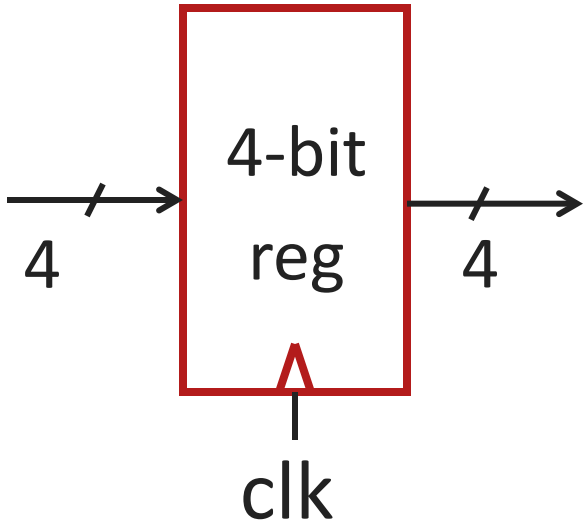
- D flip-flops in parallel

Registers

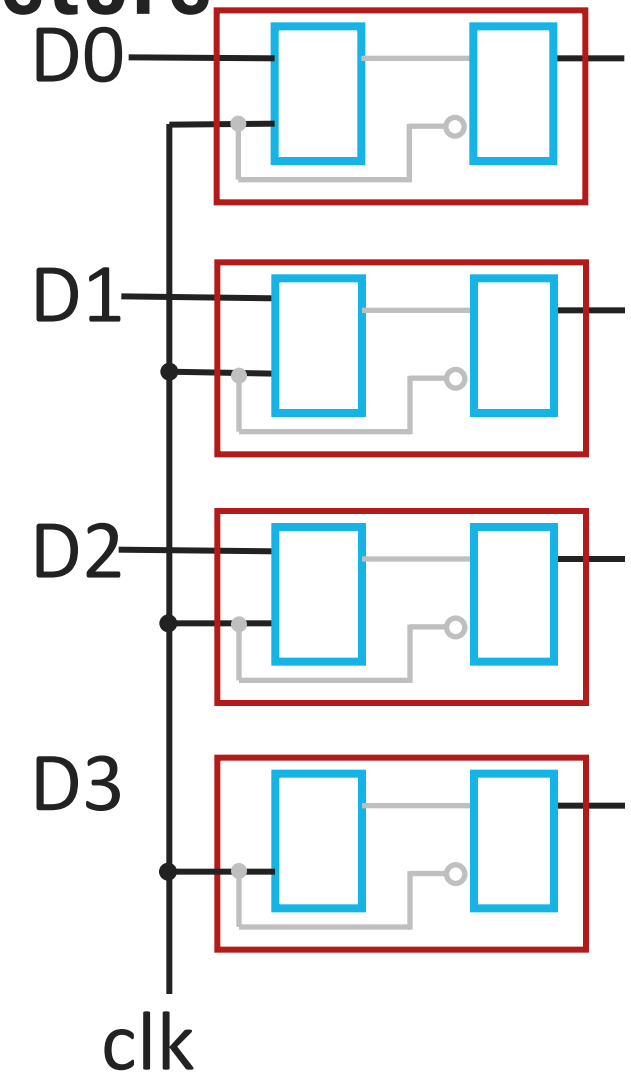


Register

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

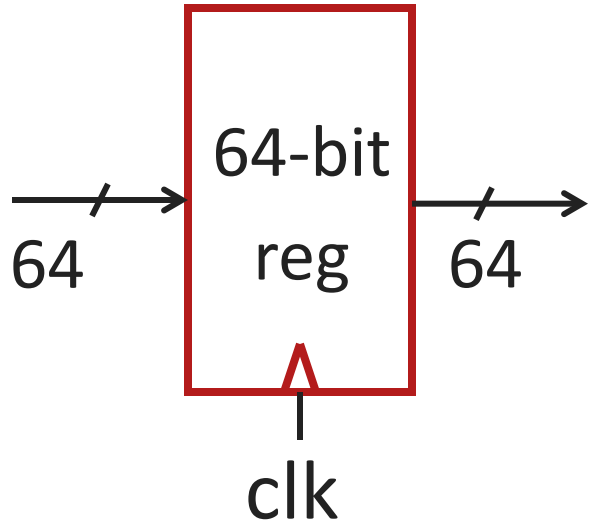


Registers



Register

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



Takeaway

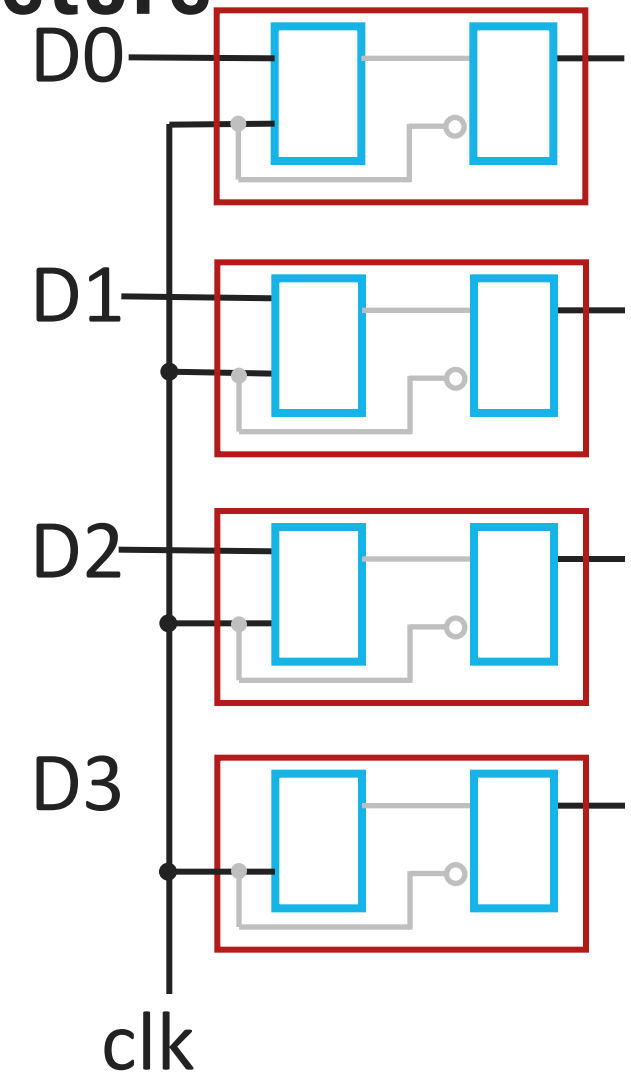
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

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

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

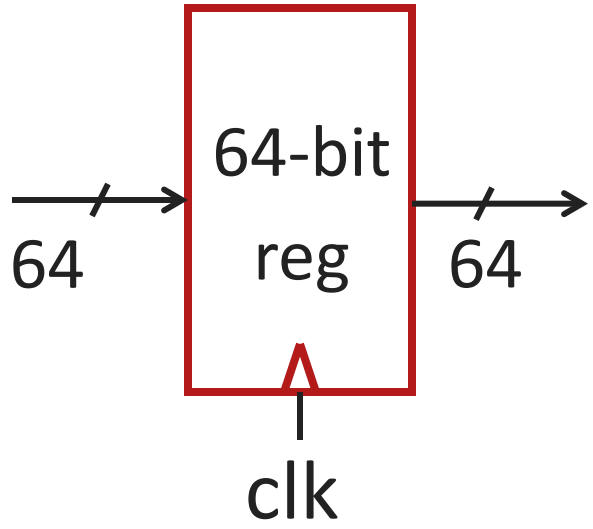
An *N*-bit register stores *N*-bits. It is created with *N* D-Flip-Flops in parallel along with a shared clock.

Registers

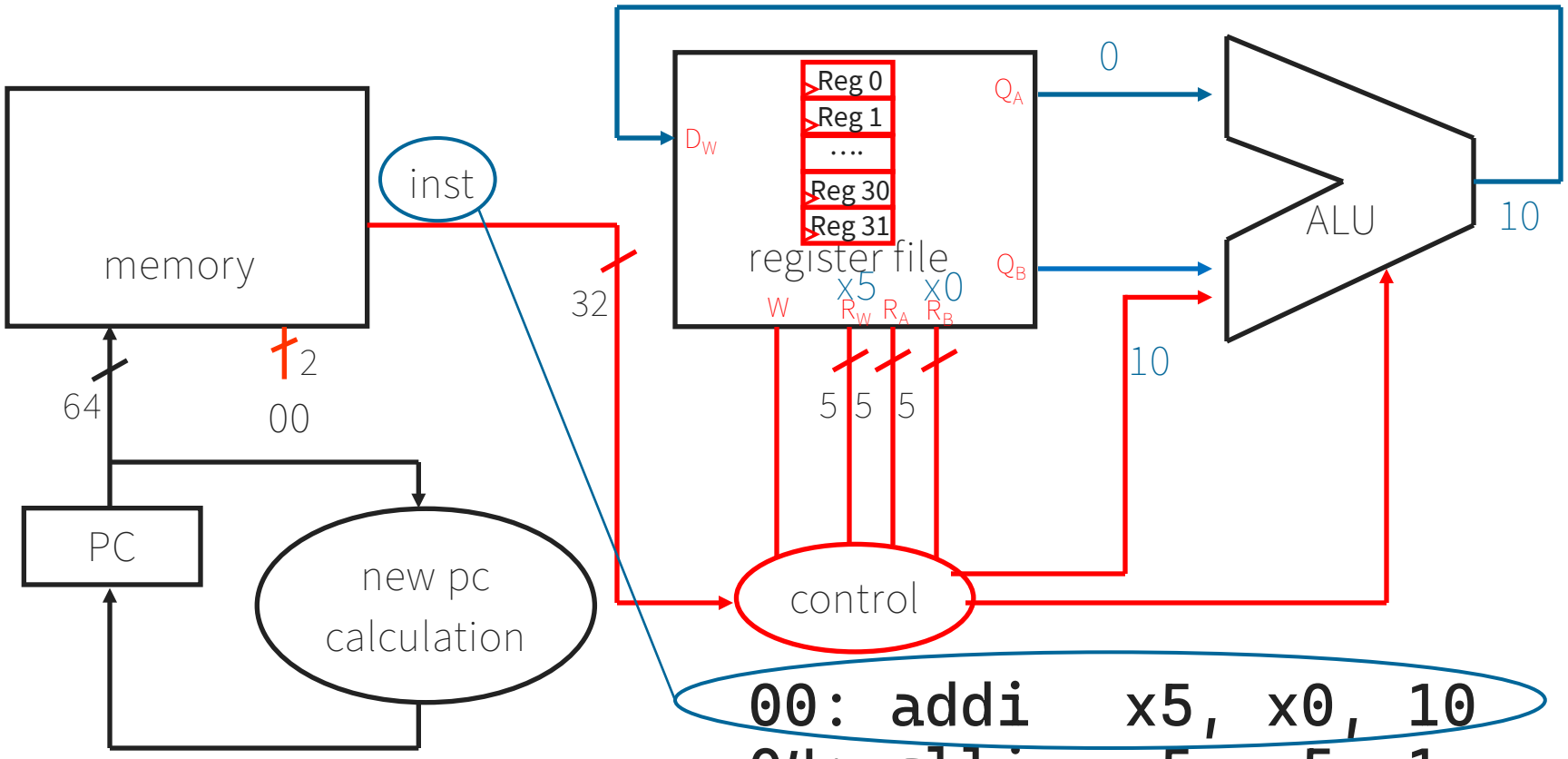


Register

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



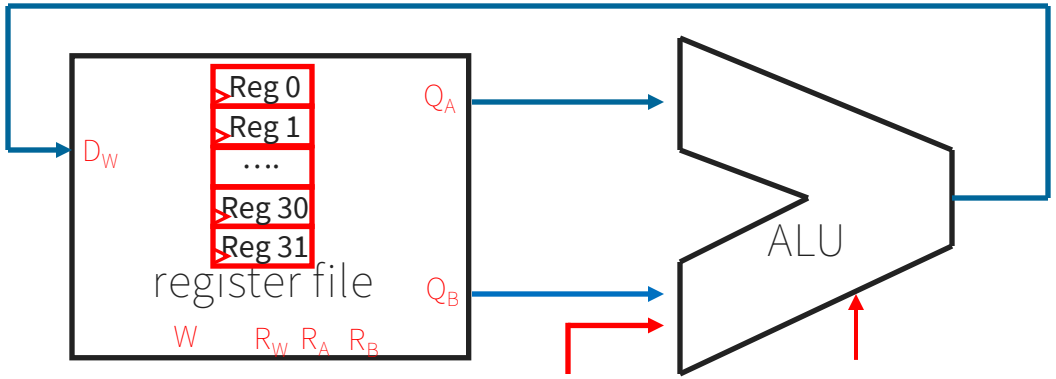
Big Picture: How to Design a Processor



Big Picture: How to Design a Processor

Register File

- N read/write registers
- Indexed by register number

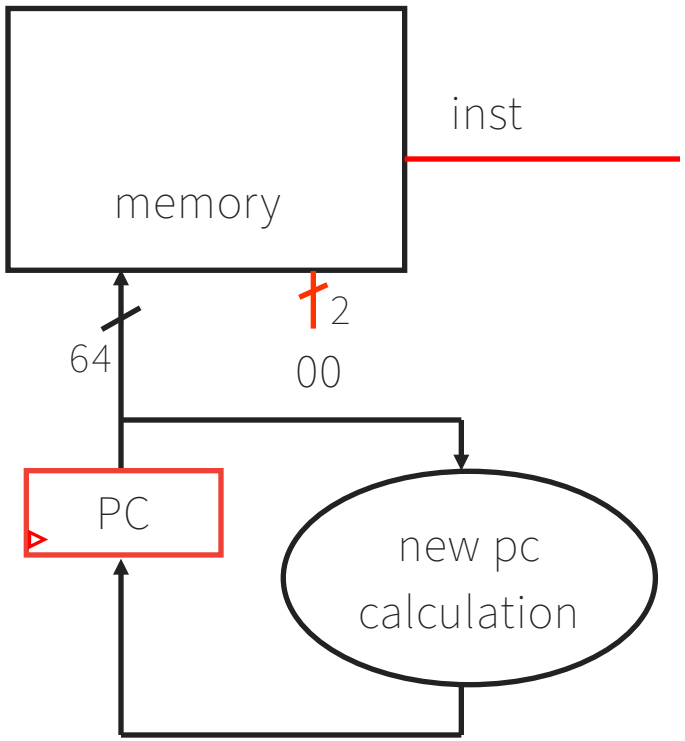


Registers

- Numbered from 0 to 31.
- Can be referred by number: x0, x1, x2, … x31
  - May also see $0, $1, $2 or r0, r1, r2
- Convention, each register also has a name:
  - x16 - x23 → s0-s7 ("s registers")
  - x8 - x15 → t0 - t7 ("t registers")

```
00: addi   x5, x0, 10
04: slli   x5, x5, 2
08: addi   x5, x5, 15
```

Big Picture: How to Design a Processor

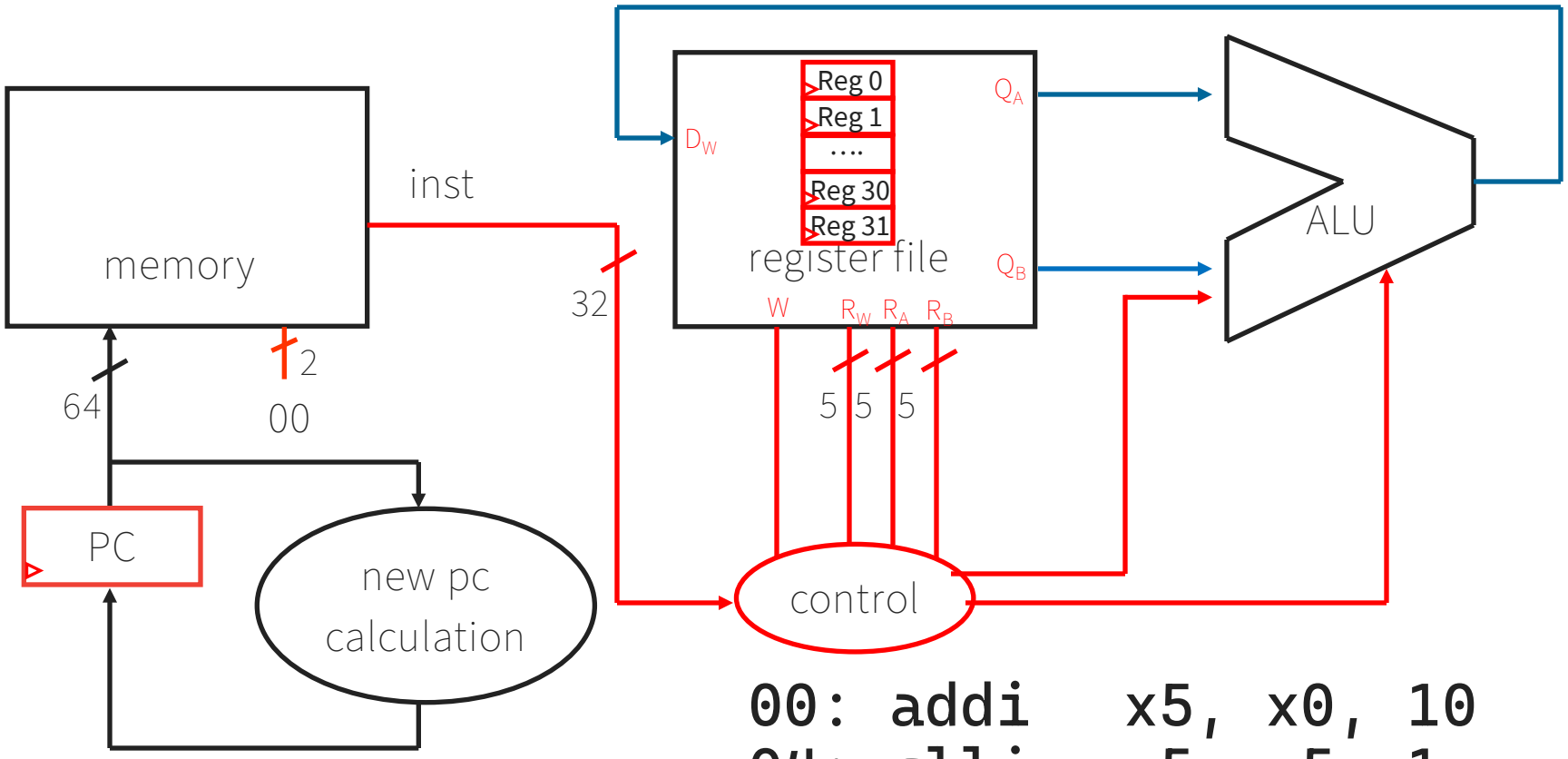


PC is register!

- PC is the Program Counter
- Stores the memory address of the next instruction

```
00: addi   x5, x0, 10
04: slli   x5, x5, 1
08: addi   x5, x5, 15
```

Big Picture: How to Design a Processor




```
00: addi   x5, x0, 10
04: slli   x5, x5, 1
08: addi   x5, x5, 15
```

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 (aka Master-Slave 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.

Summary

We store data values

- Stateful circuit elements (D Flip Flops, Registers, …)
- Clock to synchronize state changes