CSC258 Winter 2016
Lecture 5

Q2: Add two signed 4-bit integers 6 + 7, what result (also a 4-bit signed integer) will be produced?

A. 13 B. -2

C. -3 D. None of above

Answer: C

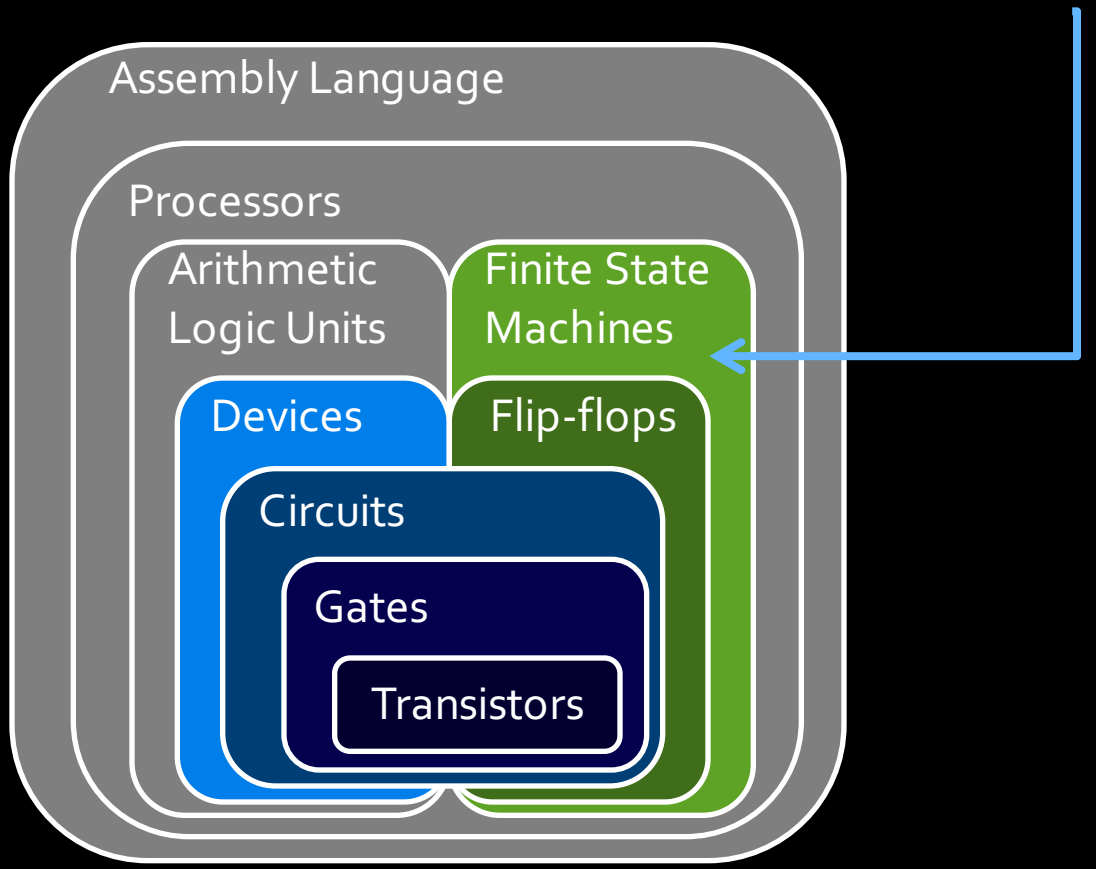
6 is 0110, 7 is 0111, adding them up we get 1101, which is -3 because 1101's 2's-complement is 0011 (3).

This situation is called an **overflow**, since the expected result 13 is exceeding the range of value that a 4-bit signed integer can represent (-8 ~ 7).

Try this C code

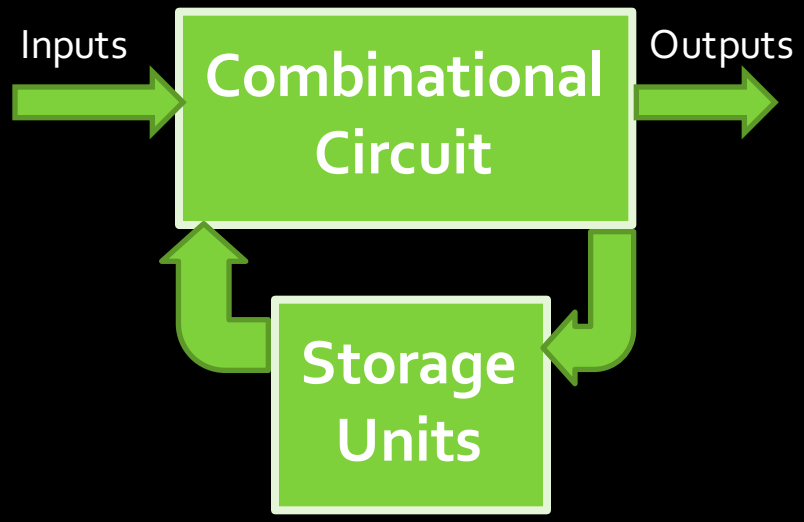
```
#include <stdio.h>


int main()
{
    /* char is 8-bit integer */
    signed char a = 100;
    signed char b = 120;
    signed char s = a + b;
    printf("%d\n", s);
}
```

We are here


Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

Circuits using flip-flops

- Now that we know about flip-flops and what they do, how do we use them in circuit design?
- What's the benefit in using flip-flops in a circuit at all?



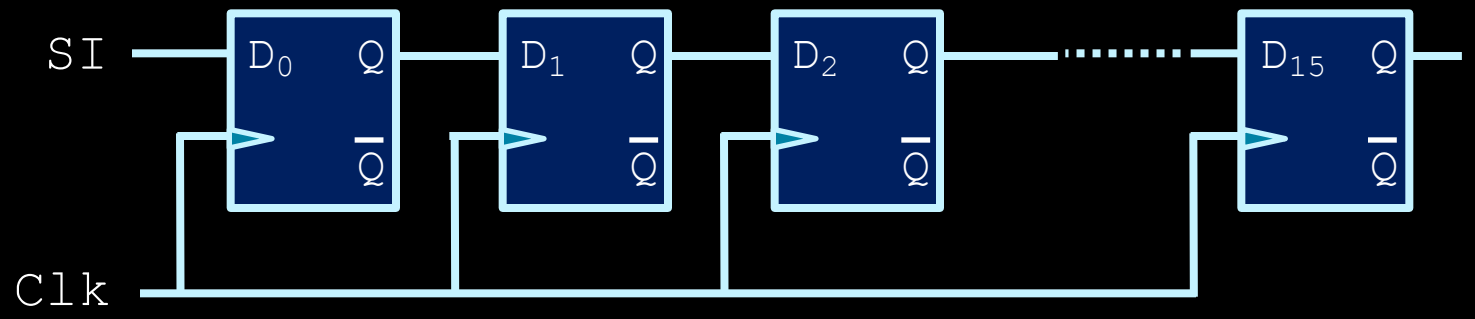
We can design much cooler circuits using flip-flops.

Example #1: Registers

For storing values

Shift registers

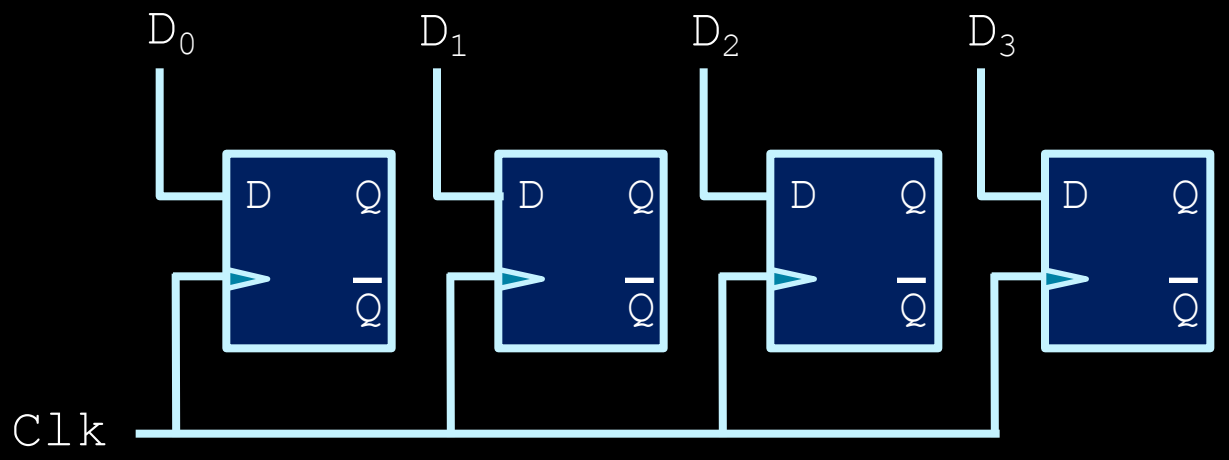
- A series of D flip-flops can store a multi-bit value (such as a 16-bit integer, for example).



- Data can be shifted into this register **one bit at a time**, over 16 clock cycles.
  - Known as a shift register.

Load registers

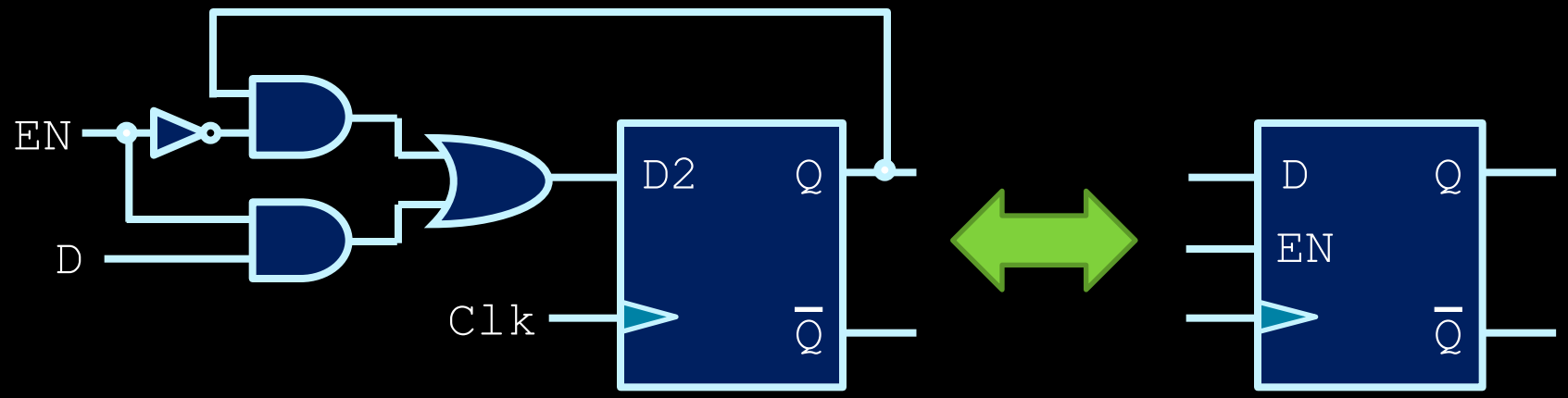
- One can also load a register's values all at once, by feeding signals into each flip-flop:
  - In this example: a 4-bit load register.



One clock pulse, 4 bits stored.

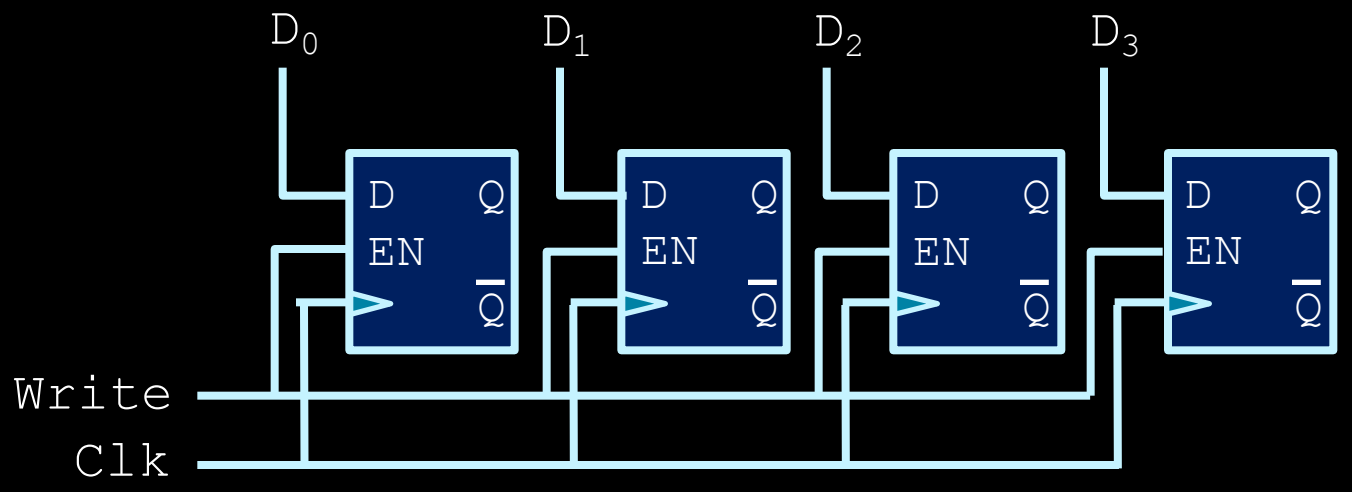
Load registers

- To control when this register is allowed to load its values, we introduce the D flip-flop with enable:



When EN = 1, D2 is whatever D is, load D
When EN = 0, D2 is whatever Q is, maintain Q

Load registers



- Implementing the register with these special D flip-flops will now maintain values in the register until overwritten by setting EN high.

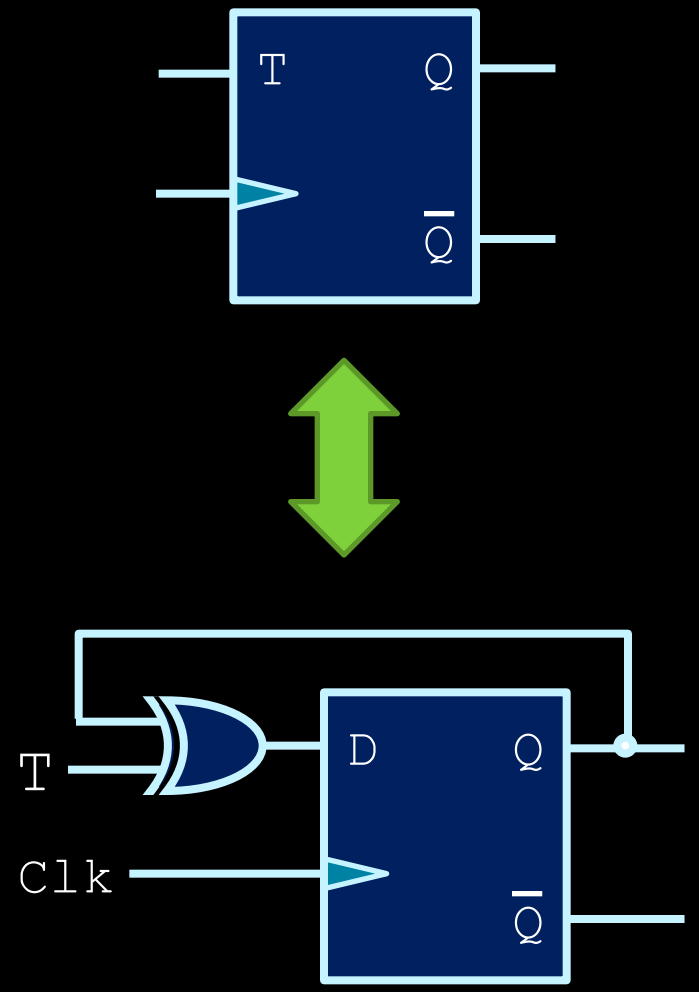
In computer architecture, registers are the CPU's most local storages (30+ of them on-chip), i.e., they are the lowest level of the memory hierarchy. They are the memory units that the CPU directly interact with.

Higher level of memory include cache, RAM, hard disc, etc.

Example #2: Counters

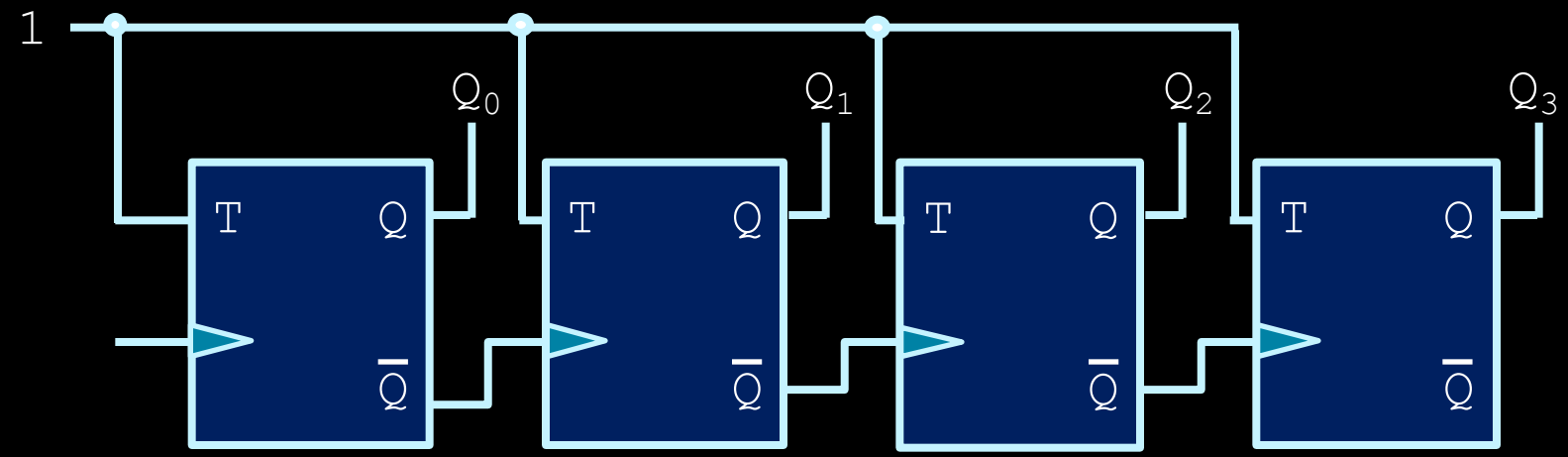
Counters

- Consider the T flip-flop:
  - Output is inverted when input T is high.
- What happens when a series of T flip-flips are connected together in sequence?
- More interesting:
  - Connect the *output* of one flip-flop to the *clock* input of the next!



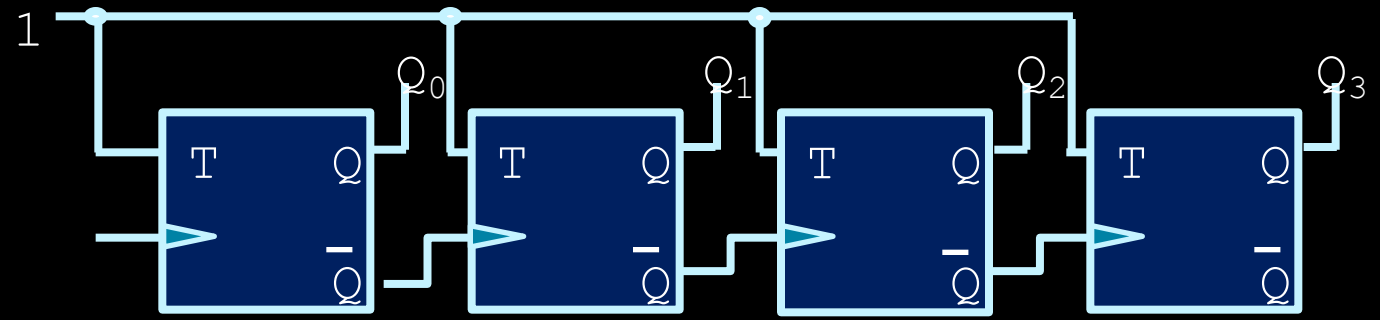
Counters

Asynchronous means the four outputs do not change upon the same clock signal.



- This is a 4-bit ripple counter, which is an example of an asynchronous circuit.
  - Timing isn't quite synchronized with the rising clock pulse.
  - Cheap to implement, but unreliable for timing.

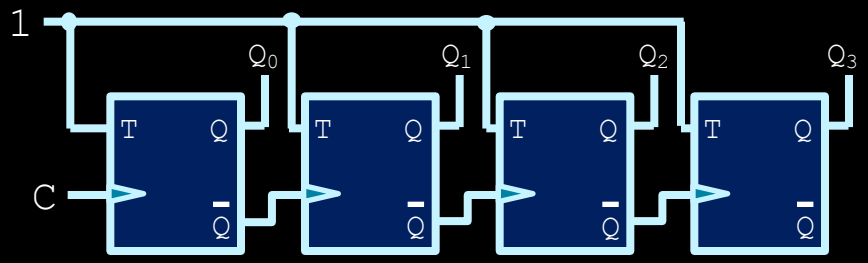
Demo: asynchronous human counter

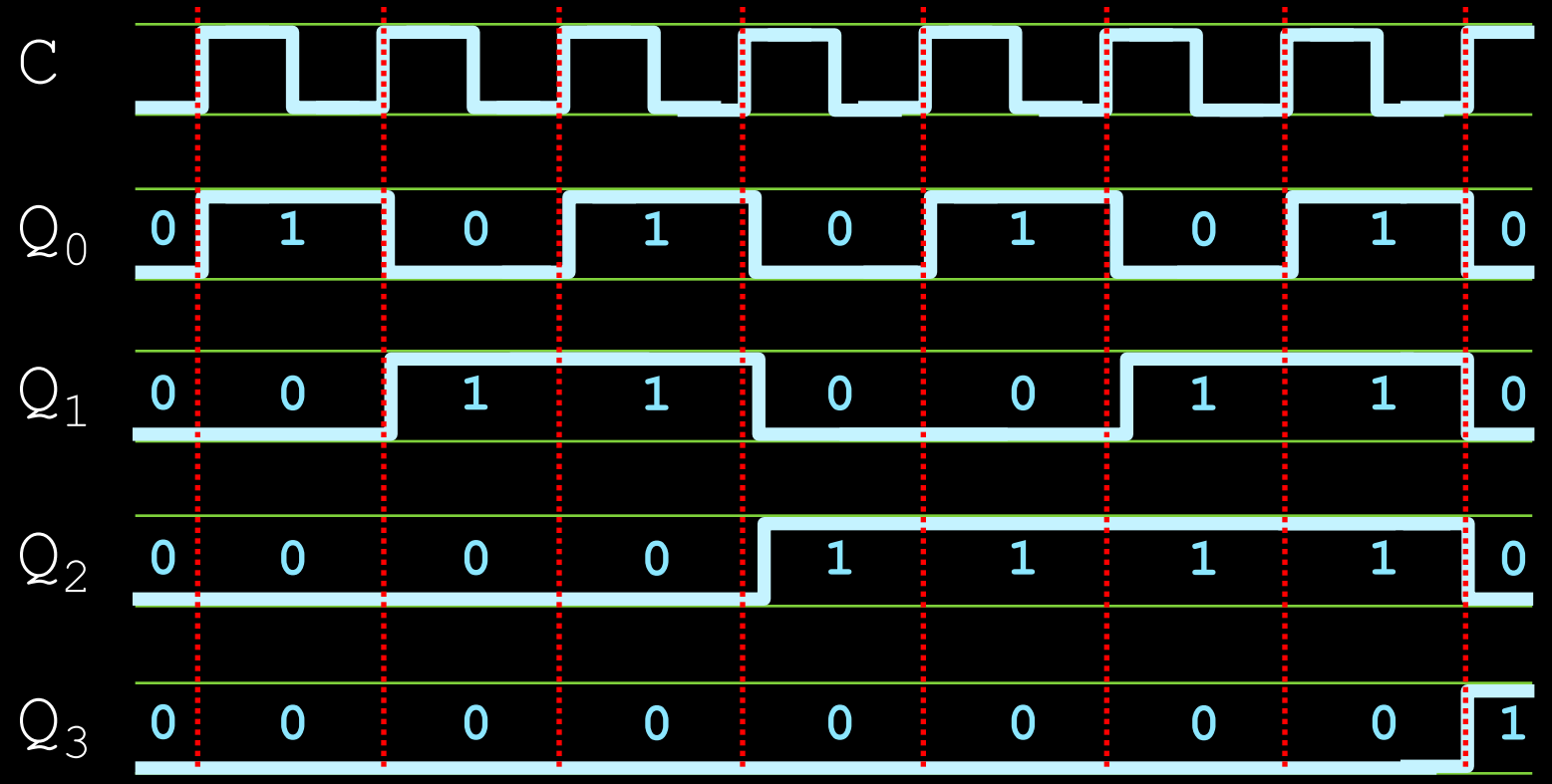


1. Need 4 volunteers, Q0, Q1, Q2, Q3
2. Q0 toggles when receives clock signal (tap on shoulder)
3. Q1 toggles when Q0 goes from 1 to 0
4. Q2 toggles when Q1 goes from 1 to 0
5. Q3 toggles when Q2 goes from 1 to 0
6. Audience read the number Q3Q2Q1Q0

Counters

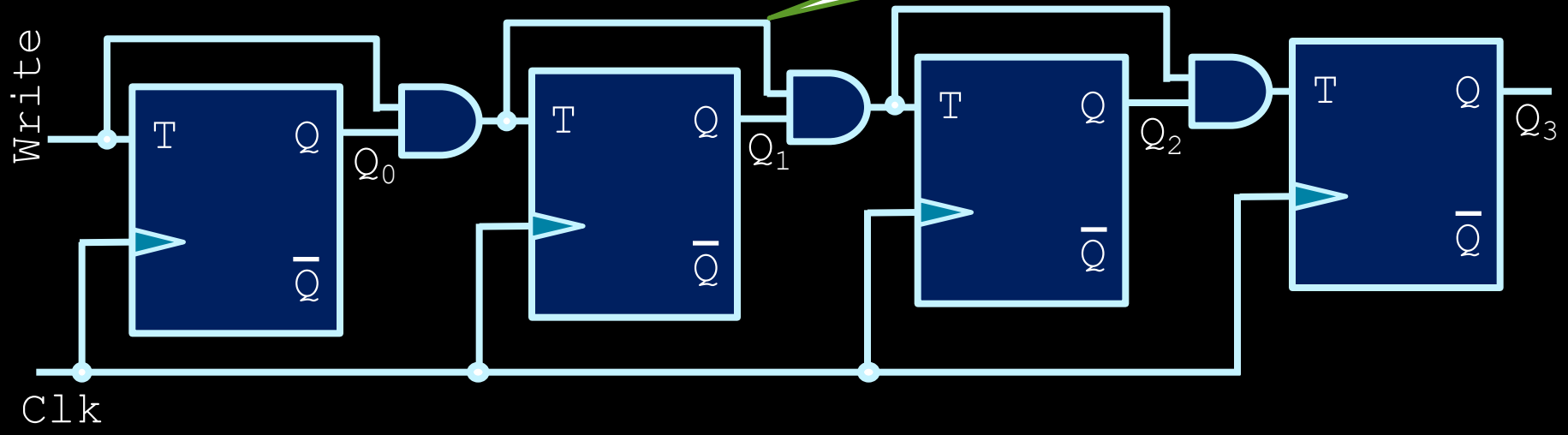
- Timing diagram





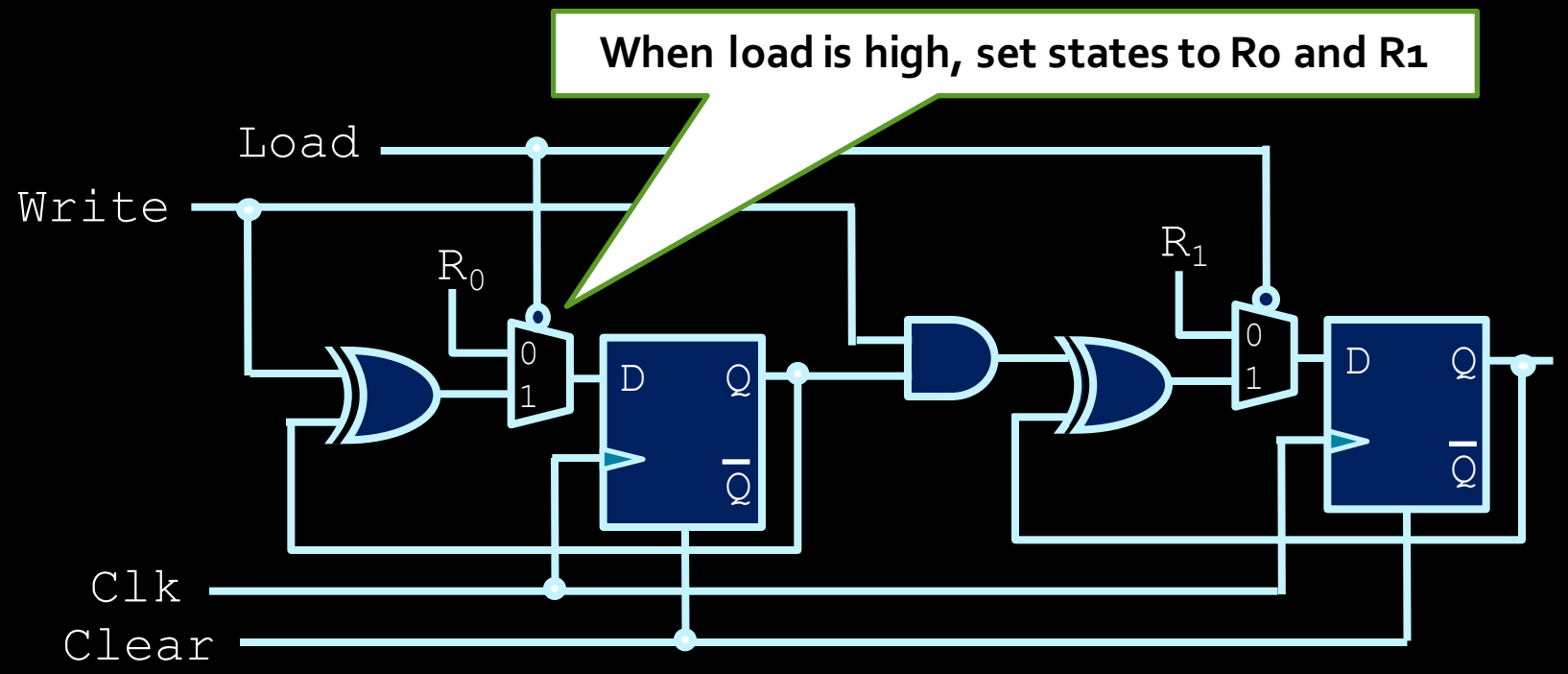
Synchronous Counter

Toggle only when the lower bit is 1 and is toggling to 0 (produce a carry)



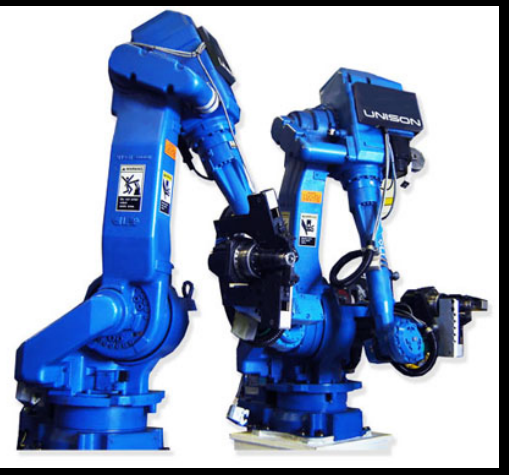
- This is a synchronous counter, because all output Q's change upon the same clock edge.

Counter with parallel load



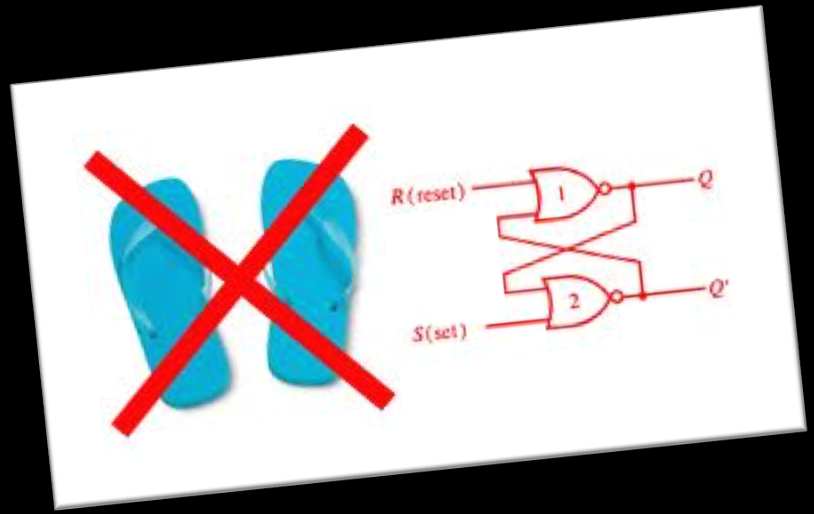
- Counters are often implemented with a parallel load and clear inputs.
  - Can set the counter to whatever value needed.

State Machines



Designing with flip-flops

- Counters and registers are examples of how flip-flops can implement useful circuits that store values.
  - How do you design these circuits?
  - What would you design with these circuits?

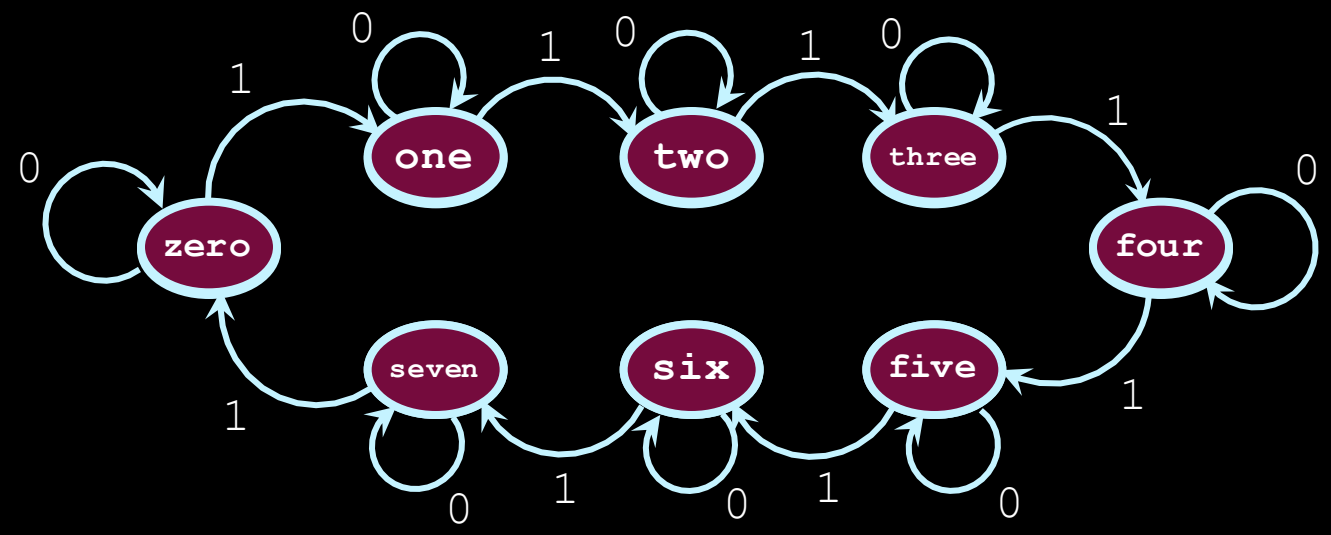


Designing with flip-flops

- Sequential circuits are the basis for memory, instruction processing, and any other operation that requires the circuit to remember past data values.
- These past data values are also called the states of the circuit.
- Need to describe the relation between the current state and the next state: use combinational circuits

State example: Counters

- With counters, each state is the current number that is stored in the counter.



Each state does not need to correspond to a binary number, they are just different states

- On each clock tick, the circuit transitions from one state to the next, based on the inputs.

State Tables

- State tables help to illustrate how the states of the circuit change with various input values.
  - Transitions are understood to take place on the clock ticks.

State	Write	State
zero	0	zero
zero	1	one
one	0	one
one	1	two
two	0	two
two	1	three
three	0	three
three	1	four
four	0	four
four	1	five
five	0	five
five	1	six
six	0	six
six	1	seven
seven	0	seven
seven	1	zero

State Tables

- Same table as on the previous slide, but with the actual flip-flop values instead of state labels.
  - Note: Flip-flop values are both inputs and outputs of the circuit here.

F_1	F_2	F_3	Write	F_1	F_2	F_3
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	1	0
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	1	0	0
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	1	1	0
1	1	0	0	1	1	0
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	0	0	0

and this brings us to…

Finite State Machines

As seen in other courses…

- You may have seen finite state machines before, but in a different context.
  - Used mainly to describe the grammars of a language, or to model sequence data.
- In CSC258, finite state machines are models for an actual circuit design.
  - The states represent internal states of the circuit, which are stored in the flip-flop values.

Finite State Machines (FSMs)

- From theory courses…
  - A Finite State Machine is an abstract model that captures the operation of a sequential circuit.
- A FSM is defined (in general) by:
  - A finite set of states,
  - A finite set of transitions between states, triggered by inputs to the state machine,
  - Output values that are associated with each state or each transition (depending on the machine),
  - Start and end states for the state machine.

Design procedures comparison (roughly)

Combinational circuits

1. Desired behaviour
2. Truth table
3. Logic expression
4. Circuit

Sequential circuits

1. Desired behaviour (cooler behaviour)
2. Finite state machine
3. Circuit with flip-flops

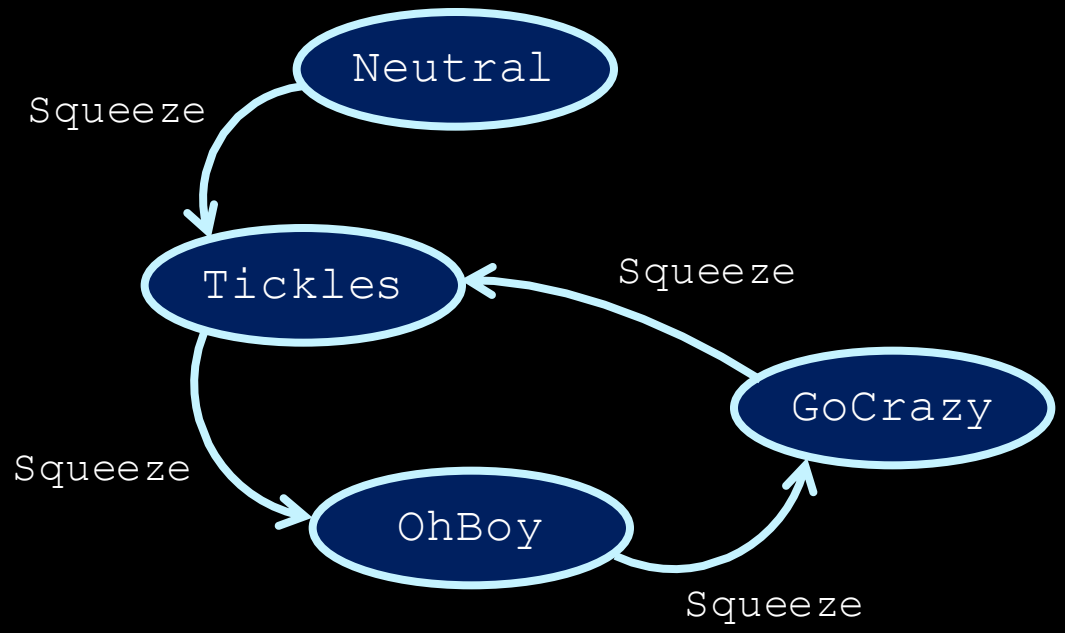
Example #1: Tickle Me Elmo

- Remember how the Tickle Me Elmo works!

Example #1: Tickle Me Elmo

- Toy reacts differently each time it is squeezed:
  - First squeeze → *"Ha ha ha…that tickles."*
  - Second squeeze → *"Ha ha ha…oh boy."*
  - Third squeeze → *"HA HA HA HA…", go crazy*
- Questions to ask:
  - What are the inputs?
  - What are the states of this machine?
  - How do you change from one state to the next?

Example #1: Tickle Me Elmo

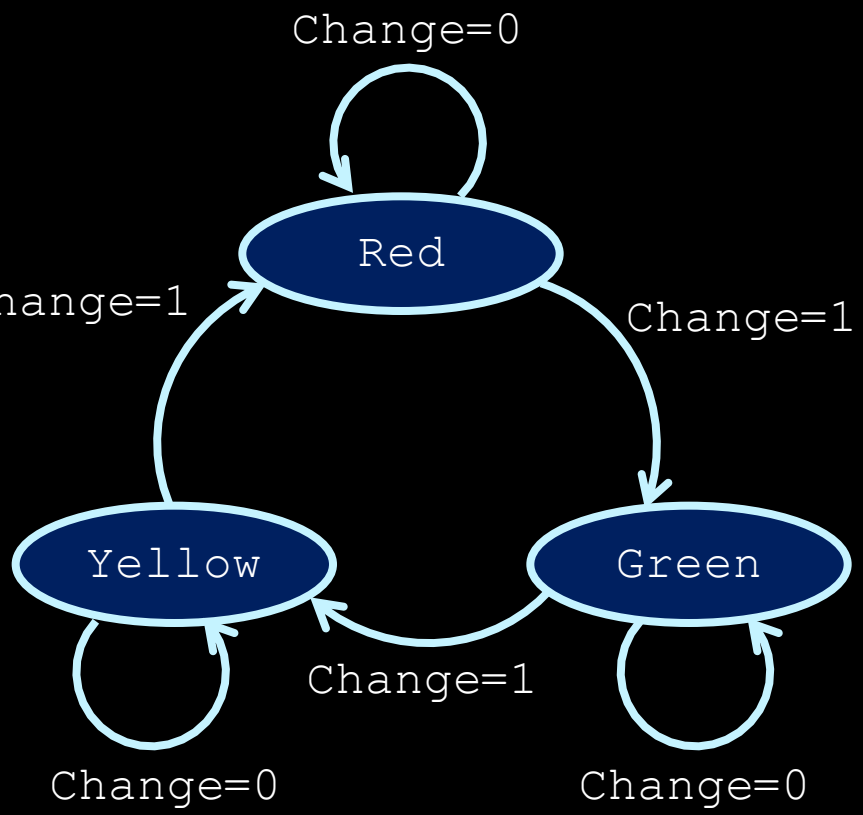


More elaborate FSMs

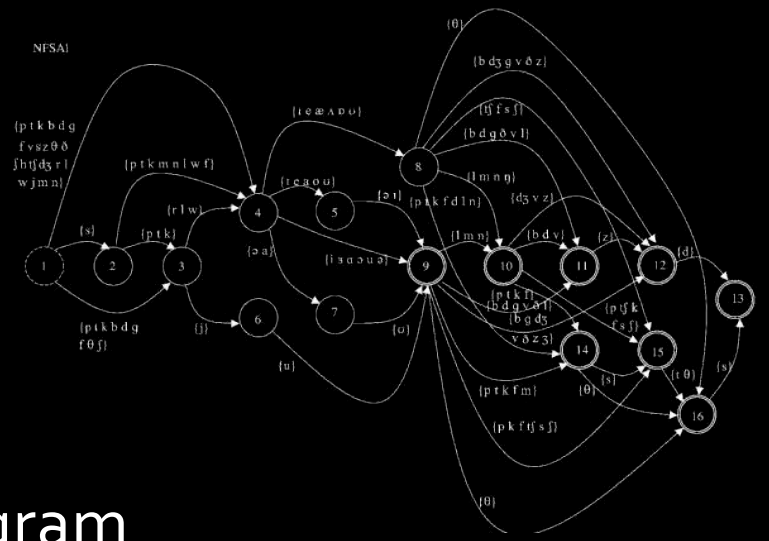
- Usually our FSM has more than one input, and will trigger a transition based on certain input values but not others.

- Also might have input values that don't cause a transition, but keep the circuit in the same state (transitioning to itself).

Example #2: Traffic Light



FSM design



- Design steps for FSM:
  1. Draw state diagram
  2. Derive state table from state diagram
  3. Assign flip-flop configuration to each state
     - Number of flip-flops needed is:
       - $\lceil \log_2(\#\text{ of states}) \rceil$
  4. Redraw state table with flip-flop values
  5. Derive combinational circuit for output and for each flip-flop input.

Example: Sequence Recognizer

- Recognize a sequence of input values, and raise a signal if that input has been seen.

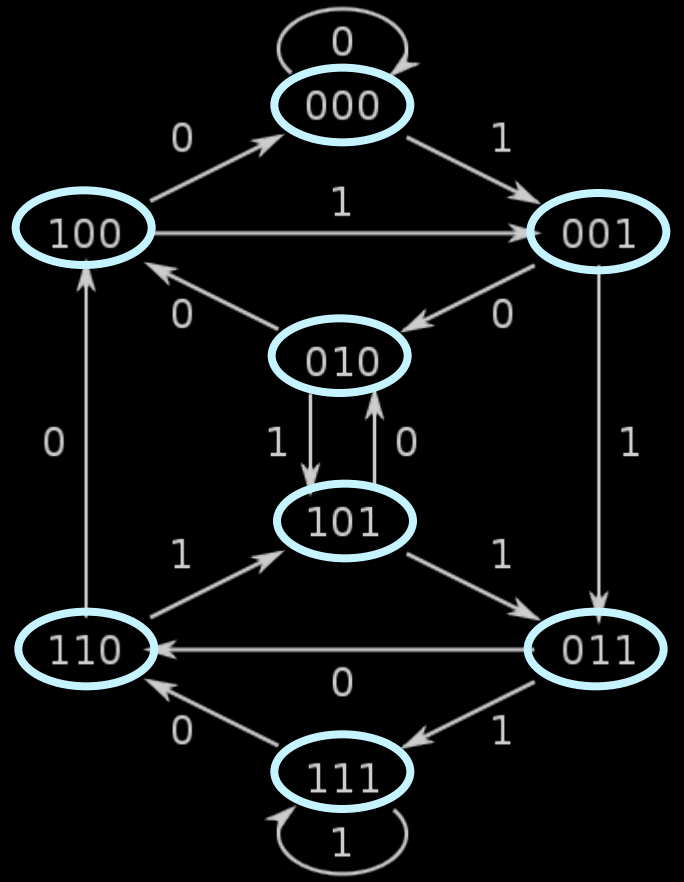
- Example: Three high values in a row
  - Detect that the input has been high for three rising clock edges.
  - Assumes a single input `X` and a single output `Z`.

What are the states?

Step 1: State diagram

- In this case, the states are labeled with the most recent three input values.

- Transitions between states are indicated by the values on the transition arrows.



Step 2: State table

- Make sure that the state table lists all the states in the state diagram, and all the possible inputs that can occur at that state.

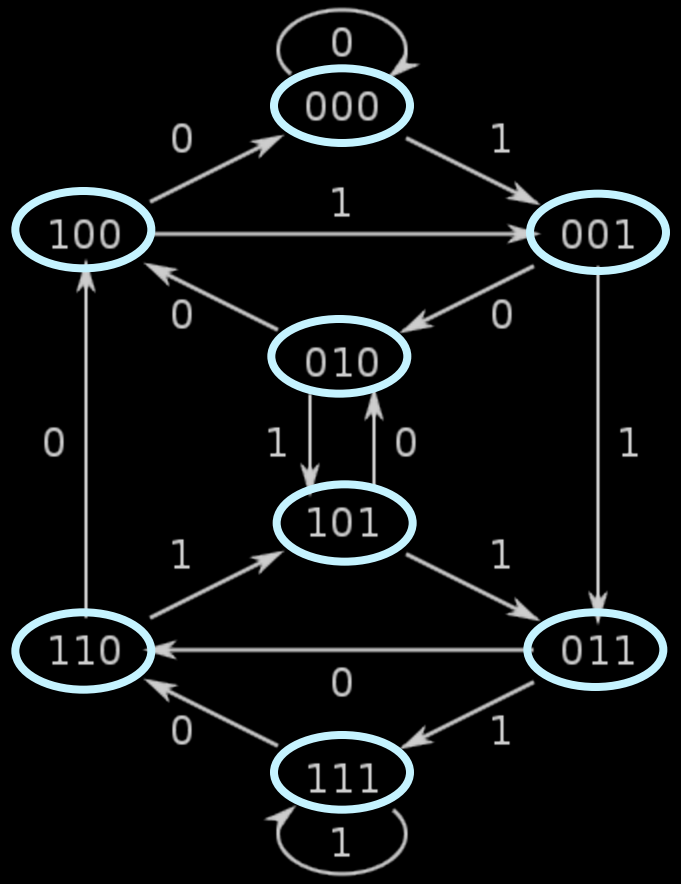
Previous State	EN	Next State
000	0	000
000	1	001
001	0	010
001	1	011
010	0	100
010	1	101
011	0	110
011	1	111
100	0	000
100	1	001
101	0	010
101	1	011
110	0	100
110	1	101
111	0	110
111	1	111

Step 3: Assign flip-flops

- Assign flip-flops for storing states.
- A single flip-flop can store two values (`0` and `1`), and thus two states.
- How many states can be stored with each additional flip-flop?
  - One flip-flop → 2 states
  - Two flip-flops → 4 states
  - Three flip-flops → 8 states
  - …
  - Eight flip-flops? → 2^8 = 256 states

n states need: $\lceil \log_2 n \rceil$ flip-flops

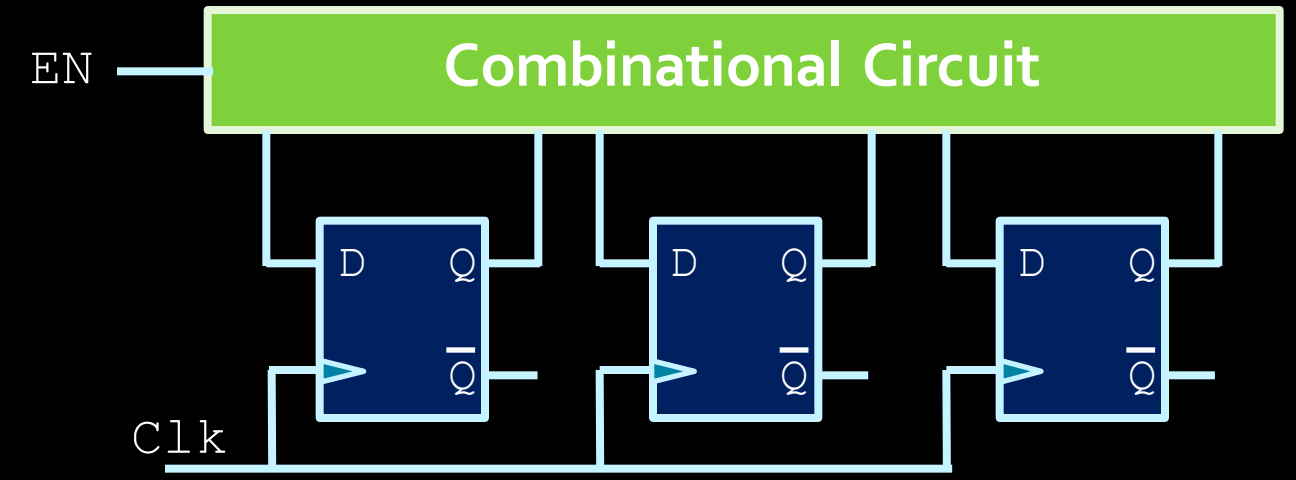
How many flip-flops for this one?



3

Step 3: Assign flip-flops

- In this case, we need to store 8 states.
  - 8 states = 3 flip-flops ($3 = \log_2 8$)
- For now, assign a flip-flop to each digit of the state names in the FSM & state table.



Step 4: State table

- Mapping states to flip-flop values
- This is NOT the only way of mapping from state to flip flop values, in fact it is not even a good way, as we will see later.

Prev. State	EN	Next State
000	0	000
000	1	001
001	0	001
001	1	010
010	0	010
010	1	011
011	0	011
011	1	100
100	0	100
100	1	101
101	0	101
101	1	110
110	0	110
110	1	111
111	0	111
111	1	000

Step 4: State table

- Mapping states to flip-flop values
- This is NOT the only way of mapping from state to flip flop values, in fact it is not even a good way, as we will see later.

F_2	F_1	F_0	EN	F_2	F_1	F_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	1	0	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	0	1	1
1	1	0	0	1	0	0
1	1	0	1	1	0	1
1	1	1	0	1	1	0
1	1	1	1	1	1	1

Step 5: Circuit design

- Karnaugh map for F_2:

	$\overline{F_0} \cdot \overline{EN}$	$\overline{F_0} \cdot EN$	$F_0 \cdot EN$	$F_0 \cdot \overline{EN}$
$\overline{F_2} \cdot \overline{F_1}$	0	0	0	0
$\overline{F_2} \cdot F_1$	1	1	1	1
$F_2 \cdot F_1$	1	1	1	1
$F_2 \cdot \overline{F_1}$	0	0	0	0

$F_2 = F_1$

Next state

Current state

Step 5: Circuit design

- Karnaugh map for F_1:

	$\overline{F_0} \cdot \overline{EN}$	$\overline{F_0} \cdot EN$	$F_0 \cdot EN$	$F_0 \cdot \overline{EN}$
$\overline{F_2} \cdot \overline{F_1}$	0	0	1	1
$\overline{F_2} \cdot F_1$	0	0	1	1
$F_2 \cdot F_1$	0	0	1	1
$F_2 \cdot \overline{F_1}$	0	0	1	1

$F_1 = F_0$

Next state

Current state

Step 5: Circuit design

- Karnaugh map for F_0:

	$\overline{F_0} \cdot \overline{EN}$	$\overline{F_0} \cdot EN$	$F_0 \cdot EN$	$F_0 \cdot \overline{EN}$
$\overline{F_2} \cdot \overline{F_1}$	0	1	1	0
$\overline{F_2} \cdot F_1$	0	1	1	0
$F_2 \cdot F_1$	0	1	1	0
$F_2 \cdot \overline{F_1}$	0	1	1	0

$F_0 = EN$

Next state

Current state

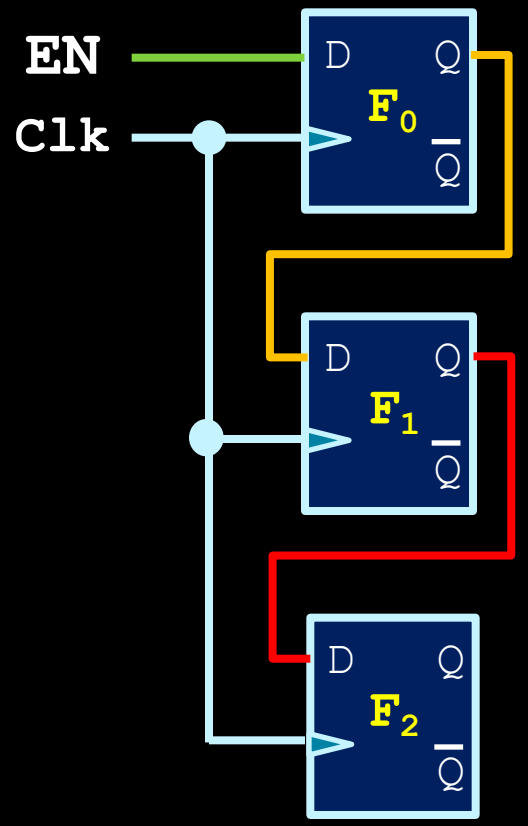
$F_2 = F_1$

$F_1 = F_0$

$F_0 = EN$

Step 5: Circuit design

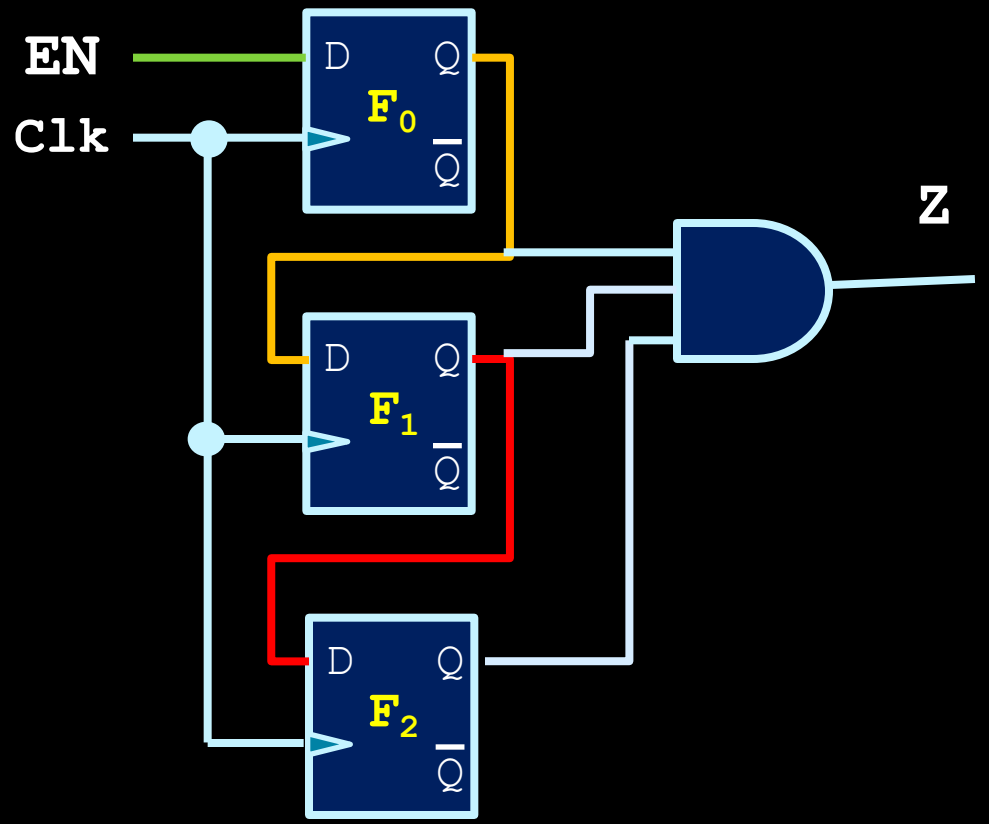
- Resulting circuit looks like the diagram on the right.
- This will record the states and make the state transitions happen based on the input,
- What about the output value Z which should go high when we have three highs in a row.



Step 5: Circuit design

- Boolean equation for Z:

$$Z = F_0 \cdot F_1 \cdot F_2$$

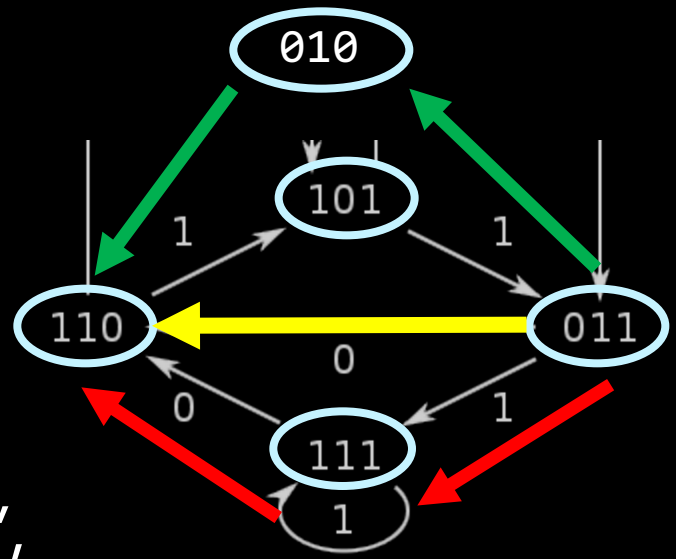


Moore machine vs Mealy machine

- Two ways to derive the circuitry needed for the output values of the state machine:
  - Moore machine:
    - The output for the FSM depends solely on the current state (based on entry actions).
  - Mealy machine:
    - The output for the FSM depends on the state and the input (based on input actions).
    - Being in state X can result in different output, depending on the input that caused that state.

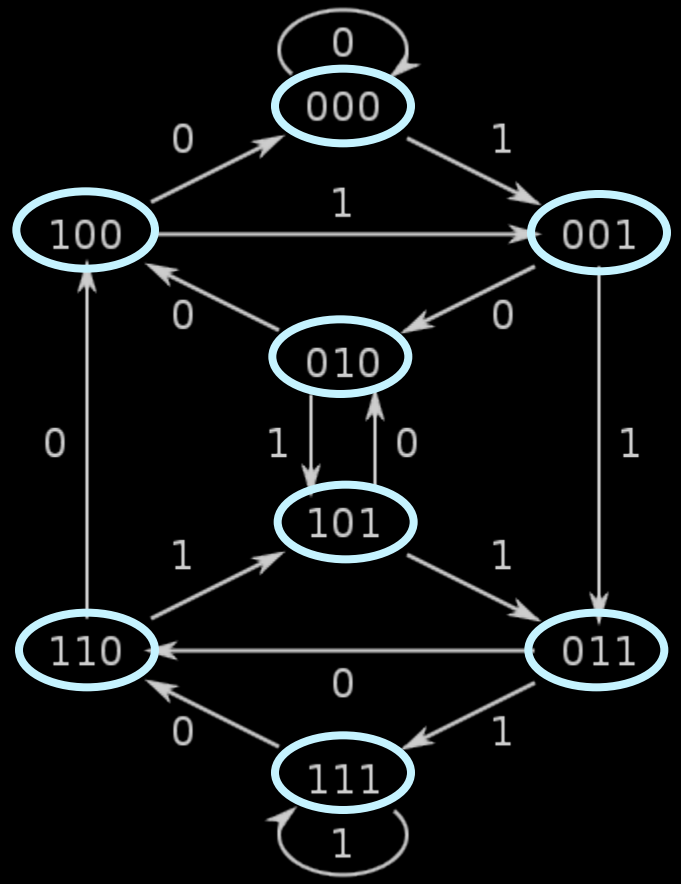
An issue: timing and state assignments

- Example: if recognizer circuit is in state 011 and gets a 0 as an input, it moves to state 110.
  - The first and last digits should change "at the same time", but they can't.
  - If the first flip-flop changes first, the state will change to 111, and the output Z would go high for an instant, which is unexpected behaviour.
  - If the second flip-flop changes first, it's fine since the output would not go wrong.



An issue: timing and state assignments

- So how do you solve this?
- Two possible solutions:
  1. Whenever possible, make flip-flop assignments such that neighbouring states differ by at most one flip-flop value.
     - Intermediate states can be allowed if the output generated by those states is consistent with the output of the starting or destination states.
  2. If the intermediate states are unused in the state diagram, you can set the output for these states to provide the output that you need.
     - Might need to add more flip-flops to create these states.



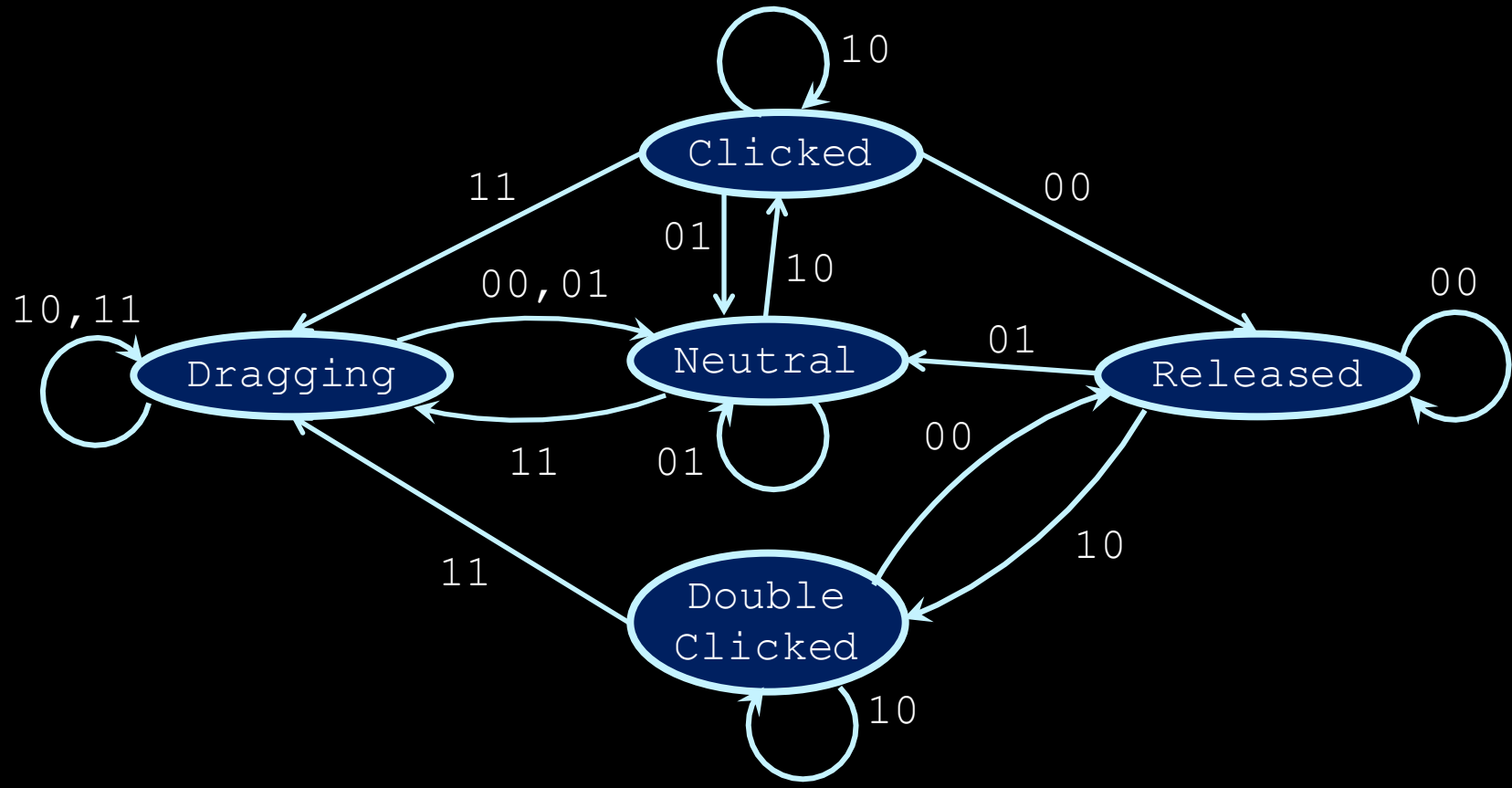
Previous State	EN	Next State
000	0	000
000	1	001
001	0	010
001	1	011
010	0	100
010	1	101
011	0	110
011	1	111
100	0	000
100	1	001
101	0	010
101	1	011
110	0	100
110	1	101
111	0	110
111	1	111

State ooo does not have to have flip-flop values ooo, it can be anything you want to assign.

Home exercise: re-assign the states so the time issue doesn't exist

Another example: Mouse clicks

- Design a circuit that takes in two signals:
  - A signal P, which is high if the user is pressing the mouse button,
  - A signal M, which is high if the mouse is being moved.
- Based on the inputs, indicate whether the user is clicking, double-clicking, or dragging the mouse on the screen.



- Transitions indicate the values of P&M.
- Outputs depend on the state (Moore machine)

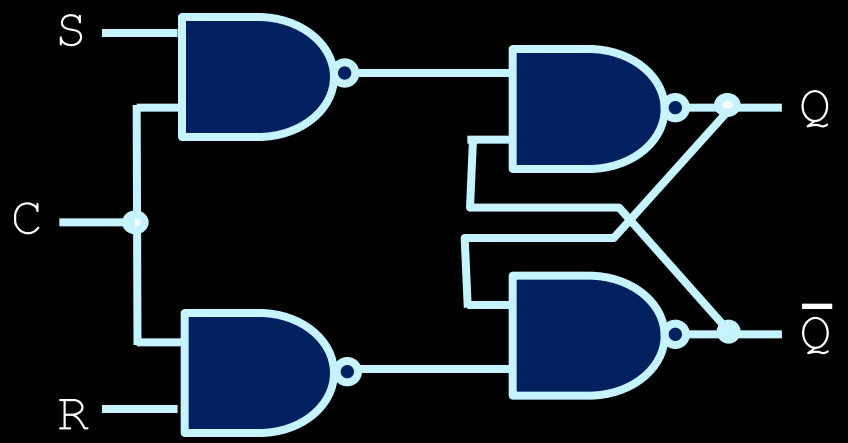
Next week

processor architecture


KEEP
CALM
IT'S
QUIZ
TIME

Q1: What is the improvement made by D latch compare to SR latch?

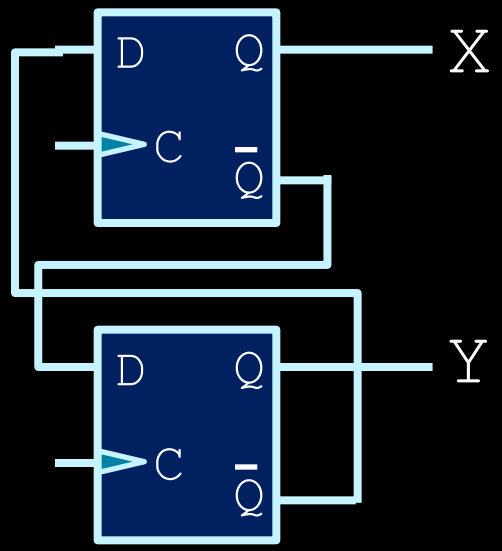
A. avoid oscillation
B. avoid forbidden state
C. become edge-triggered
D. none of above



Q2: What is the Q and Q' when S = 0, R = 1, C = 1

A. Q = 1, Q' = 0
B. Q = 0, Q' = 1
C. Q = 1, Q' = 1
D. indeterminate
E. none of above

Q3



Assuming the Q outputs of both flip-flops start off **low**, and assume positive edge trigger.

Draw the waveforms of X and Y.

(assume that signals change without delay)