CPSC 121: Models of Computation

Unit 8: Sequential Circuits

Based on slides by Patrice Belleville and Steve Wolfman

Pre-Class Learning Goals

- By the start of class, you should be able to
  - Trace the operation of a DFA (deterministic finite-state automaton) represented as a diagram on an input, and indicate whether the DFA accepts or rejects the input.
  - Deduce the language accepted by a simple DFA after working through multiple example inputs.

Quiz 8 feedback:

- Over all:
- Issues :
- Push-button light question:
  - We will revisit this problem soon.

In-Class Learning Goals

- By the end of this unit, you should be able to:
  - Translate a DFA into a sequential circuit that implements the DFA.
  - Explain how and why each part of the resulting circuit works.

Related to CPSC 121 Bib Questions

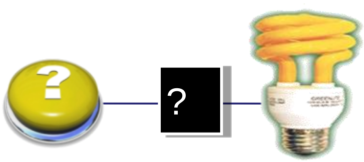
- How can we build a computer that is able to execute a user-defined program?
  - Computers execute instructions one at a time.
  - They need to remember values, unlike the circuits you designed in labs 1, 2, 3 and 4.
- NOW: We are learning to build a new kind of circuits with memory that will be the key new feature we need to build full-blown computers!

Unit Outline

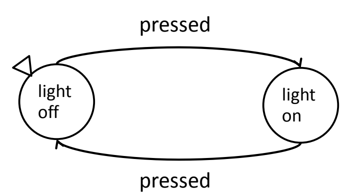
- **Sequential Circuits :Latches, and flip-flops.**
- DFA Example : Branch prediction.
- Implementing DFAs
- How Powerful are DFAs?
- Other problems and exercises.

Problem: Light Switch

- Problem:
  - Design a circuit to control a light so that the light changes state any time its "push-button" switch is pressed.



DFA for Push-Button Switch



This Deterministic Finite Automaton (DFA) isn't really about accepting/rejecting; its current state is the state of the light.

Problem: Light Switch

Problem: Design a circuit to control a light so that the light changes state any time its “push-button” switch is pressed.

Identifying inputs/outputs: consider these possible inputs and outputs:

Input$_1$: the button was pressed
Input$_2$: the button is down
Output$_1$: the light is on
Output$_2$: the light changed states

Which are most useful for this problem?

a. **Input**$_1$ and **Output**$_1$
b. **Input**$_1$ and **Output**$_2$
c. **Input**$_2$ and **Output**$_1$
d. **Input**$_2$ and **Output**$_2$
e. None of these

Departures from Combinational Circuits

- MEMORY:
  We need to “remember” the light’s state.

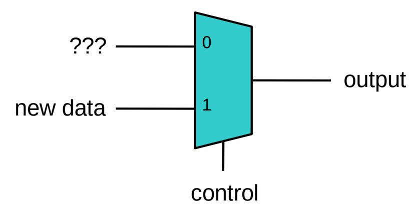
- EVENTS:
  We need to act on a button push rather than in response to an input value.

How Do We Remember?

- We want a circuit that:
  - Sometimes… remembers its current state.
  - Other times… loads a new state and remembers it.
- Sounds like a *choice*.
- What circuit element do we have for modelling choices?

“Mux Memory”

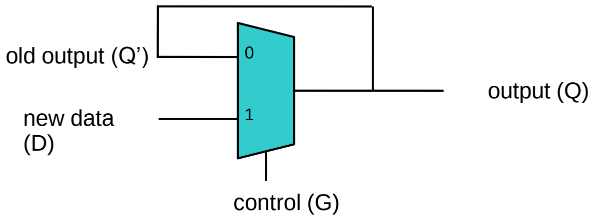
- How do we use a mux to store a bit of memory?
- We choose to remember on a control value of 0 and to load a new state on a 1.



We use “0” and “1” because that’s how MUXes are usually labelled.

“Mux Memory”

- How do we use a mux to store a bit of memory?
- We choose to remember on a control value of 0 and to load a new state on a 1.



This *violates* our basic combinational constraint: no cycles.

Truth Table for “Muxy Memory”

Fill in the MM’s truth table:

G	D	Q'	a. Q	b. Q	c. Q	d. Q
0	0	0	0	0	0	0
0	0	1	1	1	1	0
0	1	0	0	0	0	0
0	1	1	1	1	1	1
1	0	0	0	0	0	1
1	0	1	1	0	X	1
1	1	0	0	1	X	0
1	1	1	1	1	1	1

e. None of these

Truth Table for “Muxy Memory”

Worked Problem: Write a truth table for the MM:

G	D	Q'	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Like a “normal” mux table, but what happens when Q' ≠ Q?

Truth Table for “Muxy Memory”

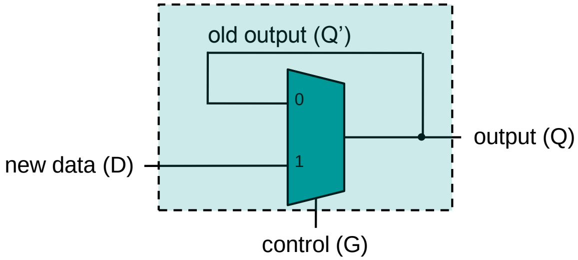
Worked Problem: Write a truth table for the MM:

G	D	Q'	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Q' “takes on” Q’s value at the “next step”.

D Latches

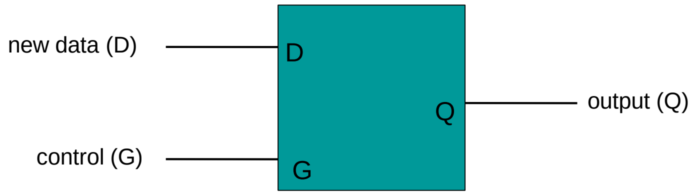
- We call a "mux-memory" a D-latch (recall from lab #5)
  - When G is 0, the latch retains its current value.
  - When G is 1, the latch loads a new value from D.



D Latch

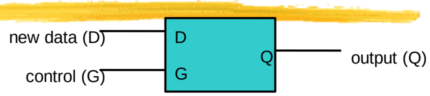
When G is 0, the latch maintains its memory.

When G is 1, the latch loads a new value from D.



D-Latch

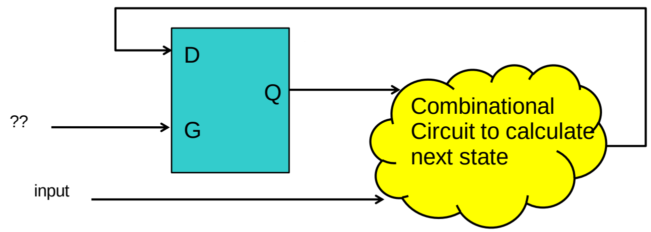
- A D-latch looks like



- Why does the D Latch have two inputs and one output when the mux inside has THREE inputs and one output?
  - A. The D Latch is broken as is; it should have three inputs.
  - B. A circuit can always ignore one of its inputs.
  - C. One of the inputs is always true.
  - D. One of the inputs is always false.
  - E. None of these (but the D Latch is not broken as is).

Using the D Latch for Circuits with Memory

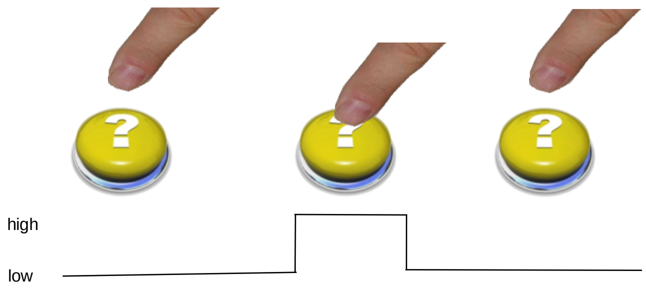
Problem: What goes in the cloud? What do we send into G?



We assume we just want Q as the output.

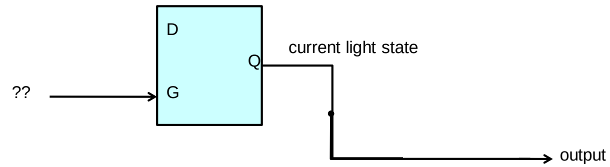
Push-Button Switch

- What signal does the button generate?



Using the D Latch for Our Light Switch

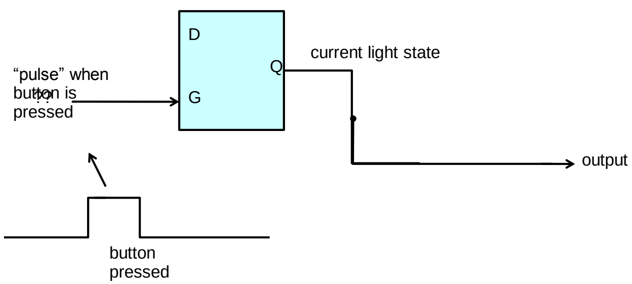
Problem: What do we send into G?



a. T if the button is down, F if it's up.
b. T if the button is up, F if it's down.
c. Neither of these.

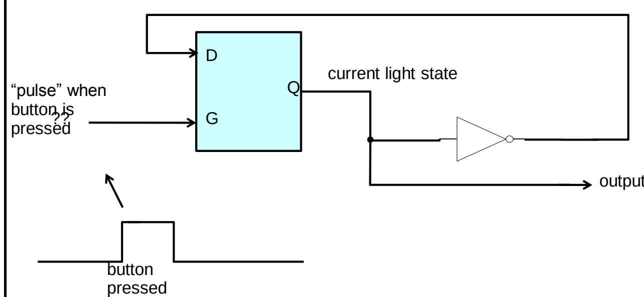
Using the D Latch for Our Light Switch

Problem: What should be the next state of the light?



Using the D Latch for Our Light Switch

Problem: Will this work?

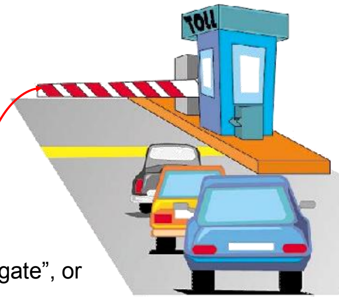


Push-Button Switch

- What is wrong with our solution?
  - A. We should have used XOR instead of NOT.
  - B. As long as the button is down, D flows to Q flows through the NOT gate and back to D...which is bad!
  - C. The delay introduced by the NOT gate is too long.
  - D. As long as the button is down, D flows to Q, and it flows through the NOT gate and back to D... which is bad!
  - E. There is some other problem with the circuit.

A Timing Problem

- This toll booth has a similar problem.
- What is wrong with this booth?



P.S. Call this a “bar”, not a “gate”, or we'll tie ourselves in (k)nots.

From MIT 6.004, Fall 2002

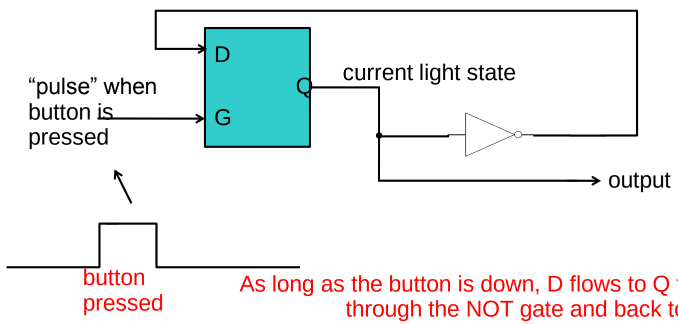
A Timing Solution

- Is this OK?

From MIT 6.004, Fall 2002

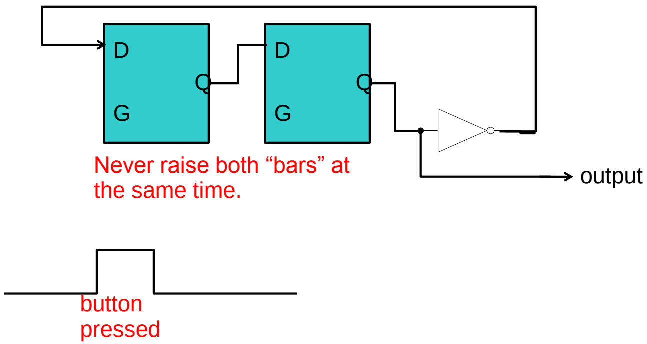
A Timing Problem

Problem: What do we send into G?



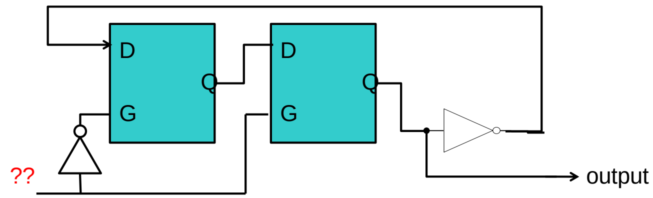
As long as the button is down, D flows to Q flows through the NOT gate and back to D... which is bad!

A Timing Solution (Almost)



29

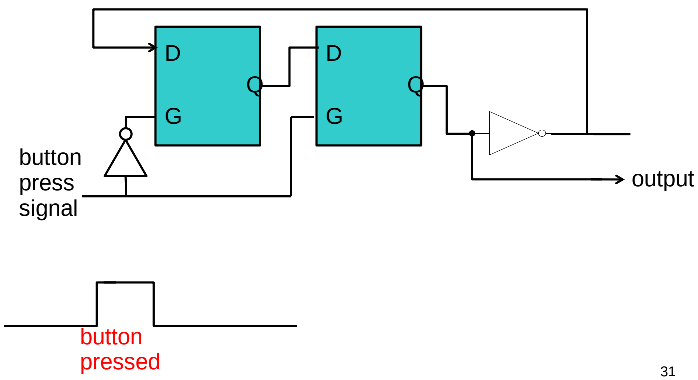
A Timing Solution



The two latches are never enabled at the same time (except for the moment needed for the NOT gate on the left to compute, which is so short that no “cars” get through).

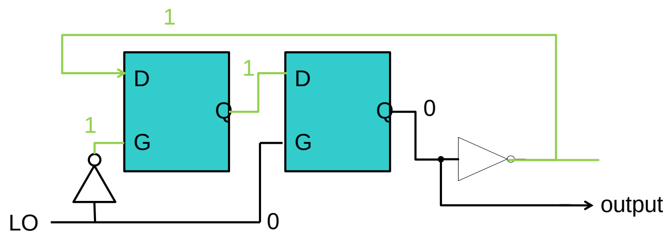
30

A Timing Solution



31

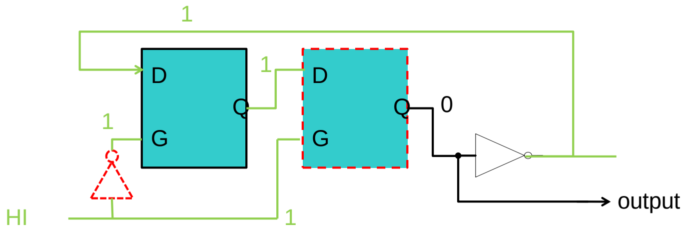
Button/Clock is LO (unpressed)



We’re assuming the circuit has been set up and is “running normally”. Right now, the light is off (i.e., the output of the right latch is 0).

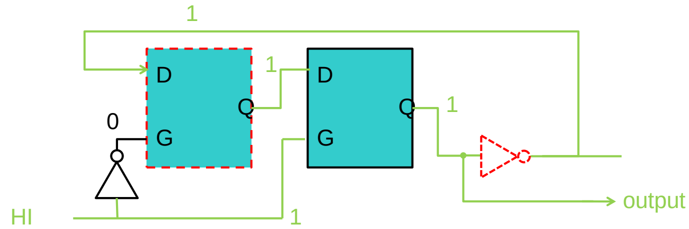
32

Button goes HI (is pressed)



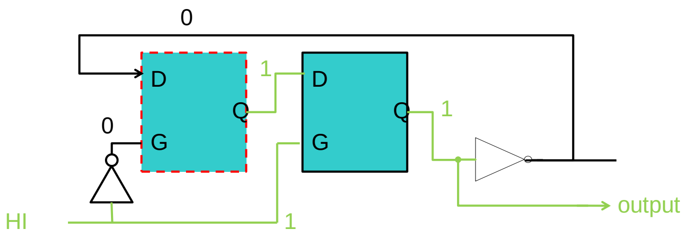
This stuff is processing a new signal.

Propagating signal..
left NOT, right latch



This stuff is processing a new signal.

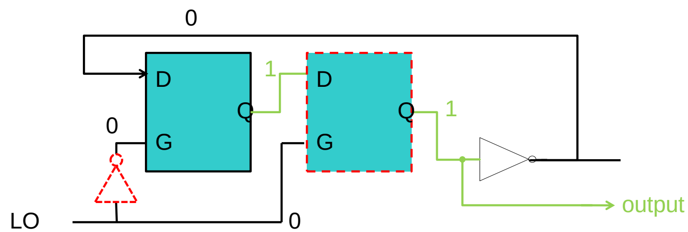
Propagating signal..
right NOT (steady state)



Why doesn't the left latch update?

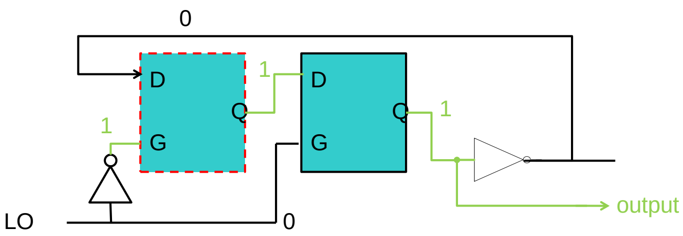
a. Its D input is 0.
b. Its G input is 0.
c. Its Q output is 1.
d. It should update!

Button goes LO (released)



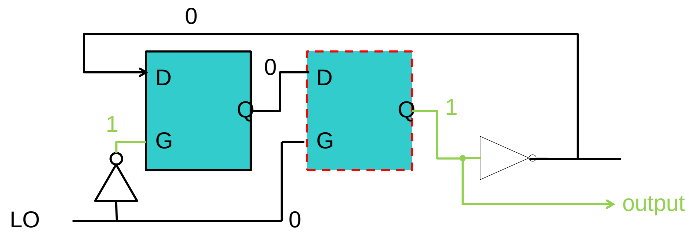
This stuff is processing a new signal.

Propagating signal.. left NOT



This stuff is processing a new signal.

Propagating signal.. left latch (steady state)

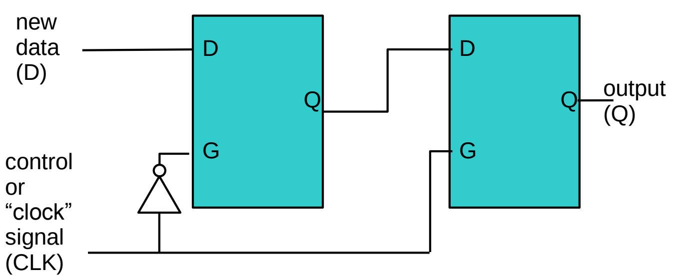


And, we're done with one cycle.
How does this compare to our initial state?

Master/Slave D Flip-Flop Symbol + Semantics

When CLK goes from 0 (low) to 1 (high), the flip-flop loads a new value from D.

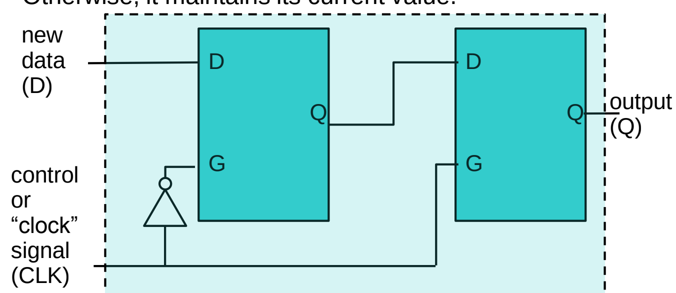
Otherwise, it maintains its current value.



Master/Slave D Flip-Flop Symbol + Semantics

When CLK goes from 0 (low) to 1 (high), the flip-flop loads a new value from D.

Otherwise, it maintains its current value.



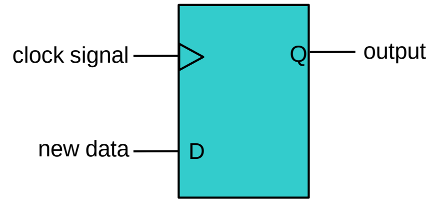
Master/Slave D Flip-Flop Symbol + Semantics

When CLK goes from 0 (low) to 1 (high), the flip-flop loads a new value from D.

Otherwise, it maintains its current value.

Master/Slave D Flip-Flop Symbol + Semantics

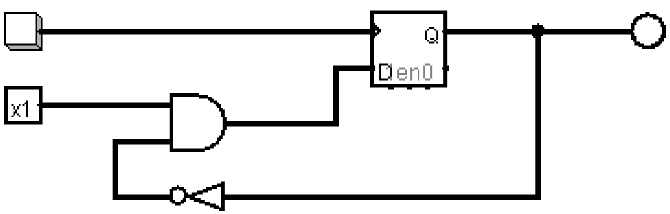
- When CLK goes from 0 (low) to 1 (high), the flip-flop loads a new value from D.
- Otherwise, it maintains its current value.



We rearranged the clock and D inputs and the output to match Logisim. Below we use a slightly different looking flip-flop.

Push-Button Switch: Solution

- Using a D- flip-flop



Why Abstract?

Logisim (and real circuits) have lots of flip-flops that all behave very similarly:

- D flip-flops,
- T flip-flops,
- J-K flip-flops,
- and S-R flip-flops.

They have slightly different implementations… and one could imagine brilliant new designs that are radically different inside.

Abstraction allows us to build a good design at a high-level without worrying about the details.

Plus… it means *you* only need to learn about D flip-flops' guts. The others are similar enough so we can just take the abstraction for granted.

Unit Outline

- Sequential Circuits :Latches, and flip-flops.
- **DFA Example : Branch prediction.**
- Implementing DFAs
- How Powerful are DFAs?
- Other problems and exercises.

Finite-State Automata

There are two types of Finite-State Automata:

- Those whose output is determined solely by the final state (Moore machines).
  - Used to match a string to a pattern.
    - Input validation.
    - Searching text for contents.
    - Lexical Analysis: the first step in a compiler or an interpreter.
      - (define (fun x) (if (<= x 0) 1 (* x (fun (- x 1)))))

(	define	(	fun	x	)	(	if	(	<=	x	0	)	1	(	*	x	(	fun	(	-	x	1	)	)	)	)	)

Finite-State Automata

- Those that produce output every time the state changes (Mealy machines).
  - Examples:
    - Simple ciphers
    - Traffic lights controller.
    - Predicting branching in machine-language programs
- A circuit that implements a finite state machine of either type needs to remember the current state:
  - It needs memory.

Computer Instructions

- How do computers really execute programs?
  - Programs written in a high-level language (Racket, Java) are translated into machine language.
  - A machine-language program is a sequence of very simple instructions.
    - Each instruction is a sequence of 0s and 1s.
    - Each instruction also has a human-readable version
      - Humans don't like looking at long sequences of 0s and 1s.
      - The human-readable version is not actually part of the program.
  - After it’s done with an instruction, the computer (usually) executes the next instruction in the list

Computer Instructions

- Example (modified to make it easier to understand):
  1. sum ← 0
  2. is n = 0?
  3. if true go to 7
  4. sum ← sum + n
  5. n ← n – 1
  6. goto 2
  7. halt
- Some instructions like instruction 3 (called ***branch instructions***) may tell the computer that the next instruction to execute is not the next in the sequence (4), but elsewhere (7).

Computer Instructions

- To speed things up, a modern computer starts executing an instruction before the previous one is finished.
- This means that when it is executing
  - if true go to 7

  it does not yet know if the condition is true, and hence does not know if the next instruction is
  - sum ← sum + n
  - or instruction number 7.

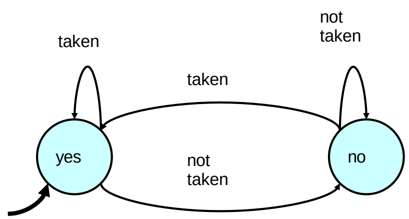
Branch Prediction: Simple Guess

- So we want to be able to predict the outcome of a branch instruction.
  - If we guess wrong, then we will ignore some of the work that was done.
- To pre-execute a "branch", the computer **guesses** which instruction comes next.
- Here's one reasonable guess:
  If the last branch was "taken" (like going to (7) from (3)), take the next.
  If it was "not taken" (like going to (4) from (3)), don't take the next.

Why? In recursion, how often do we hit the base case vs. the recursive case?

Branch Prediction: Simple Guess

- Here's the corresponding DFA. (Instead of accept/reject, we care about the current state.)



taken ⇒	the last branch was taken
not taken ⇒	the last branch was ***not*** taken
yes ⇒	we predict the next branch will be taken
no ⇒	we predict the next branch will be ***not*** taken

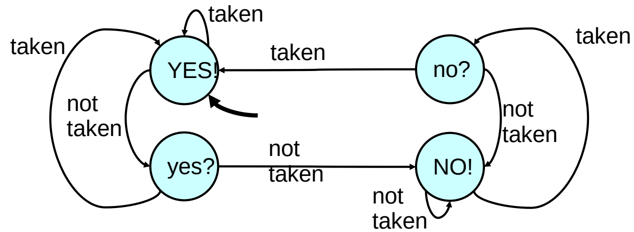
Experiments show it generally works well to add "inertia" so that it takes two "wrong guesses" to change the prediction…

DFA Example: Branch Prediction

- Suppose we keep guessing the same until we get two wrong guesses in a row and change our decision. How many states will the Finite State Automaton have now?
  - A. 2
  - B. 4
  - C. 8
  - D. Another value less than 8.
  - E. Another value larger than 8.

Branch Prediction: Using Confidence

Here's a version that takes **two wrong guesses in a row** to admit it's wrong:



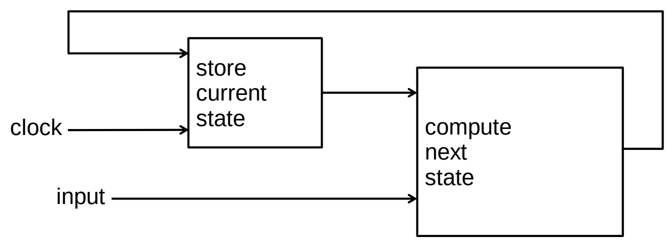
Can we build a branch prediction circuit?

Unit Outline

- Latches, toggles and flip-flops.
- DFA Example : Branch prediction.
- **Implementing DFAs**
- Other problems and exercises.

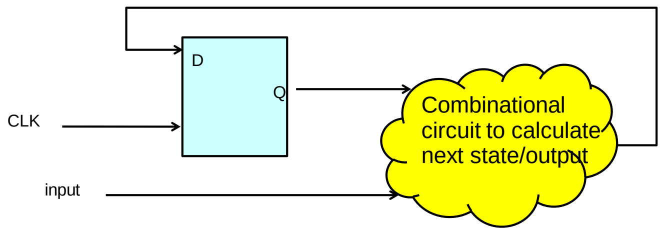
Abstract Template for a DFA Circuit

- Each time the clock "ticks" move from one state to the next.



Template for a DFA Circuit

- Each time the clock "ticks" move from one state to the next.



Each of these lines (except the clock) may carry multiple bits; the D flip-flop may be several flip-flops to store several bits.

Implementing DFAs in General

(1) Number the states and figure out **b**: the number of bits needed to store the state number.

(2) Lay out **b** D flip-flops. Together, their memory is the state as a binary number.

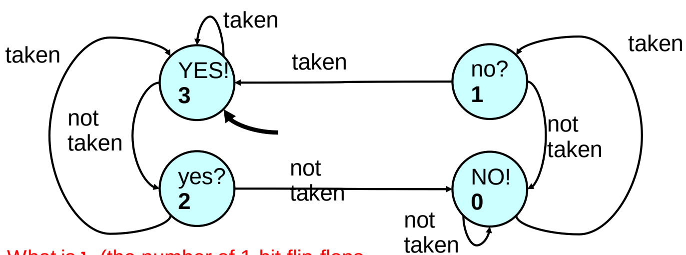
(3) ***For each state***, build a ***combinational circuit*** that determines the *next state* given the input.

(4) Send the *next states* into a MUX with the current state as the control signal: only the appropriate *next state* gets used!

(5) Use the MUX's output as the new state of the flip-flops.

With a **separate** circuit for each state, they're often very simple!

Implementing the Predictor: Step 1



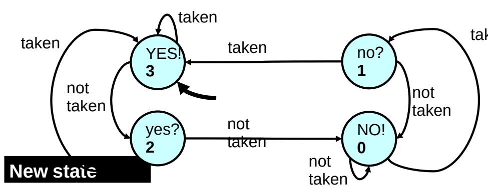
What is **b** (the number of 1-bit flip-flops needed to represent the state)?

a. 0, no memory needed
b. 1
c. 2
d. 3
e. None of these

As always, we use numbers to represent the inputs:
taken = 1
not taken = 0

Just Truth Tables...

Reminder: not taken = 0
taken = 1



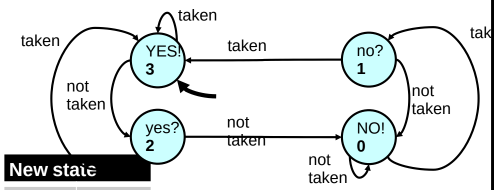
Current State		input	New state	
0	0	0	0	0
0	0	1	0	1
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

What's in this row?

a. 0 0
b. 0 1
c. 1 0
d. 1 1
e. None of these.

Just Truth Tables...

Reminder: not taken = 0
taken = 1

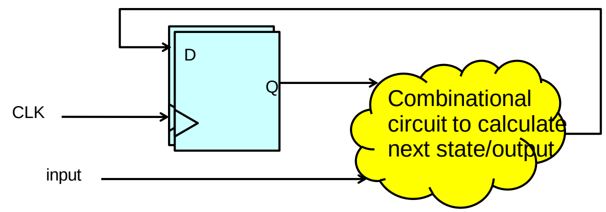


Current State		input	New state	
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

Implementing the Predictor: Step 3

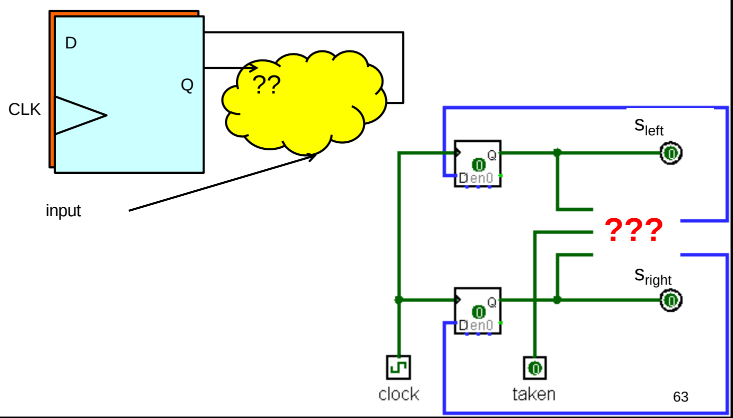
We always use this pattern.

In this case, we need two flip-flops.

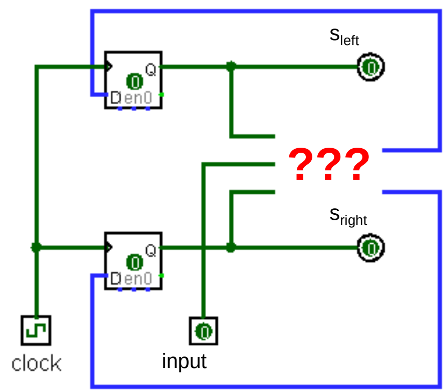


Let's switch to Logisim schematics...

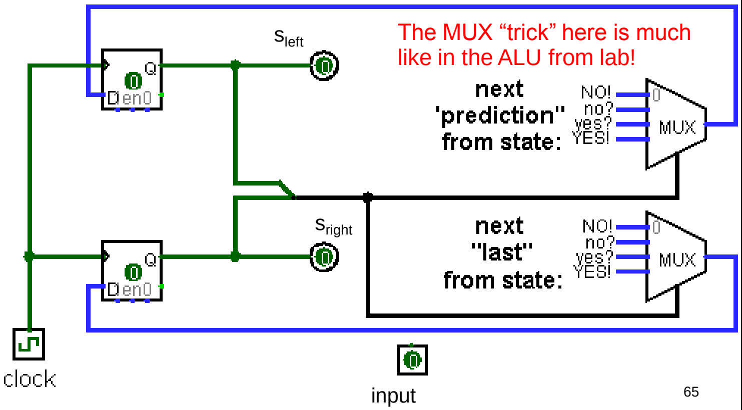
Implementing the Predictor: Step 3



Implementing the Predictor: Step 3

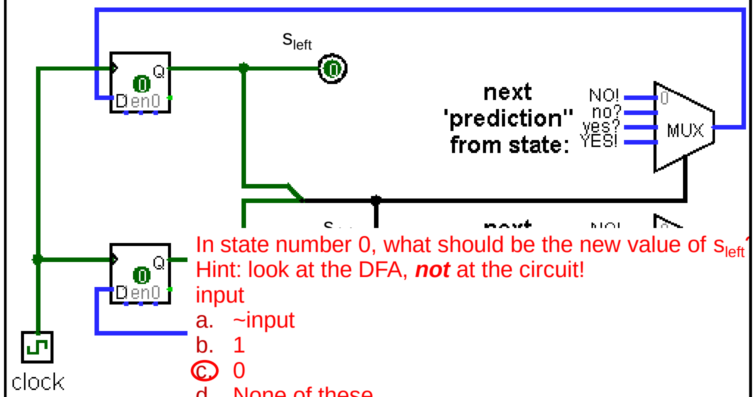


Implementing the Predictor: Step 5
(easier than 4!)



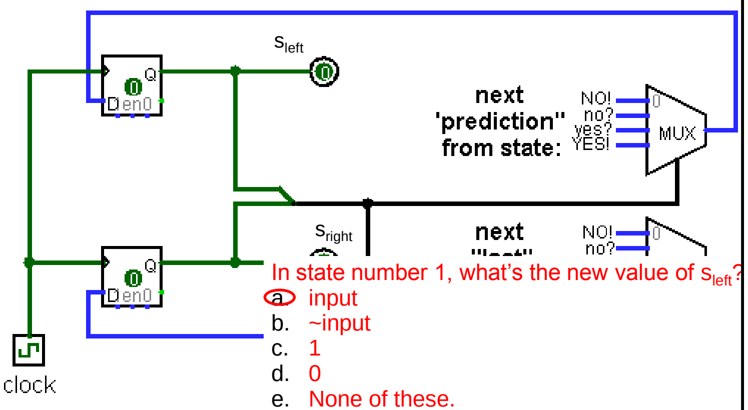
The MUX “trick” here is much like in the ALU from lab!

Implementing the Predictor: Step 4



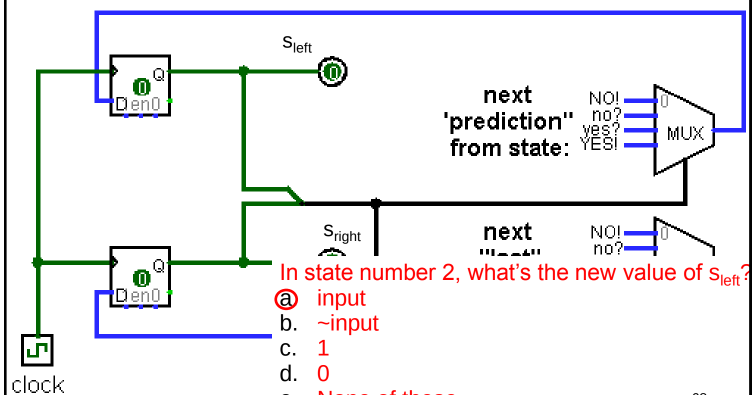
In state number 0, what should be the new value of s_{left}?
Hint: look at the DFA, ***not*** at the circuit!
input
a. ~input
b. 1
c. 0
d. None of these.

Implementing the Predictor: Step 4



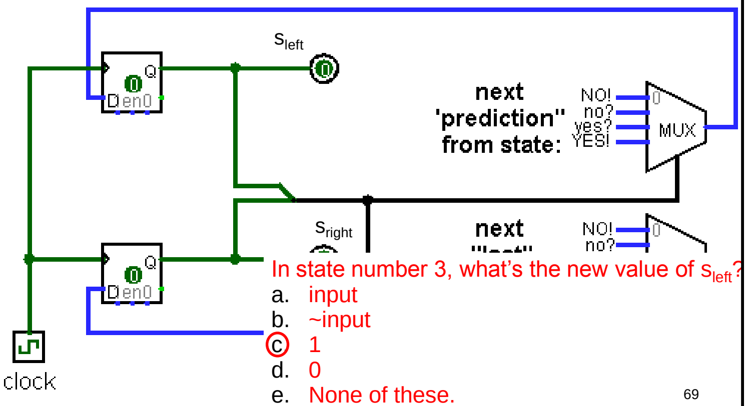
In state number 1, what’s the new value of s_{left}?
a. input
b. ~input
c. 1
d. 0
e. None of these.

Implementing the Predictor: Step 4



In state number 2, what’s the new value of s_{left}?
a. input
b. ~input
c. 1
d. 0
e. None of these.

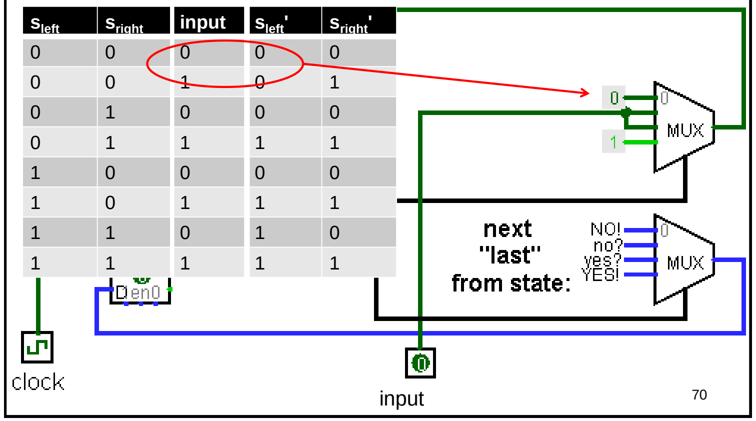
Implementing the Predictor: Step 4



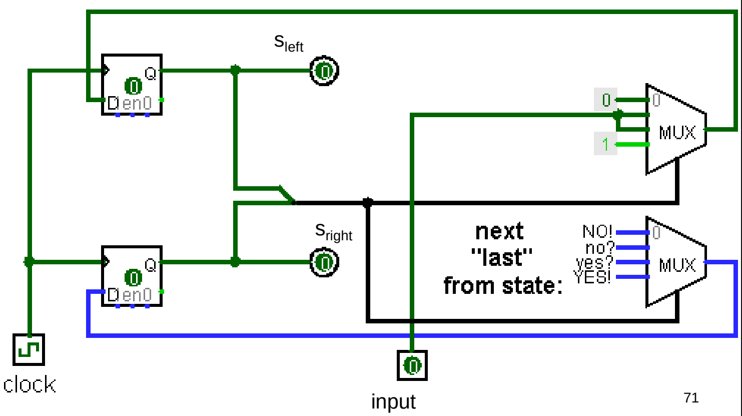
In state number 3, what's the new value of s_{left}?
a. input
b. ~input
c. 1
d. 0
e. None of these.

Just Truth Tables...

s_{left}	s_{right}	input	s_{left}'	s_{right}'
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

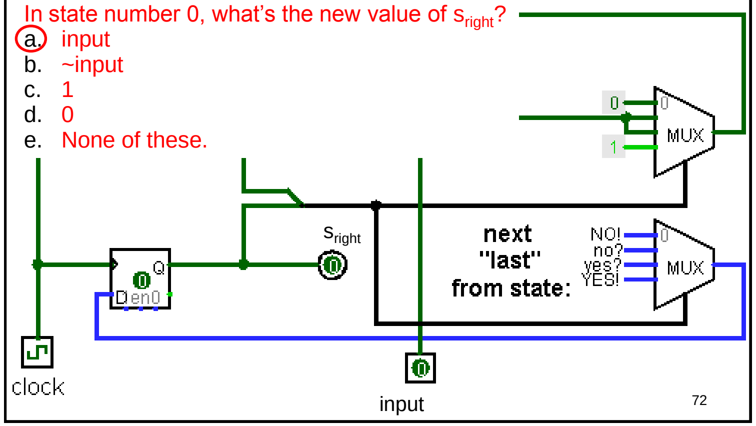


Implementing the Predictor: Step 4

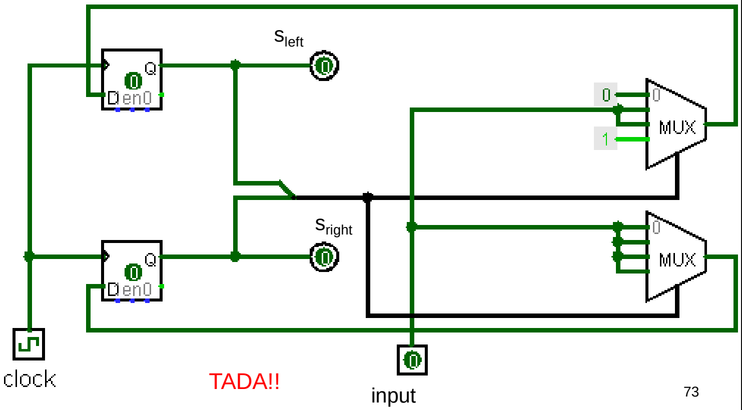


Implementing the Predictor: Step 4

In state number 0, what's the new value of s_{right}?
a. input
b. ~input
c. 1
d. 0
e. None of these.

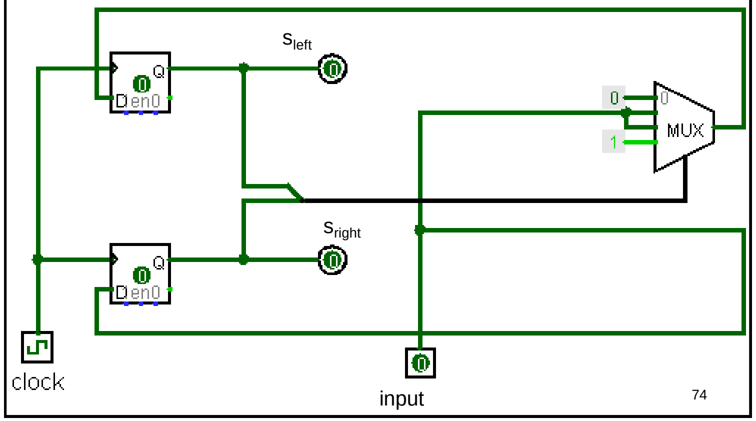


Implementing the Predictor: Step 4



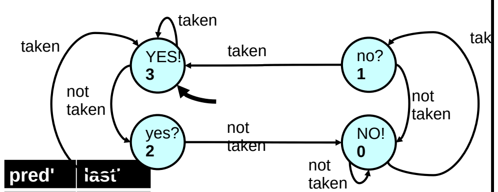
73

You *can* often simplify, but that's not the point.



74

Just for Fun: Renaming to Make a Pattern Clearer...



pred	last	input	pred'	last'
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

75

Outline

- Sequential Circuits :Latches, and flip-flops.
- DFA Example : Branch prediction.
- Implementing DFAs
- **How Powerful are DFAs?**
- Other problems and exercises.

Steps to Build a Powerful Machine

(1) Number the states and figure out **b**: the number of bits needed to store the state number.
(2) Lay out **b** D flip-flops. Together, their memory is the state as a binary number.
(3) ***For each state***, build a ***combinational circuit*** that determines the *next state* given the input.
(4) Send the *next states* into a MUX with the current state as the control signal: only the appropriate *next state* gets used!
(5) Use the MUX's output as the new state of the flip-flops.

With a **separate** circuit for each state, they're often very simple!

How Powerful Is a DFA?

DFAs can model situations with a finite amount of memory, finite set of possible inputs, and particular pattern to update the memory given the inputs.

How does a DFA compare to a modern computer?

a. Modern computer is more powerful.
b. DFA is more powerful.
c. They're the same.

Where We'll Go From Here...

We'll come back to DFAs again later in lecture.

In lab you have been and will continue to explore what you can do once you have memory and events.

And, before long, how you combine these into a working computer!

Also in lab, you'll work with a widely used representation equivalent to DFAs: **regular expressions**.

Unit Outline

- Sequential Circuits :Latches, and flip-flops.
- DFA Example : Branch prediction.
- Implementing DFAs
- How Powerful are DFAs?
- **Other problems and exercises.**

Exercises

- Real numbers:
  - We can write numbers in decimal using the format
    (-)? d+ (.d+)?
  - where the ( )? mean that the part in parentheses is optional, and d+ stands for "1 or more digits".
  - Design a DFA that will accept input strings that are valid real numbers using this format.
    - You can use else as a label on an edge instead of listing every character that does not appear on another edge leaving from a state.

Exercises

- Real numbers (continued)
  - Then design a circuit that turns a LED on if the input is a valid real number, and off otherwise.
    - Hint: Logisim has a keyboard component you can use.
    - Hint: my DFA for this problem has 6 states.
- Design a DFA with outputs to control a set of traffic lights. Thought: try allowing an output that sets a timer which in turn causes an input like our "button press" when it goes off.
- Variants to try:
  - Pedestrian cross-walks
  - Turn signals
  - Inductive sensors to indicate presence of cars
  - Left-turn signals

Quiz #9

- Due Date: Check Announcements.
- Reading for the Quiz
  Textbook sections:
  - Epp, 4th edition: 5.1 to 5.4
  - Epp, 3rd edition: 4.1 to 4.4
  - Rosen, 6th edition: 4.1, 4.2
  - Rosen, 7th edition: 5.1, 5.2