

# **CSE 2600**

# **Intro. To Digital Logic & Computer Design**

Bill Siever  
&  
Michael Hall

# This week

- Today: Review
  - TA-led Review: Wed, Feb 18th from 7-8pm in McDonnell 361
- Thursday: Exam 1 @ classtime
- Exam 1 Page: <https://washu-cse2600-sp26.github.io/schedule/>

# Course Goals

- Given a problem description (behavior and constraints) appropriate for a digital machine:
  - Identify an appropriate binary representation for relevant data.
  - Create an appropriate digital logic solution either/both structurally (gate-level designs) and behaviorally (using a Hardware Description Language (HDL)).
- Given an existing description of a digital circuit (schematic or HDL):
  - Analyze performance and implementation issues (time, space, effort to maintain, etc.).
- Given specifications for a CPU (Instruction Set) Architecture:
  - Given a problem description, write an assembly language program capable of solving the problem.
  - Be able to read and explain the behavior of assembly language.
- Given specifications for a microarchitecture:
  - Explain how instructions behave and issues impacting performance of computation.
  - Modify its control and datapath to support additional functionality.

# Course Goals: Modules 1-3

- Given a problem description (behavior and constraints) appropriate for a digital machine:
  - Identify an appropriate binary representation for relevant data.
  - Create an appropriate digital logic solution either/both structurally (gate-level designs) and behaviorally (using a Hardware Description Language (HDL)).
- Given an existing description of a digital circuit (schematic or HDL):
  - Analyze performance and implementation issues (time, space, effort to maintain, etc.).

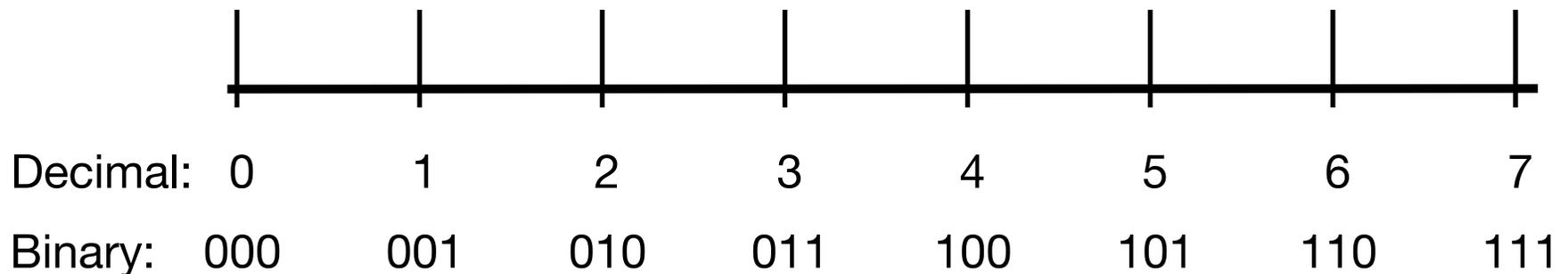
# Back to Basics: Why Binary?

- Binary: (0/1; false/true; Off/On; 0v/3v; No/Yes; ...)
- Why? Easy and cheap to build *reliable, fast* machines



# Back to Basics: Binary

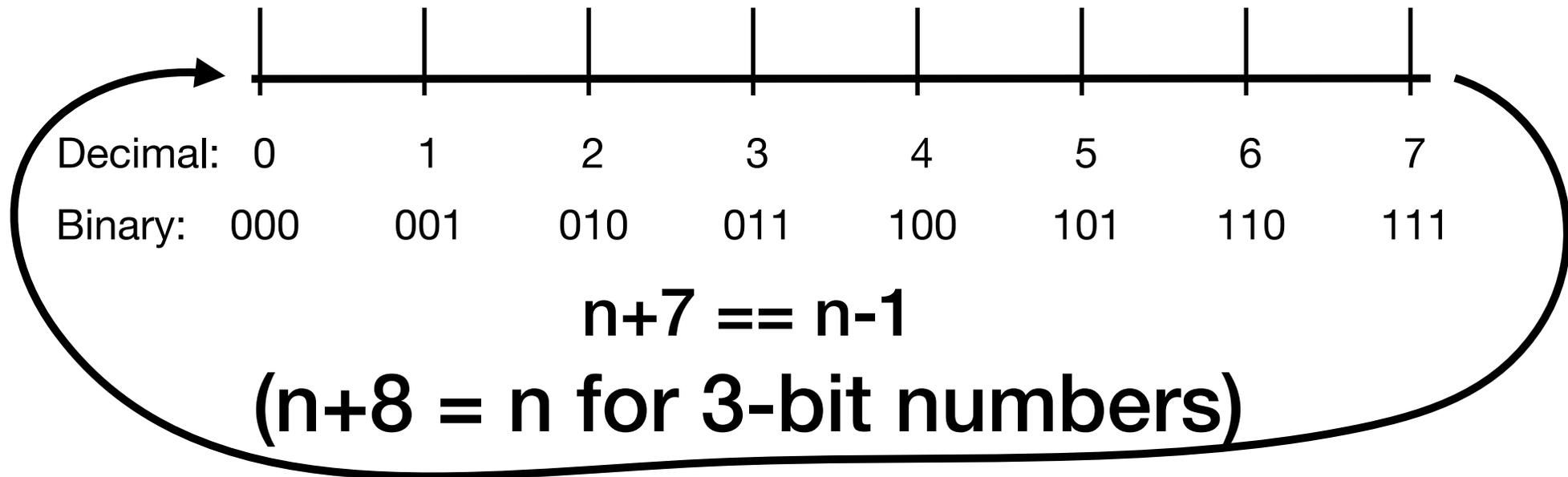
- Can encode info
- Can have one-to-one correspondence to number lines



- Also easy to convert between convenient forms:  
Binary, Decimal, Hexadecimal

# Back to Basics: Binary

- Behaving like negatives is easy (with fixed-width numbers)



# The Magic of Fixed Width numbers (modular arithmetic): Addition can emulate subtraction!



A horizontal number line with eight tick marks. The tick marks are labeled with decimal values from 0 to 7. Below the line, the corresponding 3-bit binary values are listed. The 2's complement behavior is shown for the last four values (4, 5, 6, 7), which correspond to -4, -3, -2, and -1 respectively.

Decimal:	0	1	2	3	4	5	6	7
Binary:	000	001	010	011	100	101	110	111
2's comp behavior:					-4	-3	-2	-1

# Other Data

- Enumerable things
- Including “states”:
  - Ex: Idle, Wash, Rinse, Dry

STATE	BINARY COUNTING ENCODING	ANOTHER BIN. ENC.	A ONE HOT
IDLE	00	10	0001
WASH	01	11	0010
RINSE	10	01	0100
DRY	11	00	1000

# Other Data: There'll be more

- Composite things: large numbers with fields the represent parts (instructions)
- ASCII: Enumeration / code for (English) characters
- Fixed Point & Floating Point: “Real” Numbers

# Practice

- Homework 1: Binary representations / decimal / hex
- Studio 2A: Signed Binary
- Homework 2A: More binary representations

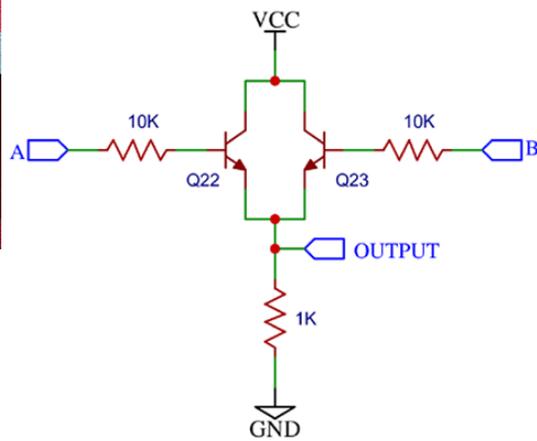
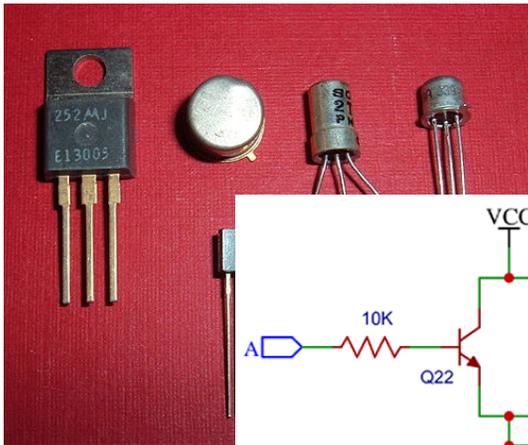
# *Practice*

- What is the decimal value of 0xAA?
- What is 29 in binary?
- What is -5 in 4-bit, 2's complement binary?
  - Shown in hex?

# Course Goals: Modules 1-3

- Given a problem description (behavior and constraints) appropriate for a digital machine:
  - Identify an appropriate binary representation for relevant data.
  - Create an appropriate digital logic solution either/both structurally (gate-level designs) and behaviorally (using a Hardware Description Language (HDL)).
- Given an existing description of a digital circuit (schematic or HDL):
  - Analyze performance and implementation issues (time, space, effort to maintain, etc.).

# Assume we can build basic blocks...

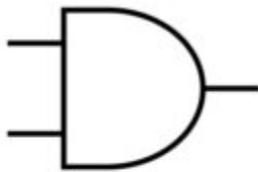


<https://circuitdigest.com/electronic-circuits/designing-o>

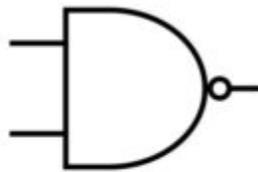


# Big Idea: Schematics for Designs

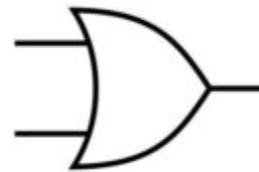
## LOGIC GATE SYMBOLS



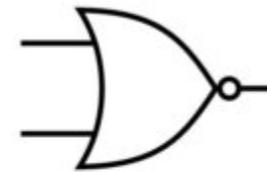
AND



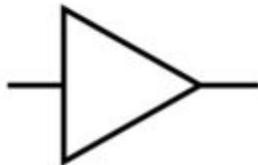
NAND



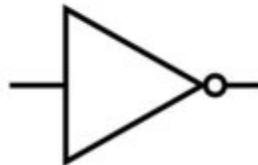
OR



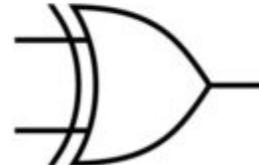
NOR



BUFFER



NOT

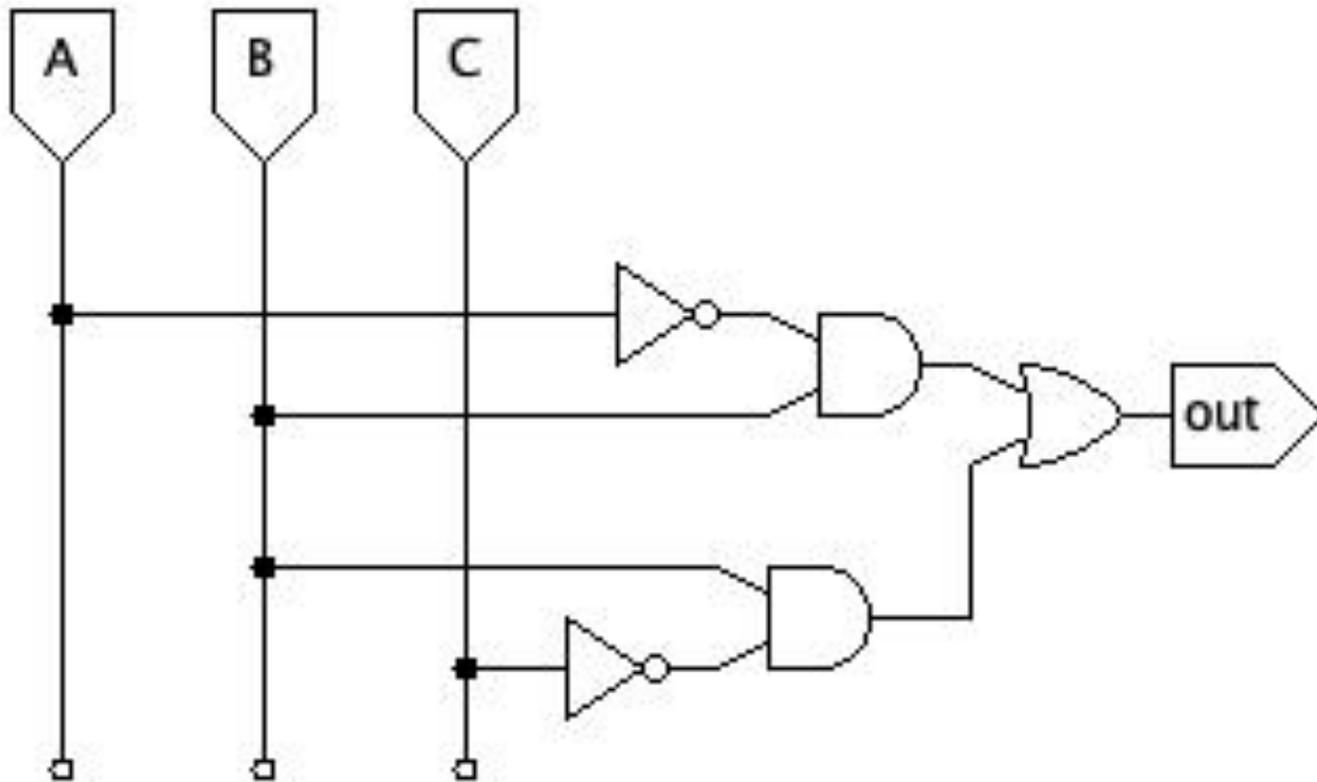


XOR



XNOR

**Can build complex things...**



# Brings some new ideas

- Can be represented (and manipulated)
  - Boolean Algebra
  - Truth table
- Implementation operates in real-world and takes time
  - I.e., Propagation Delay

$$out = \bar{A} \cdot B + B \cdot \bar{C}$$

A	B	C	OUT
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

# Brings some new ideas

- Truth table can be converted to equations and vice versa

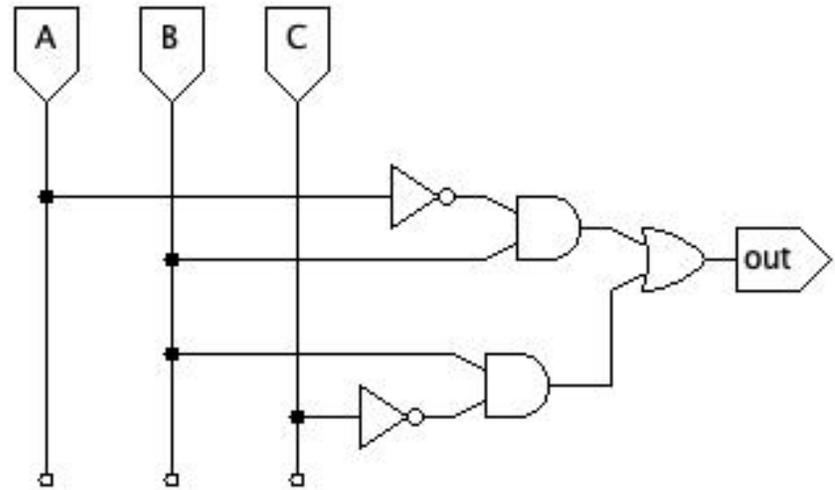
$$out = \bar{A} \cdot B + B \cdot \bar{C}$$

A	B	C	OUT
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

# Brings some new ideas

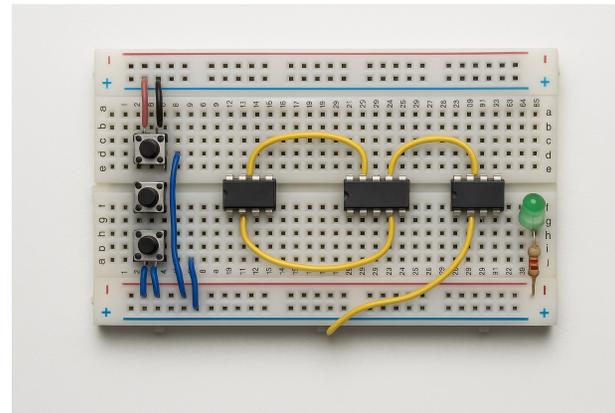
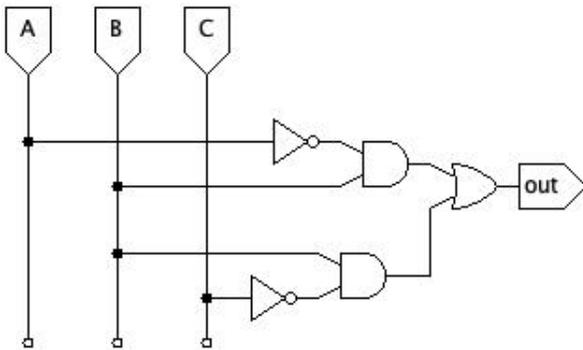
- Equations can be converted to schematics too

$$out = \bar{A} \cdot B + B \cdot \bar{C}$$



# Brings some new ideas

- And machines can be build from schematics...



ChatGPT Generated

# Concepts

- Equations can be manipulated via formal rules

**Table 2.1** Axioms of Boolean algebra

Axiom	Dual	Name
A1 $B = 0$ if $B \neq 1$	A1' $B = 1$ if $B \neq 0$	Binary field
A2 $\bar{0} = 1$	A2' $\bar{1} = 0$	NOT
A3 $0 \cdot 0 = 0$	A3' $1 + 1 = 1$	AND/OR
A4 $1 \cdot 1 = 1$	A4' $0 + 0 = 0$	AND/OR
A5 $0 \cdot 1 = 1 \cdot 0 = 0$	A5' $1 + 0 = 0 + 1 = 1$	AND/OR

**Table 2.2** Boolean theorems of one variable

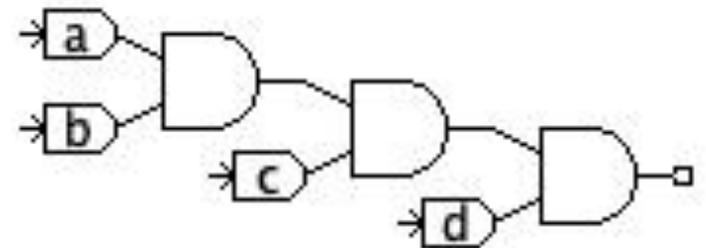
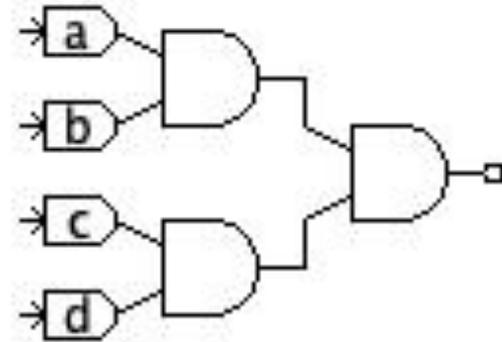
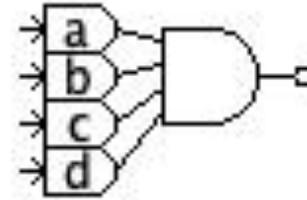
Theorem	Dual	Name
T1 $B \cdot 1 = B$	T1' $B + 0 = B$	Identity
T2 $B \cdot 0 = 0$	T2' $B + 1 = 1$	Null Element
T3 $B \cdot B = B$	T3' $B + B = B$	Idempotency
T4 $\overline{\overline{B}} = B$		Involution
T5 $B \cdot \overline{B} = 0$	T5' $B + \overline{B} = 1$	Complements

**Table 2.3** Boolean theorems of several variables

Theorem	Dual	Name
T6 $B \cdot C = C \cdot B$	T6' $B + C = C + B$	Commutativity
T7 $(B \cdot C) \cdot D = B \cdot (C \cdot D)$	T7' $(B + C) + D = B + (C + D)$	Associativity
T8 $(B \cdot C) + (B \cdot D) = B \cdot (C + D)$	T8' $(B + C) \cdot (B + D) = B + (C \cdot D)$	Distributivity
T9 $B \cdot (B + C) = B$	T9' $B + (B \cdot C) = B$	Covering
T10 $(B \cdot C) + (B \cdot \overline{C}) = B$	T10' $(B + C) \cdot (B + \overline{C}) = B$	Combining
T11 $(B \cdot C) + (\overline{B} \cdot D) + (C \cdot D) = (B \cdot C) + (\overline{B} \cdot D)$	T11' $(B + C) \cdot (\overline{B} + D) \cdot (C + D) = (B + C) \cdot (\overline{B} + D)$	Consensus
T12 $\overline{B_0 \cdot B_1 \cdot B_2 \dots} = (\overline{B_0} + \overline{B_1} + \overline{B_2} \dots)$	T12' $\overline{B_0 + B_1 + B_2 \dots} = (\overline{B_0} \cdot \overline{B_1} \cdot \overline{B_2} \dots)$	De Morgan's Theorem

# Implementation Matters

- $o = a \cdot b \cdot c \cdot d$



# Generality

- Truth Table
  - $n$  input variables ( $n$  columns of input)
  - $2^n$  rows: represent all possible combos of input values
  - $k$  columns, 1 for each output bit
- Any such truth table can be converted into an equation (and circuit)
  - Sum-of-products (or product-of-sum): Fool proof (kinda), but may be inefficient

# Simple Optimization

- Karnaugh Maps

- Use “Gray code order” (not counting order)

A changes

- Visual way to combine terms like:  $A \cdot \bar{B} \cdot C + A \cdot B \cdot C$

- Adjacent cells represent a change in a single variable

- Theorem:

---

T10  $(B \cdot C) + (B \cdot \bar{C}) = B$

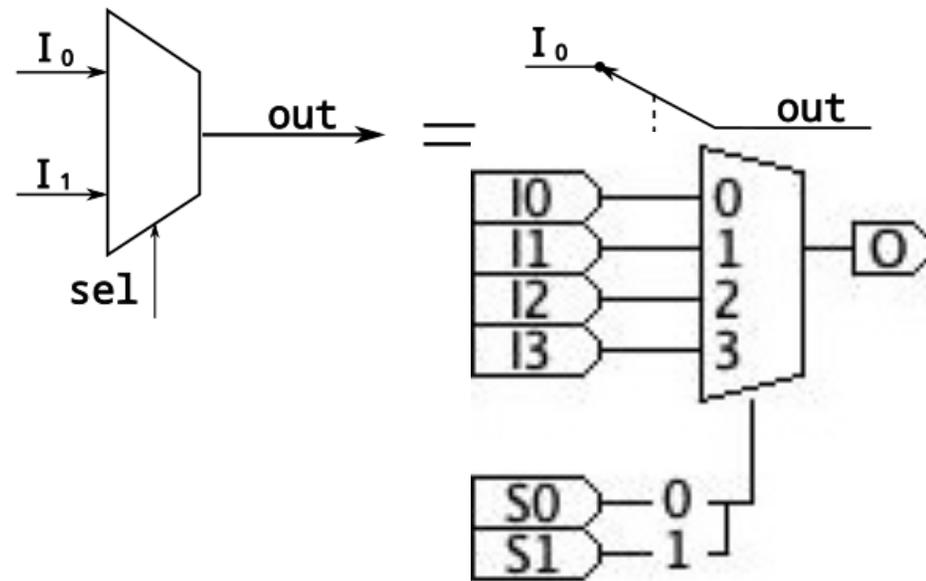
---

		A B			
		00	01	11	10
C	0	0	1	1	0
	1	1	1	0	0

C changes

# Combinational Logic

- Can be represented by tables
  - Combine inputs to produce output
  - No concept of memory
- Can represent bigger ideas
  - Selection (multiplexor)
    - May be hierarchical (rather than Sum-of-Products)
  - Encoding / Decoding ( $2^n$  inputs to  $n$  outputs or  $n$  inputs to  $2^n$  outputs)
  - Addition/subtraction, even multi-bit



# Practice

- Studio 2A: JLS simulation of combinational logic
- Homework 2A
  - Logic equations, Sum-of-Products Canonical form, propagation delay
  - JLS: Combinational logic for adding 2-bit numbers

# Practice

- Studio 2B:
  - Truth tables, equations, and Karnaugh Maps
  - “Real” behaviors: Glitches & Propagation Delay
  - Construction from Multiplexors & Nands
- Homework 2B:
  - Multiplexors from logic / equations
  - Optimization
  - JLS implementation of multiplexor

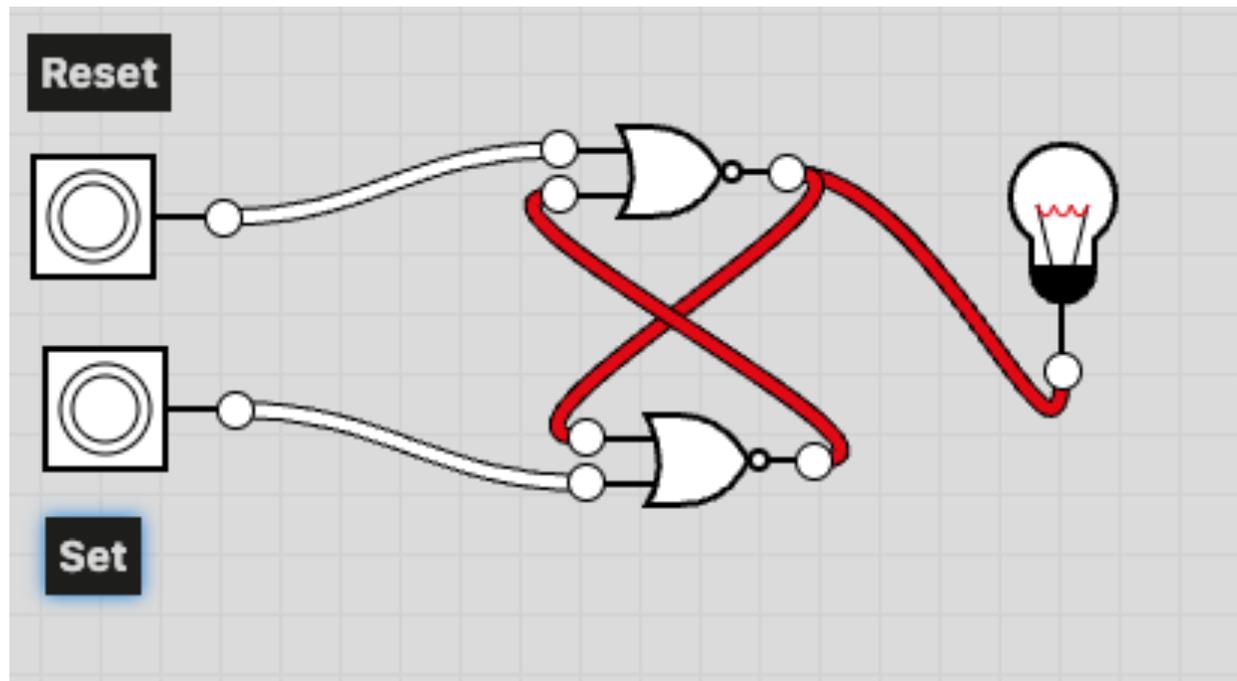
# *Practice*

- A, B, and C are inputs
- Give the SOP for Y
- Give the Min (K-Map Sense) for X
- Assume only 2-input OR gates (prop delay 8) and 2-input AND (prop delay 12) gates exist  
What's the best possible propagation delay for X?

A	B	C	X	Y
0	0	0	1	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

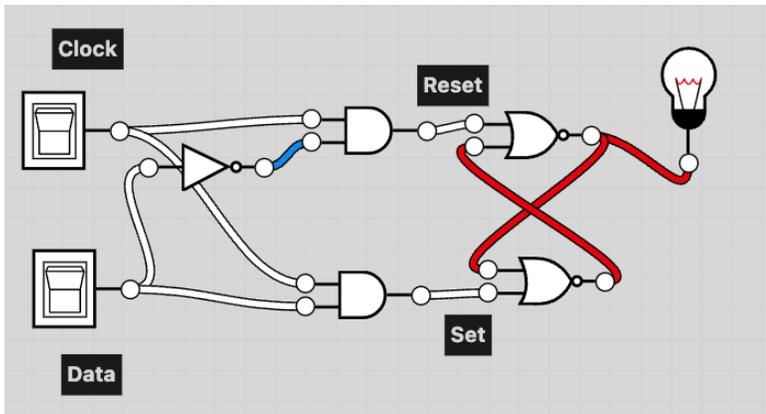
# Sequential Stuff & Feedback Paths

- Basic feedback leads to stable storage / memory



# Can combine ideas to shape behavior

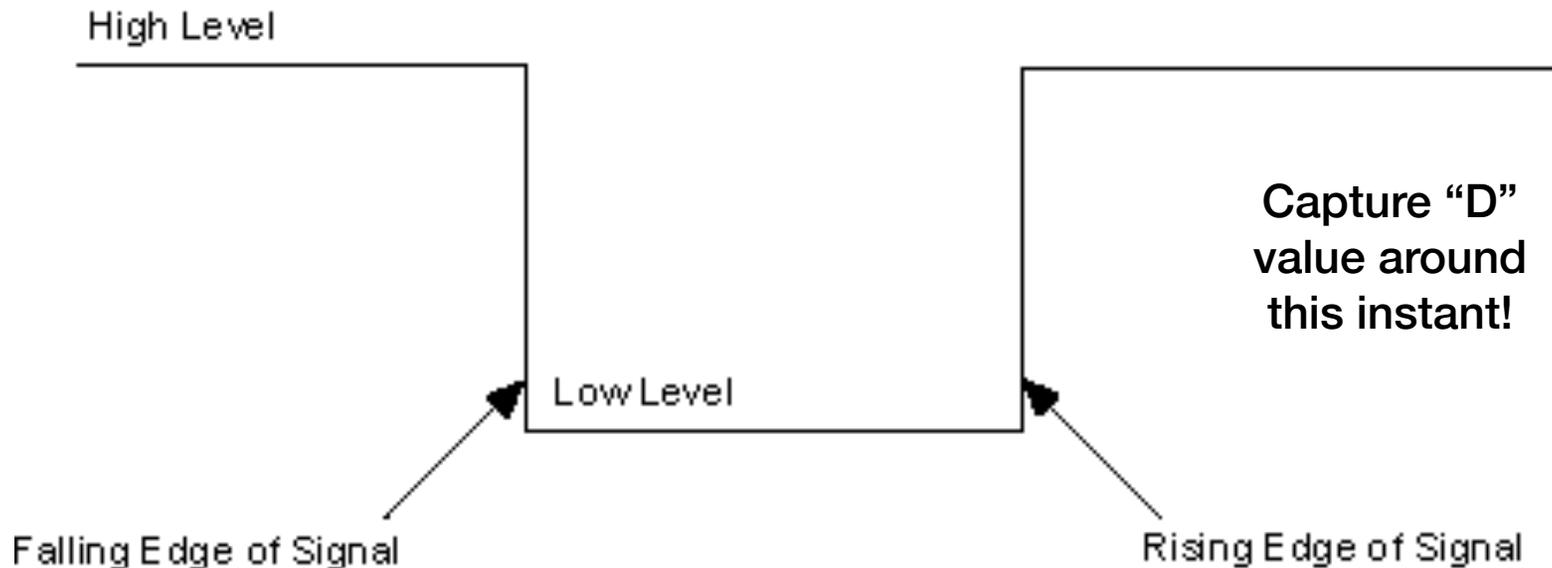
D Latch (transparent when Clock high)



CLOCK	DATA	Q
0	0	(Unchanged)
0	1	(Unchanged)
1	0	0
1	1	1



# D-Flip-Flop Clock

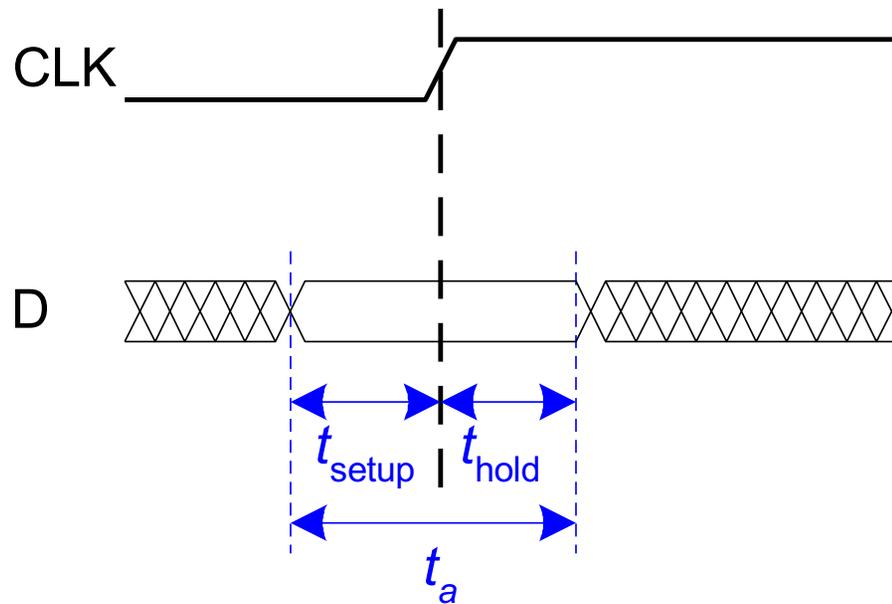


[https://www.ni.com/docs/en-US/bundle/ni-hsdio/page/hsdio/fedge\\_trigger.html](https://www.ni.com/docs/en-US/bundle/ni-hsdio/page/hsdio/fedge_trigger.html)

# D Flip Flop is built from SR Latches

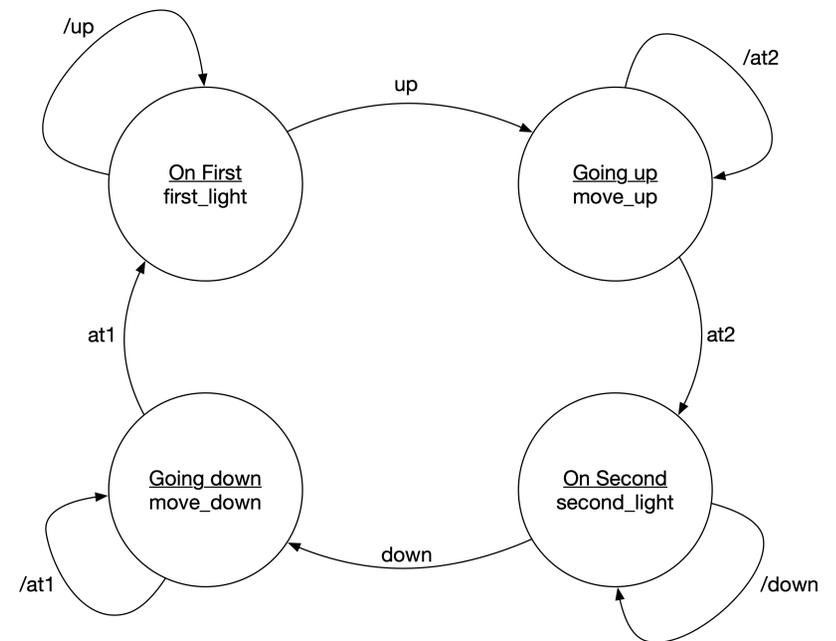
- Internal latch
  - Fails if pulses too short
  - Unstable if R/S drop at same time / too close
- Setup Time: Time needed before clock to ensure stable capture
- Hold Time: Time After clock edge value must be “held”

# Dff: Setup & Hold Time

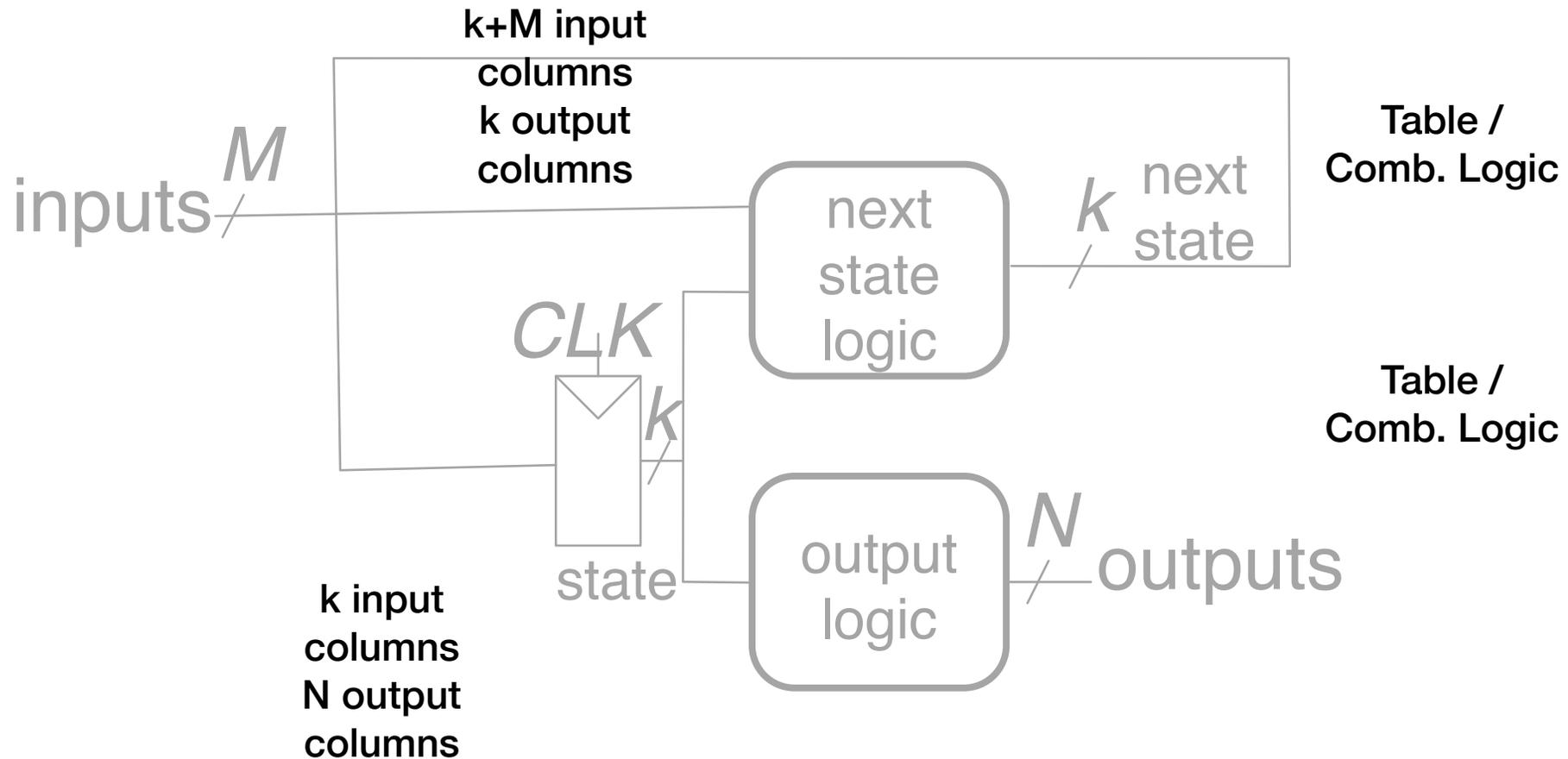


# State Machine Diagram

- **State:** Condition of system
- **Inputs:** Used by arcs / outputs
- **Arrows/arcs:** When/why state changes
- **Outputs:** Actions in the world...



# Review: Moore Machine Structure



# Practice

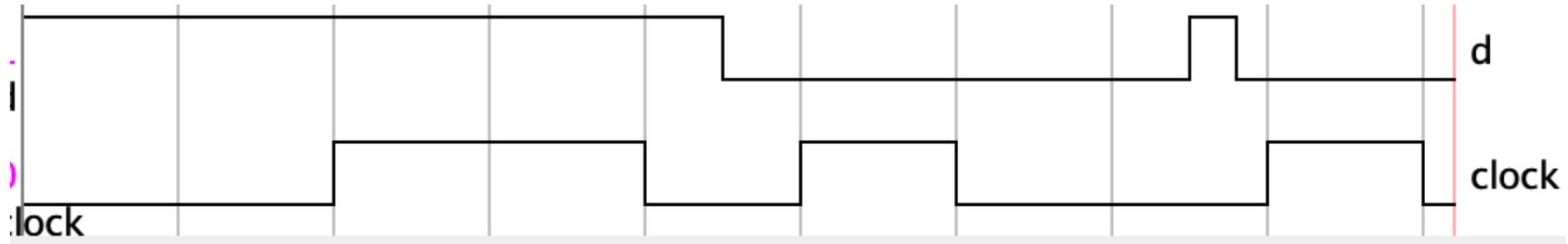
- Studio 3A:
  - JLS: Sequential Machines: Edge Triggered Flip-Flop
  - JLS: Registers
  - JLS: State Machine
- Homework 3A:
  - Review of Equations & K-Maps
  - Latch vs. Flip-Flop behaviors
  - JLS implementation of updated traffic light

# Practice

- Studio 3B:
  - State representations: One-hot vs. Binary (gates / space)
  - JLS: Flip-Flop empirical timing and behavior
- Homework 3B:
  - State machines: The Washer (part 1 ....of many...)
    - Binary representation of state; Equations; Implementation

# Practice

- Give the out from a rising-edge triggered flip flop for the following inputs



# ***Practice***

- What's the minimum number of bits needed for a state machine with the following states: {START, OPEN, PROCESS, ADJUST, ESTOP, END, IDLE}.
- Examples:
  - Given a state diagram, describe behavior.
  - Given a state machine, give table for next state equations (or outputs)

# Course Goals: Modules 1-3

- Given a problem description (behavior and constraints) appropriate for a digital machine:
  - Identify an appropriate binary representation for relevant data.
  - Create an appropriate digital logic solution either/both structurally (gate-level designs) and behaviorally (using a Hardware Description Language (HDL)).
- Given an existing description of a digital circuit (schematic or HDL):
  - Analyze performance and implementation issues (time, space, effort to maintain, etc.).

**Questions?**