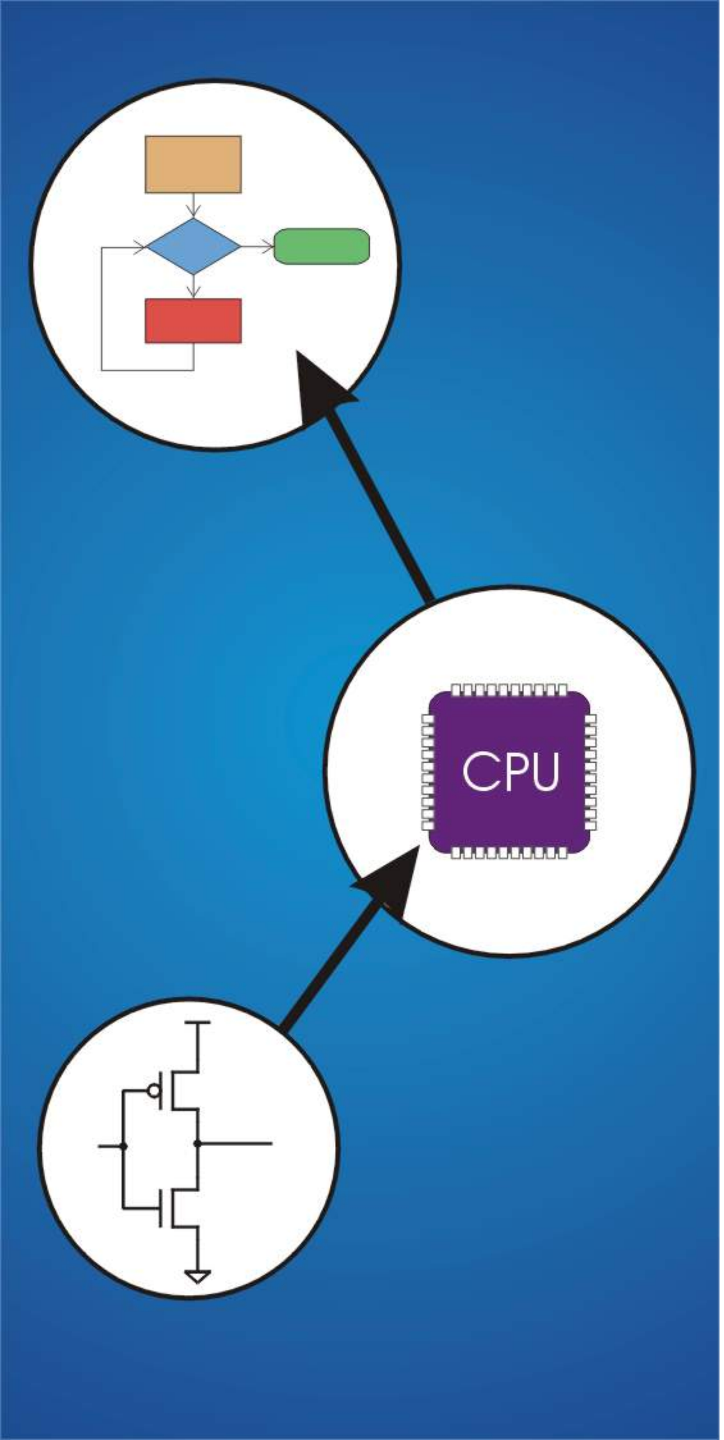
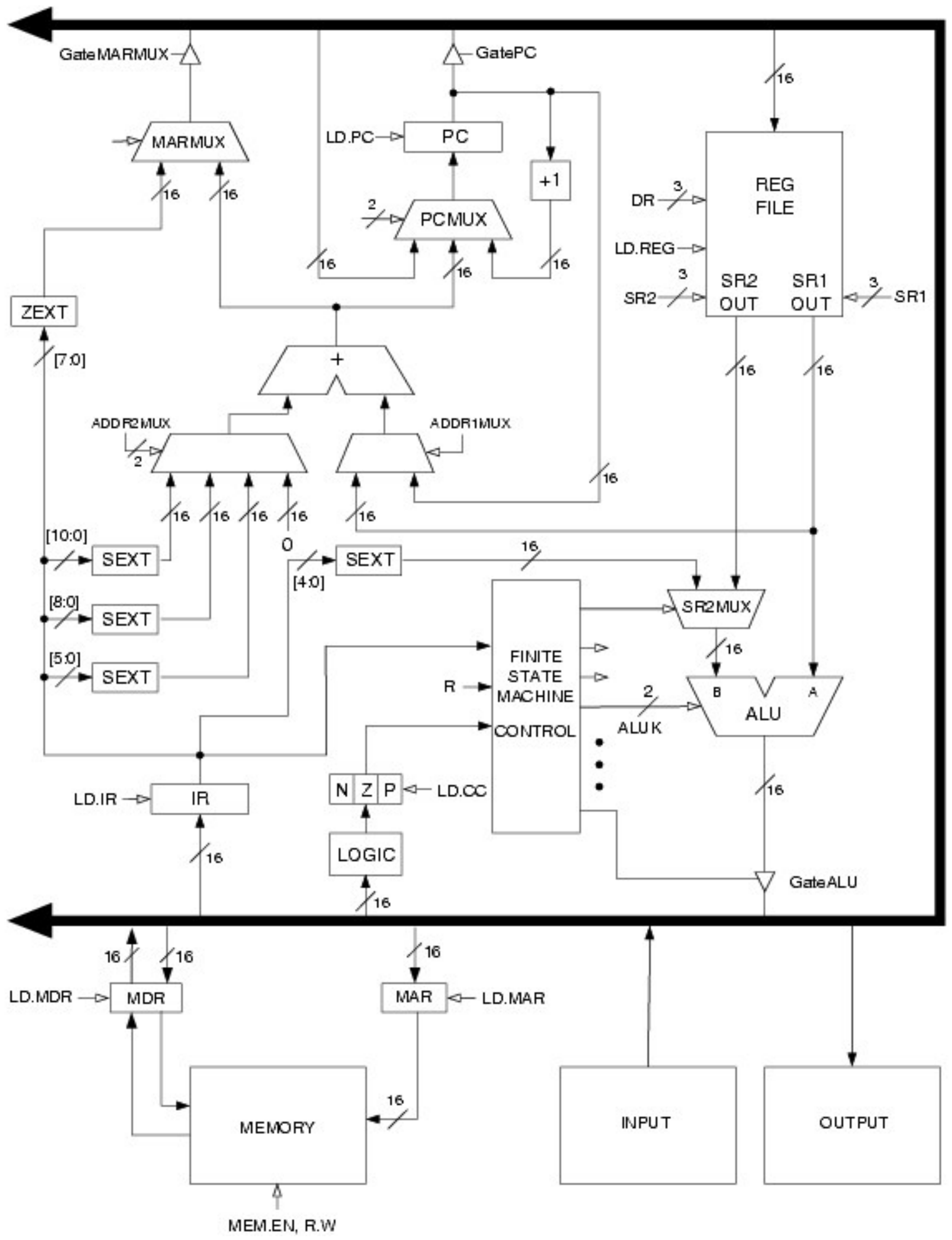


Storage Elements & Sequential Circuits

LC-3 Data Path Revisited

Now Registers and Memory



Combinational vs. Sequential

Combinational Circuit

- always gives the same output for a given set of inputs
    - ex: adder always generates sum and carry, regardless of previous inputs

Sequential Circuit

- stores information
- output depends on stored information (state) plus input
    - so a given input might produce different outputs, depending on the stored information
- *example:* ticket counter
    - advances when you push the button
    - output depends on previous state
- useful for building “memory” elements and “state machines”

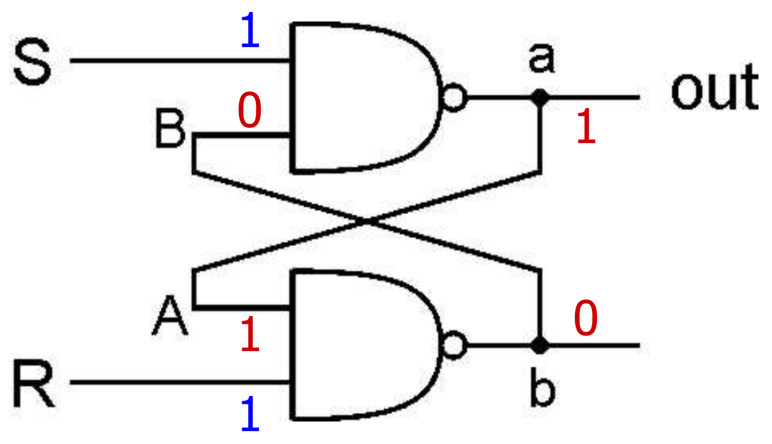
Storage Elements

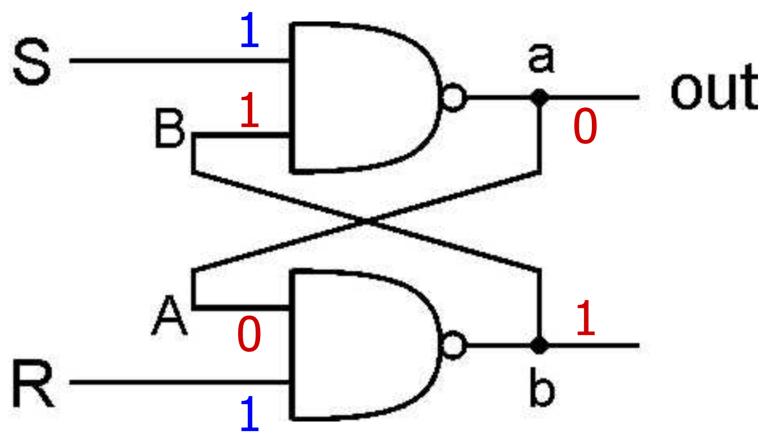
- Static: use a circuit with feedback to save a bit of information
  - flipflops
  - Static memories
- Dynamic: Use charge at a node to represent a 1 or 0
  - A cell in a dynamic memory
  - Fewer transistors hence cheaper
  - Need periodic refreshing, every few millisecs.
- Both are volatile.
- Not consideed here:
  - ROM (read only memory): combinational
  - Flash memory: semiconductor, but work like disks

R-S Latch: Simple Storage Element

R is used to "reset" or "clear" the element – set it to zero.

S is used to "set" the element – set it to one.



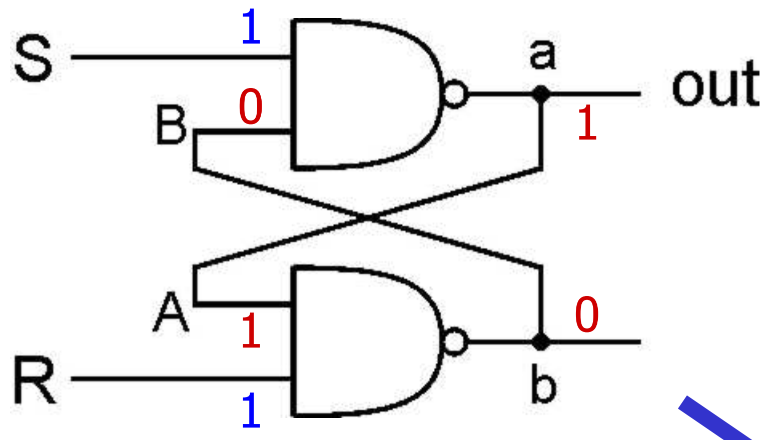


If both R and S are one, out could be <u>either</u> zero or one.

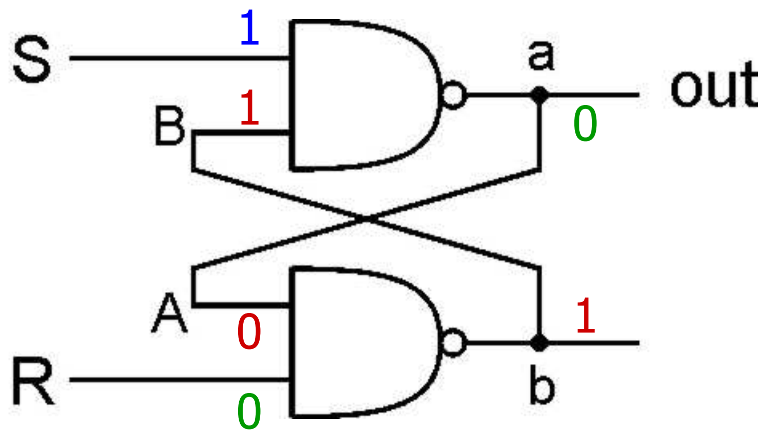
- **"quiescent" state -- holds its previous value**
- **note: if a is 1, b is 0, and vice versa**

Clearing the R-S latch

Suppose we start with output = 1, then change R to zero.



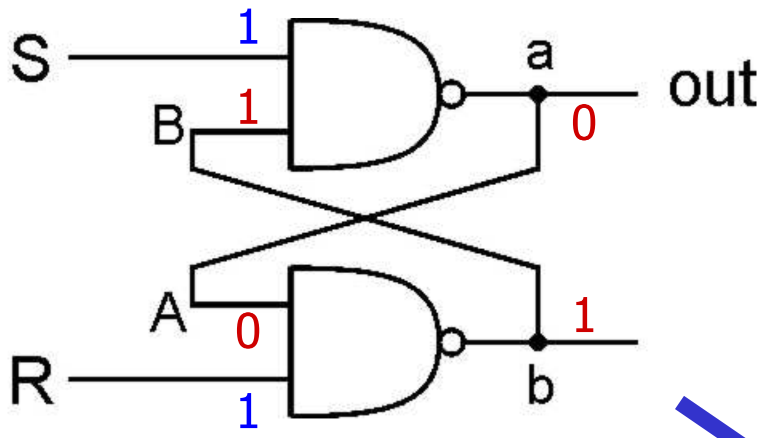
Output changes to zero.



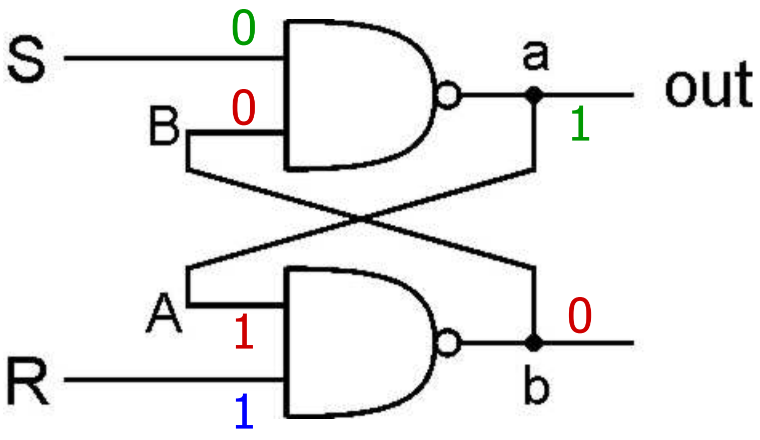
Then set R=1 to "store" value in quiescent state.

Setting the R-S Latch

Suppose we start with output = 0, then change S to zero.



Output changes to one.



Then set S=1 to “store” value in quiescent state.

R-S Latch Summary

R = S = 1

- **hold current value in latch**

S = 0, R=1

- **set value to 1**

R = 0, S = 1

- **set value to 0**

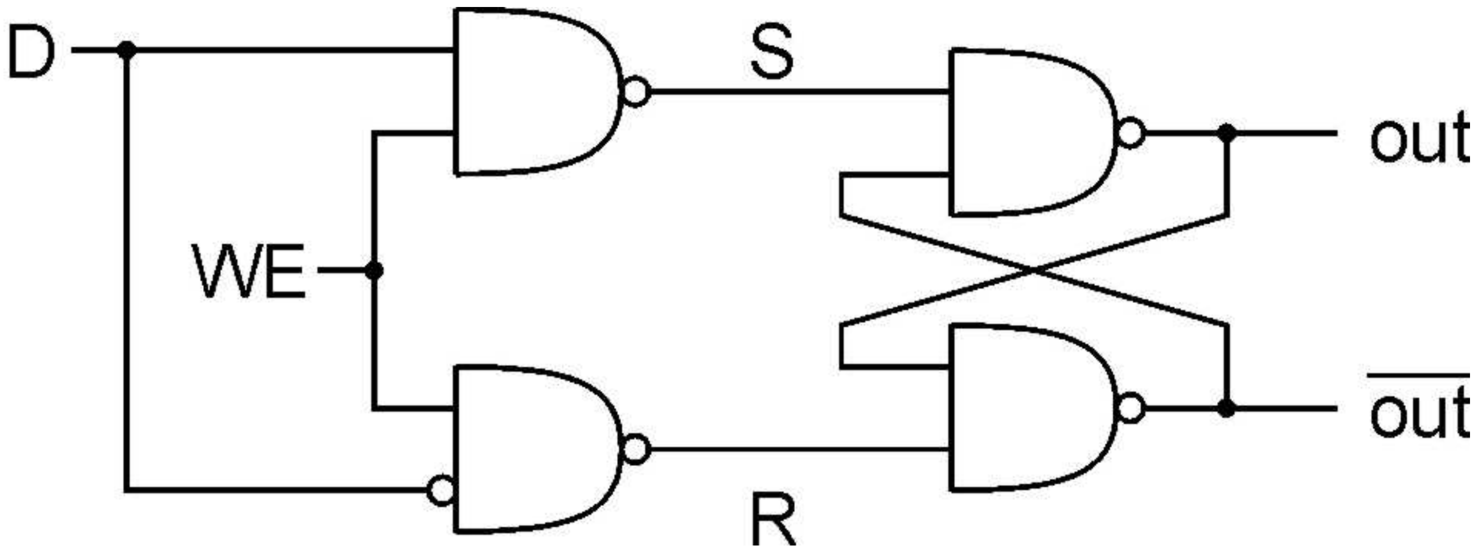
R = S = 0

- **both outputs equal one**
- **final state determined by electrical properties of gates**
- *Don't do it!*

Gated D-Latch

Two inputs: D (data) and WE (write enable)

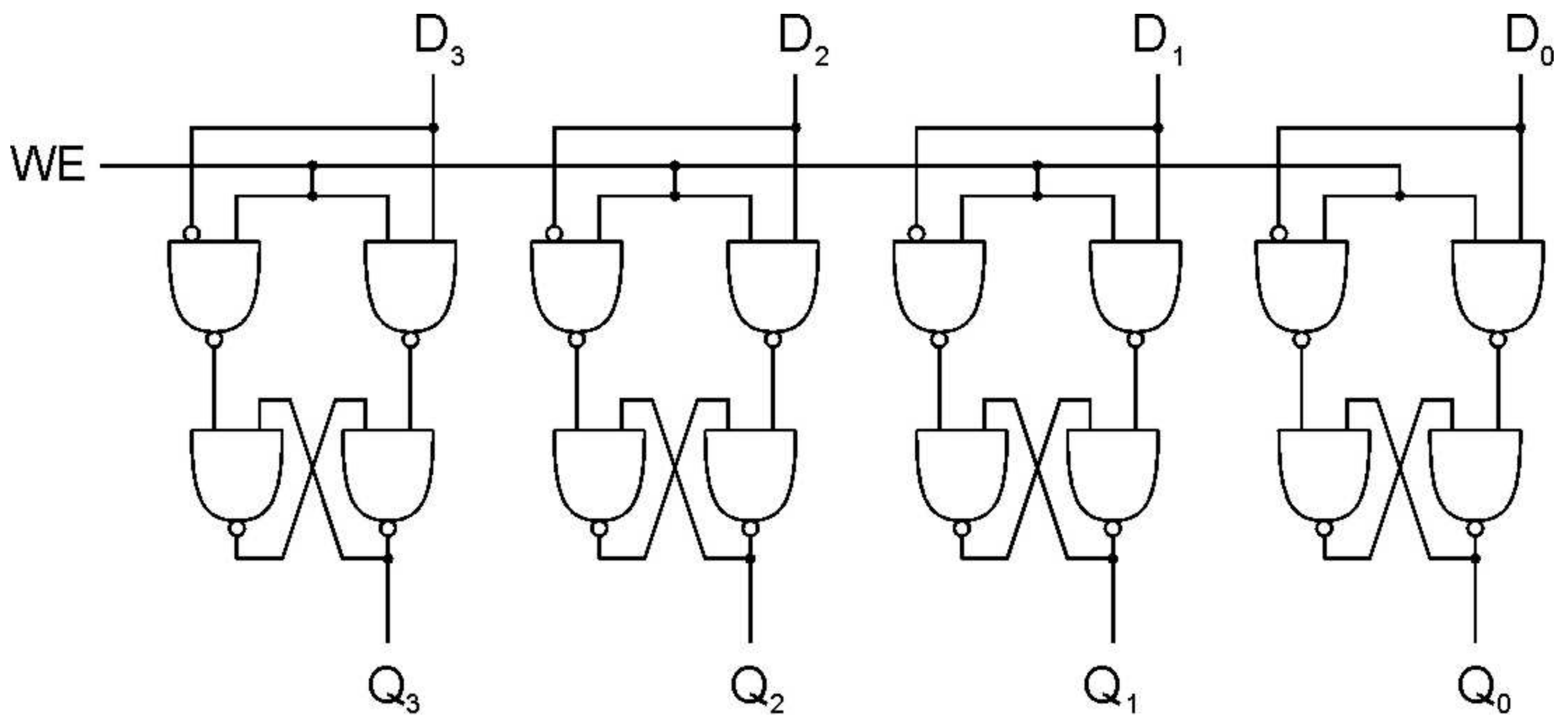
- **when WE = 1, latch is set to value of D**
  - **S = NOT(D), R = D**
- **when WE = 0, latch holds previous value**
  - **S = R = 1**



Register

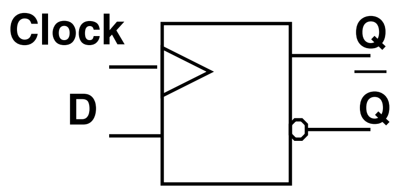
A register stores a multi-bit value.

- **We use a collection of D-latches, all controlled by a common WE.**
- **When WE=1, n-bit value D is written to register.**

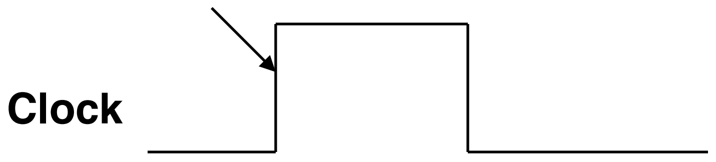


Flip-flops

D Flip-flop: a storage element, can be edge-triggered (available in logisim)



Rising edge: input sampled



D	Next Q
0	0
1	1

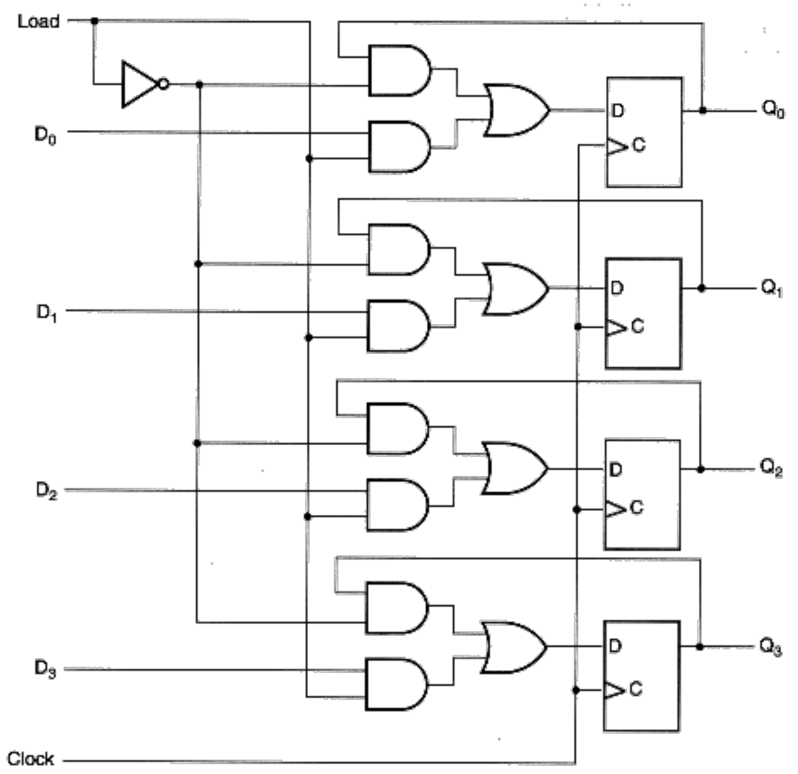
State Q is always available

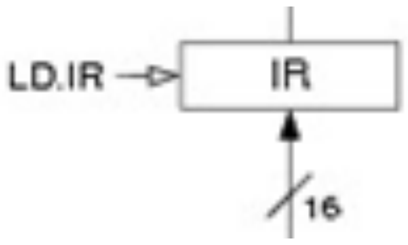
Registers

A register is a row of storage element.

Register with parallel load with a Load control line

Clock is often implied





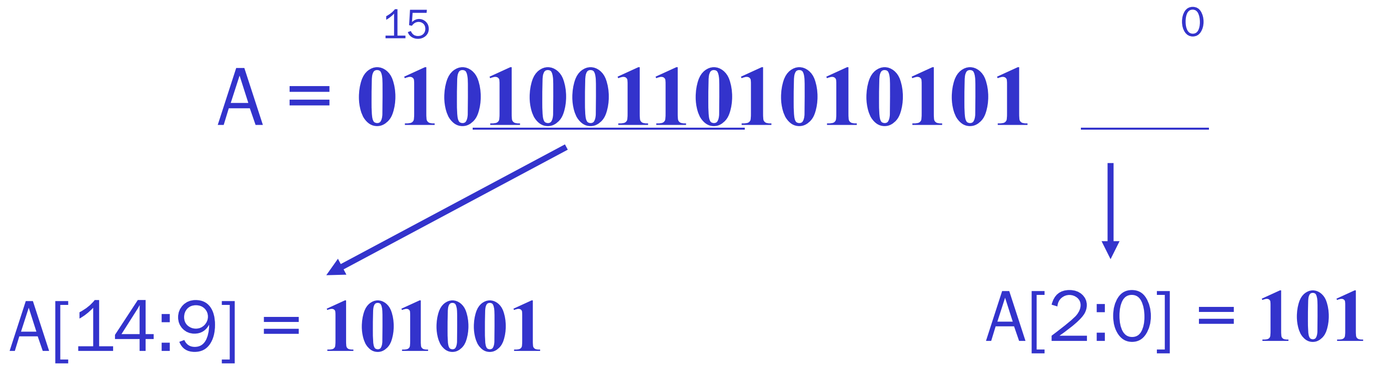
Representing Multi-bit Values

Number bits from right (0) to left (n-1)

- **just a convention -- could be left to right, but must be *consistent***

Use brackets to denote range:

D[l:r] **denotes bit** l **to bit** r, **from *left* to *right***

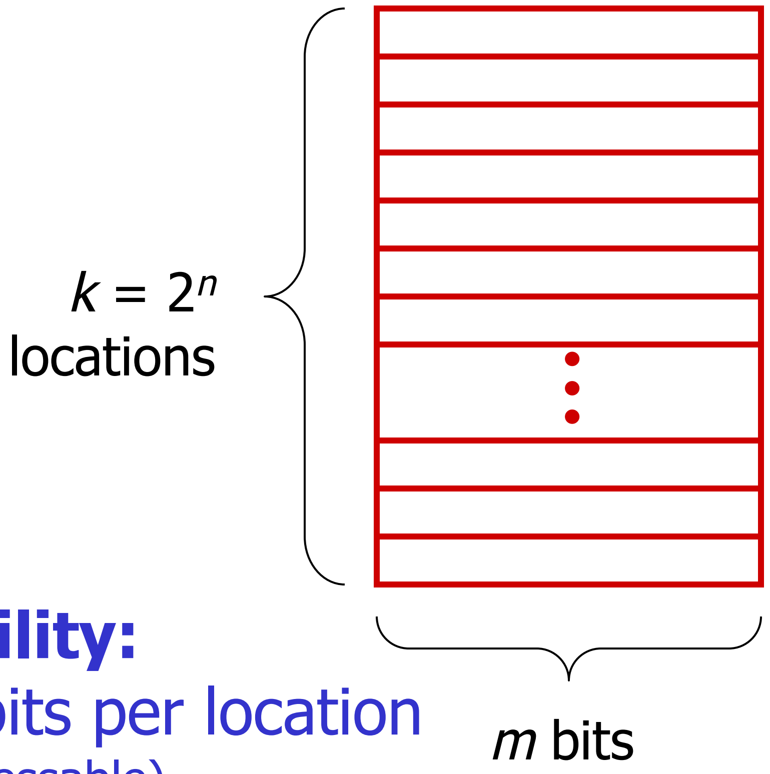


May also see A<14:9>**,**
especially in hardware block diagrams.

Memory

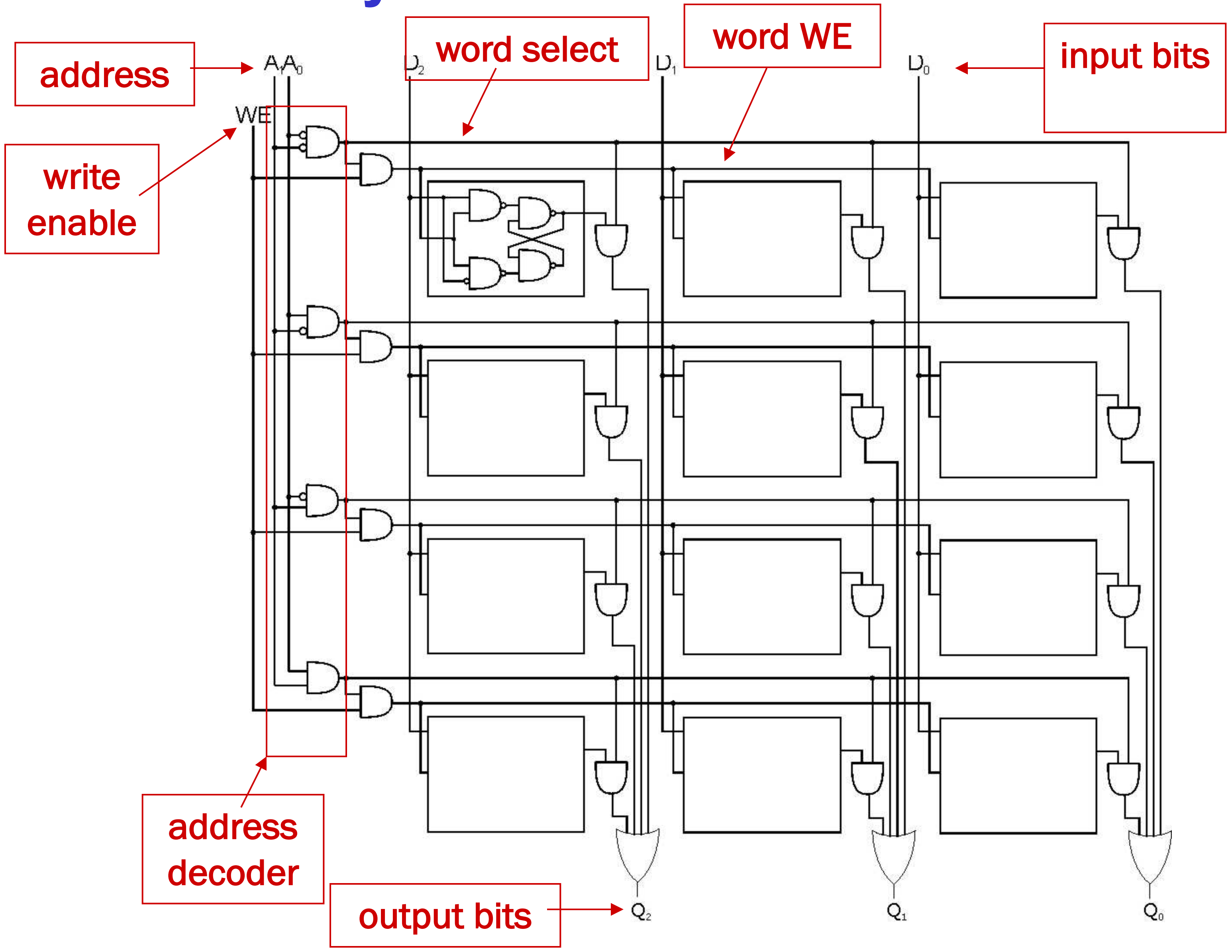
Now that we know how to store bits, we can build a memory – a logical *k* × *m* array of stored bits.

Address Space:
number of locations
(usually a power of 2)



Addressability:
number of bits per location
(e.g., byte-addressable)

2^2 x 3 Memory



address
$A_1 A_0$
write enable
WE
word select
word WE
input bits
D_2
D_1
D_0
address decoder
output bits
Q_2
Q_1
Q_0

More Memory Details

This is a not the way actual memory is implemented.

- **fewer transistors, much more dense,**
  relies on electrical properties

But the logical structure is very similar.

- **address decoder**
- **word select line**
- **word write enable**

Two basic kinds of RAM (Random Access Memory)

Static RAM (SRAM)

- **fast, maintains data as long as power applied**

Dynamic RAM (DRAM)

- **slower but denser, bit storage decays – must be periodically refreshed**

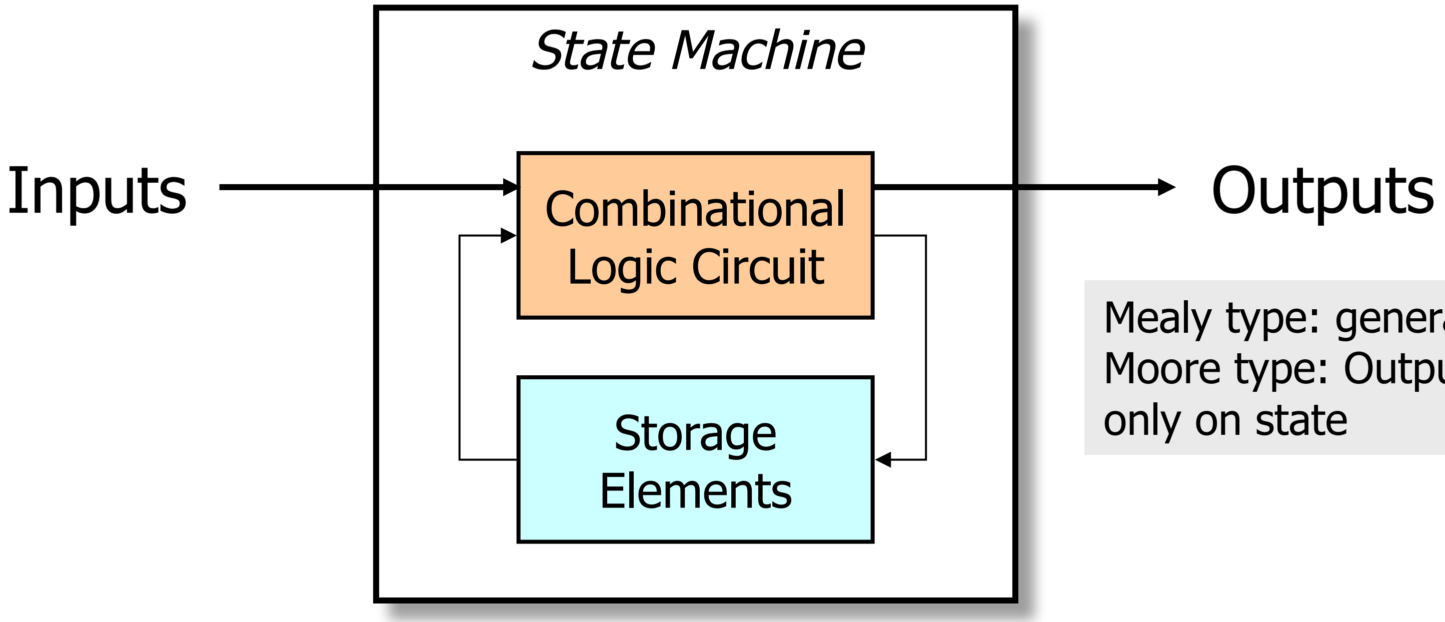
Also, non-volatile memories: ROM, PROM, flash, …

Finite State Machines

State Machine

A general sequential circuit

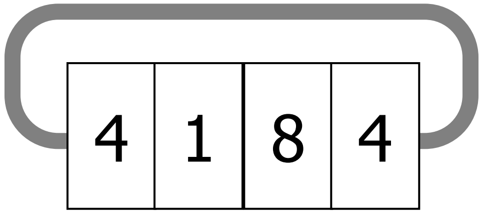
- **Combines combinational logic with storage**
- **“Remembers” state, and changes output (and state) based on inputs and current state**

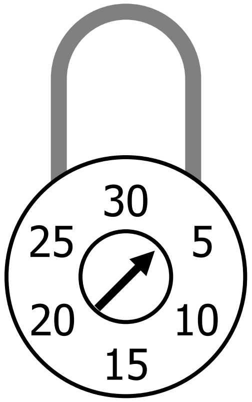


Mealy type: general
Moore type: Output depends only on state

Combinational vs. Sequential

Two types of "combination" locks





Combinational
Success depends only on the values, not the order in which they are set.

Sequential
Success depends on the sequence of values (e.g, R-13, L-22, R-3).

State

The state of a system is a snapshot of all the relevant elements of the system at the moment the snapshot is taken.

Examples:

- **The state of a basketball game can be represented by the scoreboard.**
  - **Number of points, time remaining, possession, etc.**
- **The state of a tic-tac-toe game can be represented by the placement of X' s and O' s on the board.**

State of Sequential Lock

Our lock example has four different states, labelled A-D:

**A: The lock is not open,
and no relevant operations have been performed.**

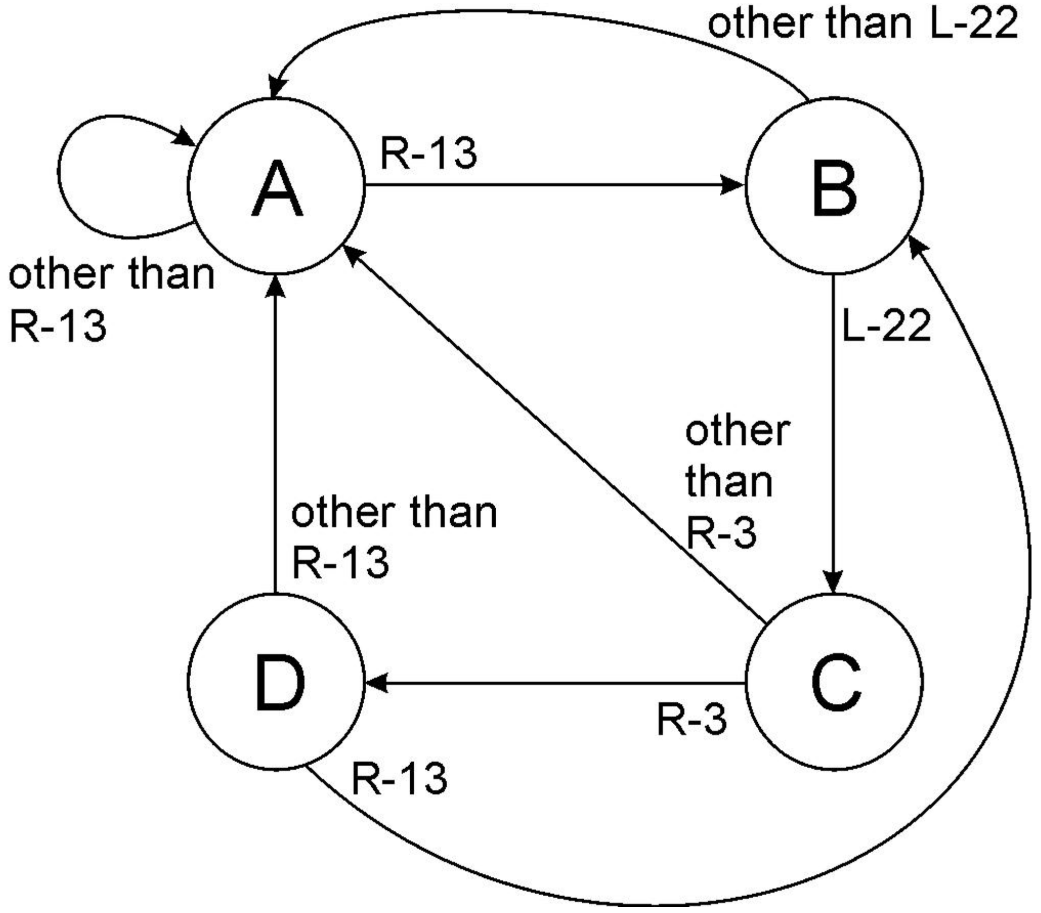
**B: The lock is not open,
and the user has completed the R-13 operation.**

**C: The lock is not open,
and the user has completed R-13, followed by L-22.**

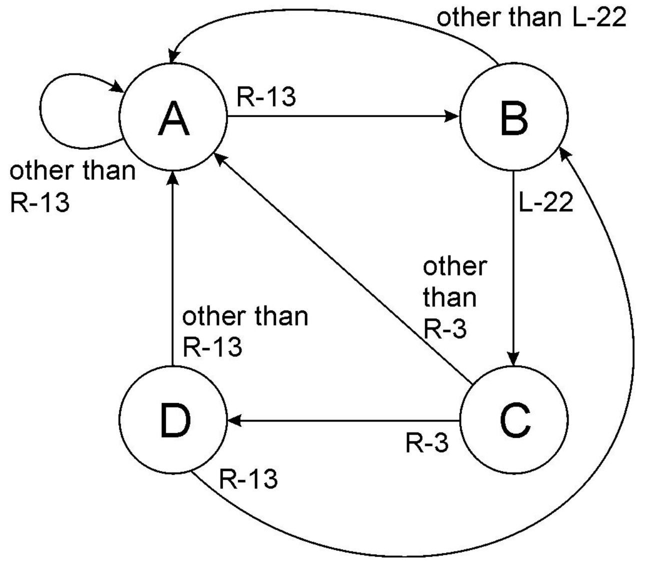
D: The lock is open.

State Diagram

Shows states and
actions that cause a transition between states.



State Table for the lock



Input	Present State	Next State	Output/action
R-32	A	B	-
Not R-32	A	A	-
L-22	B	C	-
Not L-22	B	A	-
R-3	C	D	-
Not R-3	C	A	-
R-13	D	B	Open
Not R-13	D	A	Open

Finite State Machine

A description of a system with the following components:

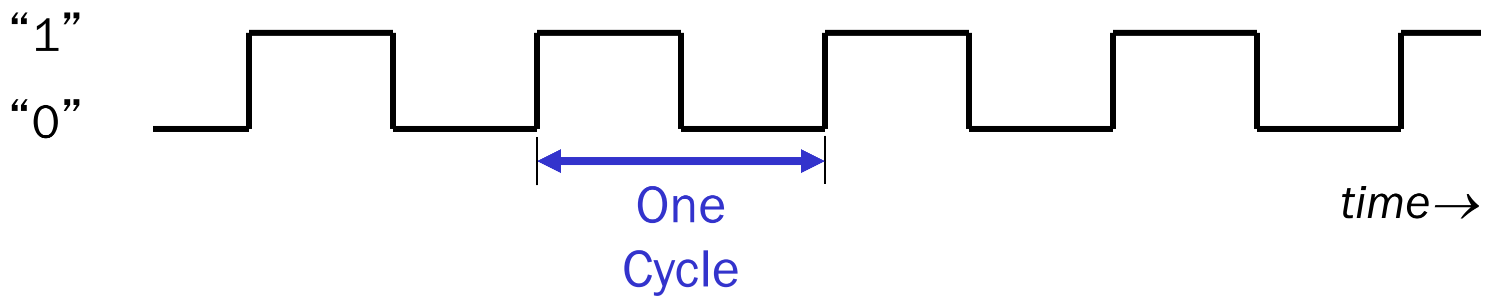
1. **A finite number of states**
2. **A finite number of external inputs**
3. **A finite number of external outputs**
4. **An explicit specification of all state transitions**
5. **An explicit specification of what determines each external output value**

Often described by a state diagram.

- **Inputs trigger state transitions.**
- **Outputs are associated with each state (or with each transition).**

The Clock

Frequently, a clock circuit triggers transition from one state to the next.



At the beginning of each clock cycle,
state machine makes a transition,
based on the current state and the external inputs.

- Not always required. In lock example, the input itself triggers a transition.

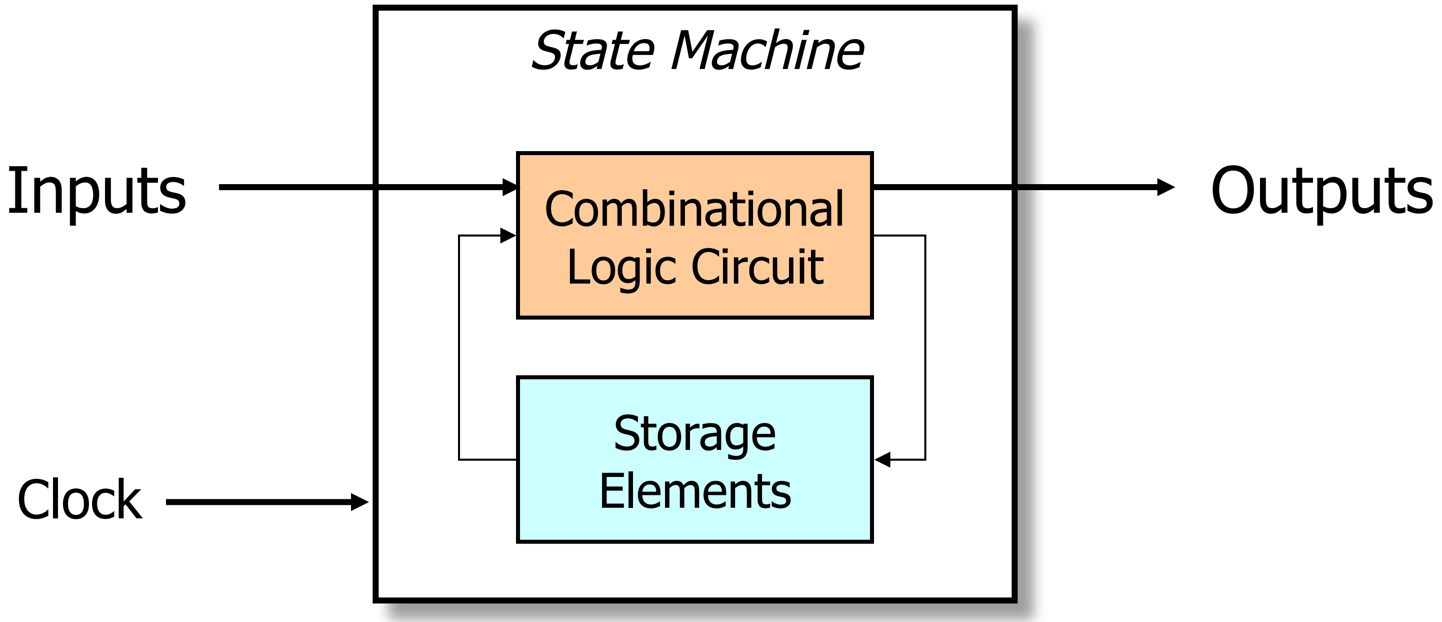
Implementing a Finite State Machine

Combinational logic

- Determine outputs and next state.

Storage elements

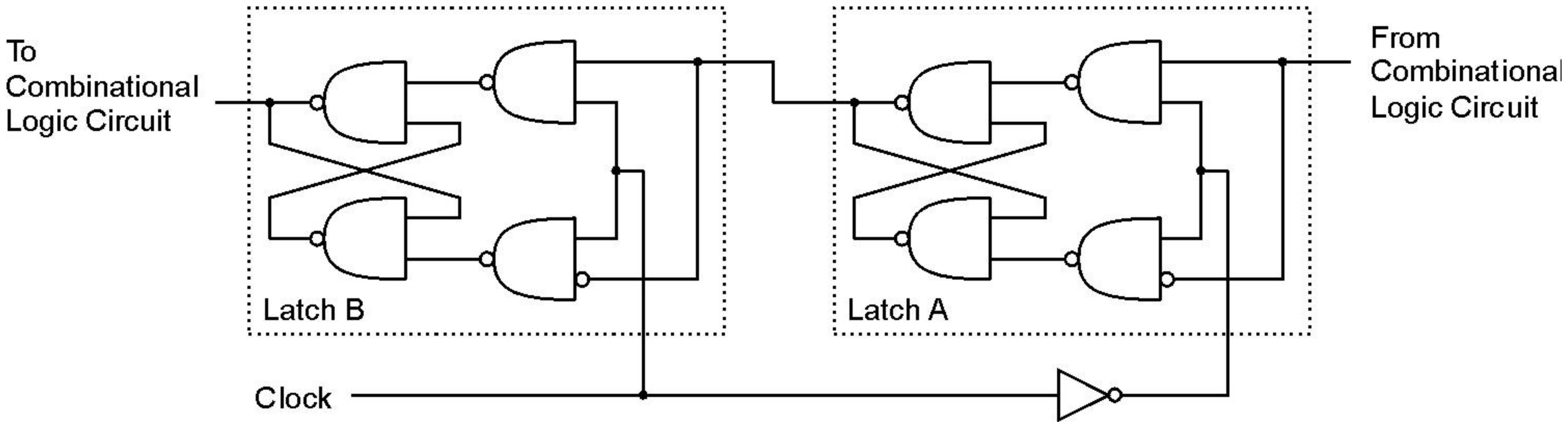
- Maintain state representation.



Storage: Master-Slave Flipflop

Skip: there are several
Possible designs

A pair of gated D-latches,
to isolate *next* state from *current* state.


To Combinational Logic Circuit
Latch B
Latch A
From Combinational Logic Circuit
Clock

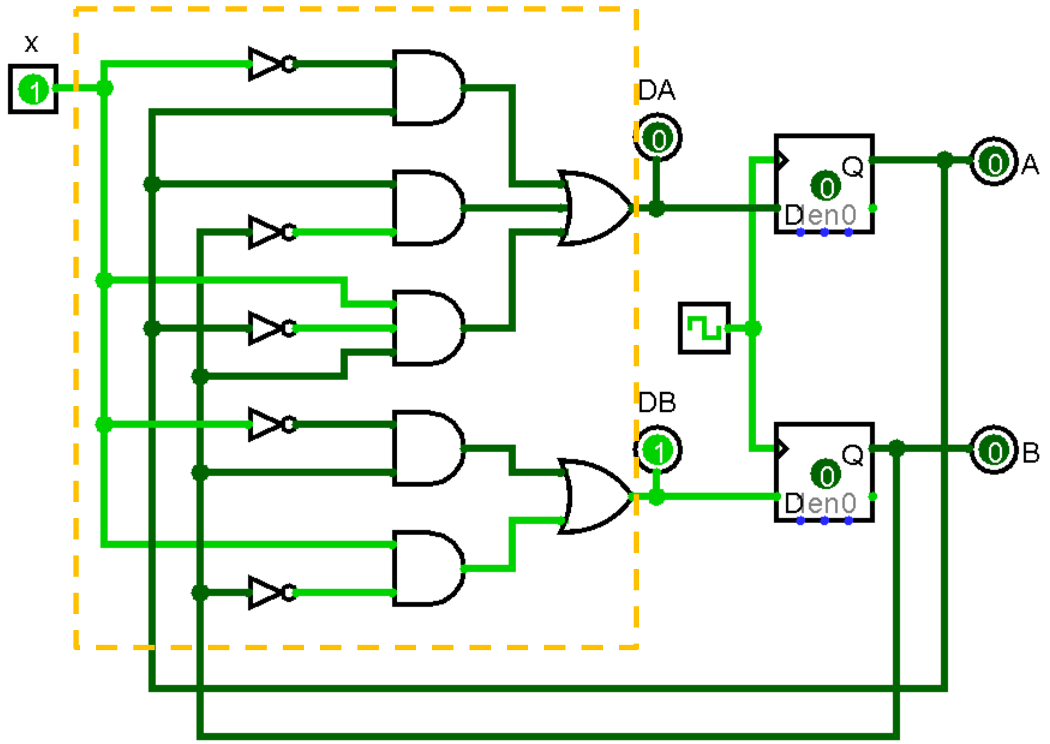
During 1st phase (clock=1), previously-computed state becomes *current* state and is sent to the logic circuit.

During 2nd phase (clock=0), *next* state, computed by logic circuit, is stored in Latch A.

Analyzing a FSM: Logic Circuit to State Diagram

1. Describe combinational circuit outputs using Boolean algebra.
2. Get the State Table for all possible Input/state combinations.
    1. Causes: Input, Present State
    2. Effects: Next State, Outputs (if different from State)
3. Get a state diagram. It provides a graphical description of the behavior.

Example 1: Analyze this FSM



Input: *x*
State: *A,B*
Output: *A, B*

Combinational block
In: *x, A, B* Out: *DA, DB*

$$DA = \overline{x}A + A\overline{B} + x\overline{A}B$$

$$DB = \overline{x}B + x\overline{B}$$

Example 1: Analyze this FSM (cont)

$$DA = \overline{x}A + A\overline{B} + x\overline{A}B$$

$$DB = \overline{x}B + x\overline{B}$$

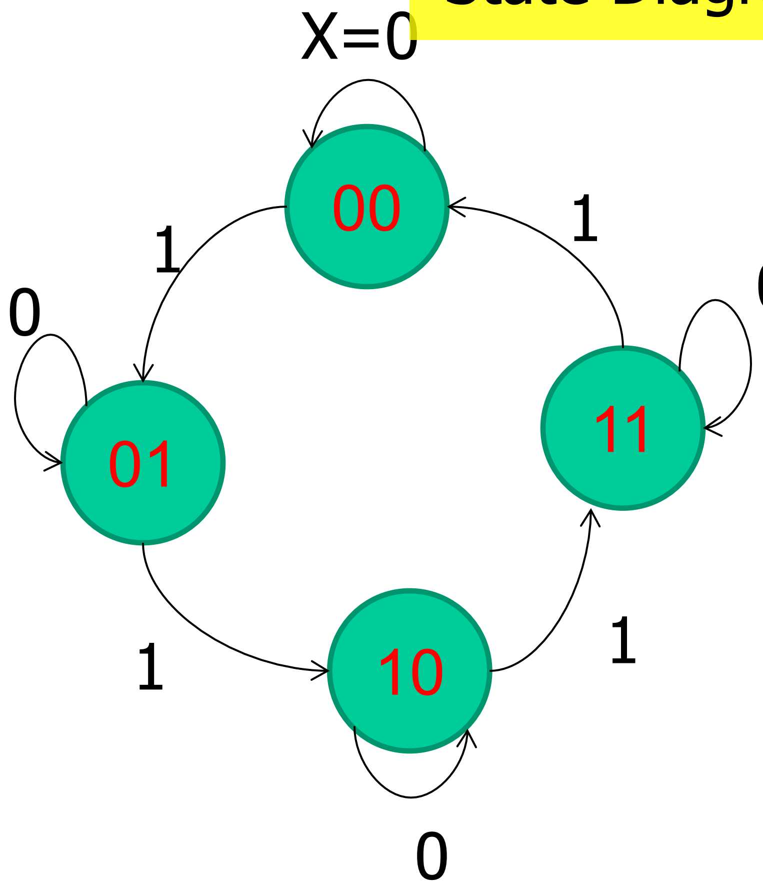
Input	Present State		Next State	
X	A	B	A	B
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Example 1: Analyze this FSM (last)

State Table

Input	Present State		Next State	
X	A	B	A	B
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

State Diagram



It is an up counter

Designing a FSM: Specification to Circuit

Reverse of Anaysis we have seen.

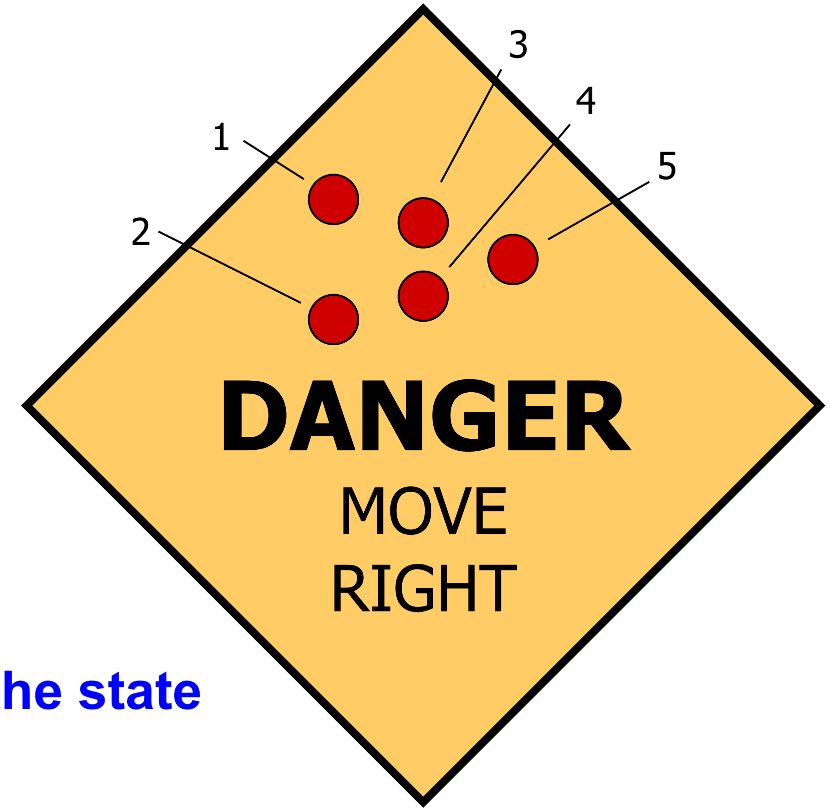
1. **Obtain a State Diagram using the specification. This is the challenging part. Determine the number of flip-flops needed (We will only use D type for convenience) and assign each state a unique binary combination.**
2. **Get the State Table for all possible Input/state combinations.**
    1. **Causes: Input, Present State**
    2. **Effects: Next State, Outputs (if different from State)**
3. **The Combinational circuit truth table is given by the State Table itself (Next state values are the D inputs). Design the combinational circuit. Optimize it as needed.**
4. **Get complete circuit with Combinational Logic, flip-flops att connected.**
5. **Simulate and verify the design.**

Example 2: Danger Sign design from P&P

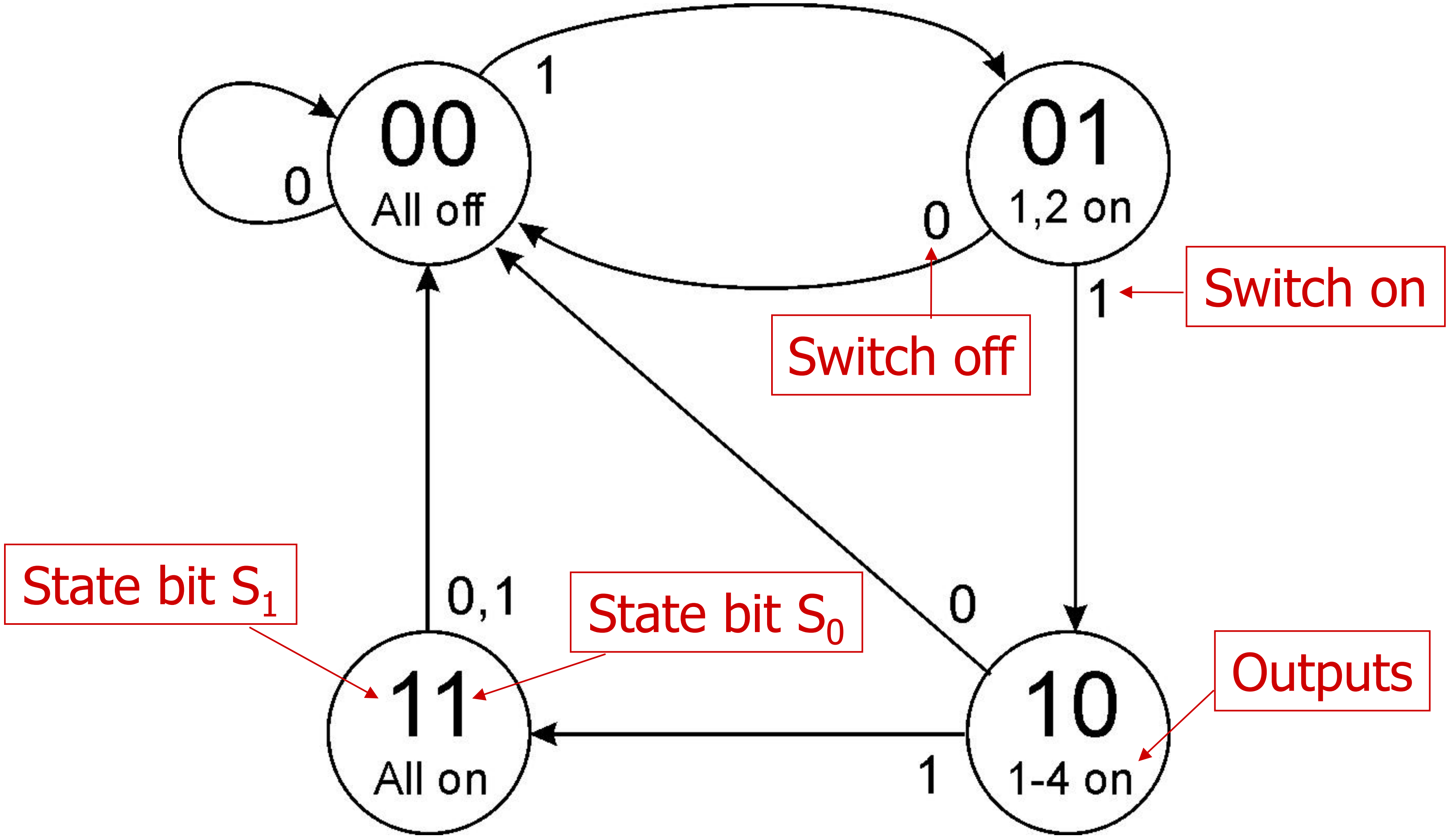
A blinking traffic sign

- No lights on
- 1 & 2 on
- 1, 2, 3, & 4 on
- 1, 2, 3, 4, & 5 on
- (repeat as long as switch is turned on)

- Input: Switch
- Outputs: Z, Y, X
- States: 4 (bits S1, S0)
- Choose: output depends on the state



Traffic Sign State Diagram



Transition on each clock cycle.

State Table

Input	Present State		Next State		Output
In	S1	S0	S1	S0	ZYX
0	0	0	0	0	000
0	0	1	0	0	100
0	1	0	0	0	110
0	1	1	0	0	111
1	0	0	0	1	000
1	0	1	1	0	100
1	1	0	1	1	110
1	1	1	0	0	111

See Logisim implementation
Present state: flipflop output
Next state: flipflop D inputs

Traffic Sign Truth Tables (from book)

Outputs
(depend only on state: S_1S_0)

Z: Lights 1 and 2

Y: Lights 3 and 4

X: Light 5

S_1	S_0	Z	Y	X
0	0	0	0	0
0	1	1	0	0
1	0	1	1	0
1	1	1	1	1

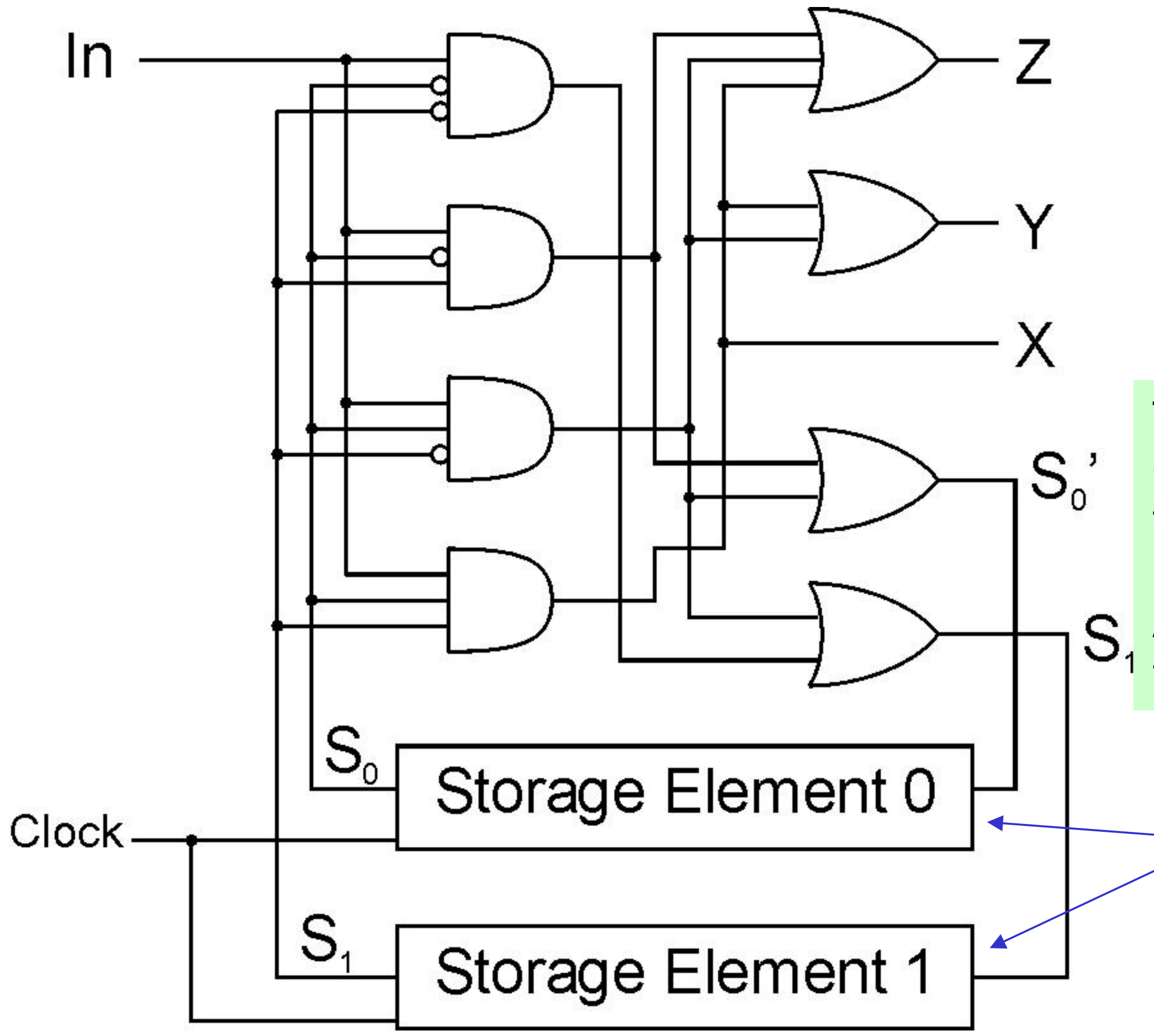
Next State: S_1' S_0'
(depend on state and input)

In: Switch

In	S_1	S_0	S_1'	S_0'
0	X	X	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Whenever In=0, next state is 00.

Traffic Sign Logic (from book)



This design is not optimized. Try Logisim to optimize.

Any edge triggered flip-flop will work.

Master-slave flipflop

From Logic to Data Path

The data path of a computer is all the logic used to process information.

- **See the data path of the LC-3 as an example.**

Combinational Logic

- **Decoders -- convert instructions into control signals**
- **Multiplexers -- select inputs and outputs**
- **ALU (Arithmetic and Logic Unit) -- operations on data**

Sequential Logic

- **State machine -- coordinate control signals and data movement**
- **Registers and latches -- storage elements**