## Overview

- Last Lecture:
  - **What is the course all about & why is it important?**
  - **What is a digital system? What is a binary digital system?**
  - **Boolean Algebra, Truth tables**
  - **Operators: inversion, and, or, xor, xnor (eq)**
  - **Design Example: Translating a word problem to a combinational logic function**
  - **Multiplexers, Implementing the example using a multiplexer**
- This Lecture:
  - **Design Example: Translating a word problem into a sequential design language**
  - **State Transition Graph**
  - **State Transition Table**
  - **Mealy and Moore Forms**



## Design Example: Automobile Lock

The automobile theft rate in Muldavia is so high that *CyclesPerSecond* rental car agency has decided to add a new security device to their cars. The initial system design has been completed and our consulting firm has been retained to implement the device. The agency designers hand us the following description:

"Please build us a small black box (2"x3"x0.5") we can attach to the dash that consists of a keypad (keys 0-9) and two LEDs, one green and one red. It should perform as follows:

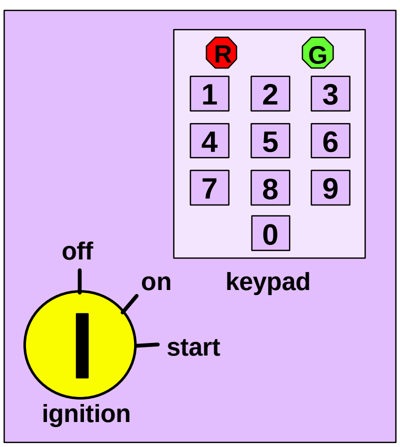
- When the **ignition is turned on**, the **red LED should light up** and the **keypad is activated**. However, **the car will not start**.
- If the **driver enters a correct four-digit code**, the **green LED goes on** as well and **the car can now be started** by **turning the ignition switch further clockwise** to the start position.
- If the **code is not correct**, the **green light will not go on and the car will not start**."

## Where Do We Start?

- Write down the inputs and outputs and list the (symbolic) values they can take.
- Choose a "language" in which to express the behavior of the machine. Is a truth table sufficient here? How would you use it?
- For sequential systems, we will start by using *state transition tables* or *state transition graphs*.
- We will be assuming *discrete-valued time* - "instant" to "instant." At any particular instant, the finite number of storage elements in the machine will have particular, well-defined values stored in them - we will be describing problems which can be implemented directly with a *Finite-State Machine.*

CS150 Newton/Pister 1.2.3

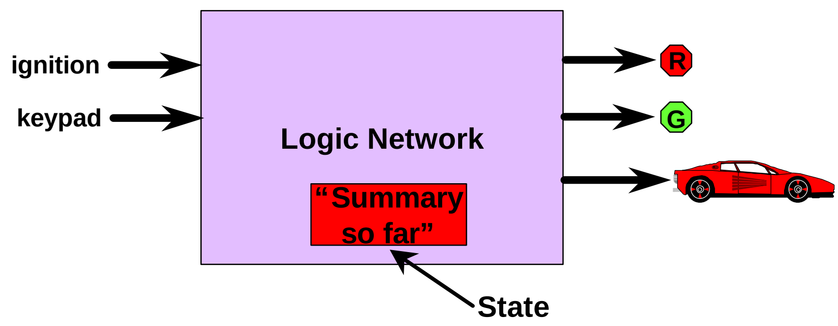
## Naming the Variables




- Inputs:
  - → Ignition: {off, on, start}
  - → Keypad: {0,…,9}
- Outputs:
  - → Red Light: {on, off}
  - → Green Light: {on, off}
  - → StartCar: {yes, no}
- What else do we need?
  - → Internal “state” (memory, store, “what’s happened up until now?”, “where are we?”

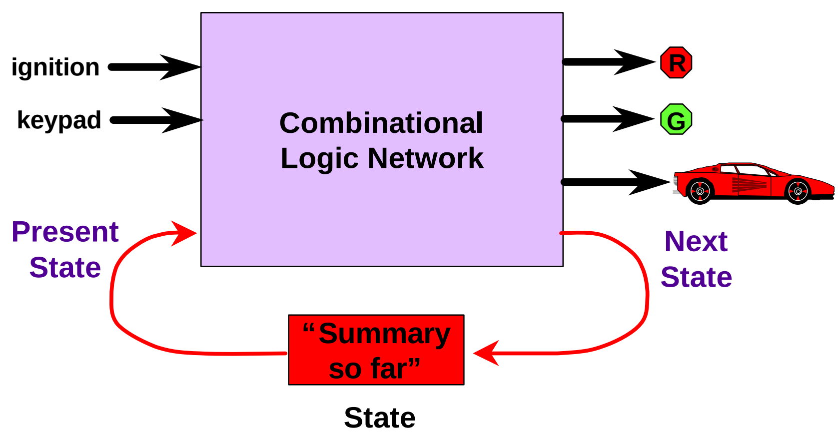
CS150 Newton/Pister 1.2.4

## Elements of a Finite-State Machine



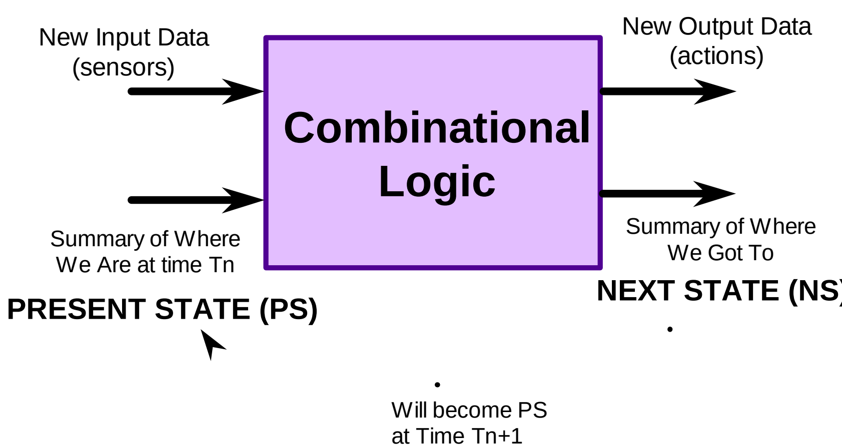

CS150 Newton/Pister 1.2.5

## Elements of a Finite-State Machine




CS150 Newton/Pister 1.2.6

## General Structure of Our Problem at Time Tn






## Encoding the Variables

- Inputs:
  - Ignition: {off, on, start}
  - Keypad: {0,…,9}

| Value: | off | on | start |
|---|---|---|---|
| ig | 00 | 01 | 11 |

| Value: | 0 | 1 | 2 | 3 | … |
|---|---|---|---|---|---|
| key | 0000 | 0001 | 0010 | 0011 | … |

- Outputs:
  - Red Light: {on, off}
  - Green Light: {on, off}
  - StartCar: {yes, no}

| Value: | off | on |
|---|---|---|
| R | 0 | 1 |

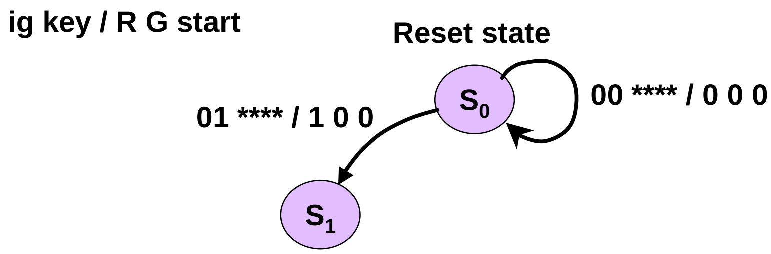
| Value: | off | on |
|---|---|---|
| G | 0 | 1 |

| Value: | no | yes |
|---|---|---|
| start | 0 | 1 |

- What else do we need?
  - Special value for when we “don’t care” what the value of an input (or an output) is.
  - ✳ or X or -

**Input vector: {ig | key }**
**output vector: { R | G | start }**
**notation: input/output**
**example:**
**00 **** / 0 0 0**

## Describing the Required Behavior




CS150 Newton/Pister 1.2.9

## Choosing a Language to Represent the Problem

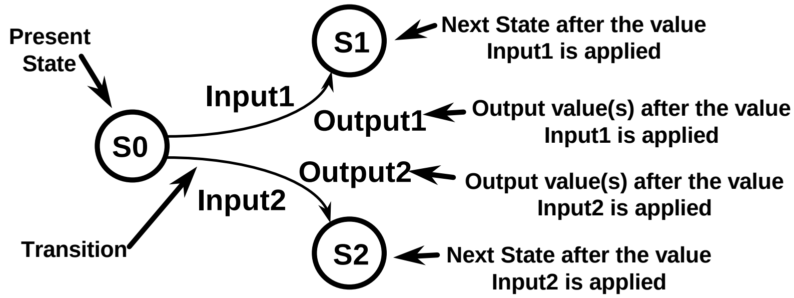
**We need a "language" to represent:**

**(1) The Present State of the machine**

**(2) For each possible input value:**

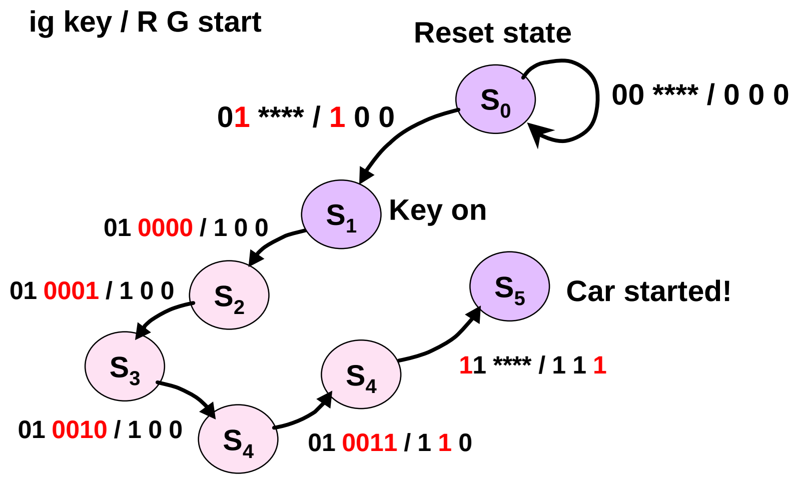
**(2a) The corresponding output value(s)**

**(2b) The corresponding Next State**




CS150 Newton/Pister 1.2.10

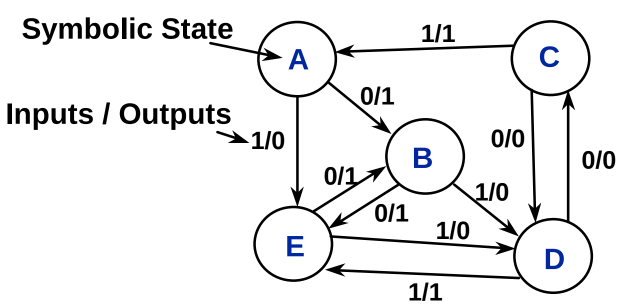
# Describing the Required Behavior




Correct: 0123



# Example Finite-State Machine State Transition Diagram (Mealy)

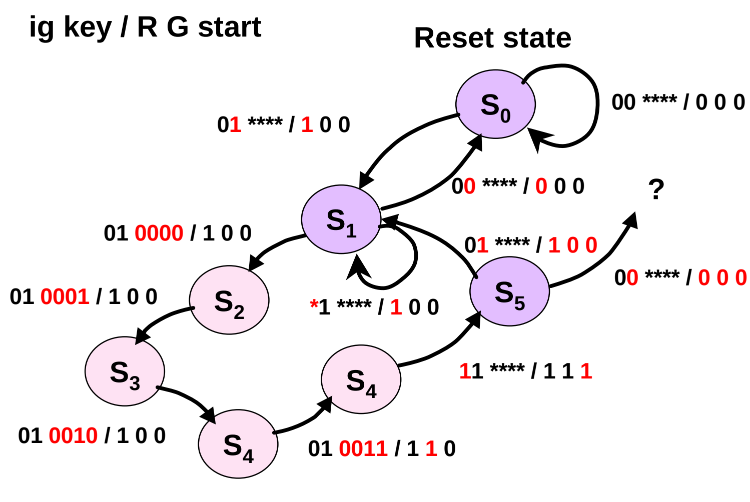


# Example Finite-State Machine State Transition Table (Mealy)

| Primary Input | 0 | | 1 | |
|---|---|---|---|---|
| **Present State** | **Next State** | **Primary Output** | **Next State** | **Primary Output** |
| A | B | 1 | E | 0 |
| B | E | 1 | D | 0 |
| C | D | 0 | A | 1 |
| D | C | 0 | E | 1 |
| E | D | 1 | D | 0 |

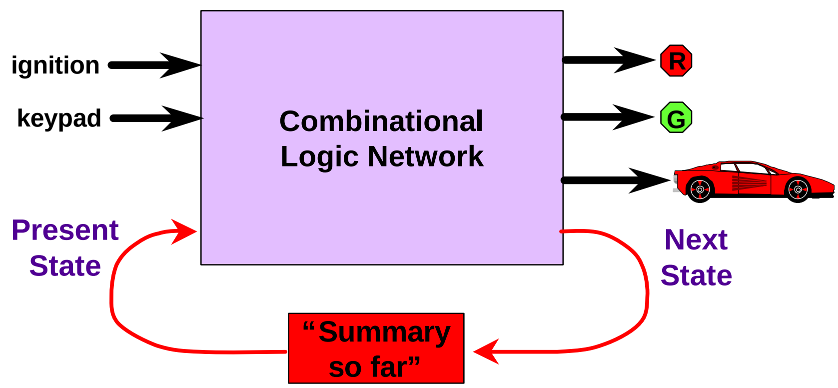


# Describing the Required Behavior




Correct: 0123

## Elements of a Finite-State Machine

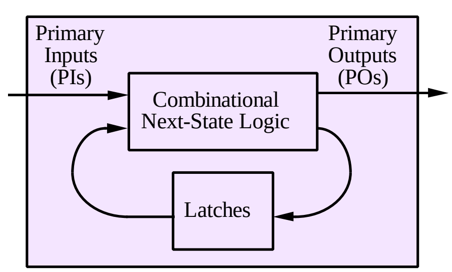



**State: $\{S_0, S_1, S_2, \ldots S_N\}$**

**Require $\lceil \log_2 N \rceil$ bits of "storage" to represent the state**

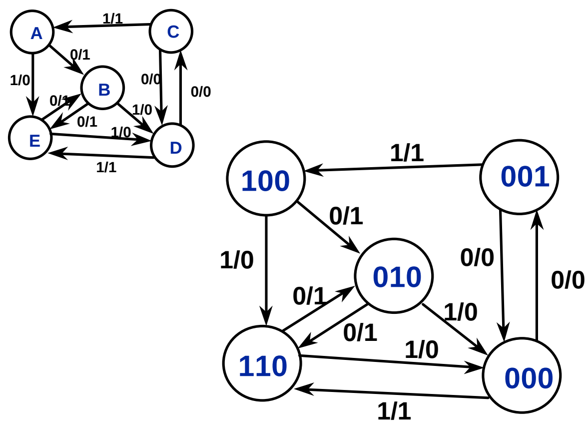


## Finite-State Machines




**Mealy Machine**

## Example Finite-State Machine Encoded States (Mealy)




011
101
111

CS150 Newton/Pister 1.2.17

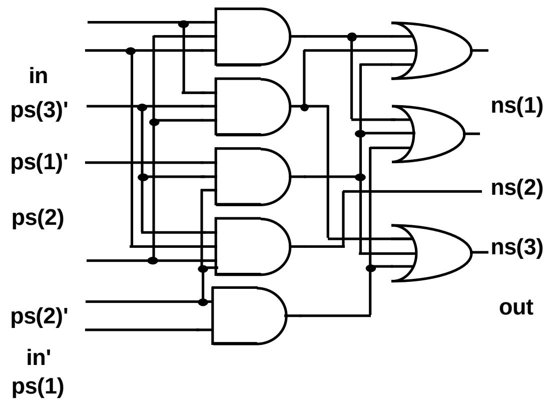
## Example Finite-State Machine Next-State Logic (Mealy)

| Inputs | | Outputs | |
|---|---|---|---|
| Present State | Input | Output | Next State |
| 100 | 0 | 1 | 010 |
| 100 | 1 | 0 | 110 |
| 010 | 0 | 1 | 110 |
| 010 | 1 | 0 | 000 |
| 001 | 0 | 0 | 000 |
| ... | ... | ... | ... |

**Combinational!**

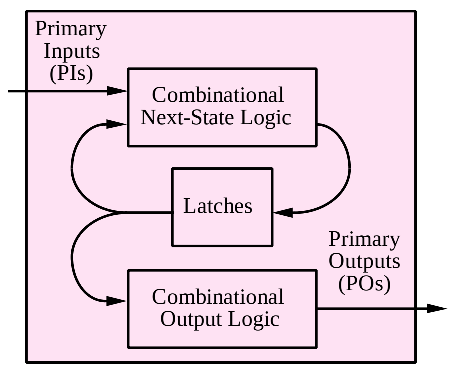
CS150 Newton/Pister 1.2.18

# Example Finite-State Machine Next-State Logic (Mealy)



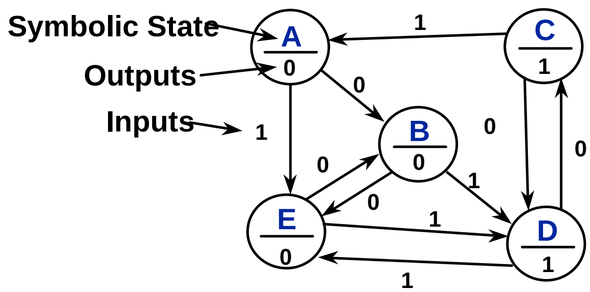



# Finite-State Machines




**Moore Machine**

# Example Finite-State Machine State Transition Diagram (Moore)



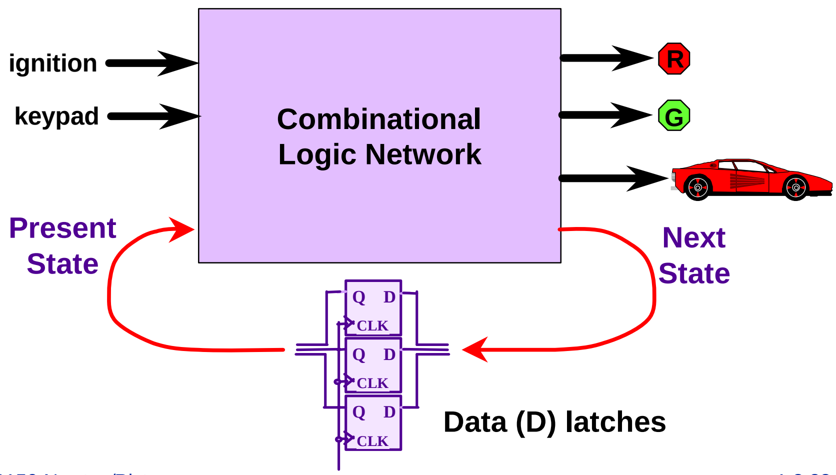

CS150 Newton/Pister 1.2.21

# Example Finite-State Machine State Transition Table (Moore)

| Present State | Next State | | Primary Output |
|---|---|---|---|
| Primary Input | 0 | 1 | |
| A | B | E | 0 |
| B | E | D | 0 |
| C | D | A | 1 |
| D | C | E | 1 |
| E | D | D | 0 |

CS150 Newton/Pister 1.2.22

## Elements of a Finite-State Machine




CS150 Newton/Pister 1.2.23