# ECE 250 / CPS 250
# Computer Architecture

# Basics of Logic Design
# Finite State Machines

**Benjamin Lee**

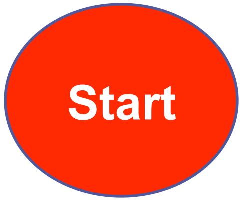
Slides based on those from

Andrew Hilton (Duke), Alvy Lebeck (Duke)
Benjamin Lee (Duke), and Amir Roth (Penn)

# Finite State Machine (FSM)

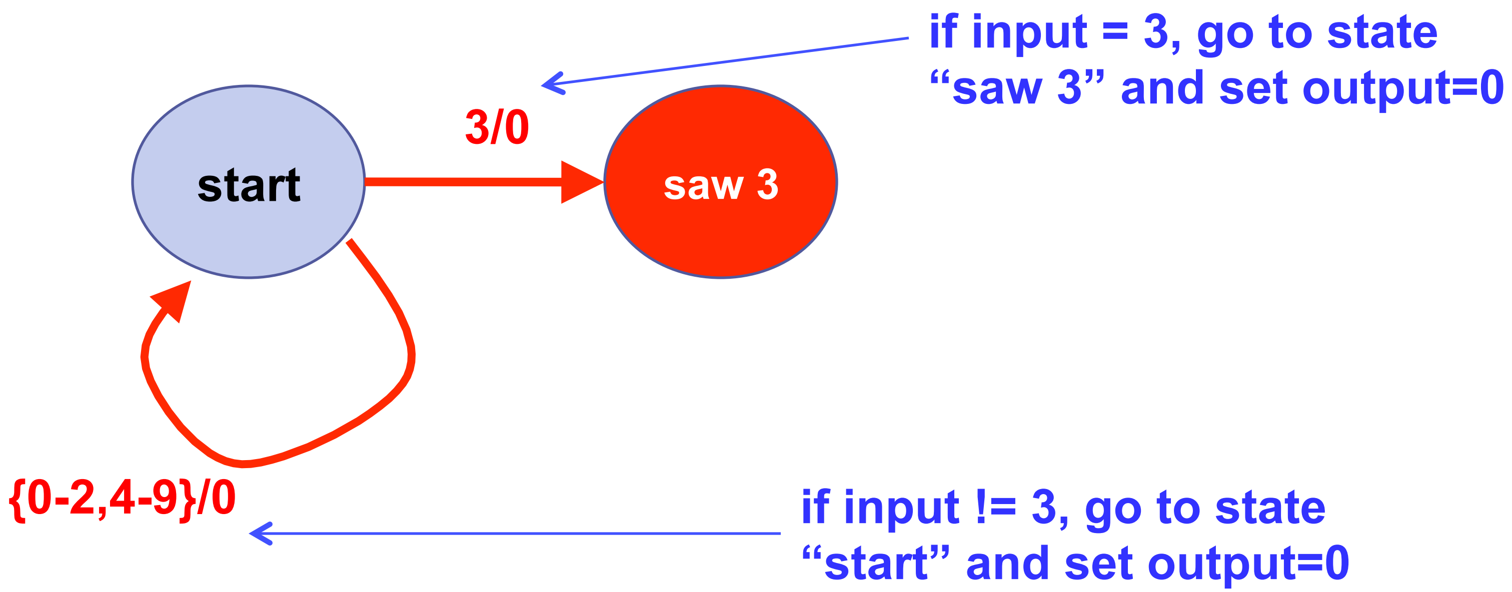
- FSM = States + Transitions
  - Next state = function (current state, inputs)
  - Outputs = function (current state, inputs)
- What you do depends on what state you're in
  - Think of a calculator … if you type "+3=", the result depends on what you did before, i.e., the state of the calculator
- Canonical Example: Combination Lock
  - Must enter 3 8 4 to unlock

# Finite State Machines: Example



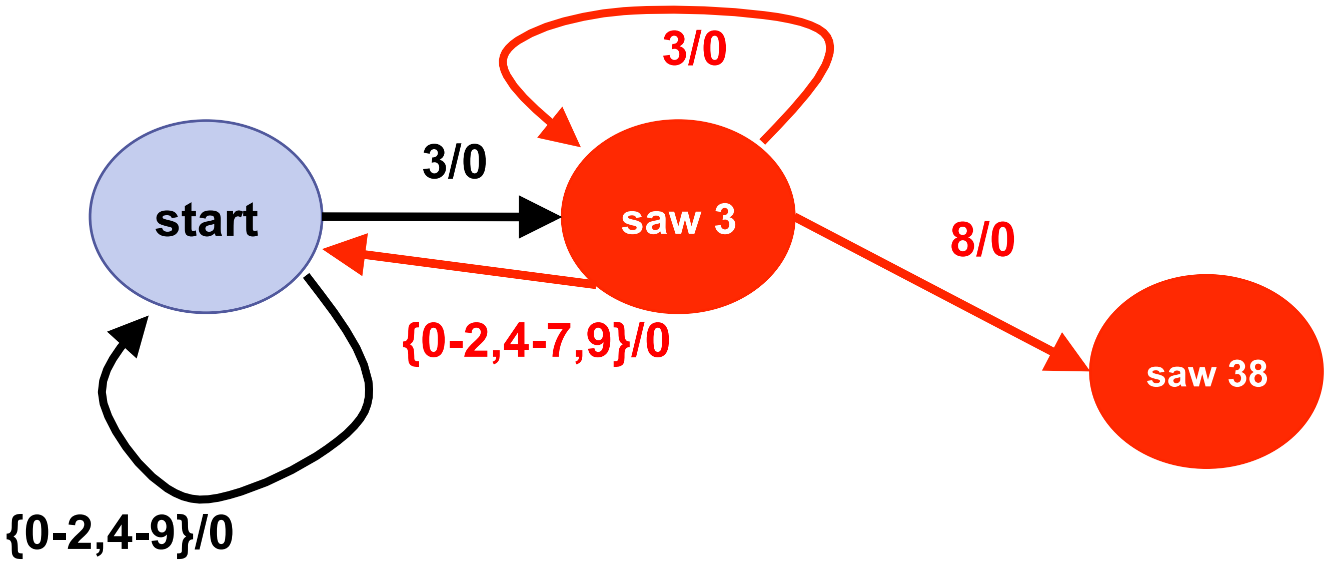

- Combination Lock Example:
  - Need to enter 3 8 4 to unlock
- Initial State called "start": no valid piece of combo seen
  - All FSMs get reset to their start state

# Finite State Machines: Example



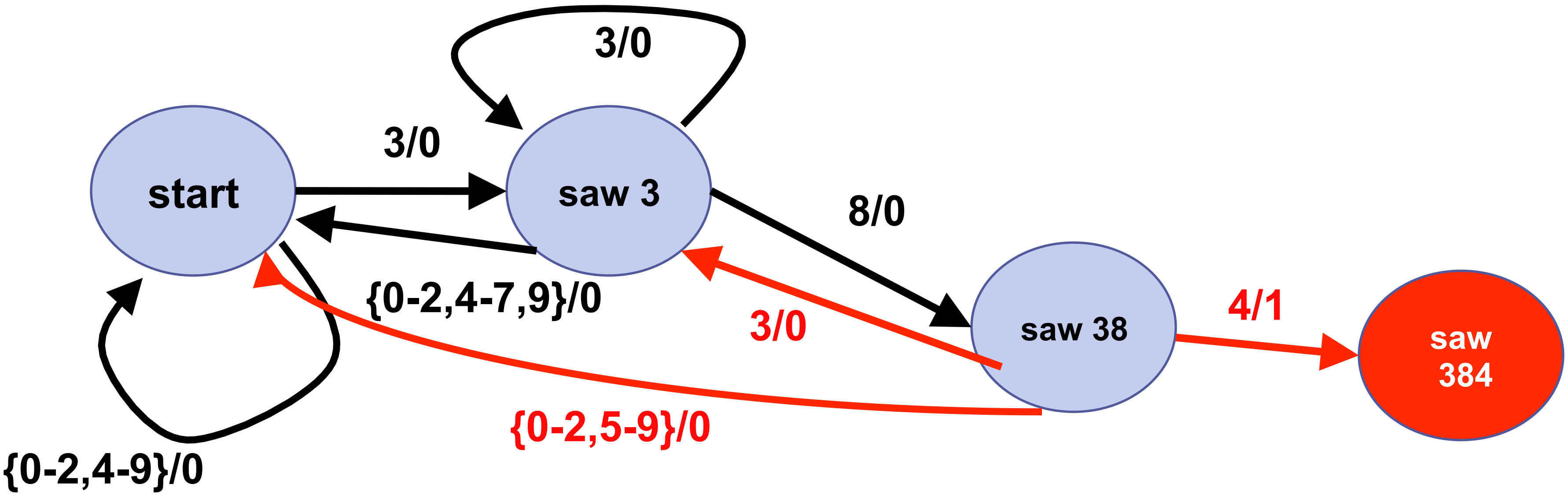

- Combination Lock Example:
  - Need to enter **3** 8 4 to unlock
- Input of 3: transition to new state, output=0
- Any other input: stay in same state, output=0

# Finite State Machines: Example



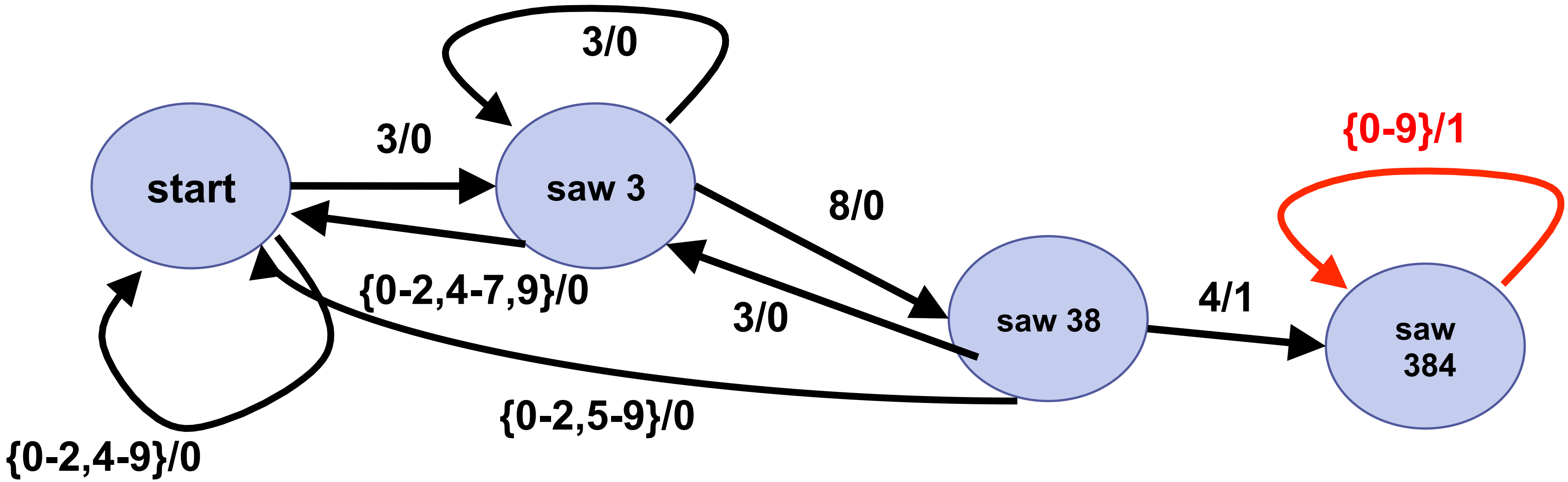

- Combination Lock Example:
  - Need to enter **3 8** 4 to unlock
- If in state "saw 3":
  - Input = 8? Goto state "saw 38" and output=0

# Finite State Machines: Example



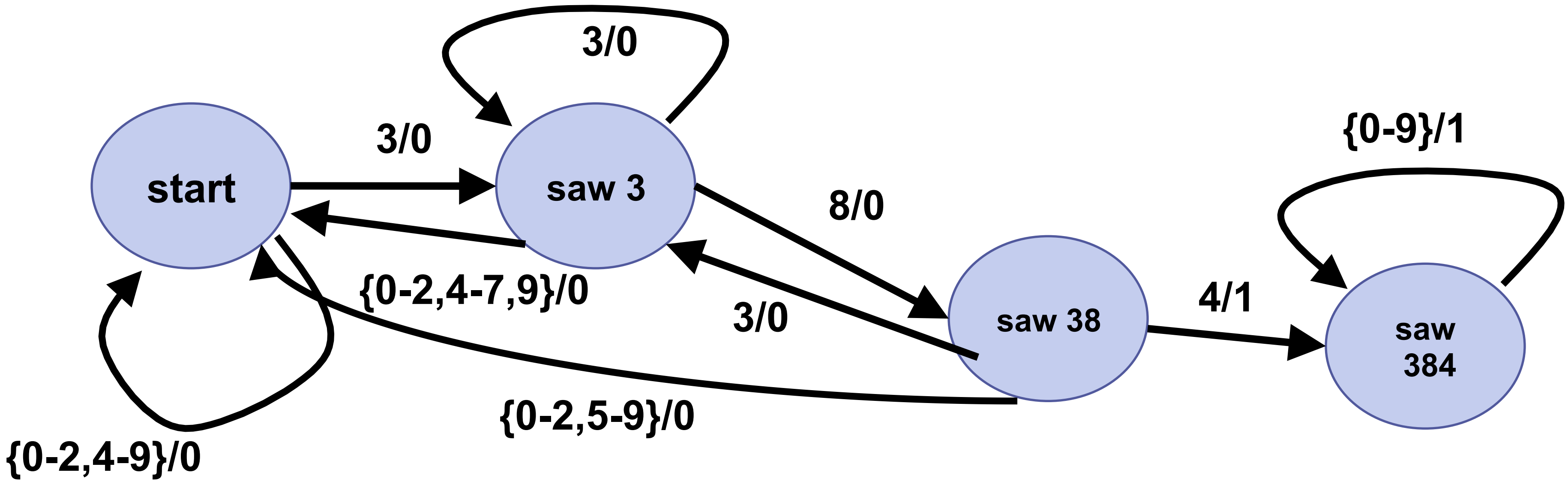

- Combination Lock Example:
  - Need to enter **3 8 4** to unlock
- If in state "saw 38":
  - Input = 4? Goto state "saw 384" and set output=1 → Unlock!

# Finite State Machines: Example




- Combination Lock Example:
  - Need to enter **3 8 4** to unlock
- If in state "saw 384":
  - Stay in this state forever and output=1

# Finite State Machines: Example




**In this picture, the circles are states.**
**The arcs between the states are transitions.**

**The figure is a state transition diagram, and it's the first thing you make when designing a finite state machine (FSM).**

# Finite State Machines: Caveats

**Do NOT assume all FSMs are like this one!**

**•A finite state machine (FSM) has at least two states, but can have many, many more. There's nothing sacred about 4 states (as in this example). Design your FSMs to have the appropriate number of states for the problem they're solving.**

- **Question: how many states would we need to detect sequence 384384?**

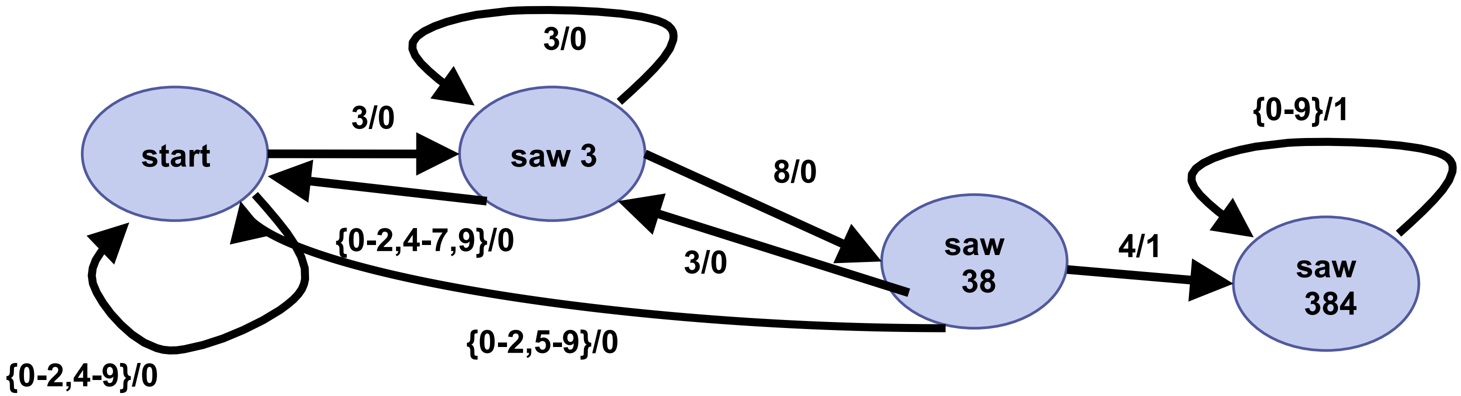
**•Most FSMs don't have state from which they can't escape.**

# FSM Types: Moore and Mealy

- Recall: FSM = States + Transitions
  - Next state = function (current state, inputs)
  - Outputs = function (current state, inputs)

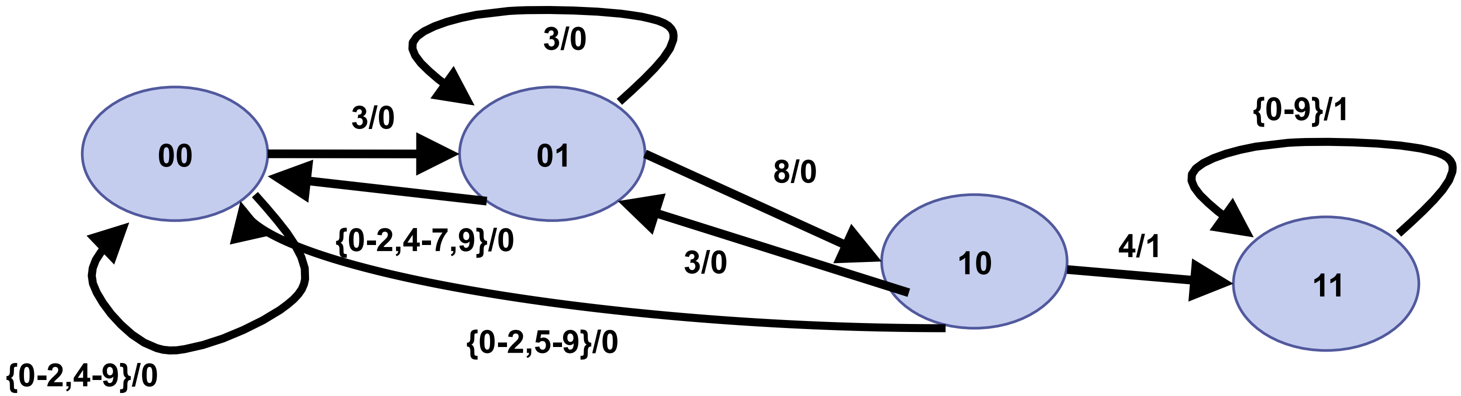
- This is the most general case
  - Called a “Mealy Machine”
  - We will assume Mealy Machines from now on
- A more restrictive FSM type is a “Moore Machine”
  - Outputs = function (current state)

# State Transition Diagram → Truth Table




| Current State | Input | Next state | Output |
|---|---|---|---|
| Start | 3 | Saw 3 | 0 (closed) |
| Start | Not 3 | Start | 0 |
| Saw 3 | 8 | Saw 38 | 0 |
| Saw 3 | 3 | Saw 3 | 0 |
| Saw 3 | Not 8 or 3 | Start | 0 |
| Saw 38 | 4 | Saw 384 | 1 (open) |
| Saw 38 | 3 | Saw 3 | 0 |
| Saw 38 | Not 4 or 3 | Start | 0 |
| Saw 384 | Any | Saw 3 | 1 |

# State Transition Diagram → Truth Table




**Digital logic → must represent everything in binary, including state names. But mapping is arbitrary!**

**We'll use this mapping:**
**start = 00**
**saw 3 = 01**
**saw 38 = 10**
**saw 384 = 11**

# State Transition Diagram → Truth Table

| Current State | Input | Next state | Output |
|---|---|---|---|
| 00 (start) | 3 | 01 | 0 (closed) |
| 00 | Not 3 | 00 | 0 |
| 01 | 8 | 10 | 0 |
| 01 | 3 | 01 | 0 |
| 01 | Not 8 or 3 | 00 | 0 |
| 10 | 4 | 11 | 1 (open) |
| 10 | 3 | 01 | 0 |
| 10 | Not 4 or 3 | 00 | 0 |
| 11 | Any | 11 | 1 |

**4 states → 2 flip-flops to hold the current state of the FSM**
**Next state given by inputs to flip-flops are $D_1D_0$**
**Current state given by outputs of flip-flops: $Q_1Q_0$**

# State Transition Diagram → Truth Table

| Q1 | Q0 | Input | D1 | D0 | Output |
|---|---|---|---|---|---|
| 0 | 0 | 3 | 0 | 1 | 0 (closed) |
| 0 | 0 | Not 3 | 0 | 0 | 0 |
| 0 | 1 | 8 | 1 | 0 | 0 |
| 0 | 1 | 3 | 0 | 1 | 0 |
| 0 | 1 | Not 8 or 3 | 0 | 0 | 0 |
| 1 | 0 | 4 | 1 | 1 | 1 (open) |
| 1 | 0 | 3 | 0 | 1 | 0 |
| 1 | 0 | Not 4 or 3 | 0 | 0 | 0 |
| 1 | 1 | Any | 1 | 1 | 1 |

**Input can be 0-9 → requires 4 bits**
**input bits are in3, in2, in1, in0**

# State Transition Diagram → Truth Table

| Q1 | Q0 | In3 | In2 | In1 | In0 | D1 | D0 | Output |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | Not 3 | | | | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | Not 8 or 3 | | | | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | Not 4 or 3 | | | | 0 | 0 | 0 |
| 1 | 1 | Any | | | | 1 | 1 | 1 |

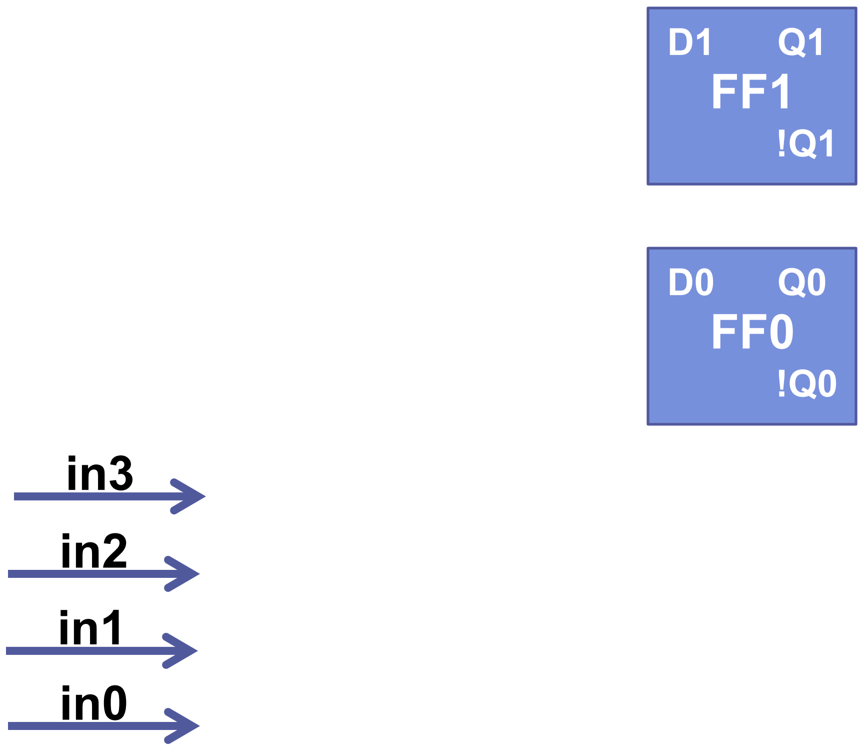
**From here, it's just like combinational logic design!**
**Write out product-of-sums equations, optimize, and build.**

# State Transition Diagram → Truth Table

| Q1 | Q0 | In3 | In2 | In1 | In0 | D1 | D0 | Output |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | Not 3 | | | | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | Not 8 or 3 | | | | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | Not 4 or 3 | | | | 0 | 0 | 0 |
| 1 | 1 | Any | | | | 1 | 1 | 1 |

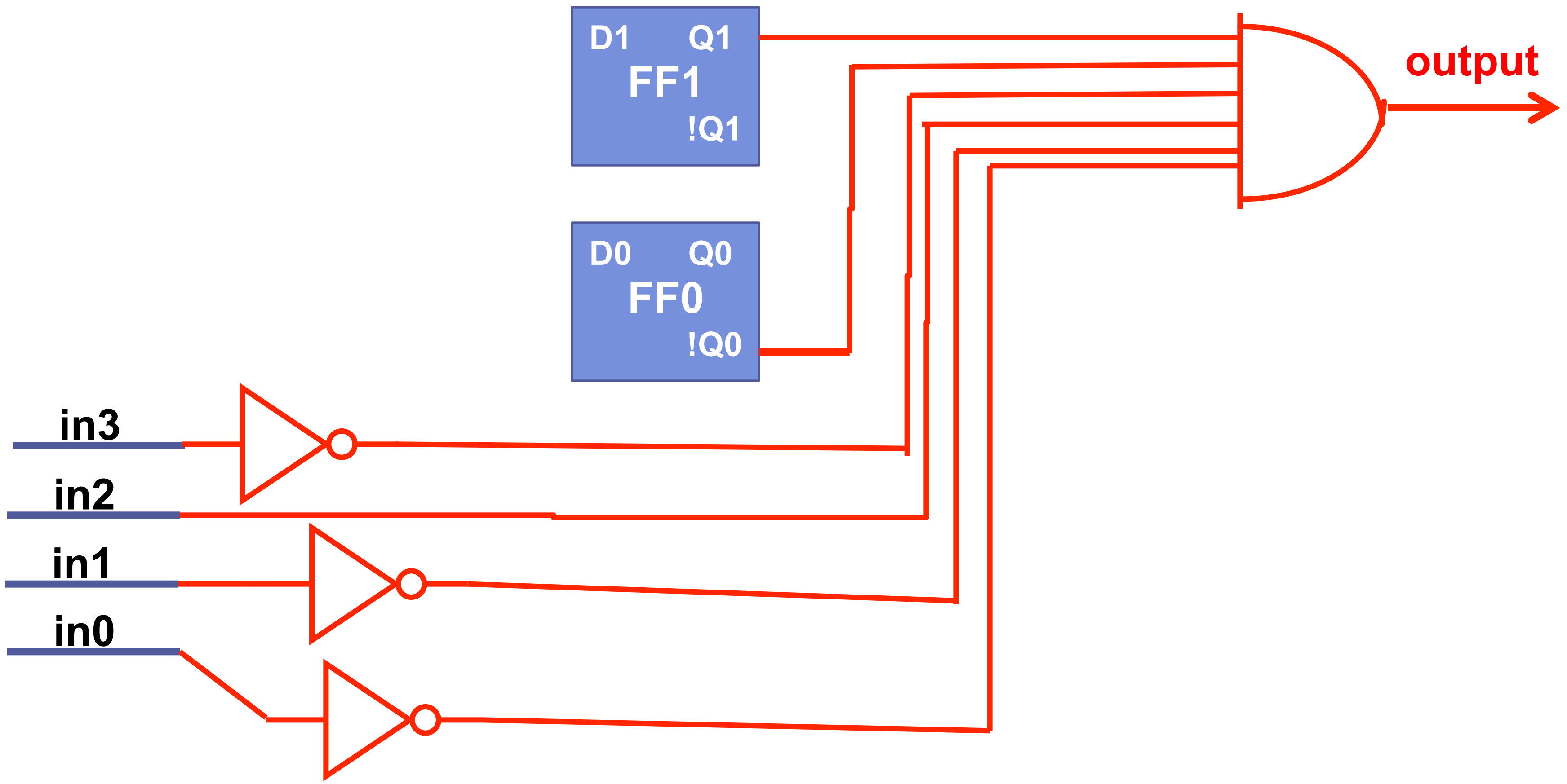
**Output = (Q1!•Q0•!In3•In2•!In1•!In0) + (Q1•Q0)**
**D1 = (!Q1•Q0•In3•!In2•!In1•!In0) + (Q1•!Q0•!In3•In2•!In1•!In0) + Q1•Q0**
**D0 = do the same thing**

# Truth Table → Sequential Circuit



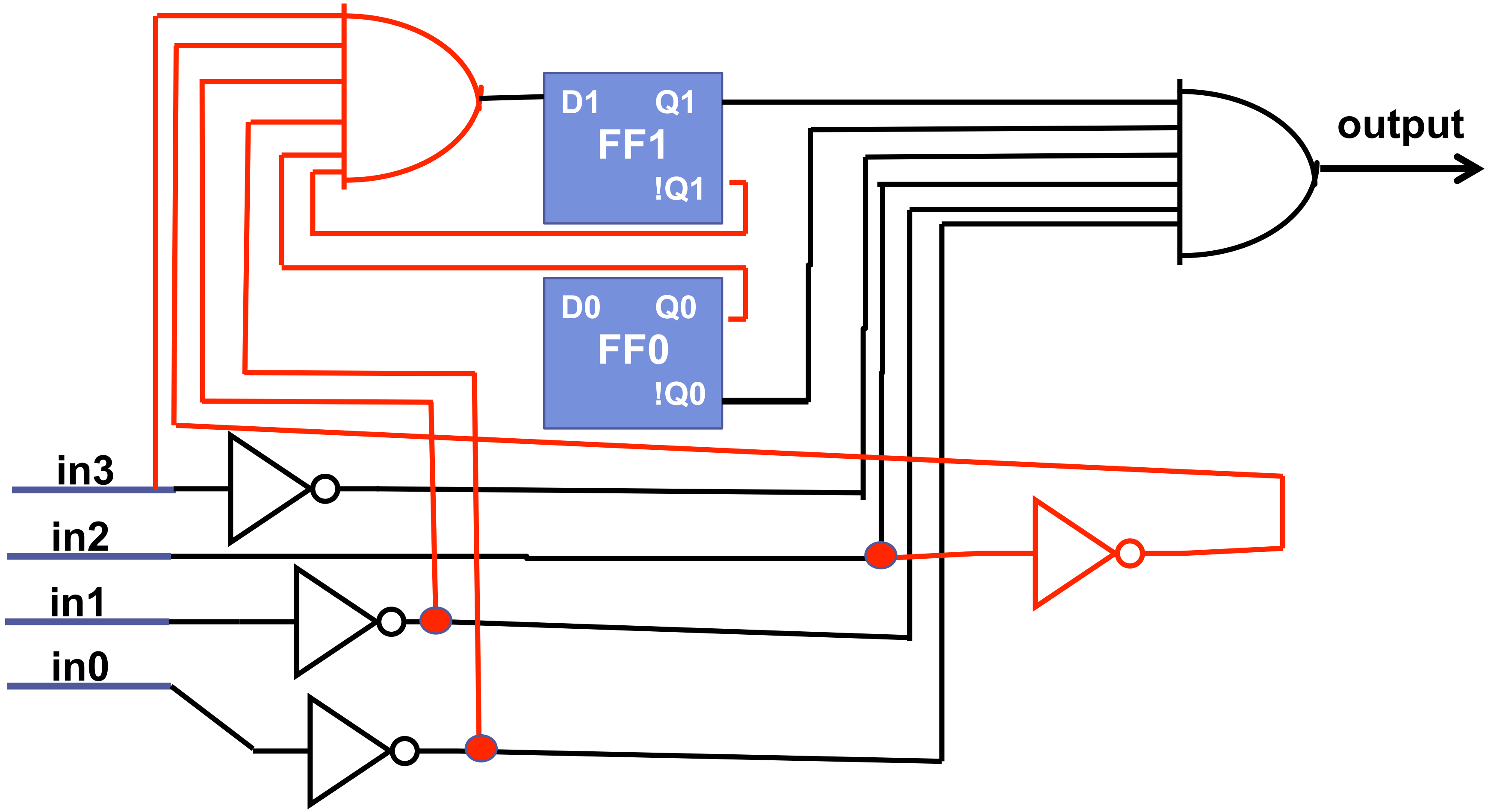

**Start with 2 FFs and 4 input bits. FFs hold current state of FSM. (not showing clock/enable inputs on flip flops)**

# Truth Table → Sequential Circuit



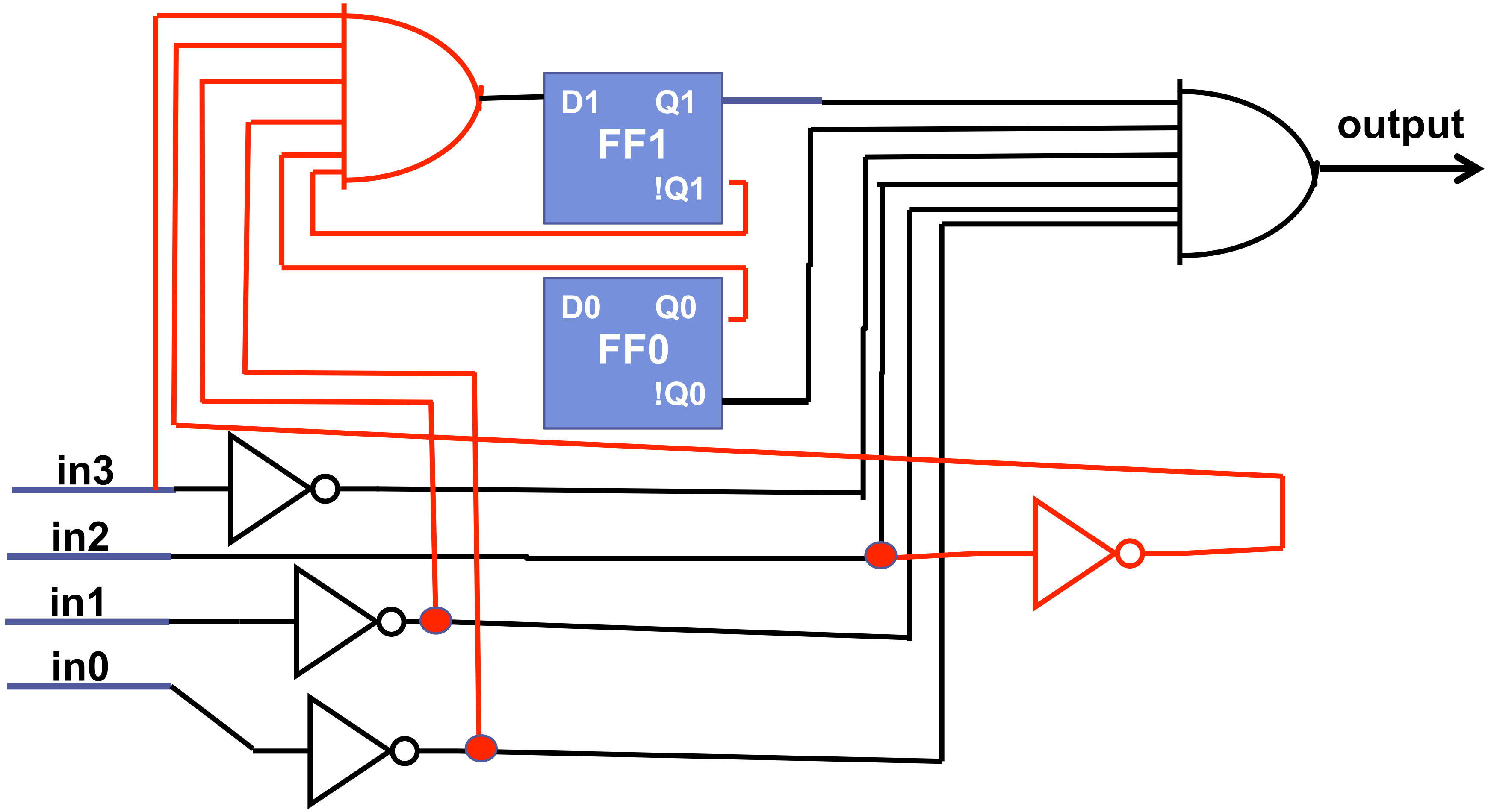

**output = (Q1•!Q0•!In3•In2•!In1•!In0)** + Q1•Q0

# Truth Table → Sequential Circuit




**D1 = (!Q1•Q0•In3•!In2•!In1•!In0)** + (Q1•!Q0•!In3•In2•!In1•!In0) + Q1•Q0

# Truth Table → Sequential Circuit




**D0 = ??   Fill this in and do it at home.**

# FSM Design Principles

- Systematic approach that always works:
  - Start with state transition diagram
  - Make truth table
  - Write out product-of-sums logic equations
  - Optimize logic equations (optional)
  - Implement logic in circuit
- Sometimes can do something non-systematic
  - Requires cleverness, but tough to do in general
- Do not do the following!
  - Use clock as an input (D input of FF)
  - Perform logic on clock signal