Finite State Machines

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Finite state machines are a common model for simple programs over fixed memory.

B. Outcomes

After this lecture, you should know

- What makes up a finite state machine (FSM) and how they execute.
- How to translate between FSMs state transition diagrams and tables and how to use them to trace FSM execution.

C. Finite State Machines

- One way to describe sequential computations uses **finite state machines (FSMs)**.
  - An FSM takes a sequence of symbols as input.
  - Different kinds of FSMs have different outputs.
  - The simplest kind of FSM is a **recognizer**: It just says "Yes" or "No" at the end of input.
    - The FSM **accepts** the input if it says "Yes,"
    - The FSM **rejects** the input if it says "No."
  - E.g. you can build an FSM that takes a sequence of 0's and 1's and accepts strings that include exactly two occurrences of 1.
  - More-complicated FSMs can produce output symbols during the calculation.
- **To specify an FSM** you specify
  - The **state set** (the set of all possible states).
  - The **start state** (the one the FSM begins execution in).

- The set of **accepting states** (when the FSM ends computation, it accepts the input iff it ends in an accepting state).
- A **transition function** that describes how the FSM executes.

- **FSM Implementation**
  - Implement FSM using sequential logic circuit
  - FSM states is unsigned bitstring of some fixed length.
  - An FSM is always in a state (which is why we need a start state).
  - The set of states isn't modifiable during execution.
- **FSM Execution**:

  -
    ```
    Initialize state ← start state;
    while there exists more input {
        read character of input
        set state to new state
        i.e., state ← transition_function(input_char, state)
    }
    if state ∈ Accepting States then
        print "Accept"
    else
        print "Reject"
    ```
- **FSM Transition Function**:
  - The transition function takes an input character and the current state; it produces the next state for the machine (may change the state or stay in the current state).
- **FSM Example**:
  - Machine M_1 will read strings of `a`'s and `b`'s and accept strings that contain exactly two occurrences of `b`.
  - We'll have four states named S_0, S_1, S_2, and S_3.
    - For $k = 0$, 1, or 2, we'll be in state S_k iff we've seen exactly k `b`'s.
    - We'll be in state S_3 if we've seen three or more `b`'s.
    - State S_0 is the initial state, state S_2 is the (only) accepting state.

- The transitions:
    - For $k \in \{0, 1, 2\}$, if we're in state S_k and the input is `b`, we go to state S_{k+1}.
    - If we're in state S_3 and the input is `b`, we stay in S_3.
    - For every state, if we see an `a`, we stay in that state.
- Sample execution
    - One way to describe an execution of an FSM uses two rows.
    - The top row contains the symbols of the input.
    - The bottom row contains the states that the FSM is in as it reads the input.
    - We stagger the columns for visibility
    - Here's a run with input `abaabaa`. Since we end in state S_2, we accept the input.
    - Note the leftmost column includes no symbol but the state S_0; this is how we describe starting in state S_0.

	a		b		a		a		b		a		a	
S_0		S_0		S_1		S_1		S_1		S_2		S_2		S_2

- (Quick question: What strings would we accept if we change the accepting state from S_2 to S_3? What if we make every state accepting *except for* S_2?)

D. Representing Transition Functions

- **State Transition Table:** One way to describe a finite state machine's transition function uses a table with three columns:
    - The first two columns list a state and input symbol. The third column lists the state we go to if we see the given state and input symbol.

- **Example**: To the right is a transition table for the machine M_1 above.
    - To get a complete description of M_1, we must also list the initial state (i.e., S_0) and accepting state(s) (i.e., S_2.)

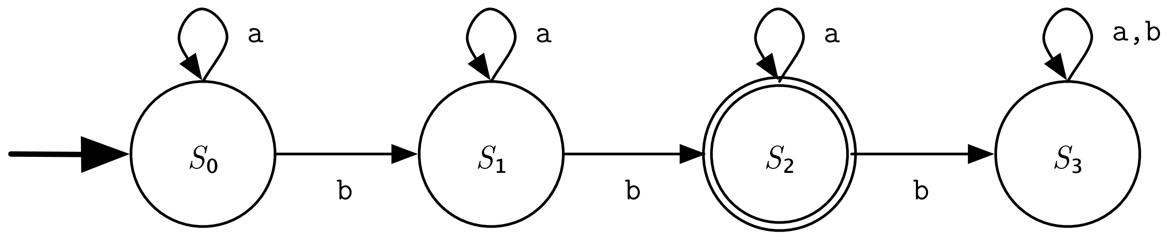
Machine M_1

State	***Input***	***New State***
S_0	a	S_0
S_0	b	S_1
S_1	a	S_1
S_1	b	S_2
S_2	a	S_2
S_2	b	S_3
S_3	a	S_3
S_3	b	S_3

Initial state: S_0; Accepting state: S_2

- **State Transition Diagram**: The other technique for representing an FSM's state transition function uses a directed graph:
    - Each state is represented as a circle.
    - Each transition is represented as an arc from the old state to the new state, with the arc labelled by the input symbol.
    - The initial state is indicated by drawing an arrow to it but with no source.
    - To indicate that a state is an accepting state, we draw a concentric circle on it.
- (By the way, it's okay for the initial state to be an accepting state, though it doesn't have to be.)

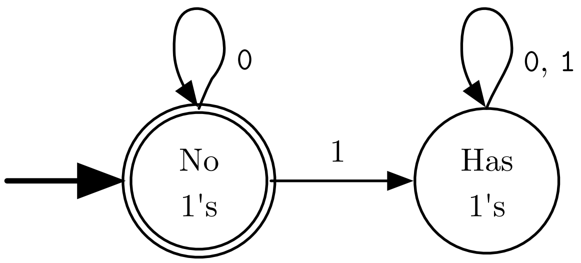
- **Example**: Here is a state transition table for the machine M_1 above.



- **Example**: Here's an FSM that reads strings of `0`'s and `1`'s and accepts iff the string only contained `0`'s.

State	Input	New State
No 1's	`0`	No 1's
No 1's	`1`	Has 1's
Has 1's	`0`	Has 1's
Has 1's	`1`	Has 1's

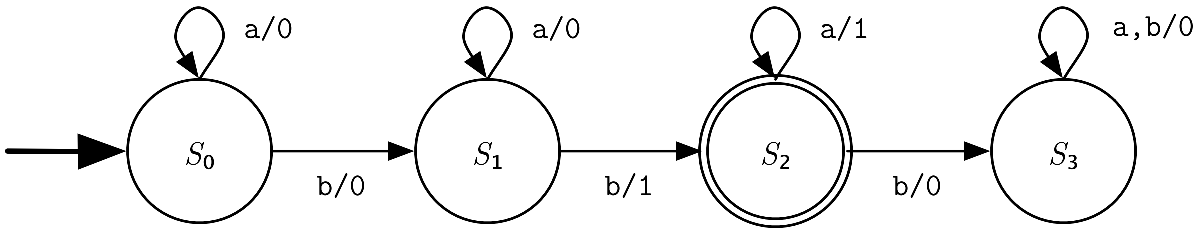
State Table for All-zero recognizer
Start and accepting state: No 1's



State Diagram for All-zero recognizer

E. Finite State Machines With Output

- An FSM with output generates a string of symbols as it processes its input.
- Two kinds of FSMs with outputs
  - A **Mealy machine** associates an output with each transition.
    - With transition table, add another column for the output.
    - With transition diagram, add slash and output to each arc.
  - A **Moore machine** associates an output with each state
    - Add another table mapping states to outputs
- **Example** (output on transition): Let machine M_2 extend M_1: Output a `1` for the two transitions that lead to S_2; output a `0` for the other transitions.
  - With input `abababab`, we output `00011000` .



State	***Input***	***New State***	***Output***
S_0	a	S_0	0
S_0	b	S_1	0
S_1	a	S_1	0
S_1	b	S_2	1
S_2	a	S_2	1
S_2	b	S_3	0
S_3	a	S_3	0
S_3	b	S_3	0

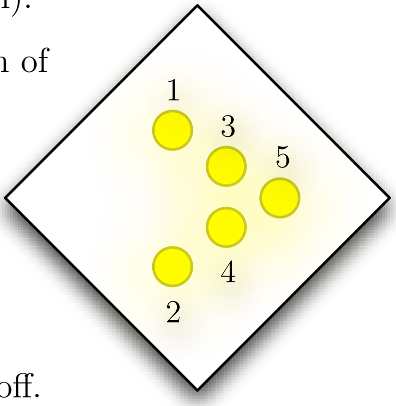
M_2: A Finite State Machine with Output
Start state: S_0; Accepting state: S_2.

- **Example** (output by state): Let M_3 extend M_1 with output `0` for states S_0, S_1, and S_3; output `1` for S_2. With input **`abababab`**, we get output `00001100` . In the table below, (*) indicates a difference in output from the M_2 machine of the previous example.

State	***Output***	***Input***	***New State***
S_0	0	a	S_0
S_0	0	b	S_1
S_1	0	a	S_1
S_1	0*	b	S_2
S_2	1	a	S_2
S_2	1*	b	S_3
S_3	0	a	S_3
S_3	0	b	S_3

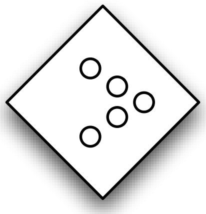
F. Textbook's Flashing Warning Sign Example

- The textbook's flashing warning sign example associates outputs with states.
  - The sign has **one bit of input** (an on/off switch).
  - It has **two bits of memory** (to remember which of four states we're in).
  - It has five outputs, one bit each, to control five lights.
  - But actually, we can get by with **three outputs**, one for (**Lights 1 & 2**) on/off, one for (**Lights 3 & 4**) on/off, and one for **Light 5** on/off.
  - The lights are arranged in a rightward V-shaped arrow.

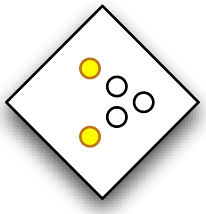


Light Numbers

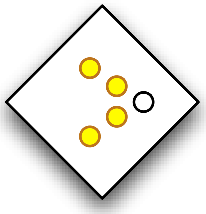
- For the finite state machine, the input will be the value of the switch S (0 or 1 for off and on). With each clock beat, we check the switch to see if it's on or off, so our input is a sequence of 0s and 1s.
- If the switch is on, the lights cycle as in the states below: 0, 1, 2, 3, 0, 1, 2, 3, etc. If the switch is off, the lights go off and we go to state 0; if we're already in state 0, we keep the lights off and remain in state 0.



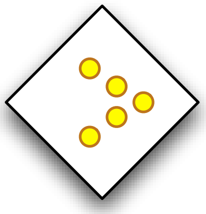
State 0
No Lights On



State 1
Lights 1–2 On



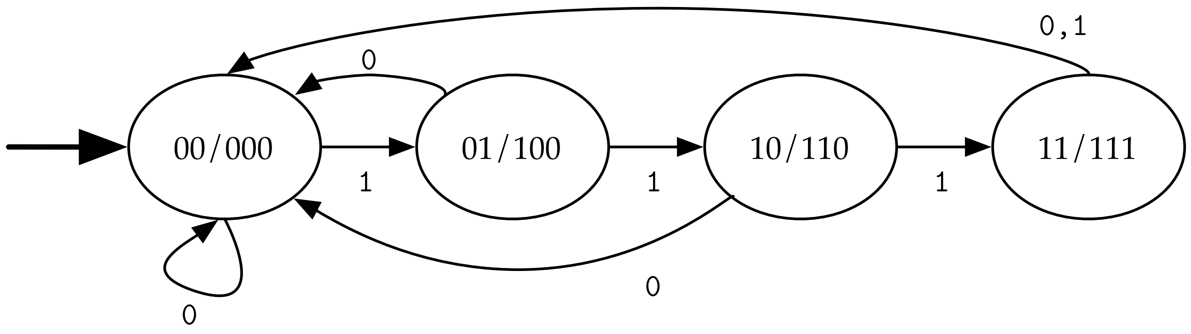
State 2
Lights 1–4 On



State 3
Lights 1–5 On

State	**Lights 1 & 2**	**Lights 3 & 4**	**Light 5**
00	0	0	0
01	1	0	0
10	1	1	0
11	1	1	1

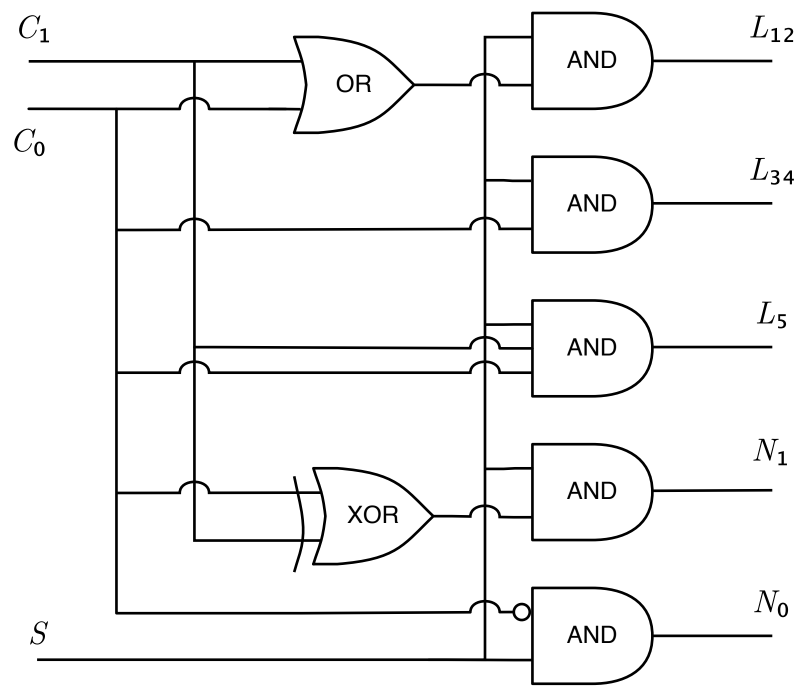
- Here's a state transition diagram: Each state label includes a slash and the three bits of output for that state (Lights 1 & 2, Lights 3 & 4, and Light 5). The arcs are labeled with the input *S*. There's no accepting state indicated because we only run the machine for its output.



State Transition Diagram for Flashing Warning Sign

- To implement this FSM in hardware, we need to describe the output bits and new state bits using combinatorial equations. In a departure from the book's notation, I'm going to use C for the current state, N for the new state, and L_{12}, L_{34}, and L_5 to indicate whether the indicated lights should be on or off. (Recall bits are numbered from right-to-left, so $C =$ the string $C_1\ C_0$ and $N =$ the string $N_1\ N_0$.)

$L_{12} = S\,(C_1 + C_0)$	Lights 1 and 2 are on in states 01, 10, and 11
$L_{34} = S\,C_1$	Lights 3 and 4 are on in states 10 or 11
$L_5 = S\,C_1\,C_0$	Light 5 is on in state 11
$N_1 = S\,(C_1 \oplus C_0)$	From states 01 and 10 we go to states 10 and 11
$N_0 = S\,\overline{C}_0$	From states 00 and 10 we go to states 01 and 11



Circuit for Flashing Warning Light

Finite State Machines

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Finite state machines are a common model for simple programs over a fixed amount of memory.

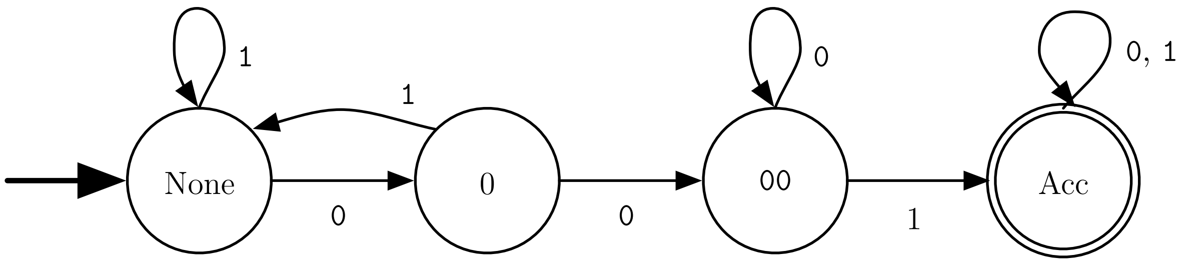
B. Objectives

At the end of this activity, you should:

- Be able to trace the execution of a finite state machine using its state transition table or state transition diagram.
- Be able to translate a state diagram for a finite state machine to an equivalent state transition table.

C. Questions

1. Here is a the state diagram for a finite state machine.



(a) Trace the execution of this machine on the input `0100010`.

(b) What is the pattern of strings accepted by this machine?

(c) Complete the state transition table below for this machine.

State	***Input***	***New State***
None	`0`	0
None	`1`	None
0		
0		
00		
00		
Acc		
Acc		

2. We'd like a finite state machine that takes an input string of `0`'s and `1`'s and has two states E and O (for even and odd), which it uses to keep track of whether the input has an even or odd number of `0`'s. It accepts strings with even numbers of `0`'s. (a) Draw a state diagram for the machine. (b) Give a state transition table for the machine.

3. Fill in the following trace of execution table for the flashing warning sign FSM.

C_1 (State hi bit)	0	0						
C_0 (State low bit)	0	0						
S (On-Off Switch)	0	1	1	1	1	1	0	0
L_{12} (Lights 1, 2)	0	0						
L_{34} (Lights 3, 4)	0	0						
L_5 (Light 5)	0	0						
N_1 (new C_1)	0	0						
N_0 (new C_0)	0	1						

Solution

1a. (Execution trace):

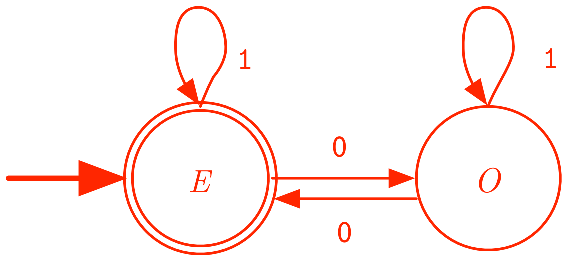
	`0`		`1`		`0`		`0`		`0`		`1`		`1`		`0`	
None		0		None		0		00		00		Acc		Acc		Acc

1b. It accepts any string that contains a `001`

1c. (Transition table): Initial state: “None”; accepting state: Acc.

State	***Input***	***New State***
None	`0`	0
None	`1`	None
0	`0`	00
0	`1`	None
00	`0`	00
00	`1`	Acc
Acc	`0`	Acc
Acc	`1`	Acc

2a. Note that since we’re tracking only the `0`’s, the `1`’s don’t change the state. Each `0` changes the state from even to odd (or vice versa).



2b. Initial and accepting state: E

State	***Input***	***New State***
E	`0`	O
E	`1`	E
O	`0`	E
O	`1`	O

3. Output and transition table:

C_1 (State hi bit)	0	0	0	1	1	0	0	0
C_0 (State low bit)	0	0	1	0	1	0	1	0
S (On-Off Switch)	0	1	1	1	1	1	0	0
L_{12} (Lights 1 & 2)	0	0	1	1	1	0	1	0
L_{34} (Lights 3 & 4)	0	0	0	1	1	0	0	0
L_5 (Light 5)	0	0	0	0	1	0	0	0
N_1 (new C_1)	0	0	1	1	0	0	0	0
N_0 (new C_0)	0	1	0	1	0	1	0	0