Finite state machines

Combinational circuit: implements Boolean function
Sequential circuit: implements finite state machine

- May also contain combinational circuit

In programming languages (i.e., 330), DFA (deterministic finite automaton)

- Essentially the same, but different purpose

DFA:

- Q, a set of states
- S, a single state which is an element of Q. This is the start state.
- F, a set of states designated as the final states
- Sigma, the input alphabet
- delta, a transition function that maps a state and a letter from the input alphabet,
  - to the next state

DFA is used to recognize a language L, which is composed of a set of strings

- made up from an input alphabet
- If DFA can recognize strings in the language, then L has a regular grammar

To use DFA:

- Start in initial state S
- Process each character in the input string, moving from state to state
- If DFA in a final state after processing the last character, string in language

Example: alphabet a, b

- Is string "aabb" recognized by a DFA?

Finite state machines

Finite state machine (FSM) with output

- Q, a set of states
- S, a single state which is an element of Q. This is the start state.
- Sigma, the input alphabet
- Pi, the output alphabet
- delta, a transition function that maps a state and a letter from the input alphabet,
  - to a state and a letter from the output alphabet

Primary differences with FSA:

- No final state
- Transition function generates output as well as determining next state

Purpose is not to recognize strings, but to generate set of outputs

- Describes how inputs and current state generate outputs

For circuits:

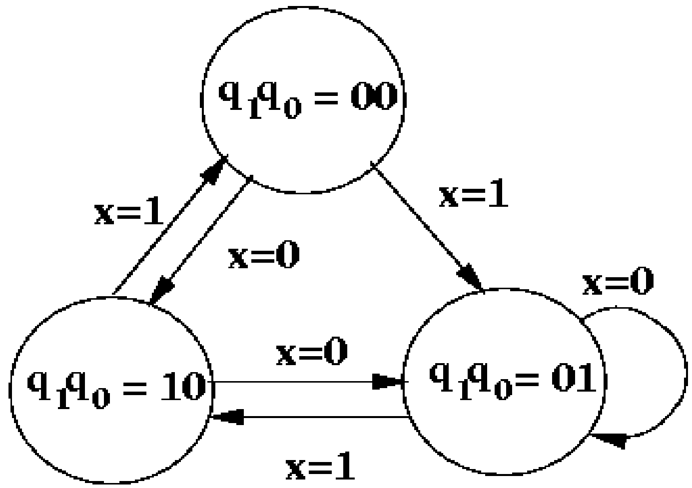
- Input alphabet: set of k-bit strings
- Output alphabet: set of m-bit strings

Transitions from a given state must be:

- Mutually exclusive: only 1 choice for any single input value
- Exhaustive: all possible inputs have a transition
  - "Nothing happens": remain in same state

Finite state machines

Example: **Start state**



States: represented by circles, 2-bit values q_1q_0

N states require ceil(lg N) bits to represent (the ceiling of log base 2 of N)

Inputs: represented by arrows labeled x (number of bits depends on number of transitions)

2^k arrows for k bits of input

Trace:	**State**	`00 (Start)`	`01`	`10`	`01`	`01`	`10`
	Input	`1`	`1`	`0`	`0`	`1`	

Finite state machines: Moore

So far: inputs tell which state to go to next, but no outputs

Moore machines: have outputs for each state

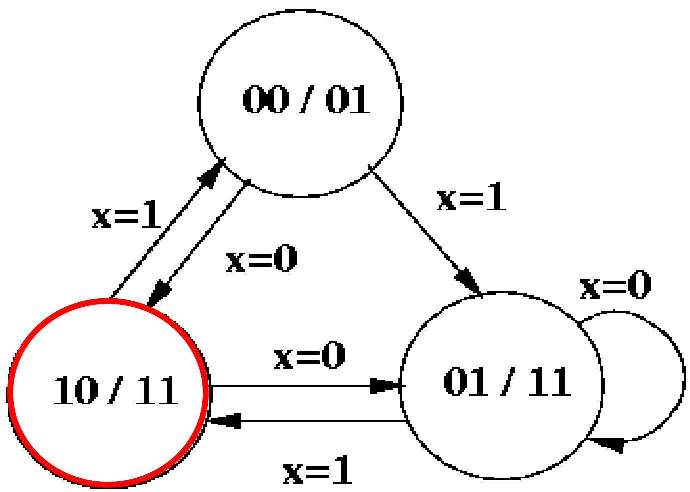
Output for a state is written following the state itself:

`01/1` **state 01, output 1** **`q1q0 = 01, z = 1`**

number of output bits depends on application

(may be more or less than for state representation)

Example:



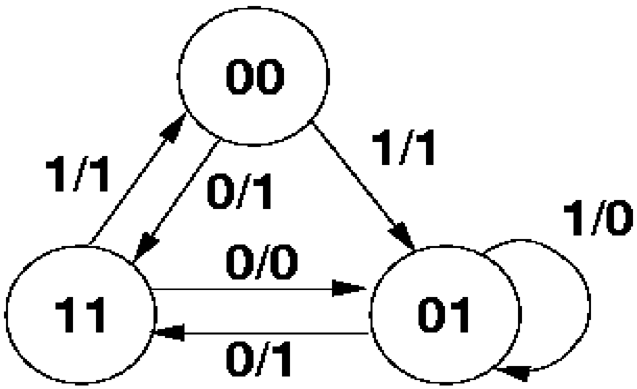
Input	**x**		**1**	**1**	**0**	**1**	**1**	**0**
State	**q1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**
	q0	**0**	**1**	**0**	**1**	**0**	**0**	**0**
Output	**z1**	**0**	**1**	**1**	**1**	**1**	**0**	**1**
	z0	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Finite state machines: Mealy

Moore machines: output is function of state
Mealy machines: output is function of state and input

output is shown on edge

Example:



In state 00, input of 1 produces output of 1: `1/1` `x = 1, z = 1`

Notice numbering of states: can select any combination of 2 bits

Input	x		1	1	0	1	1	0
State	q1	0	0	0	1	0	0	1
	q0	0	1	1	1	0	1	1
Output	z	0	1	0	1	1	1	1