Memory

- With combinatorial logic, we could only implement “stateless” functions
- By introducing flip-flops, we could remember something about the history of the inputs

Memory

- With combinatorial logic, we could only implement “stateless” functions

- By introducing sequential logic (with flip-flops), we could remember something about the history of the inputs

How do we formalize this idea of “history”?

Formalizing Memory

Combinatorial Logic

Boolean Algebra

Formalizing Memory

Combinatorial Logic   Boolean Algebra

Sequential Logic

Formalizing Memory

Combinatorial Logic	Boolean Algebra
Sequential Logic	Finite State Machines

Formalizing Memory

Combinatorial Logic	Boolean Algebra
Sequential Logic	Finite State Machines

This will allow us to express controllers that take history into account ….

Finite State Machines (FSMs)

Pure FSM form is composed of:

- A set of states
- A set of possible inputs (or events)
- A set of possible outputs
- A transition function:
  - Given the current state and an input: defines the output and the next state

Finite State Machines (FSMs)

States:

- Represent all possible “situations” that must be distinguished
- At any given time, the system is in exactly one of the states
- There is a finite number of these states

Finite State Machines (FSMs)

An example: our synchronous counter

- States: ?

Finite State Machines (FSMs)

An example: our synchronous counter

- States: the different combinations of the digits: 000, 001, 010, … 111
- Inputs: ?

Finite State Machines (FSMs)

An example: our synchronous counter

- Inputs:
  - Really only one: the event associated with the clock transitioning from high to low
  - We will call this “C”

- Outputs: ?

Finite State Machines (FSMs)

An example: our synchronous counter

- Outputs: same as the set of states
- Transition function: ?

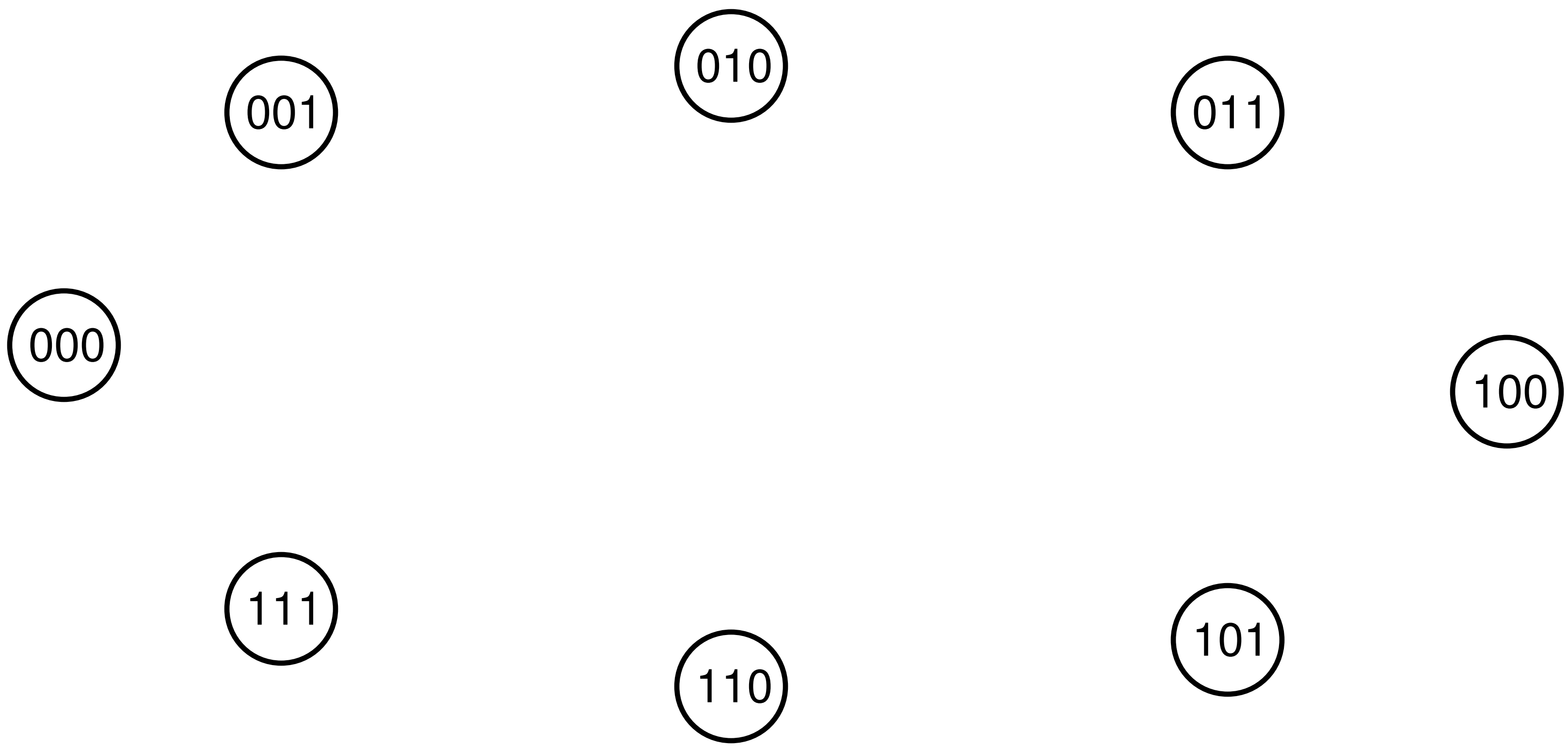
Finite State Machines (FSMs)

An example: our synchronous counter

- Transition function:
  - On the clock event, transition to the next state in the sequence

FSM Example:
Synchronous Counter

A Graphical Representation:

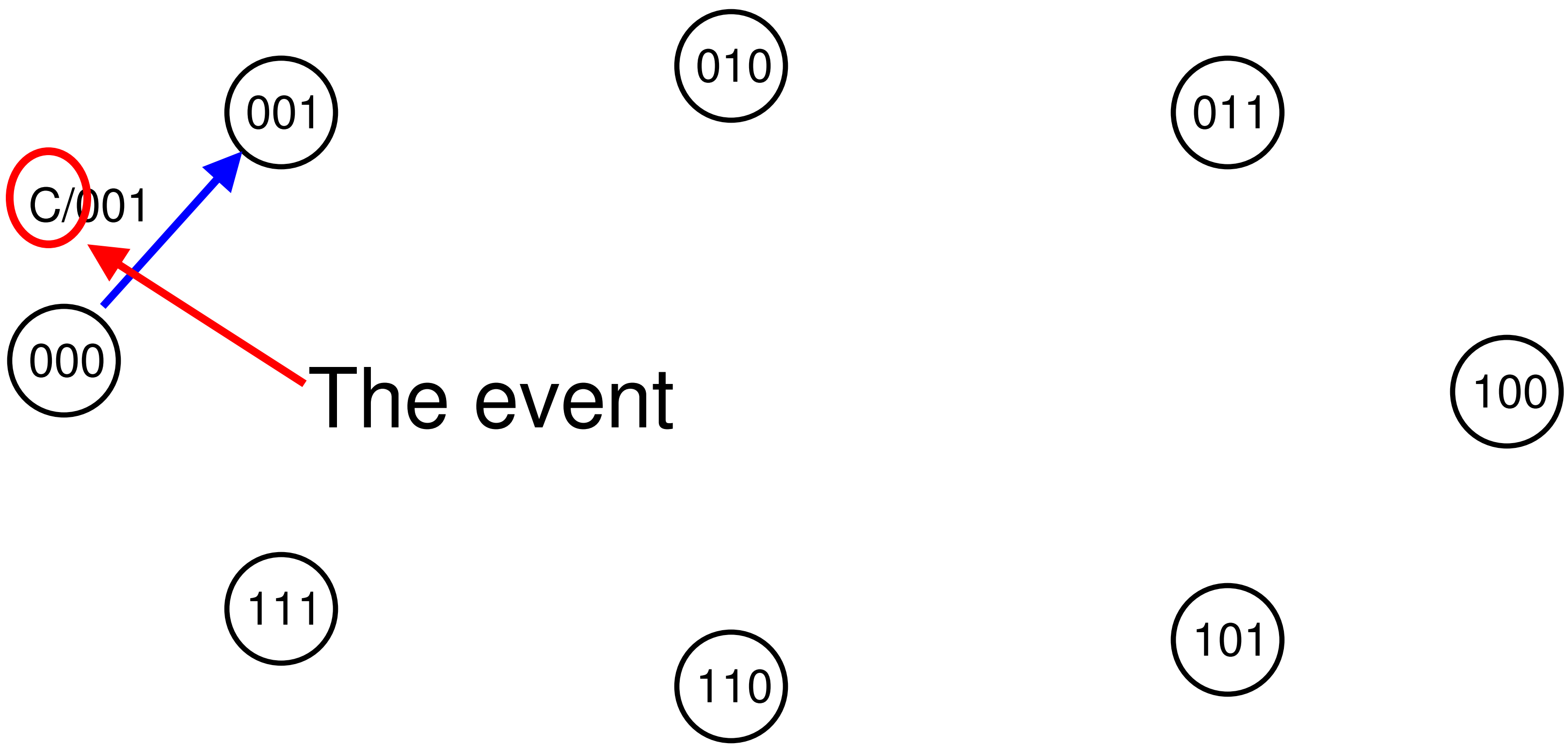


A set of states

FSM Example: Synchronous Counter

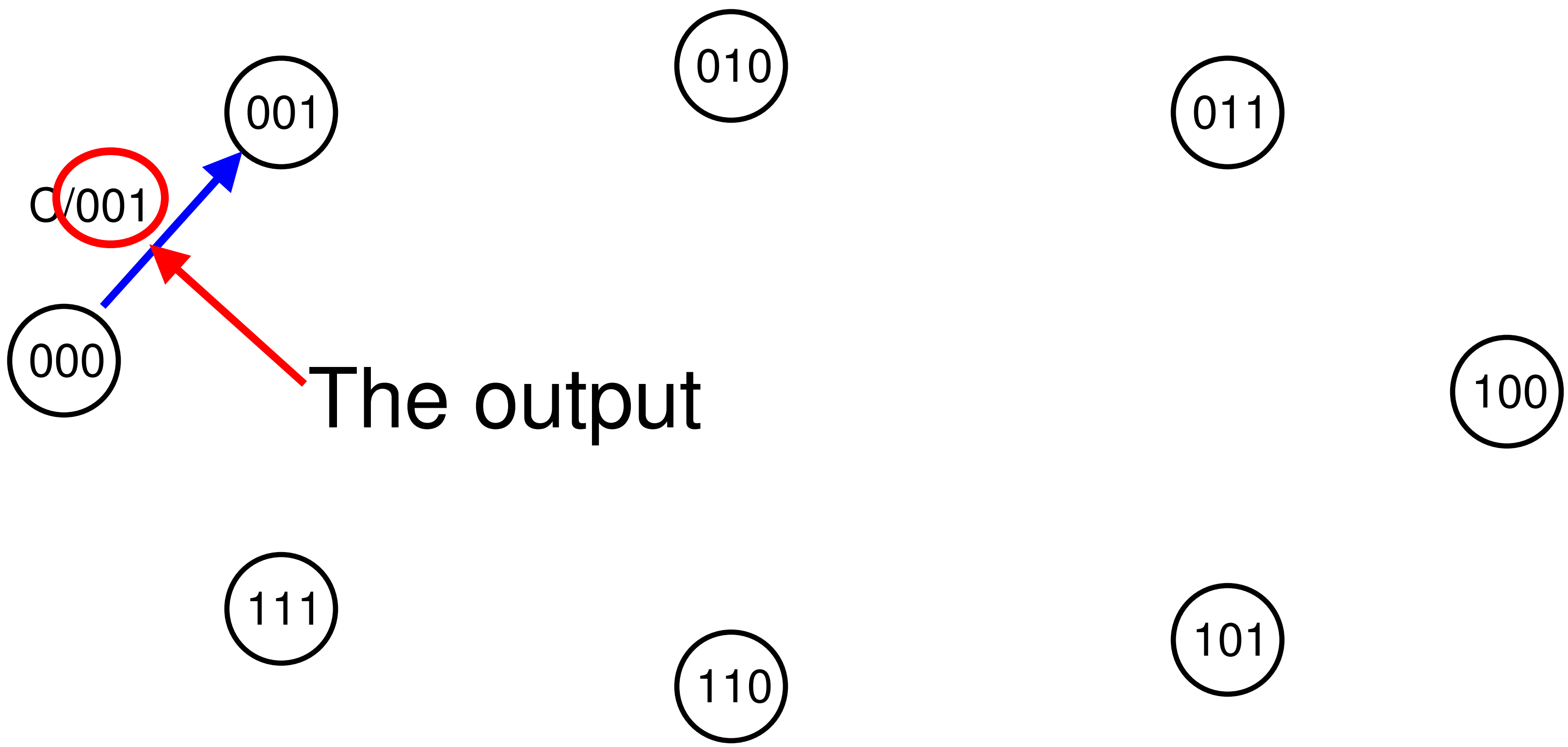
FSM Example: Synchronous Counter

A transition



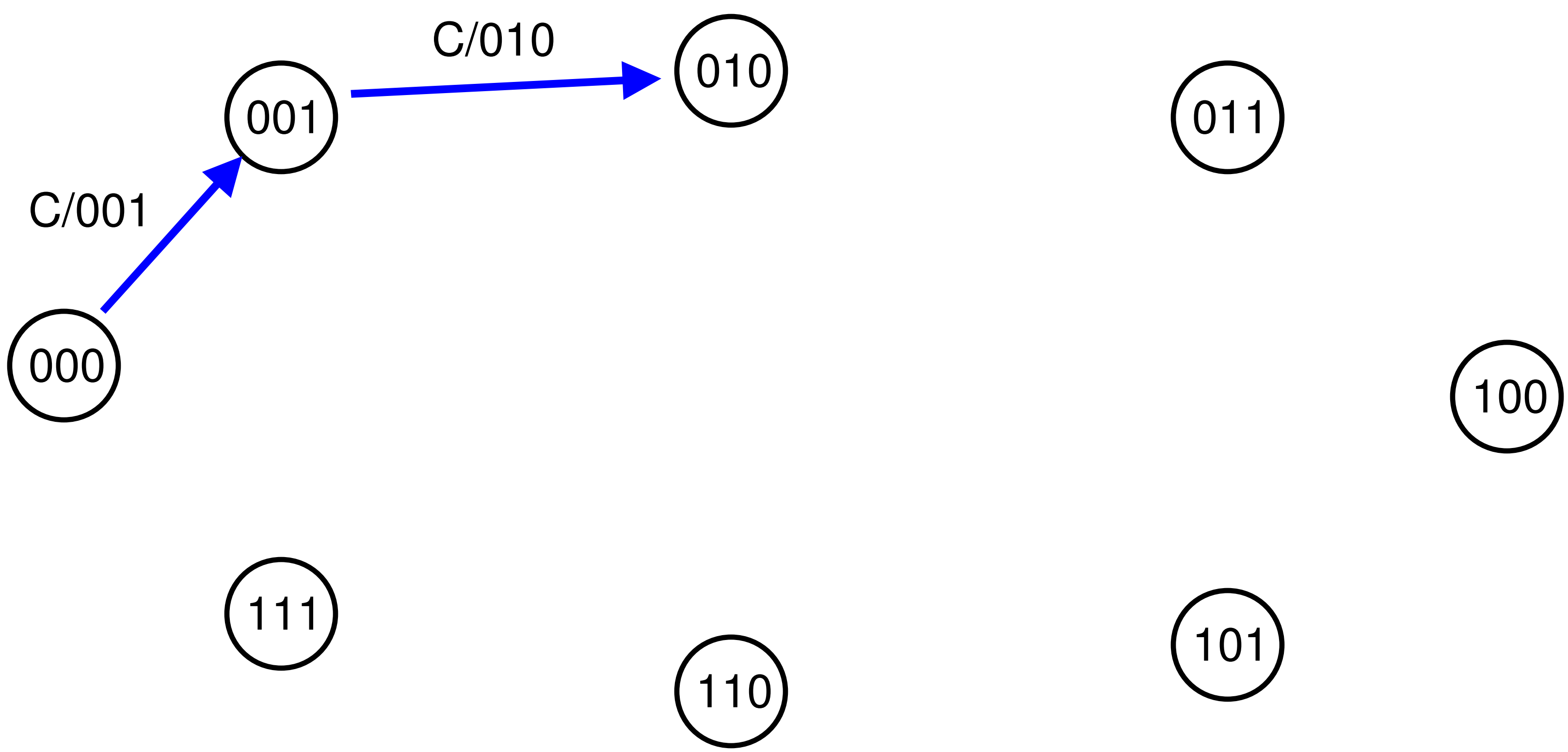
FSM Example: Synchronous Counter

A transition



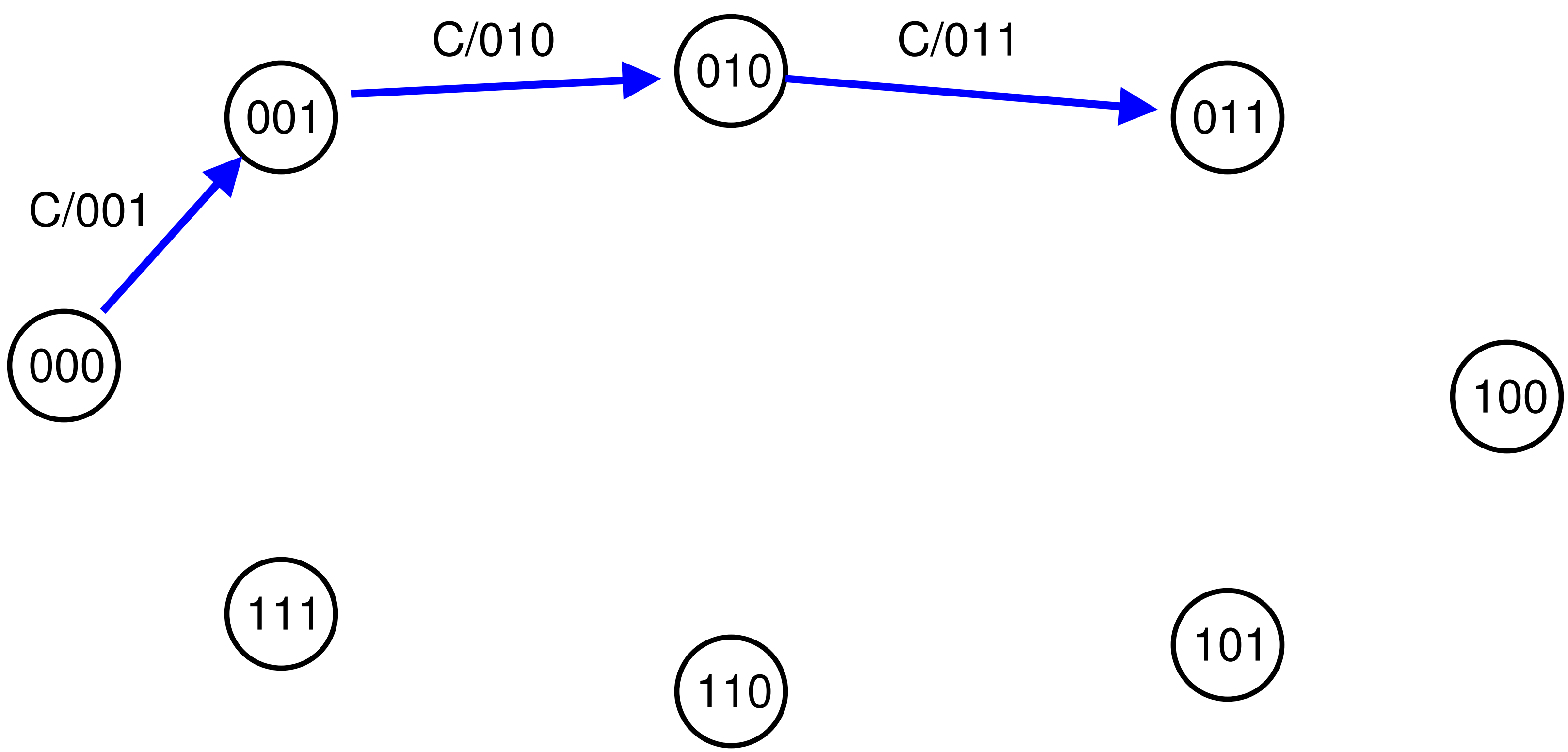
FSM Example: Synchronous Counter

The next transition



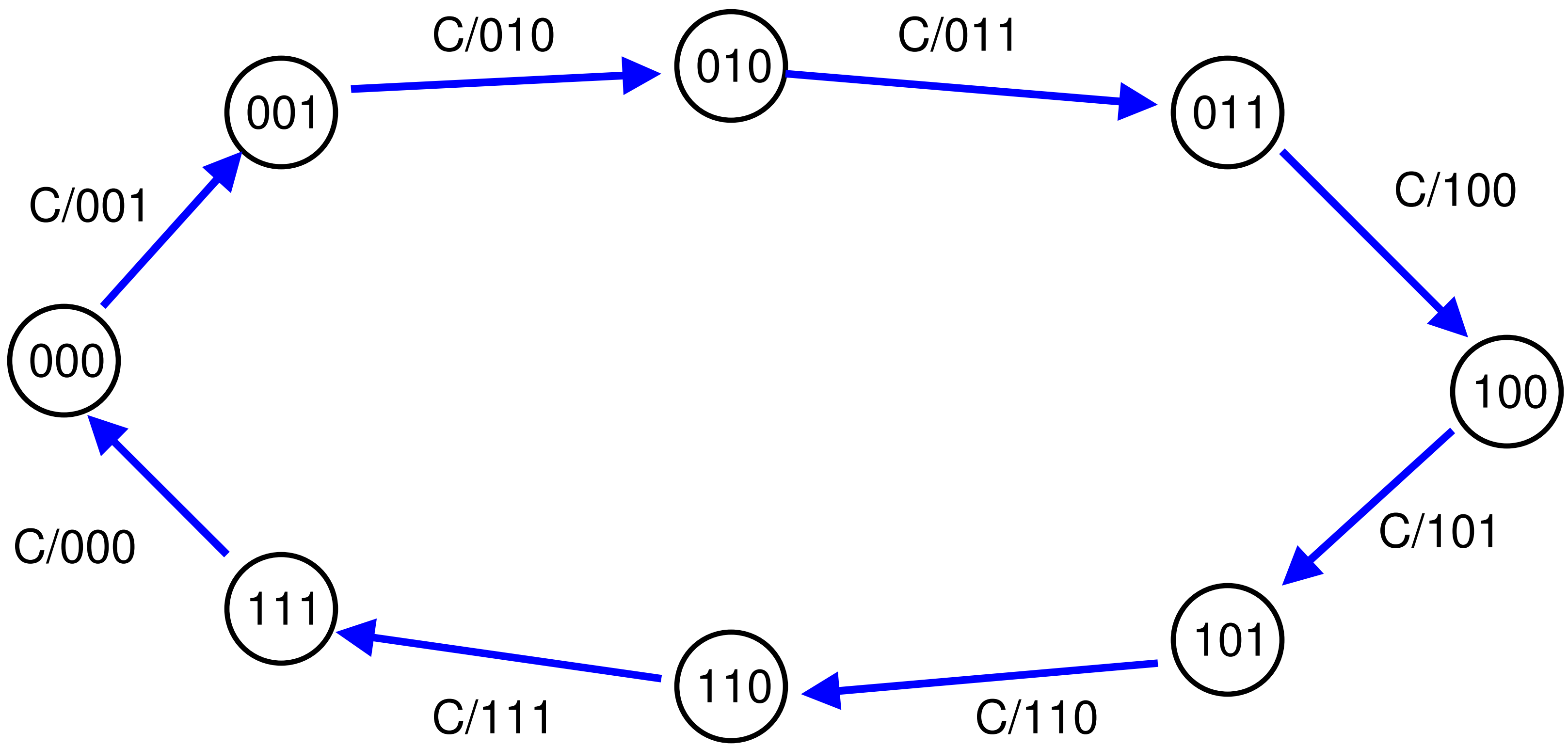
FSM Example: Synchronous Counter

The next transition

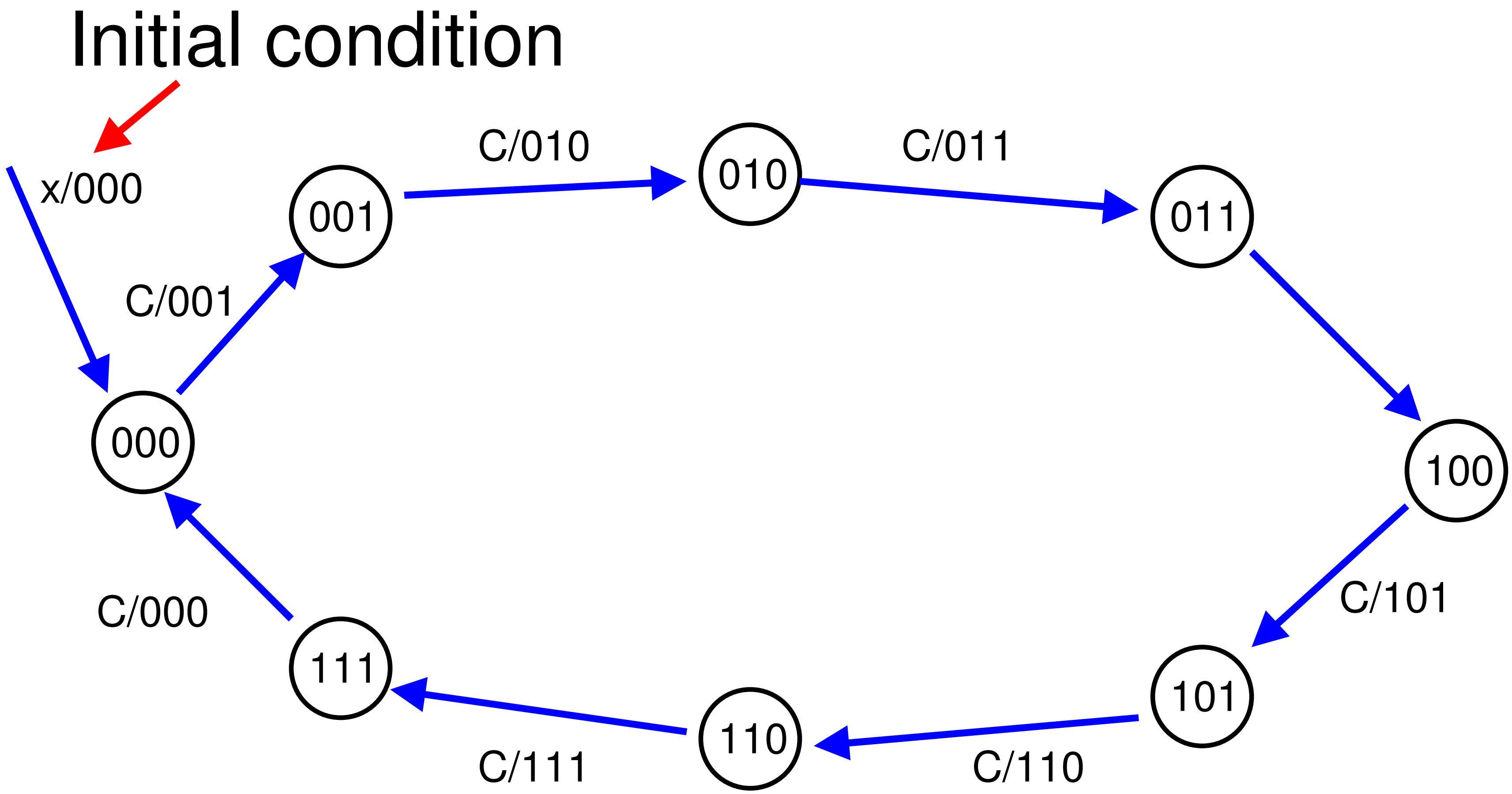


FSM Example: Synchronous Counter

The full transition set



FSM Example: Synchronous Counter



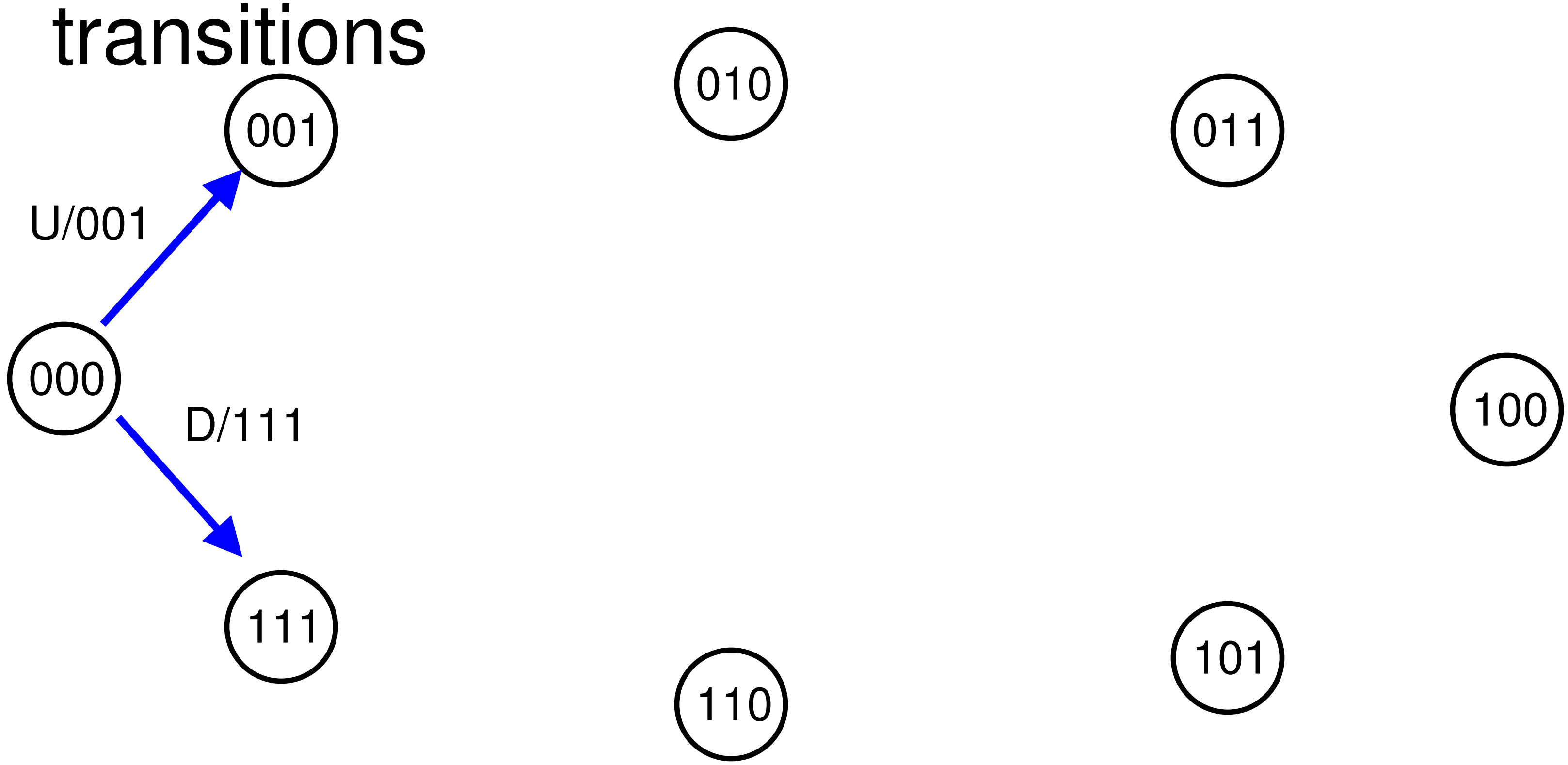
Example II: An Up/Down Counter

Suppose we have two events (instead of one): Up and down

- How does this change our state transition diagram?

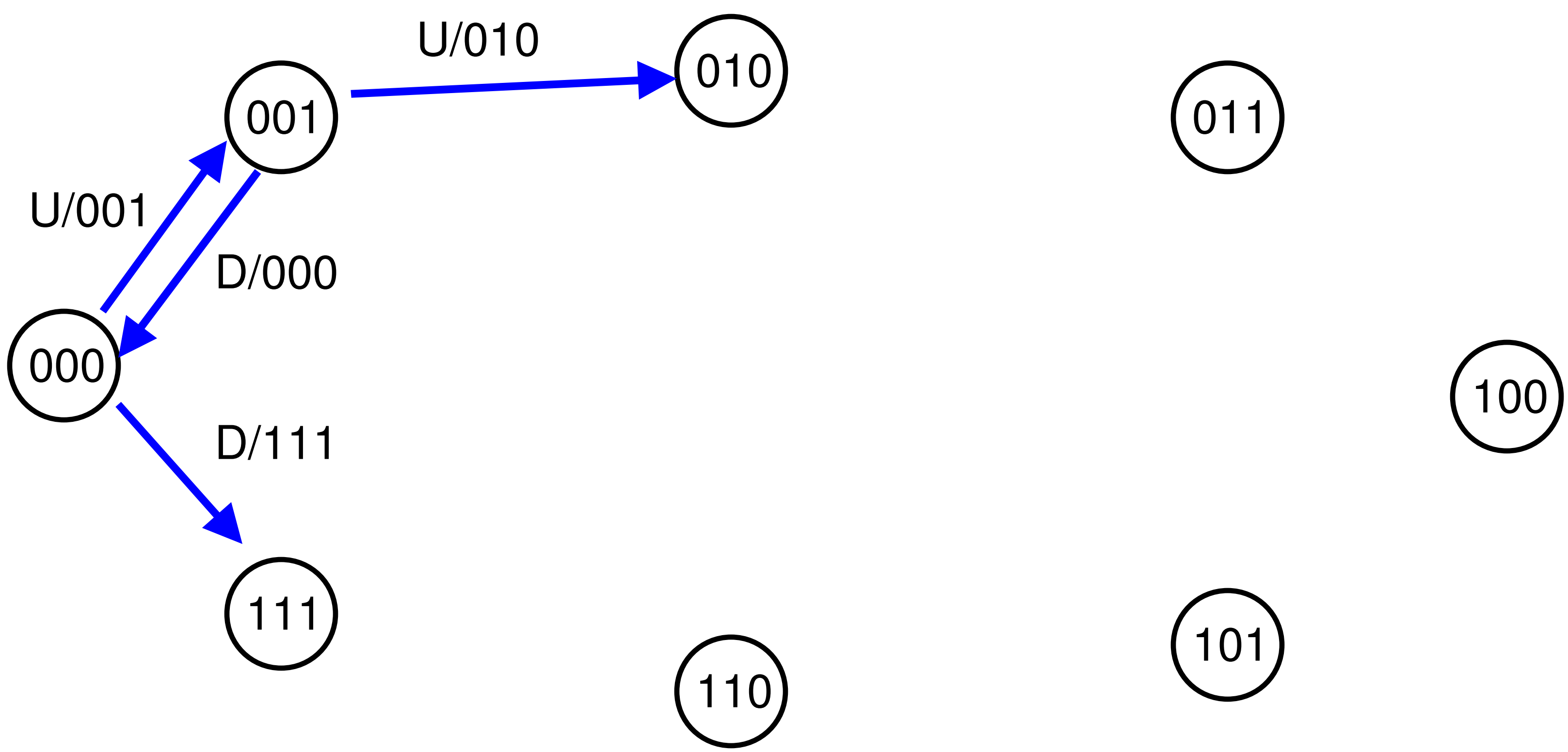
Example II: An Up/Down Counter

From state 000, there are now two possible transitions



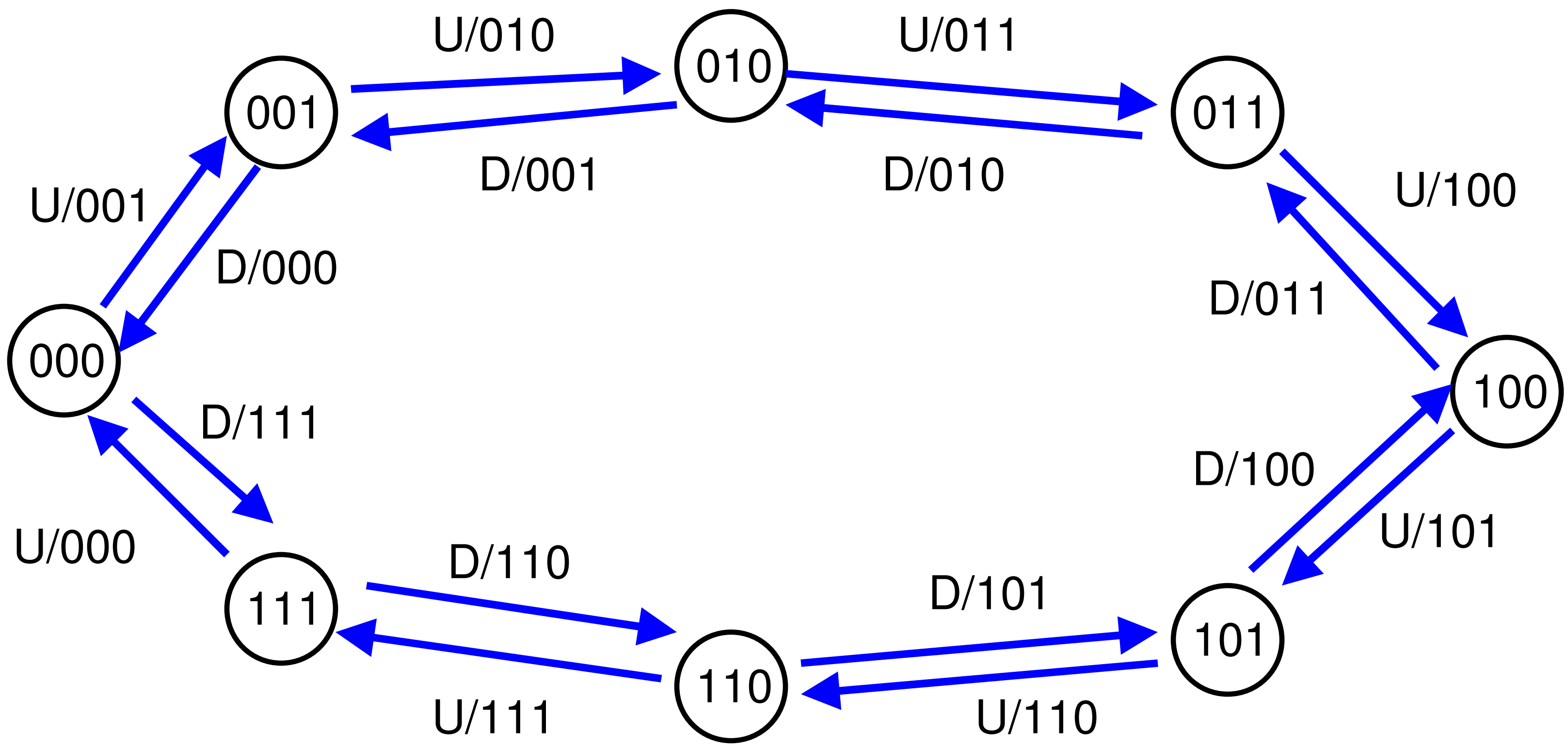
Example II: An Up/Down Counter

Likewise for state 001…



Example II: An Up/Down Counter

The full transition set



FSMs and Control

How do we relate FSMs to Control?

- States are ?

FSMs and Control

How do we relate FSMs to Control?

- States are our memory of recent inputs

- Inputs are ?

FSMs and Control

How do we relate FSMs to Control?

- States are our memory of recent inputs

- Inputs are some processed representation of what the sensors are observing

- Outputs are ?

FSMs and Control

How do we relate FSMs to Control?

- States are our memory of recent inputs

- Inputs are some processed representation of what the sensors are observing

- Outputs are the control actions

Project 2: The Problem

Project 1:

- Implementation of a feedback control circuit (in digital logic) that orients and then moves toward a beacon

Project 2:

- Integrate this capability into a sequence of movements

Project 2: The Problem

Primary behavior of the robot:

- Phase 1:
  - Move toward beacon in front of the robot
  - Scan for another beacon on the left
  - When beacon is found, turn toward it
- Phase 2:
  - Move toward beacon in front
  - Scan for another beacon on the right
  - When beacon is found, turn toward it
- Repeat

Project 2: The Problem

An exception occurs if the robot loses sight of the forward beacon (no signal on either the left or the right sensor pair)

If in phase 1:

- Rotate turret to the right
- If a beacon is found, then turn the robot toward it and continue with phase 1
- Else stop moving

Project 2: The Problem

Exception handling

If in phase 2:

- Rotate turret to the left
- If a beacon is found, then turn the robot toward it and continue with phase 2
- Else stop moving

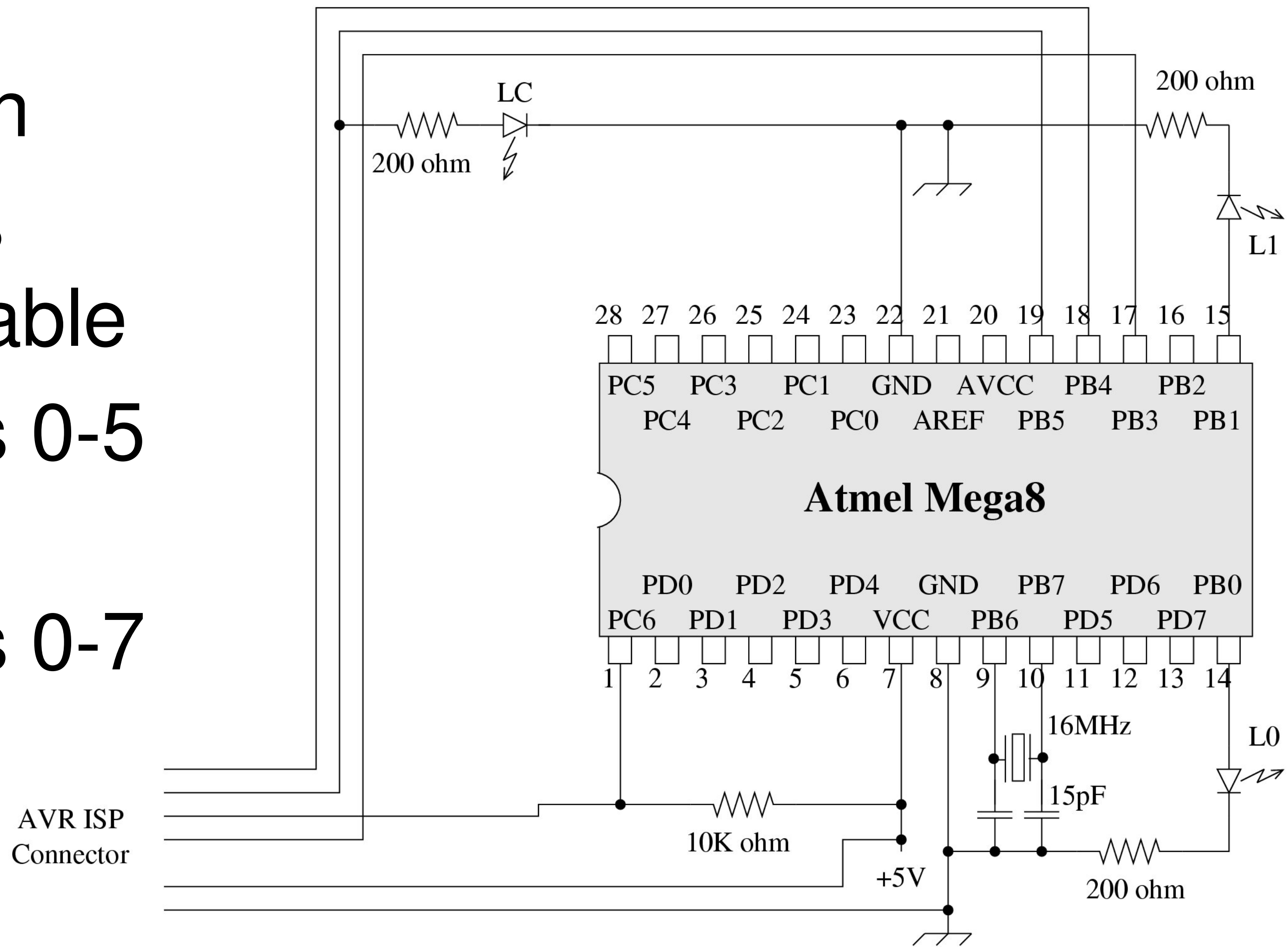
Project 2: Step -1

Low-level control with the Atmel

Project 2: Step 0

Circuit design

- PortB: pins 0,1,2 available
- PortC: pins 0-5 available
- PortD: pins 0-7 available



Project 2: Step 1

Design the FSM for this problem

- What are the states?
- What are the sensory signals?
- What are the inputs?
- What are the outputs?

Project 2: Step 2

Design the FSM for this problem

- What is the mapping from sensory signals to events?

Project 2: Step 3

Design the FSM for this problem

- What does the transition function look like?

Project 2: Step 4

Design the FSM for this problem

- What is the mapping from output to robot action?
- What must the robot do if no event occurs?

Project 2: Step 5

Implementation

- Write a C program that implements your FSM
- Burn this onto an Atmel mega8 processor
- Get it to work!

Next Time

- Homework 4 discussion
- Midterm preparation
- Another FSM control example

Implementing Finite State Machines

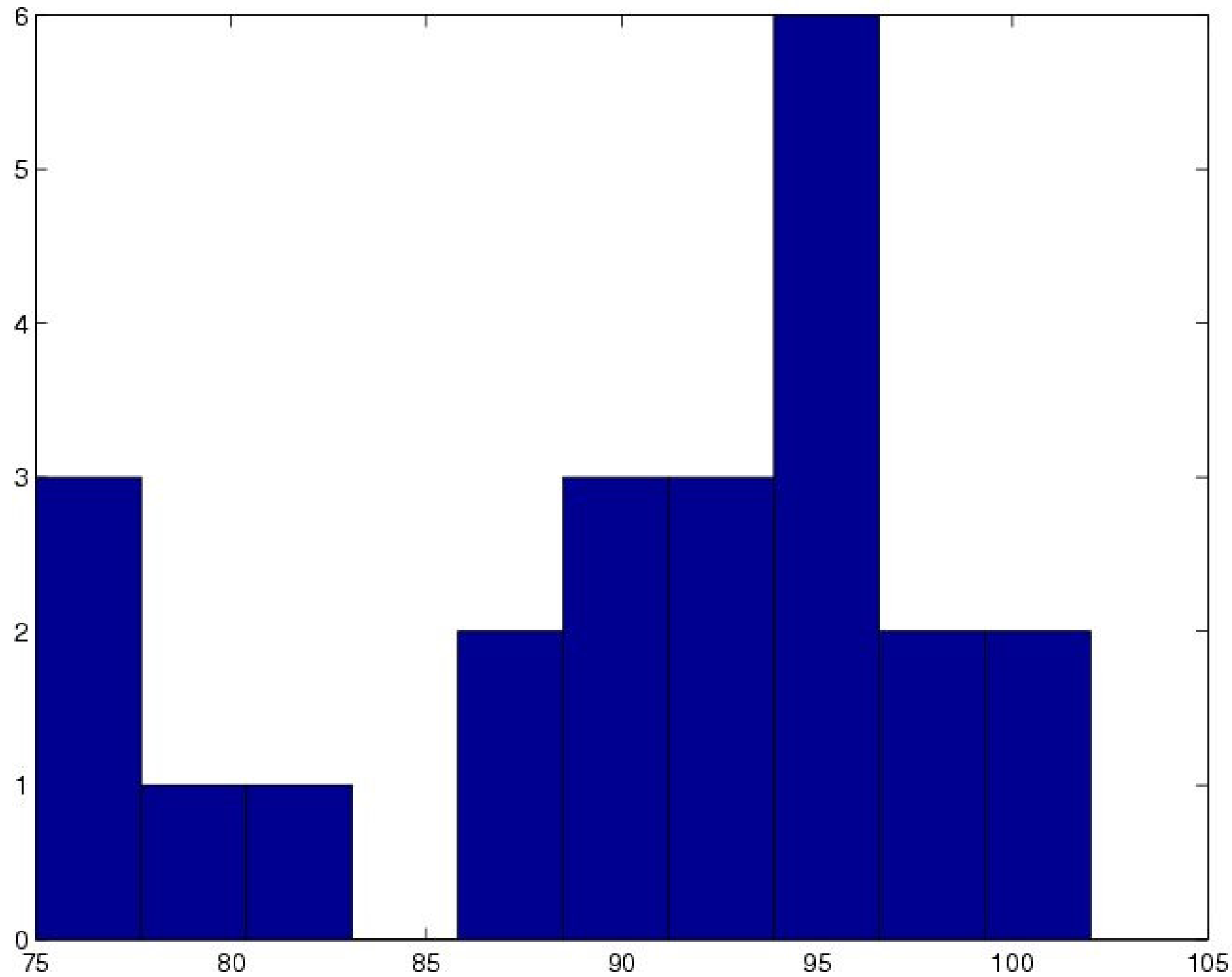
How would we implement an FSM with the logic components we have studied so far?

Today

- Midterm exam
- Lab 2 (part 1 due Thursday)
  – Demonstration & code review
  – Hand in code via D2L
- Finite State Machines
  – Control example
  – Coding

Midterm

- Mean: 90.2
- Standard deviation: 8.0



Lab 2

- You may change the prototype for one required function, e.g.:

```
uint8_t orient_new_beacon(uint8_t sensor[4], unit8_t direction)
```

- Demonstration: make sure that you show the functionality of all 5 of your required functions

FSMs: A Control Example

Suppose we have a vending machine:

- Accepts dimes and nickels
- Will dispense one of two things once $.20 has been entered: Jolt or Buzz Water
  - The “user” requests one of these by pressing a button
- Ignores select if < $.20 has been entered
- Immediately returns any coins above $.20

Vending Machine FSM

What are the states?

Vending Machine FSM

What are the states?

- $0
- $.05
- $.10
- $.15
- $.20

Vending Machine FSM

What are the inputs/events?

Vending Machine FSM

What are the inputs/events?

- Input nickel (N)
- Input dime (D)
- Select Jolt (J)
- Select Buzz Water (BW)

Vending Machine FSM

What are the outputs?

Vending Machine FSM

What are the outputs?

- Return nickel (RN)
- Return dime (RD)
- Dispense Jolt (DJ)
- Dispense Buzz Water (DBW)
- Nothing (Z)

Vending Machine Design

What is the initial state?

Vending Machine Design

What is the initial state?

- S = $0

Vending Machine Design

What can happen from S = $0?

Event	Next State	Output

Vending Machine Design

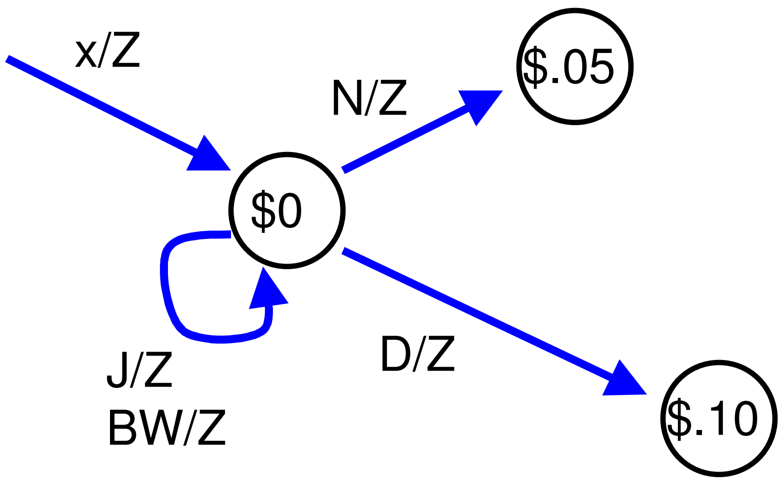
What can happen from S = $0?

What does this part of the diagram look like?

Event	Next State	Output
N	$.05	Z
D	$.10	Z
J	$0	Z
BW	$0	Z

Vending Machine Design

A piece of the state diagram:



Vending Machine Design

What can happen from S = $0.05?

Event	Next State	Output

Vending Machine Design

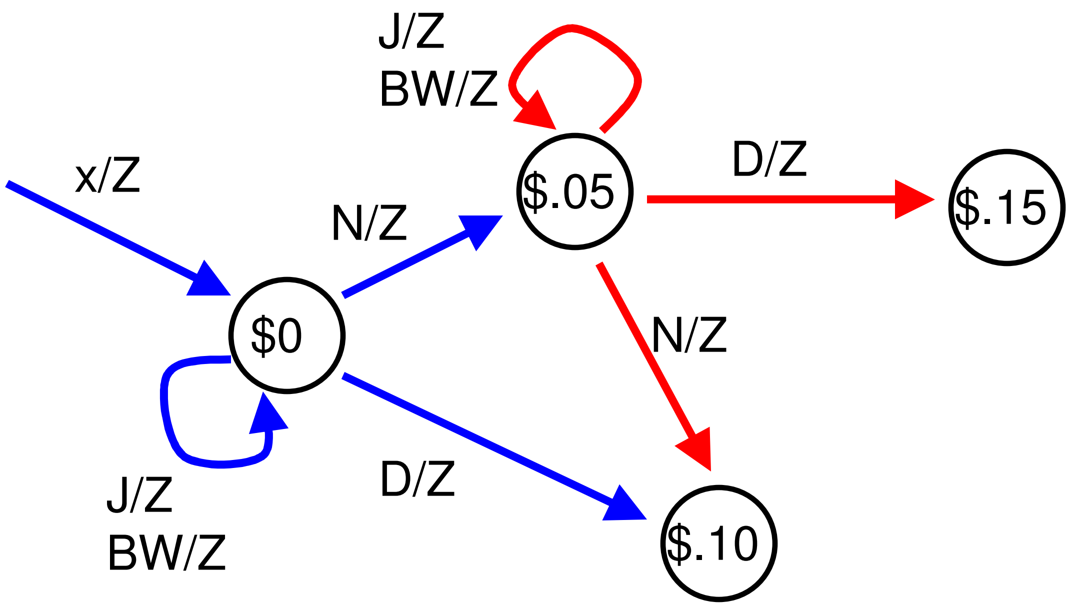
What can happen from S = $0.05?

What does the modified diagram look like?

Event	Next State	Output
N	$.10	Z
D	$.15	Z
J	$.05	Z
BW	$.05	Z

Vending Machine Design

A piece of the state diagram:



Vending Machine Design

What can happen from S = $0.10?

Event	Next State	Output

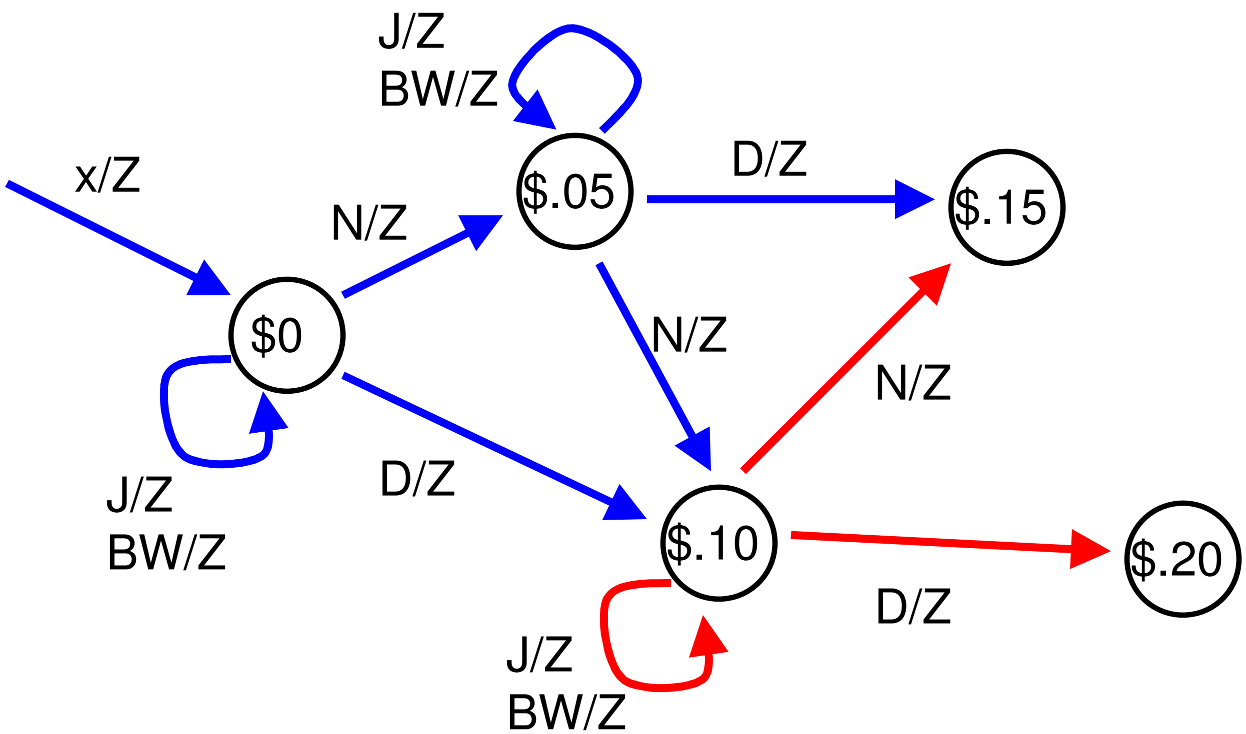
Vending Machine Design

What can happen from S = $0.10?

Event	Next State	Output
N	$.15	Z
D	$.20	Z
J	$.10	Z
BW	$.10	Z

Vending Machine Design

A piece of the state diagram:



Vending Machine Design

What can happen from S = $0.15?

Event	Next State	Output

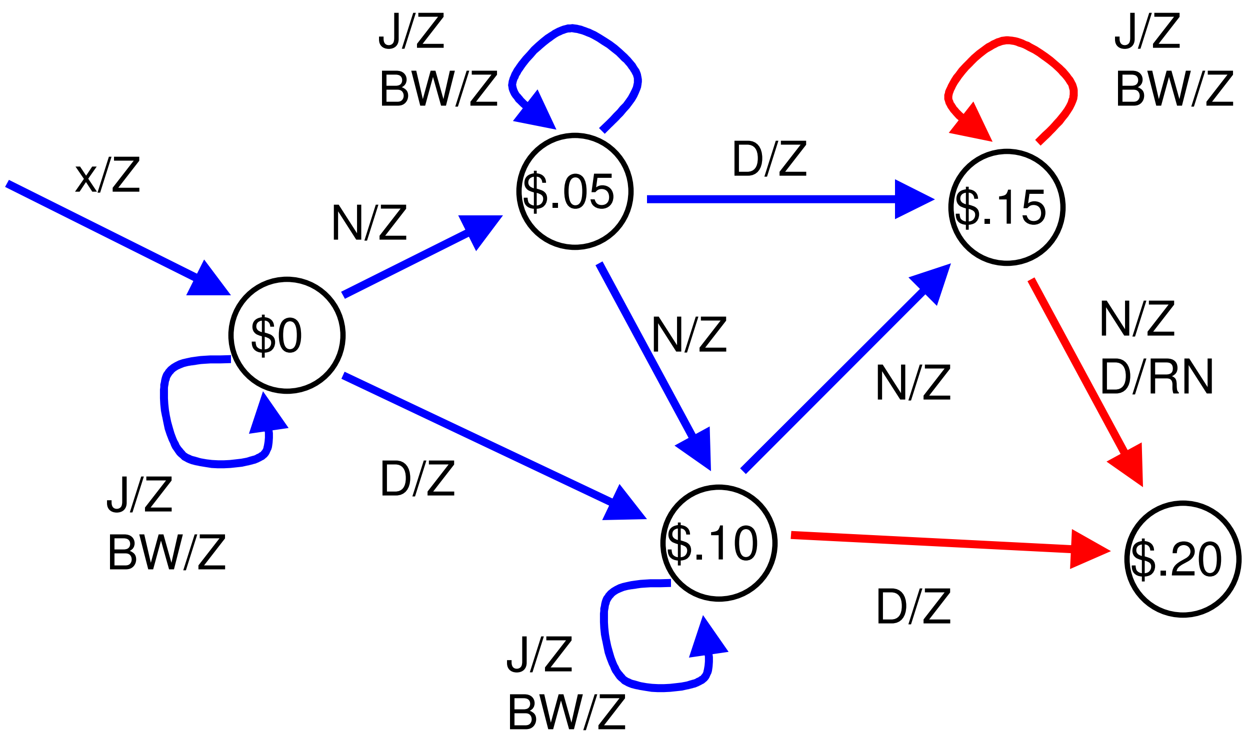
Vending Machine Design

What can happen from S = $0.15?

Event	Next State	Output
N	$.20	Z
D	$.20	RN
J	$.15	Z
BW	$.15	Z

Vending Machine Design

A piece of the state diagram:



Vending Machine Design

Finally: what can happen from S = $0.20?

Event	Next State	Output

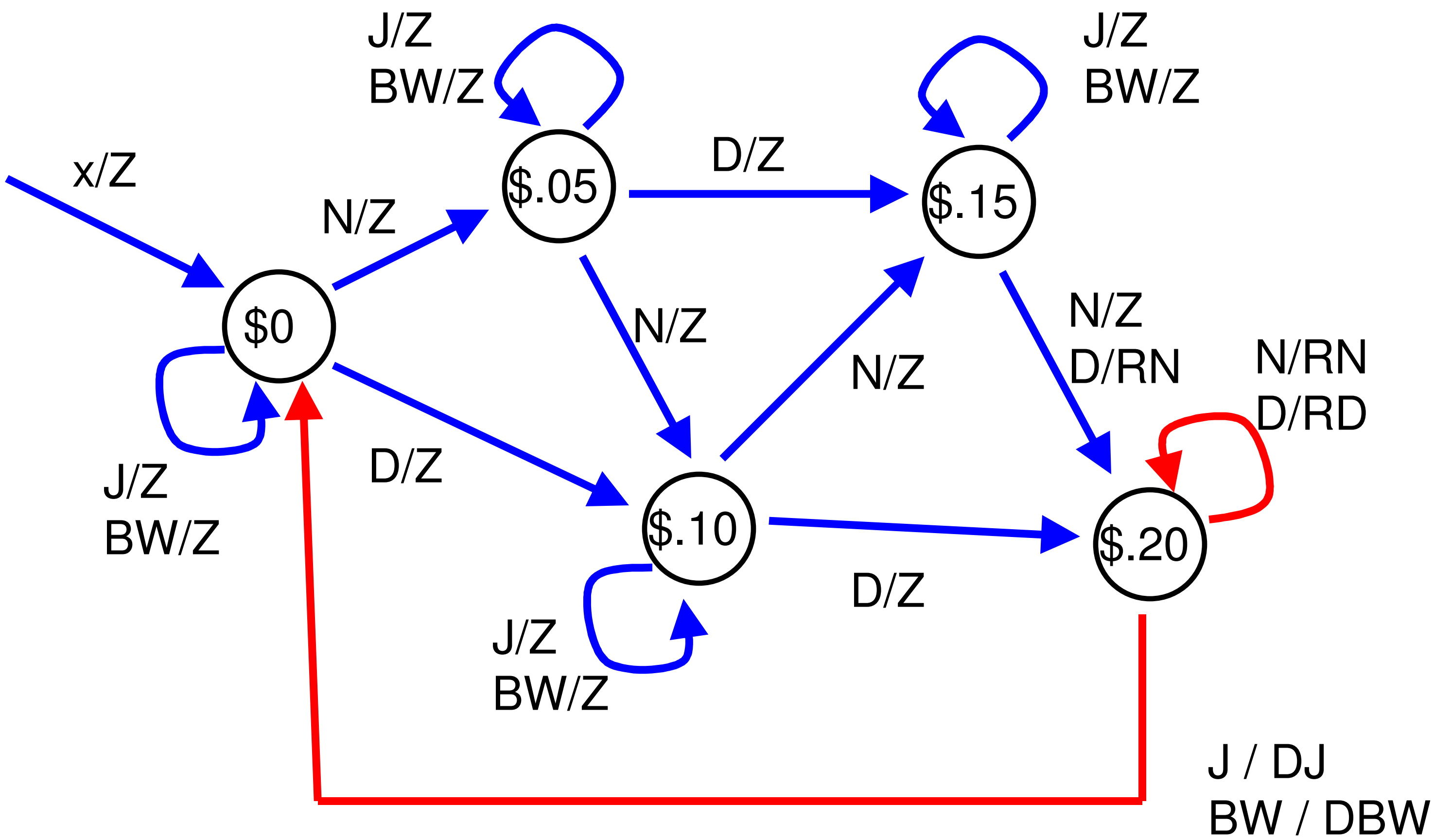
Vending Machine Design

Finally, what can happen from S = $0.20?

Event	Next State	Output
N	$.20	RN
D	$.20	RD
J	$0	DJ
BW	$0	DBW

Vending Machine Design

The complete state diagram:



A Robot Control Example

Consider the following task:

- The robot is to move toward the first beacon that it "sees"
- The robot searches for a beacon in the following order: right, left, front

What is the FSM representation?

FSMs in C

```
int state = 0;   // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
  }
}
```

FSMs in C

```
int state = 0;    // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
  }
}
```

Variable declaration and initialization

FSMs in C

```
int state = 0;   // Initial state
while(1) {
   <do some processing of the sensory inputs>
   switch(state) {
        case 0:
                <handle state 0>
                break;
        case 1:
                <handle state 1>
                break;
        case 2: …
   }
}
```

A comment (use liberally)

FSMs in C

```
int state = 0;    // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
  }
}
```

Loop forever

FSMs in C

```
int state = 0;    // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
  }
}
```

"pseudo code": not really code, but indicates what is to be done

FSMs in C

```
int state = 0;    // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
  }
}
```

In this case: we will translate the current sensory inputs into a representation of an event (if one has happened)

FSMs in C

```
int state = 0;    // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
  }
}
```

Switch/case syntax allows us to cleanly perform many "if(x==y)" operations

FSMs in C

```
int state = 0;    // Initial state
while(1) {
   <do some processing of the sensory inputs>
   switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
   }
}
```

If state==0, then execute the following code

FSMs in C

```
int state = 0;   // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
  }
}
```

This code can be as complex as necessary

FSMs in C

```
int state = 0;    // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
  }
}
```

break says to exit the switch (don't forget it or strange things will happen!)

FSMs in C

```
int state = 0;    // Initial state
while(1) {
   <do some processing of the sensory inputs>
   switch(state) {
        case 0:
               <handle state 0>
               break;
        case 1:
               <handle state 1>
               break;
        case 2: …
   }
}
```

If state==1, then …

FSMs in C

```
int state = 0;    // Initial state
while(1) {
   <do some processing of the sensory inputs>
   switch(state) {
       case 0:
              <handle state 0>
              break;
       case 1:
              <handle state 1>
              break;
       case 2: …
   }
}
```

End of the **switch** block

FSMs in C

```
int state = 0;    // Initial state
while(1) {
   <do some processing of the sensory inputs>
   switch(state) {
       case 0:
             <handle state 0>
             break;
       case 1:
             <handle state 1>
             break;
       case 2: …
   }
}
```

End of the while block

Last Time

- Finite State Machines for control
- FSM implementations in C

Today

- More on FSM implementation
- Assembly language

Administrivia

- Project 2, part 1 due TODAY
- Project 2, part 2 due in one week
- Homework 5 is on hold

Finite State Machines

- Very useful tool to describe sequential behavior.
- But – when used for control, we deviate from the theory in several key ways

FSMs As Controllers

- Need code that translates sensory inputs into FSM events
- An FSM output can require an arbitrary amount of time
  - We will often implement this control action as a separate function call
- Control actions will not necessarily be fixed (but could be a function of sensory input)

FSMs As Controllers (cont)

- We might choose to leave some events out of the implementation
  - Only some events may be relevant to certain states
- When in a state, the FSM may also issue control actions (even when a new event has not arrived)
  - Again, this may be implemented as a function call

FSMs in C

```
int state = 0;   // Initial state
while(1) {
  <do some processing of the sensory inputs>
  switch(state) {
      case 0:
            <handle state 0>
            break;
      case 1:
            <handle state 1>
            break;
      case 2: …
  }
}
```

FSMs in C (some other possibilities)

```
int state = 0;   // Initial state
while(1) {
   <do some processing of the sensory inputs>
   switch(state) {
       case 0:
             <handle state 0>
             break;
          :
       default:
             <handle default case>
             break;
   }
   <do some low-level control>
}
```

FSMs in C (some other possibilities)

```
int state = 0;    // Initial state
while(1) {
   <do some processing of the sensory inputs>
   switch(state) {
       case 0:
             <handle state 0>
             break;
          :
       default:
             <handle default case>
             break;
   }
   <do some low-level control>
}
```

Matches any state (if we reach this point)

FSMs in C (some other possibilities)

```
int state = 0;    // Initial state
while(1) {
   <do some processing of the sensory inputs>
   switch(state) {
       case 0:
             <handle state 0>
             break;
          :
       default:
             <handle default case>
             break;
   }
   <do some low-level control>
}
```

(possibly) alter some control outputs (e.g., steering direction)

FSMs in C: Processing for Individual States

```
case STATE_10cents:
   // $.10 has already been deposited
   switch(event) {
        case EVENT_NICKEL:   // Nickel
                state = STATE_15cents;  // Transition to $.15
                break;
        case EVENT_DIME:   // Dime
                state = STATE_20cents;  // Transition to $.2
                break;
        case EVENT_JOLT:   // Select Jolt
        case EVENT_BUZZ:   // Select Buzzwater
                display_NOT_ENOUGH();
                break;

        case EVENT_NONE:   // No event
                break;            // Do nothing

   };
   break;
```

FSMs in C: Processing for Individual States

```
case STATE_10cents:
   // $.10 has already been deposited
   switch(event) {
        case EVENT_NICKEL:   // Nickel
                state = STATE_15cents;  // Transition to $.15
                break;
        case EVENT_DIME:   // Dime
                state = STATE_20cents;  // Transition to $.2
                break;
        case EVENT_JOLT:   // Select Jolt
        case EVENT_BUZZ:   // Select Buzzwater
                display_NOT_ENOUGH();
                break;

        case EVENT_NONE:   // No event
                break;          // Do nothing

   };
   break;
```

Another integer

FSMs in C: Processing for Individual States

```
case STATE_10cents:
   // $.10 has already been deposited
   switch(event) {
        case EVENT_NICKEL:   // Nickel
                state = STATE_15cents;   // Transition to $.15
                break;
        case EVENT_DIME:   // Dime
                state = STATE_20cents;   // Transition to $.2
                break;
        case EVENT_JOLT:   // Select Jolt
        case EVENT_BUZZ:   // Select Buzzwater
                display_NOT_ENOUGH();
                break;

        case EVENT_NONE:   // No event
                break;           // Do nothing

   };
   break;
```

A nickel has been received

FSMs in C: Processing for Individual States

```
case STATE_10cents:
   // $.10 has already been deposited
   switch(event) {
        case EVENT_NICKEL:    // Nickel
                state = STATE_15cents;   // Transition to $.15
                break;
        case EVENT_DIME:    // Dime
                state = STATE_20cents;   // Transition to $.2
                break;
        case EVENT_JOLT:    // Select Jolt
        case EVENT_BUZZ:    // Select Buzzwater
                display_NOT_ENOUGH();
                break;

        case EVENT_NONE:    // No event
                break;            // Do nothing

   };
   break;
```

Change state for next iteration of the while() loop

FSMs in C: Processing for Individual States

```
case STATE_10cents:
   // $.10 has already been deposited
   switch(event) {
        case EVENT_NICKEL:   // Nickel
                state = STATE_15cents;  // Transition to $.15
                break;
        case EVENT_DIME:   // Dime
                state = STATE_20cents;  // Transition to $.2
                break;
        case EVENT_JOLT:   // Select Jolt
        case EVENT_BUZZ:   // Select Buzzwater
                display_NOT_ENOUGH();
                break;

        case EVENT_NONE:   // No event
                break;           // Do nothing

   };
   break;
```

If any of these match, then execute the following code (which does nothing in this example)

A Note on “Style” in C

- The numbers that we assigned to the different states are arbitrary (and at first glance, hard to interpret)
- Instead, we can define constant strings that have some meaning

- Replace: 0, 1, 2, 3, 4, 5
- With: STATE_00, STATE_05, STATE_10, STATE_15, STATE_20

A Note on “Style” in C

In C, this is done by adding some definitions to the beginning of your program (either in the .c file or the .h file):

```
#define STATE_00   0
#define STATE_05   1
#define STATE_10   2
#define STATE_15   3
#define STATE_20   4
```