CS 2630
Computer Organization

Meeting 16: Finite state machines, memories
Brandon Myers
University of Iowa
Ingredients for a processor

• Combinational logic
  • arithmetic and bitwise operations
  • multiplexors (pick 1 out of of N inputs)
  • control logic
  • branch calculation
  • address calculation

• Synchronous logic
  • registers
  • program counter, or PC
  • data memory
  • instruction memory
Peer instruction
State transition diagrams

components of a state transition diagram
• one node for each unique state
• label each state with output(s)
• label each edge with input(s)
• traverse an edge on rising edge of clock

state transition diagram for a flip-flop
Peer instruction

• What is the state transition diagram for the following circuit?
Finite state machine (FSM)

- All FSMs look like this

**Diagram:**
- **Input**
  - Next state combinational logic
  - Compute next state as a function of current state and input
- **State (finite number of stored bits)**
- **Output combinational logic**
  - Compute output as a function of current state
- **Output**
Building a FSM

• Build the circuit that outputs 1 if the most recent 3 bits were 000.
Building a FSM

• Build the circuit that outputs 1 if the most recent 3 bits were 000.

• Step 1: draw the state transition diagram
Building a FSM

- Step 2: Pick an encoding for each state.
  - Here we decided to use 2 bits to represent the 4 states

Start = 00
Saw = 01
Saw00 = 10
Saw000 = 11
## Building a FSM

- Step 3: create the truth table for nextState and out

<table>
<thead>
<tr>
<th>In</th>
<th>current State</th>
<th>nextState</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00 (start)</td>
<td>01 (saw0)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>01 (saw0)</td>
<td>10 (saw00)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>10 (saw00)</td>
<td>11 (saw000)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>11 (saw000)</td>
<td>11 (saw000)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>00 (start)</td>
<td>00 (start)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01 (saw0)</td>
<td>00 (start)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10 (saw00)</td>
<td>00 (start)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>11 (saw000)</td>
<td>00 (start)</td>
<td>1</td>
</tr>
</tbody>
</table>

- Start = 00
- Saw0 = 01
- Saw00 = 10
- Saw000 = 11
### Building a FSM

- **Step 4:** write down the expression for nextState and Out

<table>
<thead>
<tr>
<th>In</th>
<th>current State</th>
<th>nextState</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00 (start)</td>
<td>01 (saw0)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>01 (saw0)</td>
<td>10 (saw00)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>10 (saw00)</td>
<td>11 (saw000)</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>11 (saw000)</td>
<td>11 (saw000)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>00 (start)</td>
<td>00 (start)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01 (saw0)</td>
<td>00 (start)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10 (saw00)</td>
<td>00 (start)</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>11 (saw000)</td>
<td>00 (start)</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
ns_0 = \overline{in} \cdot (\overline{cs_0} \cdot \overline{cs_1} + cs_1)
\]

\[
ns_1 = \overline{in} \cdot (cs_0 \cdot \overline{cs_1} + cs_1)
\]

\[
out = cs_0 \cdot cs_1
\]
Building an FSM

- Step 5: build the circuit; must have “next state logic”, “state”, and “output logic”

\[
ns_0 = \overline{\text{in}} \cdot (\overline{cs_0} \cdot \overline{cs_1} + cs_1) \\
n_s_1 = \overline{\text{in}} \cdot (cs_0 \cdot \overline{cs_1} + cs_1) \\
\text{out} = cs_0 \cdot cs_1
\]
Peer instruction

• Build the FSM that outputs 1 when the last two inputs have been different

step 1 result:
Delay in registers
Next steps

• arrange stored bits into large memories and learn how to lookup values using an address
• Learn about delay in registers