Finite State Machines

CPS 104

Administrivia

- Homework
- Midterm
- Projects
Overview of Today’s Lecture:

- Review: Control for single cycle datapath
- Introduction to Finite State Machines (FSMs)
- Definition
- Example
- State transition diagram
- State encoding
- FSM realization
- PLAs and ROM implementation of FSMs.

Reading today Appendix B.10
Next time Chapter 5 multi cycle & Appendix C multicycle control

Review: The Single Cycle Datapath during Branch

- if (R[rs] - R[rt] == 0) then Zero <- 1; else Zero <- 0

```
Branch = 1
Jump = 0

IF (Zero = 1) THEN
    ALUctr = Subtract

Else
    ALUctr = Rb

MemWr = 0
MemtoReg = x
ExtOp = x
```
Review: The “Truth Table” for the Main Control

The “Truth Table” for RegWrite

- RegWrite = R-type + ori + lw
  = !op<5> & !op<4> & !op<3> & !op<2> & !op<1> & !op<0>(R-type) + !op<5> & !op<4> & op<3> & op<2> & !op<1> & op<0>(ori) + op<5> & !op<4> & !op<3> & !op<2> & !op<1> & !op<0>(lw)
Review: Implementation of the Main Control

Putting it All Together: A Single Cycle Processor
Worst Case Timing: lw $1, $2(offset)

Drawback of this Single Cycle Processor

- **Long cycle time:**
  - Cycle time must be long enough for the load instruction:
    - PC’s Clock -to-Q +
    - Instruction Memory Access Time +
    - Register File Access Time +
    - ALU Delay (address calculation) +
    - Data Memory Access Time +
    - Register File Setup Time +
    - Clock Skew

- Cycle time is much longer than needed for all other instructions
- What we want is to break the execution of a single instruction into multiple steps
- Before we can do that, we need to go back to logic design
Finite State Machine

- \( S = \{ s_0, s_1, \ldots s_{n-1} \} \) is a finite set of states.
- \( I = \{ i_0, i_1, \ldots i_{k-1} \} \) is a finite set of input values.
- \( O = \{ o_0, o_1, \ldots o_{m-1} \} \) is a finite set output values.

**Definition:** A finite state machine is a function \( F : (S \times I) \rightarrow (S \times O) \) that gets a sequence of input values \( I_k \in I, k = 0,1,2, \ldots \) and it produces a sequence of output values \( O_k \in O, k = 1,2, \ldots \) such that:

\[ F(s_k, i_k) = (s_{k+1}, o_{k+1}) \quad k=0,1,2, \ldots \]

---

**Finite State Machine**

*(Translation to English)*

- Finite State Machine is:
  - A machine with a finite number of possible states.
  - A machine with a finite number of possible inputs.
  - A machine with a finite number of possible different outputs.
  - At each period (Clock cycle) the machine receives an input and it produces an output.
  - The output is a function of the machine input and current state.
  - After each period the machine changes state.
  - The new state is a function of the input and current state.
Example: Traffic Light Controller

Traffic light controller at an intersection.

Finite State Machine (cont.)

- Example: Traffic lights controller:
  - There are four states:
    - NG: Green light in the north-south direction.
    - NY: Yellow light in the north-south direction.
    - EG: Green light at the East-West direction.
    - EY: Yellow light at the East-West direction.
  - There are four outputs:
    - (G;R): North-South green light, East-West red light
    - (Y;R): North-South yellow light, East-West red light
    - (R;Y): North-South red light, East-West yellow light
    - (R;G): North-South red light, East-West green light
  - There are four inputs:
    - (c, c): Car at the North-South, Car at East-West
    - (c, nc): Car at North-South, No-car at East-West
    - (nc, c): No-car at North-South, Car at East-West
    - (nc, nc): No-car at North-South, No-car at East-West
### FSM Example: Traffic Light

- **State Transitions:**

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Next-State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>NG</td>
<td>(-;NC)</td>
<td>NG</td>
<td>(G;R)</td>
</tr>
<tr>
<td>NG</td>
<td>(-;C)</td>
<td>NY</td>
<td>(G;R)</td>
</tr>
<tr>
<td>NY</td>
<td>-</td>
<td>EG</td>
<td>(Y;R)</td>
</tr>
<tr>
<td>EG</td>
<td>(NC;-)</td>
<td>EG</td>
<td>(R;G)</td>
</tr>
<tr>
<td>EG</td>
<td>(C;-)</td>
<td>EY</td>
<td>(R;G)</td>
</tr>
<tr>
<td>EY</td>
<td>-</td>
<td>NG</td>
<td>(R;Y)</td>
</tr>
</tbody>
</table>

- means don’t care

Format
(North/South; East/West)

---

### Finite State Machine (cont.)

- Finite State Machines can be represented by a graph.
- The graph is called a **State Diagram**.
- The states are the nodes in the graph.
- The arcs in the graph represent **state transitions**.
- Each arc is labeled with the **Inputs** that cause the transition.
- Nodes are labeled with the **outputs**.
FSM State Diagram
Example: Traffic light Controller

State Coding

<table>
<thead>
<tr>
<th>State</th>
<th>Code</th>
<th>Input Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NG</td>
<td>00</td>
<td>(C;C) 11</td>
</tr>
<tr>
<td>NY</td>
<td>01</td>
<td>(C;NC) 10</td>
</tr>
<tr>
<td>EG</td>
<td>10</td>
<td>(NC;C) 01</td>
</tr>
<tr>
<td>EY</td>
<td>11</td>
<td>(NC;NC) 00</td>
</tr>
</tbody>
</table>

Enumerate States

<table>
<thead>
<tr>
<th>Output Code</th>
<th>One bit per color for each light</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R;G) 001100</td>
<td>GYRGYR</td>
</tr>
<tr>
<td>(G;R) 100001</td>
<td></td>
</tr>
<tr>
<td>(Y;R) 010001</td>
<td>(North; East)</td>
</tr>
<tr>
<td>(R;Y) 001010</td>
<td></td>
</tr>
</tbody>
</table>
Finite State Machine Realization

- Finite state machines can be implemented in digital hardware by selecting binary coding for the FSM.
- Use registers to hold the state.
- Use combinational logic or PLAs, or read-only Memory (ROM) to implement the transition function.
Example: Traffic Light Controller

S = State, bits are S0 and S1
NS = Next State, bits are NS0 and NS1

<table>
<thead>
<tr>
<th>IN</th>
<th>S</th>
<th>NS</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-</td>
<td>01</td>
<td>01</td>
<td>012345</td>
</tr>
<tr>
<td>1-</td>
<td>00</td>
<td>01</td>
<td>100001</td>
</tr>
<tr>
<td>--</td>
<td>01</td>
<td>10</td>
<td>010001</td>
</tr>
<tr>
<td>--</td>
<td>10</td>
<td>10</td>
<td>001100</td>
</tr>
<tr>
<td>--</td>
<td>11</td>
<td>11</td>
<td>001010</td>
</tr>
</tbody>
</table>

NS1 = S0’*S1’*I0+S0*S1’*I1
     = S1’*(S0’*I0+S0*I1)
NS0 = S0’S1+S0*S1’*I1’+S0*S1’*I1
     = S0’*S1+S0*S1’
OUT0 = S0’*S1’
OUT1 = S0’*S1
OUT2 = S0*S1’+S0*S1= S0
OUT3 = S0*S1’
OUT4 = S0*S1
OUT5 = S0’*S1’+S0’*S1= S0’

General Method for FSM design

- Determine the problem:
  1. Draw the state diagram,
  2. Write the truth table,
  3. Write sum-of-products equations
Programmable Logic Array (PLA)

- The PLA has N inputs, K outputs and M product terms.
- Each input or its complement may be used in any product term.
- Any product term can be used in the sum.
- The PLA is “programmed” once by making connections (or putting a transistor) at the wires intersections.

FSM implementation with PLA: an example

<table>
<thead>
<tr>
<th>I</th>
<th>S</th>
<th>NS</th>
<th>OT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-</td>
<td>00</td>
<td>00</td>
<td>100001</td>
</tr>
<tr>
<td>1-</td>
<td>00</td>
<td>01</td>
<td>100001</td>
</tr>
<tr>
<td>--</td>
<td>01</td>
<td>10</td>
<td>010001</td>
</tr>
<tr>
<td>-0</td>
<td>10</td>
<td>10</td>
<td>001100</td>
</tr>
<tr>
<td>-1</td>
<td>11</td>
<td>11</td>
<td>001100</td>
</tr>
<tr>
<td>--</td>
<td>11</td>
<td>00</td>
<td>001010</td>
</tr>
</tbody>
</table>
Read Only Memory (ROM) Implementation

- Read Only Memory (ROM) is programmed at manufacturing time.
- Programmable ROM can be electrically programed (EPROM).
- To implement FSM with k-Inputs, N-bits of state, M-Outputs:
  - Connect the Inputs and State bits to the ROM address lines.
  - Connect Register to the ROM output.
  - Feed back the Next-State bits of the register into the State inputs
  - Needs: $2^{(K+N)}$ words ROM. Each word at least (N+M) bits wide.

ROM Implementation (cont.)

- Inputs $k$
- State $N$
- Register
- Outputs $M$
- Clock
- $2^{(N+K)}$ ROM
- Address
A Simple Arrow FSM

- Consider those flashing arrow signs
- No light, one light, two lights, three lights
- Let’s design the FSM to control this sign
Sequence Recognizer/Combination Lock

- A digital lock system must recognize three number input in the correct order. Must restart if incorrect number is entered.

- Design the finite state machine to recognize the combination: 8, 14, 5 (assume you have three separate inputs that indicate if the current input is equal to 8, 14, or 5, respectively)

Summary

Finite State Machines
- Inputs, Current State
- Compute Outputs and Next State
- Read Appendix B and Chapter 5 (multicycle processor)

- Homework