Outline

- Alternative controller FSM implementation approaches based on:
  - Classical Moore and Mealy machines
  - Time state: Divide and Counter
  - Jump counters
  - Microprogramming (ROM) based approaches
    - branch sequencers
    - horizontal microcode
    - vertical microcode

Alternative Ways to Implement Processor FSMs

- "Random Logic" based on Moore and Mealy Design
  - Classical Finite State Machine Design
- Divide and Conquer Approach: Time-State Method
  - Partition FSM into multiple communicating FSMs
- Exploit MSI Functionality: Jump Counters
  - Counters, Multiplexers, Decoders
- Microprogramming: ROM-based methods
  - Direct encoding of next states and outputs

Random Logic

- Perhaps poor choice of terms for "classical" FSMs
- Contrast with structured logic: PAL/PLA, FPGA, ROM
- Could just as easily construct Moore and Mealy machines with these components

Memory-Register Interface Timing

- Valid data latched on IF2 to IF3 transition because data must be valid before Wait can go low

Moore Machine State Diagram

- Note capture of MBR in these states

Moore Machine Diagram

- 16 states, 4 bit state register
- Next State Logic: 9 Inputs, 4 Outputs
- Output Logic: 4 Inputs, 18 Outputs
- These can be implemented via ROM or PAL/PLA
- Output: 16 x 8 bit ROM
### Moore Machine State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next State</th>
<th>Register Transfer Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td>1111</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>0101</td>
<td>0110</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>0100</td>
<td>0011</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>0011</td>
<td>0101</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>0010</td>
<td>0001</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>0001</td>
<td>0011</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>0000</td>
<td>1111</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>1111</td>
<td>0011</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>1110</td>
<td>0101</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>1101</td>
<td>0110</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>1100</td>
<td>0001</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>1011</td>
<td>1111</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>1010</td>
<td>1101</td>
<td>WR (1100)</td>
</tr>
<tr>
<td>1001</td>
<td>1011</td>
<td>WR (1100)</td>
</tr>
</tbody>
</table>

### Moore Machine Implementation

#### NOVA Assignment

**NOVA State Assignment SUMMARY**

- Products: 22
- Best Product: 18
- Product Terms: 414
- Improves on 21!

- States: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
- Atomic Functions: 0000, 1011, 1111, 0011, 0100, 0110, 1000, 1010, 1100, 1110

### Synchronous Mealy Machines

- Standard Mealy Machines have synchronous outputs.
- Inputs change to output changes, independent of clock.
- Revise Mealy Machine design if outputs change only on clock edges.

#### Synchronous Mealy Machine Circuitry

- **Input:** States, Outputs
- **Output:** States, Logic

---

**CS 150 - Spring 2004 – Lec #12: Control Implementation**
### Synchronous Mealy Machines

**Case I: Synchronizers at Inputs and Outputs**

A asserted in Cycle 0, \( f \) becomes asserted after 2 cycle delay!

This is clearly overkill!

### Synchronous Mealy Machines

**Case II: Synchronizers on Inputs**

A asserted in Cycle 0, \( f \) follows in next cycle

Same as using delayed signal \( A' \) in Cycle 1!

### Synchronous Mealy Machines

**Case III: Synchronized Outputs**

A asserted during Cycle 0, \( f' \) asserted in next cycle

Effect of \( f \) delayed one cycle

### Implications for Processor FSM Already Derived

- Consider inputs: Reset, Wait, IR\(15:14\), AC\(15\)
  - Latter two already come from registers, and are sync'd to clock
  - Possible to load IR with new instruction in one state
  - Perform multeway branch on opcode in next state

- Best solution for Reset and Wait: synchronized inputs
  - Place D flipflops between these external signals and the control inputs to the processor FSM
  - Sync'd versions of Reset and Wait delayed by one clock cycle

### Time State Divide and Conquer

- **Overview**
  - Classical Approach: Monolithic Implementations
  - Alternative "Divide & Conquer" Approach:
    - Decompose FSM into several communicating FSMs
    - Time state FSM (e.g., IFetch, Decode, Execute)
    - Instruction state FSM (e.g., LD, ST, ADD, BRN)
    - Condition state FSM (e.g., AC \( \neq 0 \), AC \( = 0 \))

### Time State (Divide & Conquer)

**Time State FSM**

- Most instructions follow some basic sequence
- Differ only in detailed execution sequence

**Time State FSM can be parameterized by opcode and AC states**

**Instructions State:**
- Stared in IR\(15:14\)

**Conditions State:**
- Stared at AC \( \neq 0 \)
Time State (Divide & Conquer)

Generation of Microoperations

- Initial State: PC: Reset
- PC + 1 → PC: T0
- PC → MAR: T0
- MAR → Memory Address Bus: T2 + T6 (LD + ST + ADD)
- Memory Data Bus → MBR: T2 + T6 (LD + ADD)
- MBR → IR: T4
- MBR → AC: T7 (LD)
- AC + MBR → AC: T7 + ADD
- IR[13:0] → MAR: T5 (LD + ST + ADD)
- IR[13:0] → RC: T6 (BRN)
- 1 → Request: T2 + T6 (LD + ADD)

Jump Counter

Concept

Implement FSM using ASM functionality: counters, mux, decoders

Pure jump counter only one of four possible next states

Hybrid jump counter:

Synchronous jump state

Additional states:

Logic blocks implemented via discrete logic, PALs/PLAs, ROMs
Jump Counters

Implementation Example, Continued

- CLR = CLRm + Reset
- CNT = (s2 + s6 + s9 + s11) + Wait (s1 + s3) + Wait (s2 + s6 + s9 + s11)
- CLR = Reset + s7 + s12 + s13 (s8 - WAIT)
- LD = s4

Contents of Jump State ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents (Symbolic State)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0101 (L00)</td>
</tr>
<tr>
<td>01</td>
<td>1000 (ST0)</td>
</tr>
<tr>
<td>10</td>
<td>1010 (ADD)</td>
</tr>
<tr>
<td>11</td>
<td>1101 (BR0)</td>
</tr>
</tbody>
</table>

Jump Counter

- CLR, CNT, LD implemented via Mux Logic
- CLR = CLRm + Reset
- CNT = CLRm + Reset

Active Lo outputs:
- hi input inverted at the output

Note that CNT is active high counter so invert MUX inputs!

Jump Counters

- Microoperation implementation
- 0 \rightarrow PC = Reset
- 1 \rightarrow PC \leftarrow PC + 1
- PC \leftarrow MAR = 50
- MAR \leftarrow Memory Address Bus = Walk(s1 + s2 + s5 + s6 + s8 + s9 + s11 + s12)
- Memory Data Bus \leftarrow MBR = Wait(s2 + s6 + s9 + s11)
- MBR \leftarrow Memory Data Bus = Wait(s8 + s9)

Jump Counters: CNT, CLR, LD functions of current state + Wait

Controller Implementation Summary

- Control Unit Organization
  - Register transfer operation
  - Classical Moore and Mealy machines
  - Time State Approach
  - Jump Counter
- Next Time:
  - Branch Sequencers
  - Horizontal and Vertical Microprogramming