Controller Implementation--Part II

- Alternative controller FSM implementation approaches based on:
  - Classical Moore and Mealy machines
  - Time-State: Divide and Conquer
  - Jump counters
  - Microprogramming (ROM) based approaches
    - Branch sequencers
    - Horizontal microcode
    - Vertical microcode

Branch Sequencers

4 Way Branch Sequencer

Processor CPU Design Example

Example Processor FSM

<table>
<thead>
<tr>
<th>ROM ADDRESS</th>
<th>ROM CONTENTS</th>
<th>Next State Register Transfer Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Reset, Current State, a, b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES 0 0000 X X</td>
<td>0001 (X)</td>
<td>PC = MAR, FC + 1 = FC</td>
</tr>
<tr>
<td>IF0 0 0001 0 0</td>
<td>0010 (IF0)</td>
<td>IR&lt;14&gt; + s&lt;0&gt; = IR, MAR, MBR</td>
</tr>
<tr>
<td>IF1 0 0001 1 1</td>
<td>0011 (IF1)</td>
<td>MAR = Mem, Read, Request</td>
</tr>
<tr>
<td>IF2 0 0011 0 0</td>
<td>0011 (IF2)</td>
<td>MAR = Mem, Write, Request, MBR</td>
</tr>
<tr>
<td>CD 0 0011 1 1</td>
<td>0010 (CD)</td>
<td>MAR = Mem, Write, Request, MBR</td>
</tr>
<tr>
<td>OD 0 0100 0 0</td>
<td>0101 (LOD)</td>
<td>IR = MAR</td>
</tr>
<tr>
<td>OD0 0 1000 0 0</td>
<td>1001 (STO)</td>
<td>IR = MAR, AC = MBR</td>
</tr>
<tr>
<td>OD1 0 1000 1 0</td>
<td>1001 (AD0)</td>
<td>IR = MAR</td>
</tr>
<tr>
<td>OD2 0 1100 1 1</td>
<td>1101 (BR0)</td>
<td>IR = MAR</td>
</tr>
</tbody>
</table>

Branch Sequencer

Concept

- Implement Next State Logic via ROM
- Address ROM with current state and inputs

Problem: ROM doubles in size for each additional input

- Note: Jump counter trades off ROM size vs. external logic
- Even in hybrid approach, state + input subset form ROM address

Branch Sequencer: between the extremes

- Next State stored in ROM
- Each state limited to small number of next states
- Always a power of 2

Observe: only a small set of inputs are examined in any state
Microprogramming

How to organize the control signals

Implement control signals by storing 1’s and 0’s in a ROM.

Horizontal vs. vertical microprogramming

Horizontal: 1 ROM output for each control signal

Vertical: encoded control signals in ROM, decoded externally

Some mutually exclusive signals can be combined

Helps reduce ROM length

Horizontal Microprogramming

Moore Processor ROM

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next States</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0010, 0011</td>
</tr>
<tr>
<td>0010</td>
<td>0011, 0101</td>
</tr>
<tr>
<td>0100</td>
<td>0101, 0111</td>
</tr>
<tr>
<td>0110</td>
<td>0111, 1001</td>
</tr>
<tr>
<td>1000</td>
<td>1001, 1011</td>
</tr>
<tr>
<td>1010</td>
<td>1011, 1101</td>
</tr>
<tr>
<td>1100</td>
<td>1101, 1111</td>
</tr>
</tbody>
</table>

Input: 0 = Wait, 1 = IB-15

Output Encoding:

Group mutually exclusive signals
Use external logic to decode

Example:

0: #PC, PC = 1: #PC, ABUS = #PC mutually exclusive.

Save ROM bit with external 2:4 Decoder

Horizontal Microprogramming

Advantages:

Most flexibility -- complete parallel access to datapath control points

Disadvantages:

Very long control words -- 300+ bits for real processors

NOTE: Not all microoperation combinations make sense

Moore Processor ROM

<table>
<thead>
<tr>
<th>Current State</th>
<th>Next States</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0010, 0011</td>
</tr>
<tr>
<td>0010</td>
<td>0011, 0101</td>
</tr>
<tr>
<td>0100</td>
<td>0101, 0111</td>
</tr>
<tr>
<td>0110</td>
<td>0111, 1001</td>
</tr>
<tr>
<td>1000</td>
<td>1001, 1011</td>
</tr>
<tr>
<td>1010</td>
<td>1011, 1101</td>
</tr>
<tr>
<td>1100</td>
<td>1101, 1111</td>
</tr>
</tbody>
</table>

Alpha inputs: 0 = Wait, 1 = IB-15

Beta inputs: 0 = AC-15, 1 = IB-14
Vertical Microprogramming

More extensive encoding to reduce ROM word length

- Typically use multiple microword formats:
  - Horizontal microcode: -- next state + control bits in same word
  - Separate formats for control outputs and "branch jumps"
  - May require several microwords in a sequence to implement same function as single horizontal word

- In the extreme, very much like assembly language programming

Vertical Microprogramming

Control Outputs

- Partially Encoded Control Outputs
- More extensive encoding to reduce ROM word length

Horizontal Microprogramming

- Separates formats for control outputs and "branch jumps"
- In the extreme, very much like assembly language programming
**Vertical Microprogramming**

Condition Logic

<table>
<thead>
<tr>
<th>Condition Selector</th>
<th>Condition Comparator</th>
<th>Microinstruction Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait AC&lt;15&gt; IR&lt;15&gt;</td>
<td>IR&lt;14&gt; 4:1 MUX</td>
<td>LD</td>
</tr>
</tbody>
</table>

**Writeable Control Store**

- Part of control store addresses map into RAM
  - Allows assembly language programmer to implement own instructions
  - Extend “native” instruction set with application specific instructions
  - Requires considerable sophistication to write microcode
  - Not a popular approach with today’s processors
- Make the native instruction set simple and fast
- Write “higher level” functions as assembly language sequences

**Controller Implementation Summary--Part II**

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