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>0000 X X</td>
<td>000 (3F) PC → MAR, PC + 1 → PC</td>
<td></td>
</tr>
<tr>
<td>0001 0 0</td>
<td>000 (3F) PC → MAR</td>
<td></td>
</tr>
<tr>
<td>0100 1 1</td>
<td>0010 (3F) IMAR → Mem, Read Request</td>
<td></td>
</tr>
<tr>
<td>0100 1 1</td>
<td>0010 (3F) IMAR → Mem, Read Request</td>
<td></td>
</tr>
<tr>
<td>0101 1 1</td>
<td>0010 (3F) IMAR → Mem, Read Request</td>
<td></td>
</tr>
<tr>
<td>0101 1 1</td>
<td>0010 (3F) IMAR → Mem, Read Request</td>
<td></td>
</tr>
<tr>
<td>0101 1 1</td>
<td>0010 (3F) IMAR → Mem, Read Request</td>
<td></td>
</tr>
</tbody>
</table>

Branch Sequencers

- 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 tracks off ROM size vs. external logic
- Even in hybrid approach, state + input subset form ROM address
- Branch Sequencer: between the extremes
- Next States 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
Branch Sequencers

Alternative Horizontal Implementation

Input MUX as thrashed by encoded signals, not state
Much fewer inputs than unique states!
1 example ROM input MUX can be 213

Adding length to ROM word saves 16 bits vs. doubling words
Vertical Format: (14 + 4) x 64 = 1152 ROM bits
Horizontal Format: 14 x 4 x 64 = 1152 ROM bits

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

14 Register Transfer operations become 22 Microoperations:

<table>
<thead>
<tr>
<th>PC</th>
<th>MBR</th>
<th>ABUS</th>
<th>RBUS</th>
<th>AC</th>
<th>ALU ADD</th>
<th>ALU PASS B</th>
<th>MAR</th>
<th>MBR Data Bus</th>
<th>ABUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>α</td>
<td>β</td>
<td>Request</td>
<td>Group</td>
<td>mutually exclusive signals</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output Encodings:
most flexibility: complete parallel access to datapath control points

Disadvantages:
very long control words: 100+ bits for real processors

Output Encodings:
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

Moore Processor ROM

<table>
<thead>
<tr>
<th>Current Status</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>Ab</th>
<th>Next Status</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>Ab</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0000)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(0001)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(0010)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Horizontal Microprogramming

Advantages:
most flexibility: complete parallel access to datapath control points

Disadvantages:
very long control words: 100+ bits for real processors

NOTE: Not all microoperation combinations make sense!

Output Encodings:
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
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

Condition Logic

- 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

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