Single-Cycle CPU
Control Logic

Putting it All Together: A Single Cycle Datapath

- We have everything except control signals

Okay, then, what about those Control Signals?

ALU control bits

- Recall: 5-function ALU

<table>
<thead>
<tr>
<th>ALU control input</th>
<th>Function</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>And</td>
<td>and</td>
</tr>
<tr>
<td>001</td>
<td>Or</td>
<td>or</td>
</tr>
<tr>
<td>010</td>
<td>Add</td>
<td>add, lw, sw</td>
</tr>
<tr>
<td>110</td>
<td>Subtract</td>
<td>sub, beq</td>
</tr>
<tr>
<td>111</td>
<td>Slt</td>
<td>slt</td>
</tr>
</tbody>
</table>

- based on opcode (bits 31-26) and function code (bits 5-0) from instruction
- ALU doesn’t need to know all opcodes—we will summarize opcode with ALUOp (2 bits):
  - 00 - lw, sw
  - 01 - beq
  - 10 - R-format
Generating ALU control

<table>
<thead>
<tr>
<th>Instruction opcode</th>
<th>ALUOp</th>
<th>Instruction operation</th>
<th>Function code</th>
<th>Desired ALU action</th>
<th>ALU control input</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>00</td>
<td>load word</td>
<td>xxxxxxx</td>
<td>add</td>
<td>010</td>
</tr>
<tr>
<td>sw</td>
<td>00</td>
<td>store word</td>
<td>xxxxxxx</td>
<td>add</td>
<td>010</td>
</tr>
<tr>
<td>beq</td>
<td>01</td>
<td>branch eq</td>
<td>xxxxxxx</td>
<td>subtract</td>
<td>110</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>add</td>
<td>100000</td>
<td>add</td>
<td>010</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>subtract</td>
<td>100010</td>
<td>subtract</td>
<td>110</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>AND</td>
<td>100100</td>
<td>and</td>
<td>000</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>OR</td>
<td>100101</td>
<td>or</td>
<td>001</td>
</tr>
<tr>
<td>R-type</td>
<td>10</td>
<td>slt</td>
<td>101010</td>
<td>slt</td>
<td>111</td>
</tr>
</tbody>
</table>

ALU Control Logic

Generating individual ALU signals

<table>
<thead>
<tr>
<th>ALUop</th>
<th>Function</th>
<th>ALUctr signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>xxxxx</td>
<td>010</td>
</tr>
<tr>
<td>01</td>
<td>xxxxx</td>
<td>110</td>
</tr>
<tr>
<td>10</td>
<td>00000</td>
<td>010</td>
</tr>
<tr>
<td>10</td>
<td>00100</td>
<td>110</td>
</tr>
<tr>
<td>10</td>
<td>01000</td>
<td>000</td>
</tr>
<tr>
<td>10</td>
<td>01010</td>
<td>001</td>
</tr>
<tr>
<td>10</td>
<td>10100</td>
<td>111</td>
</tr>
</tbody>
</table>

CSE 141, S2'06

Jeff Brown
sw Control

beq Control

Control Truth Table

<table>
<thead>
<tr>
<th>Opcode</th>
<th>lw</th>
<th>sw</th>
<th>beq</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format</td>
<td>000000</td>
<td>10011</td>
<td>101011</td>
</tr>
<tr>
<td>RegDst</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MemRead</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Branch</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ALUOp1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ALUOp0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Simple combinational logic (truth tables)
Single-Cycle CPU Summary

- Easy, particularly the control
- Which instruction takes the longest? By how much? Why is that a problem?
- $ET = IC \times CPI \times CT$
- What else can we do?
- When does a multi-cycle implementation make sense?
  - e.g., 70% of instructions take 75 ns, 30% take 200 ns?
  - suppose 20% overhead for extra latches
- Real machines have much more variable instruction latencies than this.