Control Overview

• Single-cycle implementation
  – Datapath: combinational logic, I-mem, regs, D-mem, PC
    • Last three written at end of cycle
  – Need control – just combinational logic!
  – Inputs:
    • Instruction (I-mem out)
    • Zero (for beq)
  – Outputs:
    • Control lines for muxes
    • ALUop
    • Write-enables
Control Overview

• Fast control
  – Divide up work on “need to know” basis
  – Logic with fewer inputs is faster
• E.g.
  – Global control need not know which ALUop
ALU Control

• Assume ALU uses

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>and</td>
</tr>
<tr>
<td>001</td>
<td>or</td>
</tr>
<tr>
<td>010</td>
<td>add</td>
</tr>
<tr>
<td>110</td>
<td>sub</td>
</tr>
<tr>
<td>111</td>
<td>slt (set less than)</td>
</tr>
<tr>
<td>others</td>
<td>don’t care</td>
</tr>
</tbody>
</table>
# ALU Control

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
<th>Opcode</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add</td>
<td>000000</td>
<td>1000000</td>
</tr>
<tr>
<td>sub</td>
<td>sub</td>
<td>000000</td>
<td>100010</td>
</tr>
<tr>
<td>and</td>
<td>and</td>
<td>000000</td>
<td>100100</td>
</tr>
<tr>
<td>or</td>
<td>or</td>
<td>000000</td>
<td>100101</td>
</tr>
<tr>
<td>slt</td>
<td>slt</td>
<td>000000</td>
<td>101010</td>
</tr>
</tbody>
</table>

- ALU-ctrl = f(opcode, function)
But...don’t forget

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
<th>Opcode</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>add</td>
<td>100011</td>
<td>xxxxxxx</td>
</tr>
<tr>
<td>sw</td>
<td>add</td>
<td>101011</td>
<td>xxxxxxx</td>
</tr>
<tr>
<td>beq</td>
<td>sub</td>
<td>000100</td>
<td>100010</td>
</tr>
</tbody>
</table>

- To simplify ALU-ctrl
  - ALUop = f(opcode)
    - 2 bits
    - 6 bits
ALU Control

- ALU-ctrl = f(ALUop, function)
- 3 bits  2 bits  6 bits
- Requires only five gates plus inverters

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>add, sub, and, …</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>lw, sw</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>beq</td>
</tr>
</tbody>
</table>
Control Signals Needed
Global Control

- **R-format**: opcode  rs  rt  rd  shamt  function
  
  \[
  \begin{array}{ccccccc}
  6 & 5 & 5 & 5 & 5 & 5 & 6 \\
  \end{array}
  \]

- **I-format**: opcode  rs  rt  address/immediate
  
  \[
  \begin{array}{ccccccc}
  6 & 5 & 5 & 5 & 16 \\
  \end{array}
  \]

- **J-format**: opcode  address
  
  \[
  \begin{array}{ccccccc}
  6 & 26 & \\
  \end{array}
  \]
Global Control

• Route instruction[25:21] as read reg1 spec
• Route instruction[20:16] are read reg2 spec
• Route instruction[20:16] (load) and and instruction[15:11] (others) to
  – Write reg mux
• Call instruction[31:26] op[5:0]
Global Control

• Global control outputs
  – ALU-ctrl - see above
  – ALU src - R-format, beq vs. ld/st
  – MemRead - lw
  – MemWrite - sw
  – MemtoReg - lw
  – RegDst - lw dst in bits 20:16, not 15:11
  – RegWrite - all but beq and sw
  – PCSrc - beq taken
Global Control

• Global control outputs
  – Replace PCsrc with
    • Branch beq
    • PCSrc = Branch * Zero

• What are the inputs needed to determine above global control signals?
  – Just Op[5:0]
### Global Control

- RegDst = ~Op[0]
- ALUSrc = Op[0]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>RegDst</th>
<th>ALUSrc</th>
</tr>
</thead>
<tbody>
<tr>
<td>rrr</td>
<td>000000</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>100011</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>sw</td>
<td>101011</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>beq</td>
<td>000100</td>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>???</td>
<td>others</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Global Control

• More complex with entire MIPS ISA
  – Need more systematic structure
  – Want to share gates between control signals

• Common solution: PLA
  – MIPS opcode space designed to minimize PLA inputs, minterms, and outputs

• Refer to MIPS Opcode map
PLA

• In AND-plane, \& selected inputs to get minterms
• In OR-plane, | selected minterms to get outputs
• E.g.
Control Signals; Add Jumps
Control Signals w/Jumps
What’s wrong with single cycle?

- Critical path probably lw:
  - I-mem, reg-read, alu, d-mem, reg-write
- Other instructions faster
  - E.g. rrr: skip d-mem
- Instruction variation much worse for full ISA and real implementation:
  - FP divide
  - Cache misses (what the heck is this? – later)

<table>
<thead>
<tr>
<th>Instructions/Program (code size)</th>
<th>Cycles/Instruction (CPI)</th>
<th>Time/Cycle (cycle time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Single Cycle Implementation

• Solution
  – Variable clock?
    • Too hard to control, design
  – Fixed short clock
    • Variable cycles per instruction

• Multicycle control (next lecture)
Summary

• Processor implementation
  – Datapath
  – Control

• Single cycle implementation