Control Unit

CPU hardware that controls instruction execution
- sends signals to the datapath to operate it
- specifies what operations to perform, what data to move, when to move it, where to move it

Control Signals

Many control signals driven by the instruction

<table>
<thead>
<tr>
<th>R-type</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode 0</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>func</td>
</tr>
<tr>
<td>31-26</td>
<td>25-21</td>
<td>20-16</td>
<td>15-11</td>
<td>10-6</td>
<td>5-0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>load/store</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode 35 or 43</td>
<td>rs</td>
<td>rt</td>
<td>displacement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31-26</td>
<td>25-21</td>
<td>20-16</td>
<td>15-0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>branch</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode 4</td>
<td>rs</td>
<td>rt</td>
<td>displacement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31-26</td>
<td>25-21</td>
<td>20-16</td>
<td>15-0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>jump</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode 2</td>
<td>address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31-26</td>
<td>25-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Regularity of the MIPS formats
- **opcode** always in bits 31-26 (Op[5-0])
- **source registers** are always rs & rt
- **base register** always rs
- **branch offset** always bits 15-0
Our R2000 Control Signals

Register file
- register write signal: \texttt{RegWrite}
  asserted for R-type instructions & load
- register destination field: \texttt{RegDst}
  \(rt, rd\)
- results value: \texttt{MemToReg}
  loaded value, R-type instruction result
- all generated by the opcode

ALU
- type of the second operand: \texttt{ALUSrc}
  register, immediate
  - generated by the opcode
- ALU operation: \texttt{ALUOp}
  add, subtract, and, or, set-on-less-than
  - generated by a small control unit
    - inputs: opcode & func field
    - output: ALU operation
  - examples:
    \texttt{lw/sw} \Rightarrow \texttt{add}
    \texttt{beq} \Rightarrow \texttt{subtract}
    R-type instruction \Rightarrow \texttt{func value}

Memory
- read signal: \texttt{MemRead}
- write signal: \texttt{MemWrite}
  - both generated by the opcode

Branch control
- new PC value: \texttt{PCSrc}
  incremented PC, target address
  - generated by the opcode AND'd with \texttt{Zero}

Jump control
- new PC value: \texttt{Jump}
  incremented PC or target address, jump address
  - generated by the opcode
Changing the Implementation

How should you approach a problem in which you had to redesign the implementation to include another instruction?

• What does the instruction do?
• What parts of the datapath does it need?
  • Can it use what is there already?
  • What new logic or registers does it need?
• How is the datapath activated?
  • What control lines does it need
  • Where should the control lines come from?