**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>opcode</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>func</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>

R-type

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

load/store

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

branch

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

jump

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: RegWrite
  asserted for R-type instructions & load
- register destination field: RegDst
  rt or rd
- results value: MemToReg
  loaded value or R-type instruction result
- all generated by the opcode

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

Memory
- read signal: MemRead
- write signal: MemWrite
  - both generated by the opcode

Branch control
- new PC value: PCSrc
  incremented PC or target address
  - generated by the opcode AND'd with Zero

Jump control
- new PC value: Jump
  incremented PC or target address, or 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?
  - Do we need a new control line?
  - Does an existing control line need to be enlarged?