Single Cycle MIPS Implementation
- All instructions take the same amount of time
  - Signals propagate along longest path
  - CPI = 1
- Lots of operations happening in parallel
  - Increment PC
  - ALU
  - Branch target computation
- Inefficient

Multicycle MIPS Implementation
- Instructions take different number of cycles
  - Cycles are identical in length
  - Share resources across cycles
    - E.g. one ALU for everything
    - Minimize hardware
  - Cycles are independent across instructions
    - R-type and memory-reference instructions do different things in their 4th cycles
  - CPI is 3, 4, or 5 depending on instruction

Multicycle versions of various instructions
- R-type (add, sub, etc.) – 4 cycles
  1. Read instruction
  2. Decode/read registers
  3. ALU operation
  4. ALU Result stored back to destination register.
- Branch – 3 cycles
  1. Read instruction
  2. Get branch address (ALU); read regs for comparing.
  3. ALU compares registers; if branch taken, update PC

Control for new instructions
- Suppose we introduce lw2r:
  - lw2r $1, $2, $3:
    - compute address as $2+$3
    - put result into $1.
    - In other words: lw $1, 0($2+$3)
  - R-type instruction
  - How does the state diagram change?

Control for new instructions
- Suppose we introduce lw2r:
  - lw2r $1, $2, $3:
    - compute address as $2+$3
    - Load value at this address into $1
    - In other words: lw $1, 0($2+$3)
  - R-type instruction
  - How does the state diagram change?
    - New states: A,B,C
    - State A: lw2r; State B: ALUOp=0, ALUSrc=1, ALUSrcB=0
    - B controls: MemRead=1, lor7 = 1
    - C controls: RegDst = 1, RegWrite = 1, MemToReg = 1
Performance

- CPI: cycles per instruction
  - Average CPI based on instruction mix
- Execution time = IC / CPI x C
  - Where IC = instruction count; C = clock cycle time
- Performance: inverse of execution time
- MIPS = million instructions per second
  - Higher is better
- Amdahl's Law:

Performance Examples

- Finding average CPI:

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>load/store</td>
<td>50%</td>
<td>2</td>
</tr>
<tr>
<td>jal/jr</td>
<td>8%</td>
<td>2</td>
</tr>
<tr>
<td>Branches</td>
<td>8%</td>
<td>3</td>
</tr>
<tr>
<td>ALU</td>
<td>34%</td>
<td>1</td>
</tr>
</tbody>
</table>

- CPI = 0.50*2 + 0.08*2 + 0.08*3 + 0.34*1
  - CPI = 1.74

Performance Examples

- CPI = 1.74, 2GHz P4, 10^9 instructions.
- Execution time = IC * CPI * Cycle time
  = 10^9 * 1.74 * 0.5 ns = 0.87 seconds

Performance Examples

- We improve the design and change CPI of load/store to 1.
  - Speedup assuming the same program?
Performance Examples

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>load/store</td>
<td>50%</td>
<td>2</td>
</tr>
<tr>
<td>in/out</td>
<td>8%</td>
<td>2</td>
</tr>
<tr>
<td>Branches</td>
<td>8%</td>
<td>3</td>
</tr>
<tr>
<td>ALU</td>
<td>34%</td>
<td>1</td>
</tr>
</tbody>
</table>

We improve the design and change CPI of load/store to 1.
- Speedup assuming the same program/cycle time?
  - \( CPI_{new} = 0.5 \times 1 + 0.08 \times 2 + 0.08 \times 3 + 0.34 \times 1 \times CPI_{new} = 1.24 \)
  - Speedup = \( \frac{1.74}{1.24} = 1.4 \)

Amdahl’s Law

\[ \text{Execution time} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected} \]

- Suppose I make my add instructions twice as fast.
  - Suppose 20% of my program is doing adds
  - Speedup?
  - What if I make the adds infinitely fast?
  - Speedup = \( \frac{5}{1.5} \times 20\% \) improvement!