Multicycle Review

Performance Examples

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

Multicycle versions of various instructions
- Load – 5 cycles
  1. Read instruction
  2. Decode/read registers
  3. ALU adds immediate to register to form address
  4. Address passed to memory; data is read into MDR
  5. Data in MDR is stored into destination register
- Store – 4 cycles
  1. Read instruction
  2. Decode/read registers
  3. ALU adds immediate to a register to form address
  4. Save data from the other source register into memory at address from cycle 3

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 1 (where the state A, B, C, back to 0
      - A controls: ALUOp=00, ALUSrcA=1, ALUSrcB=0
      - B controls: MemRead=1, MIV = 1
      - C controls: RegWrite = 1, MemToReg = 1

Performance

- CPI: cycles per instruction
  - Average CPI based on instruction mixes
- Execution time = IC * CPI * 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:
  \[
  \text{Performance after improvement} = \frac{\text{Performance before improvement}}{1 + \frac{P}{(1-P)}}
  \]

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 = 1.74
- Assume a 2GHz P4, with program consisting of 1,000,000,000 instructions.
- Find execution time

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

<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 = 1.74, 2GHz P4, 10^9 instructions.
- Execution time = IC * CPI * Cytime
  = 10^9 * 1.74 * 0.5 ns = 0.87 seconds
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>ml/mr</td>
<td>8%</td>
<td>2</td>
</tr>
<tr>
<td>Branches</td>
<td>8%</td>
<td>3</td>
</tr>
<tr>
<td>ALU</td>
<td>54%</td>
<td>1</td>
</tr>
</tbody>
</table>

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

Amdahl’s Law

(i) 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?

(ii) Suppose I make my add instructions twice as fast.
    - Suppose 20% of my program is doing adds
    - Speedup?
    - New Execute = old_execute/(4/5 + (1/5)) = 9/10 * old_execute
    - Speedup = 10/9
    - What if I make the adds infinitely fast?
      - Speedup = 5/4, only 25% improvement

Multicycle performance example

- Multicycle can have better performance than single cycle
  - Instructions take only as many cycles as they need
- CPI Example
  - Loads: 5, stores: 4, R-type: 4, branches: 3
  - % of total instructions:
    - loads: 22%, stores: 11%, R-type: 50%, branches: 17%
  - Same # of instructions for single cycle and multicycle!
  - CPI<sub>single</sub> = 1
    - but each cycle of M<sub>single</sub> is equivalent to 5 cycles of M<sub>mult</sub>
      - So effectively, CPI<sub>mult</sub> = 5 for this comparison
  - CPI<sub>mult</sub> = 5 * .22 + 4 * .11 + 4 * .50 + 3 * .17 = 4.05
  - Speedup = 5/4.05 = 1.2