G22.2243-001
High Performance Computer Architecture

Lecture 6
Multiple Issue Processors

October 10, 2007
Outline

• Announcements
  – Assignment 2 due back now
  – Lab 1 due next week @5pm

• Multiple Issue Processors

[Hennessy/Patterson CA:AQA (4th Edition): Chapters 2 and 3]
Multiple Issue Processors
Issuing Multiple Instructions Per Cycle

• Why?
  – All of the schemes described so far can at best achieve 1 instruction/cycle
  – Increasing transistor budgets, parallelism in instruction streams (independent instructions) have pushed for multiple instructions/cycle

Two variations
• **Superscalar**: varying number of instructions/cycle (1 to 8),
  – static (requires compiler support for most benefit), or dynamic
  – with or without speculation (implies hardware scheduling)
  – IBM Power2, Sun UltraSPARC, Pentium III/4, DEC Alpha, HP 8000
• **(Very) Long Instruction Words (V)LIW**: fixed set of instructions (4-16)
  – scheduled by the compiler: put ops into wide templates
  – i860, IA64 (with some hardware support)

• New metric of performance: Instructions Per Clock cycle (IPC) vs. CPI
Statically Scheduled Superscalar MIPS Processor

Superscalar MIPS: 2 instructions; 1 FP op, 1 other

• Instruction issue
  – Fetch 64-bits/clock cycle
    • Need to handle cache-line complications
  – Hardware determines whether 0, 1, or 2 instructions can be issued
    • Can only issue 2nd instruction if 1st instruction issues

• Hazard detection
  – Likelihood of hazards between two instructions in a packet
    • Simple solution: treat this as a structural hazard (issue only 1 of them)
  – 1-instruction load delay can expand to 3-instruction delay in 2-way SS
    • 2\textsuperscript{nd} instruction in the pair can’t use it without stall, nor 2 instructions in next slot
  – Branch delay becomes 2 cycles (2\textsuperscript{nd} inst. is branch) or 3 cycles (1\textsuperscript{st} inst. is branch)

• Execution
  – Additional (or pipelined) functional units to derive benefit
  – Additional port for FP registers to do FP load or FP store and FP op
  – More bypass paths
# 2-way Static Superscalar MIPS Pipeline

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Pipe Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>IF, ID, EX, MEM, WB</td>
</tr>
<tr>
<td>FP</td>
<td>IF, ID, EX, MEM, WB</td>
</tr>
<tr>
<td>Integer</td>
<td>IF, ID, EX, MEM, WB</td>
</tr>
<tr>
<td>FP</td>
<td>IF, ID, EX, MEM, WB</td>
</tr>
<tr>
<td>Integer</td>
<td>IF, ID, EX, MEM, WB</td>
</tr>
<tr>
<td>FP</td>
<td>IF, ID, EX, MEM, WB</td>
</tr>
<tr>
<td>Integer</td>
<td>IF, ID, EX, MEM, WB</td>
</tr>
<tr>
<td>FP</td>
<td>IF, ID, EX, MEM, WB</td>
</tr>
</tbody>
</table>

*Note: IF, ID, EX, MEM, WB represent the five stages of the pipeline for both integer and floating-point instructions.*
Pipeline Scheduling and Loop Unrolling

- Rare to find the ideal instruction mix of the previous slide
- Modern-day compilers apply several optimizations so as to expose **Instruction Level Parallelism (ILP)**

```plaintext
for (i=1000; i>0; i--)
    x[i] = x[i] + s
```

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 L.D F0, 0(R1)</td>
<td>1</td>
</tr>
<tr>
<td>stall</td>
<td>2</td>
</tr>
<tr>
<td>ADD.D F4 F0, F2</td>
<td>3</td>
</tr>
<tr>
<td>stall</td>
<td>4</td>
</tr>
<tr>
<td>stall</td>
<td>5</td>
</tr>
<tr>
<td>S.D F4, 0(R1)</td>
<td>6</td>
</tr>
<tr>
<td>S.D F4, 0(R1)</td>
<td>7</td>
</tr>
<tr>
<td>stall</td>
<td>8</td>
</tr>
<tr>
<td>BNE R1, R2, L1</td>
<td>9</td>
</tr>
<tr>
<td>stall</td>
<td>10</td>
</tr>
</tbody>
</table>
Pipeline Scheduling and Loop Unrolling (cont’d)

- **Loop unrolling** optimization: Replicate loop body multiple times, adjusting the loop termination code

```
L1:  L.D     F0, 0(R1)
      ADD.D  F4, F0, F2
      S.D    F4, 0(R1)
      L.D    F6, -8(R1)
      ADD.D  F8, F6, F2
      S.D    F8, -8(R1)
      L.D    F10, -16(R1)
      ADD.D  F12, F10, F2
      S.D    F12, -16(R1)
      L.D    F14, -24(R1)
      ADD.D  F16, F14, F2
      S.D    F16, -24(R1)
      DADDUI R1, R1, #-32
      BNE   R1, R2, L1
      S.D F16, 8(R1)
```

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F0, 0(R1)</td>
<td>1</td>
</tr>
<tr>
<td>L.D F6, -8(R1)</td>
<td>2</td>
</tr>
<tr>
<td>L.D F10, -16(R1)</td>
<td>3</td>
</tr>
<tr>
<td>L.D F14, -24(R1)</td>
<td>4</td>
</tr>
<tr>
<td>ADD.D F4, F0, F2</td>
<td>5</td>
</tr>
<tr>
<td>ADD.D F8, F6, F2</td>
<td>6</td>
</tr>
<tr>
<td>ADD.D F12, F10, F2</td>
<td>7</td>
</tr>
<tr>
<td>ADD.D F16, F14, F2</td>
<td>8</td>
</tr>
<tr>
<td>S.D F4, 0(R1)</td>
<td>9</td>
</tr>
<tr>
<td>S.D F4, -8(R1)</td>
<td>10</td>
</tr>
<tr>
<td>DADDUI R1, R1, #32</td>
<td>11</td>
</tr>
<tr>
<td>S.D F12, 16(R1)</td>
<td>12</td>
</tr>
<tr>
<td>BNE R1, R2, L1</td>
<td>13</td>
</tr>
<tr>
<td>S.D F16, 8(R1)</td>
<td>14</td>
</tr>
</tbody>
</table>
Loop Unrolling for Superscalar Processors

- Unroll loop 5 times to avoid extra 1-cycle delays in 2-way SS

<table>
<thead>
<tr>
<th>L1: L.D</th>
<th>Integer Instruction</th>
<th>FP Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0, 0(R1)</td>
<td>L.D F0, 0(R1)</td>
<td>1</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F4, F0, F2</td>
<td>L.D F6, -8(R1)</td>
</tr>
<tr>
<td>S.D</td>
<td>F4, 0(R1)</td>
<td>2</td>
</tr>
<tr>
<td>L.D</td>
<td>F6, -8(R1)</td>
<td>L.D F10, -16(R1)</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F8, F6, F2</td>
<td>ADD.D F4, F0, F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F8, -8(R1)</td>
<td>3</td>
</tr>
<tr>
<td>L.D</td>
<td>F10, -16(R1)</td>
<td>L.D F14, -24(R1)</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F12, F10, F2</td>
<td>ADD.D F8, F6, F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F12, 0(R1)</td>
<td>4</td>
</tr>
<tr>
<td>L.D</td>
<td>F14, -24(R1)</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F16, F14, F2</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F16, -24(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F18, -32(R1)</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F20, F18, F2</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F20, -32(R1)</td>
<td></td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1, R1, #-40</td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>R1, R2, L1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S.D F20, 8(R1)</td>
<td></td>
</tr>
</tbody>
</table>

Loop unrolling: 10 to 3.5 cycles/iteration
SS: 3.5 cycles/iteration to 2.4 (1.5 times improvement)
Dynamic Scheduling in Superscalar Processors

• How to extend Tomasulo’s algorithm?

• General solution:
  
  – Allow issue stage to work faster than rest of architecture
    • Achieved by both pipelining and widening the issue logic
    • Instructions issued, reservation stations allocated in order

  – Rest of the design already supports overlapped execution

  – Need wider CDB to store multiple results/cycle
    • Need to allow multiple instruction commits per clock cycle
Limits of Superscalar Processors

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations
  - No hazards

- Need more instructions to issue at the same time to get improved performance
  - However, greater difficulty of decode and issue
  - Even 2-way superscalar => examine 2 opcodes, 6 register specifiers, and decide if 1 or 2 instructions can issue

- Issue rates of modern processors vary between 2—8 instructions/cycle

- Motivation for VLIW and EPIC processors …
VLIW/EPIC Architectures

• Very Long Instruction Word (VLIW)
  – processor can initiate multiple operations per cycle

  \[
  \begin{align*}
  r1 &= L r4 \\
  r2 &= Add \ r1, M \\
  f1 &= Mul \ f1, f2 \\
  r5 &= Add \ r5, 4
  \end{align*}
  \]

  – Specified completely by the compiler (unlike superscalar machines)
  – Hardware is simple: issues the packet given it by the compiler

• Explicitly Parallel Instruction Computing (EPIC)
  – VLIW + new features
    • predication, rotating registers, speculations, etc.

• More later about compiling for VLIW/EPIC
Studies of the Limitations of ILP

- **Is there that much parallelism in programs?**
- Start off with a hardware model of an ideal processor
  1. **Register renaming** – infinite virtual registers and all WAW & WAR hazards are avoided
  2. **Branch prediction** – perfect; no mispredictions
  3. **Jump prediction** – all jumps perfectly predicted
  2 and 3 => no control dependencies == machine with perfect speculation == an unbounded buffer of instructions available for execution
  4. **Memory-address alias analysis** – addresses are known and a load can be moved before a store provided addresses not equal
  1 and 4 => only true data dependencies
- 1 cycle latency for all instructions
- Perfect caches (loads and stores complete in one cycle)
ILP Limit for Six SPEC92 Benchmarks

- Fair bit of instruction-level parallelism, but how much of this stems from the ideal nature of assumptions
  - Infinite registers, perfect jump/branch prediction, perfect alias analysis
More Realistic Hardware: Limiting the Instruction Window

- The set of instructions that is examined for simultaneous executions is called the window
  - Limiting factor: operand checking
    - Scales as $\text{# of inst. completing/cycle} \times \text{window size} \times \text{# operands/inst.}$
  - FP programs have loop level parallelism which benefit from large window
More Realistic Hardware: Branch Impact
Instr. window = 2000, issue width = 64

Perfect  Tournament (Adaptive 2-bit local and 2-bit correlated; 8K entries)
2-bit 512 entries  Static  Best of always
None  Taken or Not taken

Fortran Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>gcc</th>
<th>espresso</th>
<th>li</th>
<th>fppp</th>
<th>doducd</th>
<th>tomcatv</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC</td>
<td>35</td>
<td>41</td>
<td>16</td>
<td>61</td>
<td>58</td>
<td>60</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>12</td>
<td>10</td>
<td>48</td>
<td>46</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>45</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>15</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More Realistic HW: Register Impact
Instr. window = 2000, issue width = 64, bpred = 8K adaptive

Number of renaming registers

IPC

Fortran Programs
Beyond the Limits of the Study

- More aggressive optimizations
  - Address value prediction and speculation (for memory accesses)
    - Can help achieve results similar to near-perfect alias analysis
  - Speculating on multiple paths
    - Reduces recovery costs (hopefully some path is useful), and exposes more ILP

- Even perfect model has some limitations
  - WAR and WAW hazards through memory
    - Can arise due to reuse of stack locations
  - Unnecessary dependences (i.e., compilers can do better than assumed)
    - E.g., dependence on loop control variable can be eliminated by loop unrolling
  - Overcoming the data flow limit
    - Recent idea: Value Prediction
    - Speculate that a register will have a certain value, and then recover if this speculation turns out to be false
      - Can speculate both data values and address values (for alias elimination)
A Different Perspective

• So far: Parallelism among instructions in a single thread of control

• What if we *interleave* instructions from multiple threads of control?
  – These instructions are independent (modulo thread synchronization)
    • Different register sets per thread
  – Overall program finishes earlier
    • Note that behavior of a single thread has not been improved

• **Multithreading** (later)
Compiling for VLIW/EPIC Processors

Next Week