Computer Systems Organization

CSE 521/560M
Lecture 8
Prof. Patrick Crowley

Plan for Today

• Announcements
  – Office hours by appointment this Friday
  – No class on Monday
  – No office hours Tuesday; send questions via email
  – Guest lecture on Wednesday

• Questions

• Today’s discussion
Topics

- ILP concepts
- Taxonomy of dependences and hazards
- Tomasulo’s algorithm

Using ILP to Improve CPI

- Recall:
  Pipeline CPI = Ideal CPI + CPI from stalls
- Ideal CPI = 1 for single issue machine
  - Ideal CPI < 1 if multiple issue (more ILP)
- ILP will increase structural stalls
- ILP makes reduction in other stalls even more important
  - A “bubble” costs more than the loss of a single instruction
Where Can We Optimize?

- CPI contributed by data hazards can be decreased by
  - Compiler optimizations (load scheduling, dependence analysis, software pipelining, trace scheduling)
  - Forwarding
  - Register renaming
- CPI contributed by control stalls can be decreased by
  - Compiler optimizations (static b.p., loop unrolling)
  - Dynamic branch prediction
  - Speculative execution

Data Dependences

- Instruction $j$ dependent on $i$ if
  - $O_i \cap I_j \neq \emptyset$
  - Instruction $j$ dependent on $k$ and $k$ dependent on $i$
- Dependence is a program property
- Hazards (RAW in this case) and their (partial) removals is a pipeline organization property
Name Dependence

- Anti-dependence
  - $O_j \cap I_j \neq \emptyset$
  - At the instruction level, this is WAR hazard if instruction $j$ finishes first
- Output dependence
  - $O_i \cap O_j \neq \emptyset$
  - At the instruction level, this is a WAW if instruction $j$ finishes first
- In both cases, not a true dependence but a *naming* problem
  - Register renaming fixes this

Control Dependences

- Branches restrict the scheduling of instructions
- Speculation must be:
  - Safe (cannot create new exceptions)
  - Legal (cannot change the eventual outcome)
- Speculation can be implemented by:
  - Compiler (static branch prediction, loop unrolling)
  - Hardware (dynamic b.p.)
  - Both (b.p., conditional operations)
Tomasulo’s Algorithm

- Our second style of dynamic execution
- Weakness in scoreboard:
  - Centralized control
  - No forwarding (more RAW than needed)
- Tomasulo’s algorithm as it appeared in the IBM 360/91
  - Control decentralized at each functional unit
  - Forwarding
  - Concept and implementation of renaming registers that eliminates WAR and WAW hazards

Reservation Stations

- Each functional unit has a set of buffers
  - Keep operands and function to perform
  - Operands can be values or names of units that will produce the value (register renaming) with appropriate flags
- Operands arrive at reservation station as they are ready
- When both operands have values, functional unit can execute on that pair of operands
- When a functional unit computes a result, it broadcasts its name and the value
  - Might not store a result in a real register
Hazards with Tomasulo’s Solution

- Structural hazards
  - No free reservation station (stall at issue time)
- RAW hazard
  - Stall at execution time (e.g., wait for operands)
- No WAR or WAW hazards

Sample Machine
Steps

1. Issue
   - Check for structural hazard (no free reservation station or no free load-store buffer for a memory operation)
   - Rename registers if needed

2. Execute
   - If one or more operands is not ready, monitor the bus for broadcast of a result
   - When both operands have values, execute

3. Write result
   - Broadcast name of the unit and value computed (into registers and/or res. stations) and store values to mem

Implementation

- All registers (except load buffers) contain either a tag, $Q$, indicating which functional unit will compute its contents or a value
- The tag (or name) can be:
  - Zero (or special pattern) meaning that we have a value
  - The name of a load buffer
  - The name of a functional unit
- A reservation station consists of:
  - The operation to be performed
  - 2 pairs (value, tag): $(V_j, Q_j)(V_k, Q_k)$,
  - An address
  - A flag indicating whether the accompanying functional unit is busy or not
### Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue</th>
<th>Execute</th>
<th>Write Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F,34(03)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L.D F2,45(03)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MUL.D F0,F2,F4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SUB.D F6,F2,F6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIV.D F10,F0,F6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Reservation stations

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load2</td>
<td>yes</td>
<td>Load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td>yes</td>
<td>SUB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>yes</td>
<td>ADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mul1</td>
<td>yes</td>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mul2</td>
<td>yes</td>
<td>DEV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 3.3* Reservation stations and register tags shown when all of the instructions have issued, but only the first load instruction has completed and written its result to the CDB. The second load has completed effective address calculation, but it is waiting on the memory unit. We use the array Reg[] to refer to the register file and the array Mem[] to refer to the memory. Remember that an operand is specified by either a Q field or a V field at any time. Notice that the MUL instruction, which has a Write hazard at the WB stage, has issued and could complete before the DIV instruction.

### Example (cont’d)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue</th>
<th>Execute</th>
<th>Write Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F6,34(82)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L.D F2,45(83)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MUL.D F0,F2,F4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SUB.D F6,F2,F6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIV.D F10,F0,F6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Reservation stations

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mul1</td>
<td>yes</td>
<td>MUL</td>
<td></td>
<td>Mem[34 + Reg[R3]]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mul2</td>
<td>yes</td>
<td>DEV</td>
<td>Mem[34 + Reg[R2]]</td>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Register status

<table>
<thead>
<tr>
<th>Field</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>F14</th>
<th>F16</th>
<th>F18</th>
<th>F20</th>
<th>F22</th>
<th>F24</th>
<th>F26</th>
<th>F28</th>
<th>F30</th>
<th>Q0</th>
<th>Mult1</th>
<th>Mult2</th>
</tr>
</thead>
</table>

*Figure 3.4* Multiply and divide are the only instructions not finished.
Other Issues

- So far, we have assumed no load/store dependences
  - Load/store buffers must keep their addresses
  - On load, check if the address doesn’t appear in the store buffer. If so, get the value/tag from there (then load buffers have tags…)
  - Or perhaps use load/store functional units
- The IBM 360/91 didn’t, but perhaps add
  - Additional set of registers for more renaming
  - Reorder buffer (for registers with results that cannot yet be committed)
- Exceptions
  - Cannot create ‘new’ exceptions

Looking Ahead

- Not much parallelism available with a basic block
- Must find it between basic blocks
- Branch prediction accuracy is critical for good performance
- Why not issue more than one instruction per cycle?
Assignment

- Readings
  - For Monday
    • H&P: 3.7-3.9
    • Ash: Read Chapter 7
  - For Wednesday
    • Catch up and review