Plan for Today

• Questions
• Dynamic scheduling discussion
• Assignment
• HW2 discussion
Topics

• ILP concepts
• Taxonomy of dependences and hazards
• Tomasulo’s algorithm
Using ILP to Improve CPI

• Recall:
  \[ \text{Pipeline CPI} = \text{Ideal CPI} + \text{structural, data and control stalls} \]

• Ideal CPI = 1 for single issue machine
  – Ideal CPI < 1 if multiple issue (more ILP)

• Reducing Ideal CPI makes stalls more critical
  – As we will see later

• Increased 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
  – $O_i \cap I_j \neq \emptyset$

• Dependence is a program property

• Hazards (RAW in this case) and their (partial) removals is a pipeline organization property
Name Dependence

• Anti-dependence
  – $O_j \cap I_i \neq \emptyset$
  – 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

• Branch speculation must be:
  – Safe (cannot create new exceptions)
  – Legal (cannot change the eventual program outcome)

• Speculation can be implemented by:
  – Compiler (static branch prediction, loop unrolling)
  – Hardware (dynamic b.p.)
  – Both (b.p., conditional operations)

• *In following example, we assume a single basic block*
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 IBM 360/91
  – Control decentralized at each functional unit
  – Forwarding
  – Concept and implementation of renaming registers that eliminates WAR and WAW hazards
Sample Machine
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
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 (ignore for now)

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 memory
Implementation

• All registers (except load buffers) contain either a tag, $Q_i$, 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
Tomasulo Example

Time $i + 12$: MUL.D enters Write Result

<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(R2)</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>L.D F2,45(R3)</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 F8,F6,F2</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>DIV.D F10,F0,F6</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ADD.D F6,F8,F2</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>Load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>45+Regs[R3]</td>
</tr>
<tr>
<td>Add1</td>
<td>no</td>
<td>SUB</td>
<td>Mem[45+Regs[R3]]</td>
<td>Mem[34+Regs[R2]]</td>
<td>Load2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>no</td>
<td>ADD</td>
<td>Regs[F8]</td>
<td>Mem[45+Regs[R3]]</td>
<td>Add2</td>
<td>Load2</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>Mult1</td>
<td>yes</td>
<td>MUL</td>
<td>Mem[45+Regs[R3]]</td>
<td>Regs[F4]</td>
<td>Load2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>yes</td>
<td>DIV</td>
<td>Mem[34+Regs[R2]]</td>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Register Status

| Qi | F0 | F2 | F4 | F6 | F8 | F10 | F12 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mult1</td>
<td>Load2</td>
<td></td>
<td>Add2</td>
<td>Add1</td>
<td>Mult2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Latencies:
- Add/Sub: 2
- Mult: 10
- Div: 40

Find:
Status tables when MUL.D is ready to move to Write Result state
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
Other Issues

• So far, we have assumed no load/store dependences  
  – Load/store buffers must keep their addresses  
  – On load, check that 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 within a basic block
  – Must find it across basic blocks
  – Branch prediction accuracy is critical for good performance

• Why not issue more than one instruction per cycle?
Assignment

• HW2 due 10/7

• Readings
  – For Monday
    • Commentary: A Comparison of Two Pipeline Organizations
  – For Wednesday
    • H&P: 2.3, 2.6-2.7
    • Skim V&L: Chapter 6
About HW2

• You will model a simple processor in VHDL
  – Considerably more to do than in HW1, but the same amount of time!

• You will work in groups of 3 (mostly)

• Why groups?
  – Gentler introduction for some
  – Practice “working in groups”
### HW2 Groups

<table>
<thead>
<tr>
<th>Group</th>
<th>Member 1</th>
<th>Member 2</th>
<th>Member 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>David Lu</td>
<td>Michael Schultz</td>
<td>Austin Abrams</td>
</tr>
<tr>
<td>2</td>
<td>Yong Fu</td>
<td>Greg Galloway</td>
<td>Christopher Thomas</td>
</tr>
<tr>
<td>3</td>
<td>Timothy York</td>
<td>Haowei Yuan</td>
<td>Stephen Schuh</td>
</tr>
<tr>
<td>4</td>
<td>Yu-Ying Liang</td>
<td>Abu Sayeed Saifullah</td>
<td>Raphael Njuguna</td>
</tr>
<tr>
<td>5</td>
<td>Cory Flanagan</td>
<td>Jessica Schupp</td>
<td></td>
</tr>
</tbody>
</table>