Advanced Topic: Load Scheduling

- All instructions except for loads are easy in Tomasulo
  - Register inputs only
  - Register renaming captures all dependences
  - Tags tell you exactly when you can execute

- Loads not so easy
  - Must check for older active stores with same address
  - Register renaming doesn’t tell you that
The Data Memory FU

- MOB+D$ = memory FU
  - just like any other FU
  - 2 reg inputs: addr, data_in
  - 1 reg output: data_out

- what actually happens?
Store/Load Dispatch

- allocate MOB entry (tail)
- indicate store/load
- remember MOB# in RS
Store/Load Retire

- free MOB entry (head)
- load?
  - done
- store?
  - address + value to D$/TLB
Store Execute

- address + value to MOB
- can be done separately
- e.g., PentiumII: store → 2 μops

Diagram:

- MOB
- Data_in, Data_out
- Address, Value
- L/S
- D$/TLB

© 2009 by Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti
ECE 252 / CPS 220 Lecture Notes
Dynamic Scheduling II
Load Execute

- address to D$
  - read value
- address to MOB
  - compare to older store addr
  - no match $\Rightarrow$ D$ value
  - match $+$value $\Rightarrow$ MOB value
    - multiple matches? youngest
    - forwarding or bypassing
    - same latency as D$ hit (why?)
- match $-$value $\Rightarrow$ stall
  - try executing load again later
Memory Disambiguation Problem

at load execution

• what if older store address unknown?
  • how to determine match?
  • can’t determine, have to guess
  • called “memory disambiguation”

• what if older load address unknown?
  • who cares
Memory Disambiguation Alternatives

• **conservative**: loads in-order with respect to stores
  • don’t know address? assume match, wait
  + pretty simple
  – many unnecessary waits on non-matching stores

• **opportunistic**: out-of-order loads
  • don’t know address? assume no match, go
  + higher performance (most cases are not matching)
  – *mis-speculations*: went too soon? recover (complex+expensive!)

• **selective**: combination of conservative and opportunistic
  • start out opportunistic
  • load mis-speculation? remember PC in table, next time conservative
  + pretty accurate prediction ⇒ pretty good performance
<table>
<thead>
<tr>
<th></th>
<th>Pentium II</th>
<th>Pentium4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Clock</td>
<td>450MHz (Xeon)</td>
<td>2.4 (4.8 internal) GHz</td>
</tr>
<tr>
<td>Pipeline stages</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>512 entry local</td>
<td>2K entry hybrid</td>
</tr>
<tr>
<td>BTB</td>
<td>512 entries</td>
<td>4K entries</td>
</tr>
<tr>
<td>I$</td>
<td>16KB</td>
<td>64KB T$ + 8KB I$</td>
</tr>
<tr>
<td>D$</td>
<td>16KB</td>
<td>8KB</td>
</tr>
<tr>
<td>L2</td>
<td>512KB–2MB</td>
<td>256KB–2MB</td>
</tr>
<tr>
<td>Fetch Width</td>
<td>16 bytes</td>
<td>3 μops (16 bytes on miss)</td>
</tr>
<tr>
<td>Rename/Retire Width</td>
<td>3 μops</td>
<td>3 μops</td>
</tr>
<tr>
<td>Execute Width</td>
<td>5 μops</td>
<td>7 μops (X2)</td>
</tr>
<tr>
<td>Reservation Stations</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>ROB Size</td>
<td>40</td>
<td>128</td>
</tr>
<tr>
<td>Register Renaming</td>
<td>P6-style</td>
<td>MIPS R10K-style</td>
</tr>
<tr>
<td>Memory Disambiguation</td>
<td>Conservative</td>
<td>Predictor-Based</td>
</tr>
<tr>
<td>Anything else?</td>
<td>No</td>
<td>Multithreading</td>
</tr>
</tbody>
</table>
Limits of Instruction Level Parallelism (ILP)

- no need to build bigger superscalar if there isn’t ILP
  - ultimately, how much ILP is there?

- ILP study
  - assume perfect/infinite hardware
  - successively refine to more realistic hardware
  - e.g.: [Wall’88], [Wilson+Lam’92]

- some (surprising) results
  - perfect/infinite/single-cycle everything: int ILP: >50!, FP ILP: >150!!
  - on actual machines: int ILP: ~2!!, FP ILP: ~3!
  - culprits? branch prediction, memory latency, finite ROB, issue width
  - read on your own (H+P Chapter 3)
Summary

- dynamic scheduling + precise state + speculation
  - re-order buffer (ROB)
  - WB ⇒ CM (out-of-order) + RT (in-order)
  - P6 vs. R10K
  - how is recovery done?

- load scheduling using
  - MOB+D$, bypassing values from MOB
  - memory disambiguation

- ILP limits
  - read on your own (H&P chapter 3)

next up: static (compiler) exploitation of ILP