Module 2 Wrap-up and OoO

Colin Schmidt
CS 152 Section 7
3/3/2016
Agenda

• PS2 Review
• Quiz 2 Prep
• Out of Order Execution
• Lab 3 Info
Q1: Caches

- 1.{A,B}: Table questions?
- 1.C: Modeling a cache questions?
- 1.D: Average latency questions?
Q2: Coding for caches

• 2.\{A,B,C\}: Calculate cache misses questions?
  – Cache access pattern
Q3: New Cache Design

• 3.A: Same cycle time as before questions?
• 3.B: AMAT questions?
• 3.C: 3C’s questions?
• 3.D: Virtual aliasing questions?
• 3.E: Preventing aliasing questions?
Q4 Victim Cache

• 4.A: Access time questions?
• 4.B: Cache behavior questions?
• 4.C: AMAT questions?
Q5 3C’s

• doubling assoc?
• halving line size?
• doubling sets?
• adding prefetching
Q6 Memory Hierachy

- Hit Time
- Miss Rate
- Miss Penalty
Topics

• Caches
  – 3 C’s
  – associativity
  – replacement policy
  – write policy
  – access time
  – AMAT
  – Writing good code
Translation/Protection

• Virtual Memory
• TLB
  – Cache interaction
  – Aliasing
• Page Tables
  – Entries
  – Organization
• Protection
Complex Pipelines

• Adding long latency operations is what causes complications
• Why wasn’t this a problem with loads
• How do we prevent this?
Data Hazards: An Example

$\begin{array}{llll}
I_1 & \text{FDIV.D} & f_6, & f_6, & f_4 \\
I_2 & \text{FLD} & f_2, & & 45(x3) \\
I_3 & \text{FMUL.D} & f_0, & f_2, & f_4 \\
I_4 & \text{FDIV.D} & f_8, & f_6, & f_2 \\
I_5 & \text{FSUB.D} & f_{10}, & f_0, & f_6 \\
I_6 & \text{FADD.D} & f_6, & f_8, & f_2 \\
\end{array}$

RAW Hazards
WAR Hazards
WAWW Hazards
Scoreboard for In-order Issues

Busy[FU#] : a bit-vector to indicate FU’s availability.
(FU = Int, Add, Mult, Div)
These bits are hardwired to FU's.

WP[reg#] : a bit-vector to record the registers for which writes are pending.
These bits are set by Issue stage and cleared by WB stage

Issue checks the instruction (opcode dest src1 src2) against the scoreboard (Busy & WP) to dispatch

FU available?       Busy[FU#]
RAW?               WP[src1] or WP[src2]
WAR?               cannot arise
WAW?               WP[dest]
## Scoreboard Dynamics

<table>
<thead>
<tr>
<th>Functional Unit Status</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int(1)</td>
<td></td>
</tr>
<tr>
<td>Add(1)</td>
<td></td>
</tr>
<tr>
<td>Mult(3)</td>
<td></td>
</tr>
<tr>
<td>Div(4)</td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t0</th>
<th>$I_1$</th>
<th>f6</th>
<th>f6</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>$I_2$</td>
<td>f2</td>
<td>f6</td>
<td>f6, f2</td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td>f6</td>
<td>f2</td>
<td>f6, f2</td>
</tr>
<tr>
<td>t3</td>
<td>$I_3$</td>
<td>f0</td>
<td>f6</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td>f0</td>
<td>f6</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t5</td>
<td>$I_4$</td>
<td>f0</td>
<td>f8</td>
<td>f0, f8</td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td>f8</td>
<td>f0</td>
<td>f0, f8</td>
</tr>
<tr>
<td>t7</td>
<td>$I_5$</td>
<td>f10</td>
<td>f8</td>
<td>f8, f10</td>
</tr>
<tr>
<td>t8</td>
<td></td>
<td>f8</td>
<td>f10</td>
<td>f8, f10</td>
</tr>
<tr>
<td>t9</td>
<td></td>
<td>f8</td>
<td>f8</td>
<td></td>
</tr>
<tr>
<td>t10</td>
<td>$I_6$</td>
<td>f6</td>
<td></td>
<td>f6</td>
</tr>
<tr>
<td>t11</td>
<td></td>
<td>f6</td>
<td>f6</td>
<td></td>
</tr>
</tbody>
</table>

$I_1$: FDIV.D

$I_2$: FLD

$I_3$: FMULT.D

$I_4$: FDIV.D

$I_5$: FSUB.D

$I_6$: FADD.D

2/29/2016

CS152, Spring 2016
## Issue Limitations: In-Order and Out-of-Order

<table>
<thead>
<tr>
<th></th>
<th>Instruction</th>
<th>Fields</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FLD</td>
<td>f2, 34(x2)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>FLD</td>
<td>f4, 45(x3)</td>
<td>long</td>
</tr>
<tr>
<td>3</td>
<td>FMULT.D</td>
<td>f6, f4, f2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>FSUB.D</td>
<td>f8, f2, f2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>FDIV.D</td>
<td>f4', f2, f8</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>FADD.D</td>
<td>f10, f6, f4'</td>
<td>1</td>
</tr>
</tbody>
</table>

**In-order:**  
1 (2,1) . . . . . . 2 3 4 4 3 5 . . . 5 6 6

**Out-of-order:**  
1 (2,1) 4 4 5 . . . 2 (3,5) 3 6 6

Any antidependence can be eliminated by renaming.  
*(renaming => additional storage)*

Can it be done in hardware? **yes!**
Register Renaming

- Decode does register renaming and adds instructions to the issue-stage instruction reorder buffer (ROB)
  \[\Rightarrow\text{renaming makes WAR or WAW hazards impossible}\]

- Any instruction in ROB whose RAW hazards have been satisfied can be issued.
  \[\Rightarrow\text{Out-of-order or dataflow execution}\]
IBM 360/91 Floating-Point Unit
R. M. Tomasulo, 1967

Distribute instruction templates by functional units

Common bus ensures that data is made available immediately to all the instructions waiting for it.
Match tag, if equal, copy value & set presence “p”.

Load buffers (from memory)
Store buffers (to memory)

Adder
Mult

Instructions

Regfile

Floating-Point Unit
Phases of Instruction Execution

Fetch: Instruction bits retrieved from cache.

Decode: Instructions dispatched to appropriate issue-stage buffer.

Execute: Instructions and operands issued to execution units. When execution completes, all results and exception flags are available.

Commit: Instruction irrevocably updates architectural state (aka “graduation”).
Unified Physical Register File
(MIPS R10K, Alpha 21264, Intel Pentium 4 & Sandy Bridge)

- Rename all architectural registers into a single *physical* register file during decode, no register values read
  - $x1 \rightarrow P1$
- Functional units read and write from single unified register file holding committed and temporary registers in execute
- Commit only updates mapping of architectural register to physical register, no data movement

```
<table>
<thead>
<tr>
<th>Decode Stage</th>
<th>Unified Physical Register File</th>
<th>Committed Register Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read operands at issue</td>
<td>Write results at completion</td>
<td></td>
</tr>
<tr>
<td>Register Mapping</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Functional Units</td>
<td></td>
</tr>
</tbody>
</table>
```

3/2/2016
## Physical Register Management

### Rename Table

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P8</td>
<td></td>
<td>P7</td>
<td></td>
<td></td>
<td>P5</td>
<td>P6</td>
</tr>
</tbody>
</table>

### Physical Regs

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x6&gt;</td>
<td>&lt;x7&gt;</td>
<td>&lt;x3&gt;</td>
<td>&lt;x1&gt;</td>
</tr>
</tbody>
</table>

### Free List

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
</table>

### ROB

<table>
<thead>
<tr>
<th>use</th>
<th>ex</th>
<th>op</th>
<th>p1</th>
<th>PR1</th>
<th>p2</th>
<th>PR2</th>
<th>Rd</th>
<th>LPRd</th>
<th>PRd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **ld x1, 0(x3)**
- **addi x3, x1, #4**
- **sub x6, x7, x6**
- **add x3, x3, x6**
- **ld x6, 0(x1)**

(LPRd requires third read port on Rename Table for each instruction)
**Lifetime of Physical Registers**

- Physical regfile holds committed and speculative values
- Physical registers decoupled from ROB entries (*no data in ROB*)

```
ld x1, (x3)
addi x3, x1, #4
sub x6, x7, x9
add x3, x3, x6
ld x6, (x1)
add x6, x6, x3
sd x6, (x1)
ld x6, (x11)
```

```
ld P1, (Px)
addi P2, P1, #4
sub P3, Py, Pz
add P4, P2, P3
ld P5, (P1)
add P6, P5, P4
sd P6, (P1)
ld P7, (Pw)
```

**Rename**

When can we reuse a physical register?

*When next write of same architectural register commits*
Lab 3

• Not ready yet...
• Later this week/weekend
• Info at http://ccelio.github.io/riscv-boom-doc/
Questions