CS 152, Spring 2011
Section 2

Christopher Celio
About Me

• Christopher Celio
• celio @ eecs
• Office Hours: Tuesday 1–2pm, 751 Soda
• Any questions?
Lab 1

- Account forms
- New Image: cashew

```bash
celio@t7400-1:~/simics-workspace$./simics targets/sunfire/cashew-common.simics
```
• Questions?
PSet 1: Precise Exceptions

• Definition
• Single commit point
  – All state committed before exception
  – No state committed after exception
Newsgroup
ucb.class.cs152
news.csua.berkeley.edu

Can use EECS web UI:
http://inst.eecs.berkeley.edu/webnews
PSet 1: Iron Law

- Reduce number of registers in the ISA

- Adding 16-bit versions of the most common instructions in MIPS (normally 32-bits in length) to the ISA (i.e., make MIPS a variable length ISA)

- For a given CISC ISA, changing the implementation of the micro-architecture from a microcoded engine to a RISC pipeline (with a CISC-to-RISC decoder on the front-end)
Clock period can be reduced by dividing the execution of an instruction into multiple cycles

\[ t_C > \max \{ t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW} \} \] \( ( = t_{DM} \text{ probably}) \)

However, CPI will increase unless instructions are pipelined
Fully Bypassed Datapath

Is there still a need for the stall signal?

\[
\text{stall} = (rs_D = ws_E) \cdot (\text{opcode}_E = \text{LW}_E) \cdot (ws_E \neq 0) \cdot \text{re1}_D + (rt_D = ws_E) \cdot (\text{opcode}_E = \text{LW}_E) \cdot (ws_E \neq 0) \cdot \text{re2}_D
\]
Mem–ALU Pipeline Question

- Re-arrange EXE and MEM stages

IF ➔ DEC ➔ EXE ➔ MEM ➔ WB

IF ➔ DEC ➔ MEM ➔ EXE ➔ WB

- only support register indirect addressing
- Can now implement mem–reg instructions
  - **CAdd** rd,rs0,rs1
  - rd ← M[rs0] + rs1
Mem–ALU Pipeline Question

- Re-arrange EXE and MEM stages

- Assume fully bypassed (and correct)
Pipeline Question Part A

• Re-arrange EXE and MEM stages

IF → DEC → EXE → MEM → WB

IF → DEC → MEM → EXE → WB

• Provide a sequence that:
  – causes a bubble with the old pipeline
  – no bubble with the new pipeline
Pipeline Question Part A

- Fixes the Load–use dependency

```
LW r2, 0(r1)
ADD r4, r2, r3
```
Pipeline Question Part B

Give a sequence that would cause a bubble in the new pipeline, but not the old
Pipeline Question Part B

An arithmetic instruction followed by a dependent memory access

ADD r2, r1, r5
LW r4, 0(r2)
Pipeline Question Part C

• Compare the two pipelines
  – Which would you recommend?
Pipeline Question Part D

• How can we continue to provide full MIPS ISA support?
  – as in, reg-imm indirect addressing
  – **LW r2, 8(r3)**
Pipeline Question Part D

• Option A
  – include an adder in decode or mem (increase in cycle time)

• Option B
  – add a address compute stage (increase CPI, b/c more hazards)

• Option C
  – put adder in parallel with mem stage
  – take two cycles on mem access
  – (increase CPI)
Pipeline Question Part D

• Option D
  – Add a bypass from ALU back to MEM
  – same cycle time
  – increases CPI (more hazards, complexity)
Pipeline Question Part E

• What problems might arise when implementing precise exceptions
Pipeline Question Part E

• Where do you draw the commit point?
• How do you handle stores?

• Solution:
  – buffer stores until all earlier instructions have committed
### Double Adder Pipeline Question

#### Figure 2-A. Dual ALU Pipeline

<table>
<thead>
<tr>
<th>IF</th>
<th>ID</th>
<th>EX1</th>
<th>EX2/MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction fetch</td>
<td>Instruction decode and register read</td>
<td>ALU1 execution and address calculation</td>
<td>ALU2 execution and memory access</td>
<td>Writeback to register file</td>
</tr>
</tbody>
</table>

Monday, February 7, 2011
Double Adder Pipeline Part A

```
add r1, r2, r3
lw  r4, 0(r1)
add r5, r4, r6
add r7, r5, r8
add r1, r2, r3
lw  r4, 0(r1)
add r5, r1, r6

ALU1 or ALU2?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU 1</td>
<td></td>
</tr>
<tr>
<td>ALU 2</td>
<td>ALU 2</td>
</tr>
<tr>
<td>ALU 2</td>
<td>ALU 1</td>
</tr>
<tr>
<td>ALU 1</td>
<td>ALU 1</td>
</tr>
</tbody>
</table>
```

Monday, February 7, 2011
<table>
<thead>
<tr>
<th>stall? (yes/no)</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>add r1, r2, r3</td>
<td></td>
</tr>
<tr>
<td>lw r4, 0(r1)</td>
<td></td>
</tr>
<tr>
<td>lw r1, 0(r2)</td>
<td></td>
</tr>
<tr>
<td>add r3, r1, r4</td>
<td></td>
</tr>
<tr>
<td>lw r5, 0(r1)</td>
<td></td>
</tr>
<tr>
<td>lw r1, 0(r2)</td>
<td></td>
</tr>
<tr>
<td>lw r3, 0(r1)</td>
<td></td>
</tr>
<tr>
<td>lw r1, 0(r2)</td>
<td></td>
</tr>
<tr>
<td>sw r1, 0(r3)</td>
<td></td>
</tr>
<tr>
<td>lw r1, 0(r2)</td>
<td></td>
</tr>
<tr>
<td>add r3, r1, r4</td>
<td></td>
</tr>
<tr>
<td>sw r5, 0(r3)</td>
<td></td>
</tr>
<tr>
<td>lw r1, 0(r2)</td>
<td></td>
</tr>
<tr>
<td>add r3, r1, r4</td>
<td></td>
</tr>
</tbody>
</table>

Note that the base address operand for both LW and SW must be available by the end of ID, but the data operand for SW must only be available by the end of EX1.
### Double Adder Pipeline Part B

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Stall? (yes/no)</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>add r1, r2, r3</td>
<td>No</td>
<td>The add (in EX1) uses ALU1 and bypasses its result to the LW (in ID).</td>
</tr>
<tr>
<td>lw r4, 0(r1)</td>
<td>No</td>
<td>The first LW (in EX2/MEM) bypasses its result to the add (in EX1) which will use ALU2, and also to the second LW (in ID).</td>
</tr>
<tr>
<td>add r3, r1, r4</td>
<td>No</td>
<td>The result of the first LW (in EX1) is not available in time for the second LW (in ID), so the second LW must stall.</td>
</tr>
<tr>
<td>lw r5, 0(r1)</td>
<td>Yes</td>
<td>The LW (in EX2/MEM) bypasses its result to the SW (in EX1) in time for it to store the data in EX2/MEM.</td>
</tr>
<tr>
<td>lw r1, 0(r2)</td>
<td>No</td>
<td>The LW (in EX2/MEM) bypasses its result to the add (in EX1) which will use ALU2. But, the result of the add (in EX1) is not available in time for the SW (in ID), so the SW must stall.</td>
</tr>
<tr>
<td>sw r1, 0(r3)</td>
<td>Yes</td>
<td>The LW (in EX2/MEM) bypasses its result to the add (in EX1) which will use ALU2.</td>
</tr>
<tr>
<td>lw r1, 0(r2)</td>
<td>No</td>
<td>The LW (in EX2/MEM) bypasses its result to the add (in EX1) which will use ALU2.</td>
</tr>
</tbody>
</table>

Note that the base address operand for both LW and SW must be available by the end of ID, but the data operand for SW must only be available by the end of EX1.
Questions?

- HW
- Lab
- Class Logistics