Pipeline Hazards

or

Danger! Danger! Danger!

Data Hazards

ADD R1, R2, R3
SUB R4, R5, R1
AND R6, R1, R7
OR R8, R1, R9
XOR R10, R1, R11

Data __________ may result in data __________.
Data Dependence

- Data hazards are caused by data dependences.
- Data dependences, and thus data hazards, come in 3 flavors (not all of which apply to this pipeline).
  - (read-after-write)
  - (write-after-write)
  - (write-after-read)

RAW Hazard

- later instruction tries to read an operand before earlier instruction writes it.
- The dependence:
  - add R1, R2, R3
  - sub R5, R1, R4

- The hazard:
  - add R1, R2, R3
  - sub R5, R1, R4

- RAW hazard is extremely common.

WAW Hazard

- later instruction tries to write an operand before earlier instruction writes it.
- The dependence:
  - add R1, R2, R3
  - sub R1, R2, R4

- The hazard:
  - lw R1, R2, R3
  - sub R1, R2, R4

- WAW hazard possible in a reasonable pipeline, but not in the very simple pipeline we’re assuming.

WAR Hazard

- later instruction tries to write an operand before earlier instruction reads it.
- The dependence:
  - add R1, R2, R3
  - sub R2, R5, R4

- The hazard:
  - add R1, R2, R3
  - sub R2, R5, R4

- WAR hazard is uncommon/impossible in a reasonable (in-order) pipeline.
Solutions?

Dealing with Data Hazards through Forwarding

Forwarding Options

- ADD -> ADD
- ADD -> LW
- ADD -> SW (2 operands)
- LW -> ADD
- LW -> LW
- LW -> SW (2 operands)

(I’m letting ADD stand in for all ALU operations)
More Forwarding

Forwarding and Stalling

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw R1, 0(R2)</td>
<td>IF ID EX WB ID</td>
</tr>
<tr>
<td>sub R4, R1, R6</td>
<td></td>
</tr>
<tr>
<td>and R6, R1, R7</td>
<td></td>
</tr>
<tr>
<td>or R8, R1, R9</td>
<td></td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td></td>
</tr>
<tr>
<td>SW R1, 1000(R2)</td>
<td></td>
</tr>
<tr>
<td>LW R7, 2000(R2)</td>
<td></td>
</tr>
<tr>
<td>ADD R5, R7, R1</td>
<td></td>
</tr>
<tr>
<td>LW R8, 2004(R2)</td>
<td></td>
</tr>
<tr>
<td>SW R7, 2008(R8)</td>
<td></td>
</tr>
<tr>
<td>ADD R8, R8, R2</td>
<td></td>
</tr>
<tr>
<td>LW R9, 1012(R8)</td>
<td></td>
</tr>
<tr>
<td>SW R9, 1016(R8)</td>
<td></td>
</tr>
</tbody>
</table>
Avoiding Pipeline Stalls

lw R1, 1000(R2)
lw R3, 2000(R2)
add R4, R1, R3
lw R1, 3000(R2)
add R6, R4, R1
sw R6, 1000(R2)

• this is a compiler technique called instruction scheduling.

How big a problem are these pipeline stalls?

• 13% of the loads in FP programs
• 25% of the loads in integer programs

Detecting ALU Input Hazards

Inserting Bubbles

• Set all control values in the EX/MEM register to safe values (equivalent to a nop)
• Keep same values in the ID/EX register and IF/ID register
• Keep PC from incrementing
**Adding Datapaths**

**Control Hazards**

- Result from *branch* or *control* ____________
- Instructions are not only dependent on instructions that produce their operands, but also on all previous control flow (branch, jump) instructions that lead to that instruction.

**Old Datapath**

**Branch Hazards**

<table>
<thead>
<tr>
<th>Branch</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>I3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>I4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>
Branch Stall Impact

- If CPI = 1, 30% branch, Stall 3 cycles => new CPI = ????
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- (limited MIPS) branch tests if register = 0 or ≠ 0
- MIPS Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

Branch Hazards

<table>
<thead>
<tr>
<th>Branch</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>correct instruction</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>I4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>I5</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

What We Know About Branches

- more conditional branches than unconditional
- more forward than backward
- 67% of branches taken
- backward branches taken 80%
Four Branch Hazard Alternatives

1. _______ until branch direction clear
   - Problems?

2. Predict Branch Not Taken
   - Execute successor instructions in sequence
   - “Squash” instructions in pipeline if branch actually taken
   - Advantage of late pipeline state update
   - 33% MIPS branches not taken on average
   - PC+4 already calculated, so use it to get next instruction
   - This is what the pipeline is doing, anyway

3. Predict Branch Taken
   - 67% MIPS branches taken on average
   - But haven’t calculated branch target address in this MIPS architecture
     - MIPS still incurs 1 cycle branch penalty
     - Other machines: branch target known before outcome
Fourth Branch Hazard Alternatives -- Delayed Branch

- Define branch to take place *AFTER* a following instruction

  - branch instruction
  - sequential successor₁
  - sequential successor₂
  - .........
  - sequential successorₙ
  - branch target if taken

  Branch delay of length \( n \)

- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- MIPS uses this

Delayed Branch

- Where to get instructions to fill branch delay slot?
  - Before branch instruction
  - From the target address: only valuable when branch taken
  - From fall through: only valuable when branch not taken
  - Cancelling branches allow more slots to be filled

- Compiler effectiveness for single branch delay slot:
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% (60% x 80%) of slots usefully filled

Key Points

- Hard to keep the pipeline completely full
- Data Hazards require dependent instructions to wait for the producer instruction
  - Most of the problem handled with forwarding (bypassing)
  - Sometimes stall still required (especially in modern processors)
- Control hazards require control-dependent (post-branch) instructions to wait for the branch to be resolved
- \( ET = IC \times CPI \times CT \)