Computer Systems Organization

CSE 521/560M
Lecture 6
Prof. Patrick Crowley

Plan for Today

• Announcements:
  – Shortened lecture, ends at 3:10 PM
  – Get started on HW 1
• Questions
• Assignment
• Pipelining discussion 3
Assignment

• HW 1, Due Oct 1 but start today
• Readings
  – For Monday
    • H&P: 3.1-3.4
    • Ash: Skim Chapters 2-4 (syntax), Read Chapter 5
  – For Wednesday
    • H&P: 3.5-3.6
    • Ash: Chapter 6

The Story with ILP So Far

• Pipelining (a good idea)
  – Pipeline CPI = Ideal CPI + CPI from stalls
• 5-stage RISC integer pipeline
  – Must watch for hazards and exceptions, but not too bad
• Multicycle (e.g., floating-point) operations
  – Hazard and exception detection harder, more stalls
• Since instructions execute in fetch order, one stalled instruction stalls all subsequent ones
Dynamic Scheduling

- Idea: re-order instructions in HW to reduce stalls without altering the program outcome
- Implics possibility of
  - Out of order execution
  - Out of order completion
  - Imprecise exceptions (deal with this later)

Example

DIV.D  R1,R2,R3; long latency
ADD.D  R2,R1,R5; stall, RAW on R1
SUB.D  R6,R7,R8; must this wait?
Checking for Issue

- Split ID stage into:
  - **Issue**: decode instructions; check for structural hazards (stall if nec.). Instructions flow in-order.
  - **Read operands**: wait until no data hazards then read operands
- After ID stage, instruction enters EX as before
- Note that WAR hazards may now occur
  
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operands</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV.D</td>
<td>R1, R2, R3</td>
<td>long latency</td>
</tr>
<tr>
<td>ADD.D</td>
<td>R2, R1, R4</td>
<td>stall, RAW on R1</td>
</tr>
<tr>
<td>SUB.D</td>
<td>R4, R7, R8</td>
<td>WAR hazard on R4</td>
</tr>
</tbody>
</table>

Implementations of Dynamic Scheduling

- In order to compute correct results, need to keep track of:
  - Execution pipelines (recall that we now have several)
  - Register usage for read and write
  - Operation completion
- Two major techniques
  - Scoreboard (Seymour Cray for CDC 6600 in 1964)
  - Tomasulo’s algorithm (in IBM 360/91 in 1967)
Scoreboarding Steps

1. Issue
   - The execution unit must be free
   - There should be no WAW hazard
   - If either of these are false, the instruction stalls.
     No further issue is allowed (although fetching may continue if there is an instruction fetch buffer)

Scoreboarding Steps

2. Read Operands
   - When the instruction is issued, the execution unit is reserved (i.e., becomes busy)
   - Operands are read in the execution unit when they are ready (i.e., there is no RAW hazard)

3. Execution
   - One or more cycles depending on functional unit latency
   - When execution completes, the unit notifies the scoreboard that it’s ready to write the result
Scoreboarding Steps

4. Write result
   – Before writing, check for WAR hazards. If one exists, the unit is stalled until all WAR hazards are cleared

5. Delay
   – Because forwarding is not implemented, there should be one unit of delay between writing and reading the same register
   – Similarly, it takes on unit of time between the release of a unit and its next possible occupancy

CDC 6600 Scoreboard Structure
Simplifications and Optimizations

- We have assumed that there could be concurrent updates to the register file
  - Can be solved (dynamically) by grouping execution units together and preventing concurrent writes in the same group
- Opportunities for optimization
  - Forwarding
  - Buffer first-available operand in functional unit rather than waiting for both (reduce WAR hazards)

What the Scoreboard Contains

- Status of each functional unit
  - Free or busy
  - Operation to be performed
  - Names of the result $F_l$ and source $F_j,F_k$ registers
  - Flags $R_j,R_k$ indicating whether the source registers are ready
  - Names $Q_j,Q_k$ of the units (if any) producing values for $F_j,F_k$
- Status of result registers
  - For each $F_l$ the name of the unit (if any) that will produce its contents
- The instruction status
  - Has it been issued, is it in execution, is it ready to write?
Scoreboard Example

![Scoreboard Example Diagram]

Figure A.52 Components of the scoreboard. Each instruction that has issued or is pending is boxed in red.
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue</th>
<th>Read operands</th>
<th>Execution complete</th>
<th>Write result</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F6, 34(R2)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L.D F2, 45(R3)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MUL.D F0, F2, F4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SUB.D F8, F6, F2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIV.D F10, F9, F6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADD.D F6, F8, F2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj</th>
<th>Fk</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Divide</td>
<td>Yes</td>
<td>Div</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td>No</td>
<td>No</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status**

<table>
<thead>
<tr>
<th>FU</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>...</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Divide</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure A.34: Scoreboard tables just before the DIV.D goes to write result. ADD.D was able to complete as soon as DIV.D passed through read operands and got a copy of F6. Only the DIV.D remains to finish.