Last time...

- Scoreboard
  - Data structure that keeps track of dependencies among instructions
- In-order limitations
  - Out-of-order alone cannot solve
- Register renaming
  - Overcoming the restriction caused by the # of registers

Instruction-level Parallelism via Renaming

Latency

1
2
3
4
5
6

In-order: 1 2 3 4 5 6
Out-of-order: 1 2 3 4 5 6

Any antidependence can be eliminated by renaming.
(renaming \Rightarrow additional storage)
Can it be done in hardware? yes!

Register Renaming

- Decode does register renaming and adds instructions to the issue stage reorder buffer (ROB)
\Rightarrow renaming makes WAR or WAW hazards impossible
- Any instruction in ROB whose RAW hazards have been satisfied can be dispatched.
\Rightarrow Out-of-order or dataflow execution

Dataflow Execution

Instruction slot is candidate for execution when:
- It holds a valid instruction ("use" bit is set)
- It has not already started execution ("exec" bit is clear)
- Both operands are available (p1 and p2 are set)

Renaming & Out-of-order Issue

An example

- When are tags in sources replaced by data?
  Whenever an FU produces data
- When can a name be reused?
  Whenever an instruction completes
Data-Driven Execution

- Instruction template (i.e., tag t) is allocated by the Decode stage, which also associates tag with register in regfile.
- When an instruction completes, its tag is deallocated.

Reorder buffer

Replacing the tag by its value is an expensive operation.

Simplifying Allocation/Deallocation

Instruction buffer is managed circularly.

- "exec" bit is set when instruction begins execution.
- When an instruction completes its "use" bit is marked free.
- ptr2 is incremented only if the "use" bit is marked free.

Effectiveness?

Renaming and Out-of-order execution was first implemented in 1969 in IBM 360/91 but did not show up in the subsequent models until mid-Nineties.

Why?

Reasons:
1. Effective on a very small class of programs.
2. Memory latency a much bigger problem.
3. Exceptions not precise!

One more problem needed to be solved.

Precise Interrupts

It must appear as if an interrupt is taken between two instructions (say I_i and I_{i+1})

- the effect of all instructions up to and including I_i is totally complete.
- no effect of any instruction after I_i has taken place.

The interrupt handler either aborts the program or restarts it at I_{i+1}.

CSE 490/590 Administrivia

- No office hours this week.
- Appointment via email if needed.
- Project-related questions → fastest: Safwan or Jangyoung.
- Guest Lecture by Prof. Kris Schindler on Wed.
- Guest lecture by Prof. Tevfik Kosar on Fri.
### Effect on Interrupts

**Out-of-order Completion**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>DIVD</td>
<td>$f_6$, $f_6$, $f_4$</td>
</tr>
<tr>
<td>$I_2$</td>
<td>LD</td>
<td>$f_2$, $45(r_3)$</td>
</tr>
<tr>
<td>$I_3$</td>
<td>MULTD</td>
<td>$f_0$, $f_2$, $f_4$</td>
</tr>
<tr>
<td>$I_4$</td>
<td>DIVD</td>
<td>$f_8$, $f_6$, $f_2$</td>
</tr>
<tr>
<td>$I_5$</td>
<td>SUBD</td>
<td>$f_{10}$, $f_0$, $f_6$</td>
</tr>
<tr>
<td>$I_6$</td>
<td>ADDD</td>
<td>$f_6$, $f_8$, $f_2$</td>
</tr>
</tbody>
</table>

Consider interrupts 1 2 3 4 5 6

**Out-of-order completion**

Precise interrupts are difficult to implement at high speed.
- Want to start execution of later instructions before exception checks finished on earlier instructions.

### Exception Handling

**In-Order Five-Stage Pipeline**

- Hold exception flags in pipeline until commit point (M stage).
- Exceptions in earlier pipe stages override later exceptions.
- Inject external interrupts at commit point (override others).
- If exception at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage.

### Phases of Instruction Execution

- **Fetch:** Instruction bits retrieved from cache.
- **Decode:** Instructions placed in appropriate issue (aka “dispatch”) stage buffer.
- **Issue:** Instructions and operands sent to execution units. When execution completes, all results and exception flags are available.
- **Commit:** Instruction irrevocably updates architectural state (aka “graduation” or “completion”).

### Acknowledgements

- These slides heavily contain material developed and copyright by:
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252