Advanced Computer Architecture- 06CS81
Hardware Based Speculation (unit 3)
Tomasulo algorithm and Reorder Buffer

Tomasulo idea:
1. Have reservation stations where register renaming is possible
2. Results are directly forwarded to the reservation station along with the final registers. This is also called short circuiting or bypassing.

ROB:
1. The instructions are stored sequentially but we have indicators to say if it is speculative or completed execution.
2. If completed execution and not speculative and reached head of the queue then we commit it.

Speculating on Branch Outcomes
- To optimally exploit ILP (instruction-level parallelism) – e.g. with pipelining, Tomasulo, etc. – it is critical to efficiently maintain control dependencies (=branch dependencies)
**Key idea:** Speculate on the outcome of branches (=predict) and execute instructions *as if* the predictions are correct

- of course, we must proceed in such a manner as to be able to recover if our speculation turns out wrong

**Three components of hardware-based speculation**

1. *dynamic branch prediction* to pick branch outcome
2. *speculation* to allow instructions to execute before control dependencies are resolved, i.e., before branch outcomes become known – with ability to undo in case of incorrect speculation
3. *dynamic scheduling*

**Speculating with Tomasulo**

**Key ideas:**

1. *separate execution from completion*: instructions to execute speculatively but no instructions update registers or memory until no more speculative
2. therefore, add a final step – after an instruction is no longer speculative, called *instruction commit* – when it is allowed to make register and memory updates
3. *allow instructions to execute and complete out of order but force them to commit in order*
4. Add hardware called the *reorder buffer (ROB)*, with registers to hold the result of an instruction *between completion and commit*

**Tomasulo’s Algorithm with Speculation: Four Stages**

1. **Issue:** get instruction from Instruction Queue
   - if reservation station and ROB slot free (no structural hazard), control issues instruction to reservation station and ROB, and sends to reservation station operand values (or reservation station source for values) as well as allocated ROB slot number
2. **Execution:** operate on operands (EX)
   - when both operands ready then execute; if not ready, watch CDB for result
3. **Write result:** finish execution (WB)
   - write on CDB to all awaiting units and ROB; mark reservation station available
4. **Commit:** update register or memory with ROB result
   - when instruction reaches head of ROB and results present, update register with result or store to memory and remove instruction from ROB
   - if an incorrectly predicted branch reaches the head of ROB, flush the ROB, and restart at correct successor of branch

**ROB Data Structure**

**ROB entry fields**

- Instruction type: branch, store, register operation (i.e., ALU or load)
- State: indicates if instruction has completed and value is ready
- Destination: where result is to be written – register number for register operation (i.e. ALU or load), memory address for store
• branch has no destination result
  • Value: holds the value of instruction result till time to commit

**Additional reservation station field**
  • Destination: Corresponding ROB entry number

**Example**
1. L.D F6, 34(R2)
2. L.D F2, 45(R3)
3. MUL.D F0, F2, F4
4. SUB.D F8, F2, F6
5. DIV.D F10, F0, F6
6. ADD.D F6, F8, F2

The position of Reservation stations, ROB and FP registers are indicated below:

*Assume latencies load 1 clock, add 2 clocks, multiply 10 clocks, divide 40 clocks*

*Show data structures just before MUL.D goes to commit…*

**Reservation Stations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
<th>Dest</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td>yes</td>
<td>MUL</td>
<td>Mem[45+Regs[R3]]</td>
<td>Regs[F4]</td>
<td></td>
<td></td>
<td>#3</td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>yes</td>
<td>DIV</td>
<td>Mem[34+Regs[R2]]</td>
<td></td>
<td>#3</td>
<td></td>
<td>#5</td>
<td></td>
</tr>
</tbody>
</table>

At the time MUL.D is ready to commit only the two L.D instructions have already committed, though others have completed execution.
Actually, the MUL.D is at the head of the ROB – the L.D instructions are shown only for understanding purposes.

#X represents value field of ROB entry number X

**Floating point registers**

<table>
<thead>
<tr>
<th>Field</th>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reorder#</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Busy</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>…</td>
<td>yes</td>
</tr>
</tbody>
</table>
Reorder Buffer

<table>
<thead>
<tr>
<th>Entry</th>
<th>Busy</th>
<th>Instruction</th>
<th>State</th>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no</td>
<td>L.D</td>
<td>Commit</td>
<td>F6</td>
<td>Mem[34+Regs[R2]]</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>L.D</td>
<td>Commit</td>
<td>F2</td>
<td>Mem[45+Regs[R3]]</td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>MUL.D</td>
<td>Write result</td>
<td>F0</td>
<td>#2 \times \text{Regs}[F4]</td>
</tr>
<tr>
<td>4</td>
<td>yes</td>
<td>SUB.D</td>
<td>Write result</td>
<td>F8</td>
<td>#1 – #2</td>
</tr>
<tr>
<td>5</td>
<td>yes</td>
<td>DIV.D</td>
<td>Execute</td>
<td>F10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>yes</td>
<td>ADD.D</td>
<td>Write result</td>
<td>F6</td>
<td>#4 + #2</td>
</tr>
</tbody>
</table>

Example

Loop:

- LD 
- MULD
- SD
- SUBI
- BNEZ

Assume instructions in the loop have been issued twice
Assume L.D and MUL.D from the first iteration have committed and all other instructions have completed
Assume effective address for store is computed prior to its issue
Show data structures
Reorder Buffer

<table>
<thead>
<tr>
<th>Entry</th>
<th>Busy</th>
<th>Instruction</th>
<th>State</th>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no</td>
<td>L.D</td>
<td>F0, 0(R1)</td>
<td>Commit</td>
<td>F0 Mem[0+Regs[R1]]</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>MUL.D</td>
<td>F4, F0, F2</td>
<td>Commit</td>
<td>F4 #1 × Regs[F2]</td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>S.D</td>
<td>F4, 0(R1)</td>
<td>Write result</td>
<td>0 + Regs[R1] #2</td>
</tr>
<tr>
<td>4</td>
<td>yes</td>
<td>DADDUI</td>
<td>R1, R1, #8</td>
<td>Write result</td>
<td>R1 Regs[R1] – 8</td>
</tr>
<tr>
<td>5</td>
<td>yes</td>
<td>BNE</td>
<td>R1, R2, Loop</td>
<td>Write result</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>yes</td>
<td>L.D</td>
<td>F0, 0(R1)</td>
<td>Write result</td>
<td>F0 Mem[#4]</td>
</tr>
<tr>
<td>7</td>
<td>yes</td>
<td>MUL.D</td>
<td>F4, F0, F2</td>
<td>Write result</td>
<td>F4 #6 × Regs[F2]</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>S.D</td>
<td>F4, 0(R1)</td>
<td>Write result</td>
<td>0 + #4 #7</td>
</tr>
<tr>
<td>9</td>
<td>yes</td>
<td>DADDUI</td>
<td>R1, R1, #8</td>
<td>Write result</td>
<td>R1 #4 – 8</td>
</tr>
<tr>
<td>10</td>
<td>yes</td>
<td>BNE</td>
<td>R1, R2, Loop</td>
<td>Write result</td>
<td></td>
</tr>
</tbody>
</table>

Notes

- If a branch is mispredicted, recovery is done by flushing the ROB of all entries that appear after the mispredicted branch
  - entries before the branch are allowed to continue
  - restart the fetch at the correct branch successor
- When an instruction commits or is flushed from the ROB then the corresponding slots become available for subsequent instructions

Advantages of hardware-based speculation:

- able to disambiguate memory references;
- better when hardware-based branch prediction is better than software-based branch prediction done at compile time; maintains a completely precise exception model even for speculated instructions;
- does not require compensation or bookkeeping code;
- main disadvantage:
  - complex and requires substantial hardware resources;