Lecture: Out-of-order Processors

• Topics: more ooo design details, timing, load-store queue
The Alpha 21264 Out-of-Order Implementation

Branch prediction and instr fetch

Instr Fetch Queue

- R1 ← R1 + R2
- R2 ← R1 + R3
- BEQZ R2
- R3 ← R1 + R2
- R1 ← R3 + R2

Decode & Rename

Speculative Reg Map
- R1 → P36
- R2 → P34

Instr Fetch Queue

- Instr 1
- Instr 2
- Instr 3
- Instr 4
- Instr 5
- Instr 6

Reorder Buffer (ROB)

Committed Reg Map
- R1 → P1
- R2 → P2

Register File P1-P64

Issue Queue (IQ)

- P33 ← P1 + P2
- P34 ← P33 + P3
- BEQZ P34
- P35 ← P33 + P34
- P36 ← P35 + P34

ALU

Results written to regfile and tags broadcast to IQ

ALU

ALU
Additional Details

• When does the decode stage stall? When we either run out of registers, or ROB entries, or issue queue entries.

• Issue width: the number of instructions handled by each stage in a cycle. High issue width ➞ high peak ILP.

• Window size: the number of in-flight instructions in the pipeline. Large window size ➞ high ILP.

• No more WAR and WAW hazards because of rename registers – must only worry about RAW hazards.
Branch Mispredict Recovery

• On a branch mispredict, must roll back the processor state: throw away IFQ contents, ROB/IQ contents after branch

• Committed map table is correct and need not be fixed

• The speculative map table needs to go back to an earlier state

• To facilitate this spec-map-table rollback, it is checkpointed at every branch
Waking Up a Dependent

- In an in-order pipeline, an instruction leaves the decode stage when it is known that the inputs can be correctly received, not when the inputs are computed.

- Similarly, an instruction leaves the issue queue before its inputs are known, i.e., wakeup is speculative based on the expected latency of the producer instruction.
### Out-of-Order Loads/Stores

<table>
<thead>
<tr>
<th>Ld</th>
<th>R1 ← [R2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ld</td>
<td>R3 ← [R4]</td>
</tr>
<tr>
<td>St</td>
<td>R5 → [R6]</td>
</tr>
<tr>
<td>Ld</td>
<td>R7 ← [R8]</td>
</tr>
<tr>
<td>Ld</td>
<td>R9 ← [R10]</td>
</tr>
</tbody>
</table>

What if the issue queue also had load/store instructions? Can we continue executing instructions out-of-order?
Memory Dependence Checking

- The issue queue checks for register dependences and executes instructions as soon as registers are ready.
- Loads/stores access memory as well – must check for RAW, WAW, and WAR hazards for memory as well.
- Hence, first check for register dependences to compute effective addresses; then check for memory dependences.
Memory Dependence Checking

- Load and store addresses are maintained in program order in the Load/Store Queue (LSQ)
- Loads can issue if they are guaranteed to not have true dependences with earlier stores
- Stores can issue only if we are ready to modify memory (can not recover if an earlier instr raises an exception)
The Alpha 21264 Out-of-Order Implementation

Branch prediction and instr fetch

Instr Fetch Queue

R1 ← R1+R2
R2 ← R1+R3
BEQZ R2
R3 ← R1+R2
R1 ← R3+R2
LD R4 ← 8[R3]
ST R4 → 8[R1]

Instr 1
Instr 2
Instr 3
Instr 4
Instr 5
Instr 6
Instr 7

Decoding & Rename

R1 → P1
R2 → P2
R3 → P3
R4 → P4
R5 → P5
R6 → P6
R7 → P7

Speculative Reg Map
R1→P36
R2→P34

P33 ← P1+P2
P34 ← P33+P3
BEQZ P34
P35 ← P33+P3
P36 ← P35+P3
P37 ← 8[P35]
P37 → 8[P36]

Issue Queue (IQ)

P37 ← [P35 + 8]
P37 → [P36 + 8]

Reorder Buffer (ROB)

Committed Reg Map
R1→P1
R2→P2

Commit

Register File P1-P64

Results written to regfile and tags broadcast to IQ

Register File

ALU

ALU

ALU

D-Cache

LSQ

ALU
Title

• Bullet