Lecture: Out-of-order Processors

- Topics: more ooo design details, timing, load-store queue
Problem 3

- Show the renamed version of the following code: Assume that you have 36 physical registers and 32 architected registers. When does each instr leave the IQ?

R1 ← R2+R3
R1 ← R1+R5
BEQZ R1
R1 ← R4 + R5
R4 ← R1 + R7
R1 ← R6 + R8
R4 ← R3 + R1
R1 ← R5 + R9
Problem 3

• Show the renamed version of the following code:
   Assume that you have 36 physical registers and 32 architected registers. When does each instr leave the IQ?

\[
\begin{align*}
R1 & \leftarrow R2 + R3 & P33 & \leftarrow P2 + P3 \\
R1 & \leftarrow R1 + R5 & P34 & \leftarrow P33 + P5 \\
BEQZ & \hspace{1em} R1 & BEQZ & \hspace{1em} P34 \\
R1 & \leftarrow R4 + R5 & P35 & \leftarrow P4 + P5 \\
R4 & \leftarrow R1 + R7 & P36 & \leftarrow P35 + P7 \\
R1 & \leftarrow R6 + R8 & P1 & \leftarrow P6 + P8 \\
R4 & \leftarrow R3 + R1 & P33 & \leftarrow P3 + P1 \\
R1 & \leftarrow R5 + R9 & P34 & \leftarrow P5 + P9 \\
\end{align*}
\]
Problem 3

• Show the renamed version of the following code:
  Assume that you have 36 physical registers and 32
  architected registers. When does each instr leave the IQ?

\[
\begin{align*}
R1 & \leftarrow R2 + R3 & P33 & \leftarrow P2 + P3 & \text{cycle } i \\
R1 & \leftarrow R1 + R5 & P34 & \leftarrow P33 + P5 & i + 1 \\
\text{BEQZ } & R1 & \text{BEQZ } P34 & i + 2 \\
R1 & \leftarrow R4 + R5 & P35 & \leftarrow P4 + P5 & i \\
R4 & \leftarrow R1 + R7 & P36 & \leftarrow P35 + P7 & i + 1 \\
R1 & \leftarrow R6 + R8 & P1 & \leftarrow P6 + P8 & j \\
R4 & \leftarrow R3 + R1 & P33 & \leftarrow P3 + P1 & j + 1 \\
R1 & \leftarrow R5 + R9 & P34 & \leftarrow P5 + P9 & j + 2 \\
\end{align*}
\]

Width is assumed to be 4.

\( j \) depends on the \#stages between issue and commit.
• Assume there are 36 physical registers and 32 logical registers, and width is 4

• Estimate the issue time, completion time, and commit time for the sample code
Assumptions

- Perfect branch prediction, instruction fetch, caches

- ADD $\rightarrow$ dep has no stall; LD $\rightarrow$ dep has one stall

- An instr is placed in the IQ at the end of its 5\textsuperscript{th} stage, an instr takes 5 more stages after leaving the IQ (ld/st instrs take 6 more stages after leaving the IQ)
OOO Example

Original code
ADD R1, R2, R3
LD R2, 8(R1)
ADD R2, R2, 8
ST R1, (R3)
SUB R1, R1, R5
LD R1, 8(R2)
ADD R1, R1, R2

Renamed code

IQ
OOO Example

Original code
ADD R1, R2, R3
LD R2, 8(R1)
ADD R2, R2, 8
ST R1, (R3)
SUB R1, R1, R5
LD R1, 8(R2)
ADD R1, R1, R2

Renamed code
ADD P33, P2, P3
LD P34, 8(P33)
ADD P35, P34, 8
ST P33, (P3)
SUB P36, P33, P5

Must wait
## OOO Example

<table>
<thead>
<tr>
<th>Original code</th>
<th>Renamed code</th>
<th>InQ</th>
<th>Iss</th>
<th>Comp</th>
<th>Comm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td>ADD P33, P2, P3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD R2, 8(R1)</td>
<td>LD P34, 8(P33)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R2, R2, 8</td>
<td>ADD P35, P34, 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST R1, (R3)</td>
<td>ST P33, (P3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB R1, R1, R5</td>
<td>SUB P36, P33, P5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD R1, 8(R2)</td>
<td>ADD R1, R1, R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## OOO Example

### Original code

<table>
<thead>
<tr>
<th>Original code</th>
<th>Renamed code</th>
<th>InQ</th>
<th>Iss</th>
<th>Comp</th>
<th>Comm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td>ADD P33, P2, P3</td>
<td>i</td>
<td>i+1</td>
<td>i+6</td>
<td>i+6</td>
</tr>
<tr>
<td>LD R2, 8(R1)</td>
<td>LD P34, 8(P33)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+8</td>
</tr>
<tr>
<td>ADD R2, R2, 8</td>
<td>ADD P35, P34, 8</td>
<td>i</td>
<td>i+4</td>
<td>i+9</td>
<td>i+9</td>
</tr>
<tr>
<td>ST R1, (R3)</td>
<td>ST P33, (P3)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+9</td>
</tr>
<tr>
<td>SUB R1, R1, R5</td>
<td>SUB P36, P33, P5</td>
<td>i+1</td>
<td>i+2</td>
<td>i+7</td>
<td>i+9</td>
</tr>
<tr>
<td>LD R1, 8(R2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD R1, R1, R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## OOO Example

<table>
<thead>
<tr>
<th>Original code</th>
<th>Renamed code</th>
<th>InQ</th>
<th>Iss</th>
<th>Comp</th>
<th>Comm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R1, R2, R3</td>
<td>ADD P33, P2, P3</td>
<td>i</td>
<td>i+1</td>
<td>i+6</td>
<td>i+6</td>
</tr>
<tr>
<td>LD R2, 8(R1)</td>
<td>LD P34, 8(P33)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+8</td>
</tr>
<tr>
<td>ADD R2, R2, 8</td>
<td>ADD P35, P34, 8</td>
<td>i</td>
<td>i+4</td>
<td>i+9</td>
<td>i+9</td>
</tr>
<tr>
<td>ST R1, (R3)</td>
<td>ST P33, (P3)</td>
<td>i</td>
<td>i+2</td>
<td>i+8</td>
<td>i+9</td>
</tr>
<tr>
<td>SUB R1, R1, R5</td>
<td>SUB P36, P33, P5</td>
<td>i+1</td>
<td>i+2</td>
<td>i+7</td>
<td>i+9</td>
</tr>
<tr>
<td>LD R1, 8(R2)</td>
<td>LD P1, 8(P35)</td>
<td>i+7</td>
<td>i+8</td>
<td>i+14</td>
<td>i+14</td>
</tr>
<tr>
<td>ADD R1, R1, R2</td>
<td>ADD P2, P1, P35</td>
<td>i+9</td>
<td>i+10</td>
<td>i+15</td>
<td>i+15</td>
</tr>
</tbody>
</table>
Branch prediction and instr fetch

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

Instr Fetch Queue

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

Reorder Buffer (ROB)

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

Committed Reg Map
R1→P1
R2→P2

Speculative Reg Map
R1→P36
R2→P34

Decode & Rename

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

Speculative Reg Map
R1→P36
R2→P34

Issue Queue (IQ)

ALU
ALU
ALU

Results written to regfile and tags broadcast to IQ

Register File P1-P64

The Alpha 21264 Out-of-Order Implementation
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 $\Rightarrow$ high peak ILP

• Window size: the number of in-flight instructions in the pipeline. Large window size $\Rightarrow$ 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.
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) – happens at commit
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]

Decode & Rename

Speculative Reg Map
R1→P36
R2→P34

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

Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7

Committed Reg Map
R1→P1
R2→P2

Issue Queue (IQ)

P37 ← [P35 + 8]
P37 → [P36 + 8]

Reorder Buffer (ROB)

ALU
ALU
ALU

Register File
P1-P64

Results written to regfile and tags broadcast to IQ

LSQ

D-Cache

ALU

ALU

ALU

Committed
Reg Map
R1→P1
R2→P2

ALU

ALU

ALU

Result written to regfile and tags broadcast to IQ

Committed
Reg Map
R1→P1
R2→P2

ALU

ALU

ALU

Result written to regfile and tags broadcast to IQ

Committed
Reg Map
R1→P1
R2→P2

ALU

ALU

ALU

Result written to regfile and tags broadcast to IQ

Committed
Reg Map
R1→P1
R2→P2

ALU

ALU

ALU

Result written to regfile and tags broadcast to IQ

Committed
Reg Map
R1→P1
R2→P2
Problem 2

Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1</td>
<td></td>
<td></td>
<td>abcd</td>
</tr>
<tr>
<td>LD</td>
<td>R3</td>
<td></td>
<td></td>
<td>adde</td>
</tr>
<tr>
<td>ST</td>
<td>R5</td>
<td>R6</td>
<td></td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7</td>
<td>R8</td>
<td></td>
<td>abce</td>
</tr>
<tr>
<td>ST</td>
<td>R9</td>
<td>R10</td>
<td></td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11</td>
<td>R12</td>
<td></td>
<td>abba</td>
</tr>
</tbody>
</table>
Problem 2

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ← [R2]</td>
<td>3</td>
<td>abcd</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>LD</td>
<td>R3 ← [R4]</td>
<td>6</td>
<td>adde</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>4</td>
<td>7</td>
<td>abba</td>
<td>5</td>
</tr>
<tr>
<td>LD</td>
<td>R7 ← [R8]</td>
<td>2</td>
<td>abce</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>8</td>
<td>3</td>
<td>abba</td>
<td>9</td>
</tr>
<tr>
<td>LD</td>
<td>R11 ← [R12]</td>
<td>1</td>
<td>abba</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>
## Problem 3

Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ← [R2]</td>
<td>3</td>
<td></td>
<td></td>
<td>abcd</td>
</tr>
<tr>
<td>LD</td>
<td>R3 ← [R4]</td>
<td>6</td>
<td></td>
<td></td>
<td>adde</td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>5 7</td>
<td></td>
<td>abba</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>R7 ← [R8]</td>
<td>2</td>
<td></td>
<td>abce</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>1 4</td>
<td></td>
<td>abba</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>R11 ← [R12]</td>
<td>2</td>
<td></td>
<td>abba</td>
<td></td>
</tr>
</tbody>
</table>
Problem 3

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume no memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1 ← [R2]</td>
<td>3</td>
<td>abcd</td>
<td>4</td>
</tr>
<tr>
<td>LD</td>
<td>R3 ← [R4]</td>
<td>6</td>
<td>adde</td>
<td>7</td>
</tr>
<tr>
<td>ST</td>
<td>R5 → [R6]</td>
<td>5</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7 ← [R8]</td>
<td>2</td>
<td>abce</td>
<td>3</td>
</tr>
<tr>
<td>ST</td>
<td>R9 → [R10]</td>
<td>1</td>
<td>4</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11 ← [R12]</td>
<td>2</td>
<td>abba</td>
<td>3</td>
</tr>
</tbody>
</table>
Problem 4

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD R1</td>
<td>[R2]</td>
<td>3</td>
<td>abcd</td>
<td></td>
</tr>
<tr>
<td>LD R3</td>
<td>[R4]</td>
<td>6</td>
<td>adde</td>
<td></td>
</tr>
<tr>
<td>ST R5</td>
<td>[R6]</td>
<td>4</td>
<td>7</td>
<td>abba</td>
</tr>
<tr>
<td>LD R7</td>
<td>[R8]</td>
<td>2</td>
<td>abce</td>
<td></td>
</tr>
<tr>
<td>ST R9</td>
<td>[R10]</td>
<td>8</td>
<td>3</td>
<td>abba</td>
</tr>
<tr>
<td>LD R11</td>
<td>[R12]</td>
<td>1</td>
<td>abba</td>
<td></td>
</tr>
</tbody>
</table>
Problem 4

Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume memory dependence prediction.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R1</td>
<td>[R2]</td>
<td>3</td>
<td>abcd</td>
</tr>
<tr>
<td>LD</td>
<td>R3</td>
<td>[R4]</td>
<td>6</td>
<td>adde</td>
</tr>
<tr>
<td>ST</td>
<td>R5</td>
<td>[R6]</td>
<td>4</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R7</td>
<td>[R8]</td>
<td>2</td>
<td>abce</td>
</tr>
<tr>
<td>ST</td>
<td>R9</td>
<td>[R10]</td>
<td>8</td>
<td>abba</td>
</tr>
<tr>
<td>LD</td>
<td>R11</td>
<td>[R12]</td>
<td>1</td>
<td>abba</td>
</tr>
</tbody>
</table>
Title

• Bullet