Superscalar (III)

Limitations of Superscalar (cont.)

Exercise
- Given the instructions below, list all the dependencies (RAW, WAW, WAR) it has, and apply register renaming to address output dependency (WAW) and anti-dependency (WAR).

\[
\begin{align*}
I0: & \quad R3 + 1 \rightarrow R3 \\
I1: & \quad R3 + R2 \rightarrow R4 \\
I2: & \quad R3 \text{ op } R4 \rightarrow R7 \\
I3: & \quad \text{Store } R0 \rightarrow R4
\end{align*}
\]

• True data dependency (RAW):
  - \textbf{I1 depends} on the result of \textbf{I0 (R3)}
  - \textbf{I2 depends} on the result of \textbf{I0 and I1 (R3, R4)}

• WAW: \textbf{I3 writes} after \textbf{I1 write} to R4
• WAR: \textbf{I3 writes} after \textbf{I2 reads} R4
• Register renaming:
  \[
  \begin{align*}
  I0: & \quad R3 + 1 \rightarrow R3(a) \\
  I1: & \quad R3(a) + R2 \rightarrow R4 \\
  I2: & \quad R3(a) \text{ op } R4 \rightarrow R7 \\
  I3: & \quad \text{Store } R0 \rightarrow R4(a)
  \end{align*}
  \]

Instruction Issue Policy
- In essence, the processor is trying to \textbf{look ahead of current point} of execution to \textbf{locate instructions} that can be brought \textbf{into the pipeline}.
- Three types of ordering are important in this regards:
  • Order in which instructions are \textbf{fetched}
  • Order in which instructions are \textbf{executed} (constrained by data dependencies)
  • Order in which instructions \textbf{update} registers and memory values (order of completion)
- One \textbf{constraint}: \textbf{results must be correct}. So, the processor must \textbf{accommodate} the various \textbf{dependencies and conflicts} discussed earlier.

- Four categories:
  • In-order issue (order to execute), in-order completion (order to write the result)
  • In-order issue, out-of-order completion
• Out-of-order issue, out-of-order completion
• Out-of-order issue, in-order completion

- Example:
  • Assume a superscalar pipeline is capable of fetching and decoding 2 instructions at a time
    - Instructions are fetched and decoded in pair. The next two instructions must wait to be decoded until the pair of decode pipeline stages has cleared.
  • having 3 separate ALUs (e.g., two for integer arithmetic and one for floating-point arithmetic)
  • 2 instances of the write-back pipeline stage
  • 6 instruction code fragment with the following constraints:
    - I1 requires two cycles to execute
    - I3 and I4 conflict for the same ALU (e.g., both need floating-point arithmetic)
    - I5 depends on the value produced by I4
    - I5 and I6 conflict for an ALU (which may be different from the one I3 and I4 need)
  • To fetch, decode, and write back an instruction, each stage need 1 clock cycle.
  • When there is a conflict for a functional unit, or when a functional unit requires more than one cycle to generate a result, instructions temporarily stall.

---

In-order issue, in-order completion
- **Sequential execution** (in-order issue) and to write results in that same order (in-order completion)
- It is the simplest policy. **Not very efficient.** Instruction must **stall if necessary.**

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Decode</th>
<th>ALU1</th>
<th>ALU2</th>
<th>ALU3</th>
<th>WriteBack1</th>
<th>WriteBack2</th>
<th>Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>I2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>I3</td>
<td>I4</td>
<td>I1</td>
<td>I2</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>I5</td>
<td>I6</td>
<td>I3</td>
<td>I4</td>
<td>I1</td>
<td>I2</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>I3</td>
<td>I4</td>
<td>I1</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>I4</td>
<td>I3</td>
<td>I1</td>
<td>I2</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>I5</td>
<td>I6</td>
<td>I4</td>
<td>I3</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>I5</td>
<td>I4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>I6</td>
<td>I5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>I6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>
- Note: in this example, I3 can use ALU2 or ALU3 at cycle 4. But they must be written back after I1 and I2 have been written (in-order completion). So, put I3 in cycle 4 won't shorten the time. I5 can use ALU1, ALU2 or ALU3. This example lets I5 use ALU3. I5 depends on the value produced by I4, so I5 has to wait until I4 finish execution in cycle 6.