In-order vs. Out-of-order Execution

In-order instruction execution
- instructions are fetched, executed & committed in compiler-generated order
  - if one instruction stalls, all instructions behind it stall
- instructions are statically scheduled by the hardware
  - scheduled in compiler-generated order
  - how many of the next \( n \) instructions can be issued, where \( n \) is the superscalar issue width
    - superscalars can have structural & data hazards within the \( n \) instructions
- advantage of in-order instruction scheduling: simpler implementation
  - faster clock cycle
  - fewer transistors
  - lower power
  - faster design/development/debug time

Out-of-order instruction execution
- instructions are fetched in compiler-generated order
- instruction commit is in-order (today) or out-of-order (older computers)
- in between they may be executed in some other order
- instructions are dynamically scheduled by the hardware
  - hardware decides in what order instructions can be executed
  - instructions behind a stalled instruction can pass it if not dependent upon it
- advantages: higher performance
  - better at hiding latencies, less processor stalling
  - higher utilization of functional units
In-order instruction issue: Alpha 21164

2 styles of static instruction scheduling
- dispatch buffer & instruction slotting (Alpha 21164)
- shift register model (UltraSPARC-1)

Instruction slotting
- can issue up to 4 instructions
  - completely empty the instruction buffer before filling it again
  - compiler can pad with \texttt{nops} so a conflicting instruction is issued with the following instructions, not alone
**21164 Instruction Unit Pipeline**

**Fetch & issue**
- **S0**: instruction fetch
  - branch prediction bits read
- **S1**: opcode decode
  - target address calculation
  - if predict taken, redirect the fetch
- **S2**: instruction slotting: decide which of the next 4 instructions can be issued
  - intra-cycle structural hazard check
  - intra-cycle data hazard check
- **S3**: instruction dispatch
  - inter-cycle load-use hazard check
  - register read
In-order instruction issue: UltraSparc 1

Shift register model
• can issue up to 4 instructions per cycle
• shift in new instructions after every group of instructions is issued

Code Scheduling on Superscalars

Original code
Loop:
\[
\begin{align*}
\text{lw } &\ R1, \ 0(R5) \\
\text{addu } &\ R1, \ R1, \ R6 \\
\text{sw } &\ R1, \ 0(R5) \\
\text{addi } &\ R5, \ R5, \ -4 \\
\text{bne } &\ R5, \ R0, \ Loop
\end{align*}
\]
**Code Scheduling on Superscalars**

Original code

<table>
<thead>
<tr>
<th>ALU/branch instructions</th>
<th>Memory instructions</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loop:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw R1, 0(R5)</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>addu R1, R1, R6</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>sw R1, 0(R5)</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>addi R5, R5, -4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>bne R5, R0, Loop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With load-latency-hiding code

<table>
<thead>
<tr>
<th>ALU/branch instructions</th>
<th>Memory instructions</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loop:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw R1, 0(s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi R5, R5, -4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addu R1, R1, R6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw R1, 4(R5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bne R5, $0, Loop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Code Scheduling on Superscalars: Loop Unrolling**

<table>
<thead>
<tr>
<th>ALU/branch instruction</th>
<th>Data transfer instruction</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loop:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi R5, R5, -16</td>
<td>lw R1, 0(R5)</td>
<td>1</td>
</tr>
<tr>
<td>lw R2, 12(R5)</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>addu R1, R1, R6</td>
<td>lw R3, 8(R5)</td>
<td>3</td>
</tr>
<tr>
<td>addu R2, R2, R6</td>
<td>lw R4, 4(R5)</td>
<td>4</td>
</tr>
<tr>
<td>addu R3, R3, R6</td>
<td>sw R1, 16(R5)</td>
<td>5</td>
</tr>
<tr>
<td>addu R4, R4, R6</td>
<td>sw R2, 12(R5)</td>
<td>6</td>
</tr>
<tr>
<td>bne R5, R0, Loop</td>
<td>sw R3, 8(R5)</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>sw R4, 4(R5)</td>
<td>8</td>
</tr>
</tbody>
</table>

What is the cycles per iteration?
What is the IPC?
Code Scheduling on Superscalars: Loop Unrolling

Advantages:
+ 

Disadvantages:
- 
-