1 In-Order Dual-Issue Superscalar PARCv1 Processor 2

2 Superscalar Pipeline Hazards 4
   2.1. RAW Hazards ................................................. 4
   2.2. Control Hazards ............................................. 6
   2.3. Structural Hazards ........................................... 10
   2.4. WAW and WAR Name Hazards ................................. 10

3 Analyzing Performance of Superscalar Processors 11
1. In-Order Dual-Issue Superscalar PARCv1 Processor

- Processors studied so far are fundamentally limited to CPI $\geq 1$
- Superscalar processors enable CPI $< 1$ (i.e., IPC $> 1$) by executing multiple instructions in parallel
- Can have both in-order and out-of-order superscalar processors, but we will start by exploring in-order

- Continue to assume combinational memories
- **F Stage**: fetch two instructions at once
- **D Stage**: 4 read ports, decode 2 inst, “issue” inst to correct pipe
- **X/M Stage**: separate into A and B pipes (see next page)
- **W Stage**: 2 write ports
More abstract way to illustrate same dual-issue superscalar pipeline

Different instructions use the A-pipe and/or the B-pipe

<table>
<thead>
<tr>
<th></th>
<th>addu</th>
<th>addiu</th>
<th>mul</th>
<th>lw</th>
<th>sw</th>
<th>j</th>
<th>jal</th>
<th>jr</th>
<th>bne</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-Pipe</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>B-Pipe</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Example pipeline diagram for dual-issue superscalar processor

- Multiple instructions in stages F, D, W allowed because superscalar processor has duplicated hardware to avoid structural hazards
- **Fetch Block** – group of instructions fetched as unit
- **Swizzle** – instructions “swapped” from natural fetch position to appropriate execution pipe
2. Superscalar Pipeline Hazards

Seems so easy, but why is pipelining hard?

- RAW Hazards
- Control Hazards
- Structural Hazards
- WAR/WAR Name Hazards

2.1. RAW Hazards

Let's first assume we only use stalling to resolve RAW hazards

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>addiu r1, r2, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addiu r3, r4, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addu r5, r1, r3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addiu r6, r5, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addiu r7, r8, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addiu r9, r8, 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A fully-bypassed superscalar processor is possible, but expensive
Revisit previous assembly sequence with full bypassing

<table>
<thead>
<tr>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>addiu r1, r2, 1</td>
</tr>
<tr>
<td>addiu r3, r4, 1</td>
</tr>
<tr>
<td>addu r5, r1, r3</td>
</tr>
<tr>
<td>addiu r6, r5, 1</td>
</tr>
<tr>
<td>addiu r7, r8, 1</td>
</tr>
<tr>
<td>addiu r9, r8, 1</td>
</tr>
</tbody>
</table>

Activity: Draw a pipeline diagram for following instruction sequence. Include all microarchitectural dependency arrows.

<table>
<thead>
<tr>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>addiu r1, r2, 1</td>
</tr>
<tr>
<td>lw r3, 0(r4)</td>
</tr>
<tr>
<td>lw r5, 0(r3)</td>
</tr>
<tr>
<td>addiu r6, r7, 1</td>
</tr>
<tr>
<td>addiu r8, r5, 1</td>
</tr>
<tr>
<td>addiu r9, r8, 1</td>
</tr>
</tbody>
</table>
2.2. Control Hazards

Consider following two static instruction sequences.

```
1 0x1000 addiu r1, r2, 1
2 0x1004 j foo
3 ...
4 foo:
5 0x2000 addiu r3, r4, 1
6 0x2004 addiu r5, r6, 1
```

Pipeline diagram for left sequence. Jumps are resolved in D stage.

```
```

Pipeline diagram for right sequence. Branches are resolved in A0 stage.

```
```
Unaligned fetch blocks

Consider the following static instruction sequence

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x000 opA</td>
</tr>
<tr>
<td>2</td>
<td>0x004 opB</td>
</tr>
<tr>
<td>3</td>
<td>0x008 opC</td>
</tr>
<tr>
<td>4</td>
<td>0x00c j 0x100</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>0x100 opD</td>
</tr>
<tr>
<td>7</td>
<td>0x104 j 0x204</td>
</tr>
<tr>
<td>8</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>0x204 opE</td>
</tr>
<tr>
<td>10</td>
<td>0x208 j 0x30c</td>
</tr>
<tr>
<td>11</td>
<td>...</td>
</tr>
<tr>
<td>12</td>
<td>0x30c opF</td>
</tr>
<tr>
<td>13</td>
<td>0x310 opG</td>
</tr>
<tr>
<td>14</td>
<td>0x314 opH</td>
</tr>
</tbody>
</table>

- Unaligned fetch blocks within a cache line are challenging
- Unaligned fetch blocks across cache lines are very challenging
2. Superscalar Pipeline Hazards

2.2. Control Hazards

**Aligned fetch blocks**

Only fetch aligned fetch blocks, possibly discarding first instruction. Reconsider the same static instruction sequence

```plaintext
1  0x000  opA
2  0x004  opB
3  0x008  opC
4  0x00c  j  0x100
5  ...
6  0x100  opD
7  0x104  j  0x204
8  ...
9  0x204  opE
10 0x208  j  0x30c
11  ...
12 0x30c  opF
13 0x310  opG
14 0x314  opH
```

Layout of fetch blocks in instruction cache. Numbers indicate which instructions belong to which fetch block.
## Supporting precise exceptions

Consider following instruction sequence. Assume commit point is in the A1/B1 stage and the `xxx` instruction causes an illegal instruction exception originating in the D stage.

```plaintext
1. addu r1, r2, r3
2. xxx               # causes illegal instruction exception
3. addiu r4, r5, 1
4. addiu r6, r7, 1
5. ...
6. exception_handler:
   7. opX
   8. opY
   9. opZ
```

What if `addu` caused an arithmetic overflow exception?
2.3. Structural Hazards

Structural hazards are not possible in the canonical single-issue PARCv1 pipeline, but structural hazards are possible in the canonical dual-issue PARCv1 pipeline if two instructions in the same fetch block want to use the same pipe.

```
mul r1, r2, r3
mul r4, r5, r6
lw r7, 0(r8)
sw r9, 0(r10)
```

2.4. WAW and WAR Name Hazards

WAW name hazards are not possible in the canonical single-issue PARCv1 pipeline, but WAW name hazards are possible in the canonical dual-issue PARCv1 pipeline if two instructions in the same fetch block write the same register.

```
addiu r1, r2, 1
addiu r1, r3, 1
```

WAR name hazards are not possible in the canonical single-issue PARCv1 pipeline. Are WAR name hazards possible in the canonical dual-issue PARCv1 pipeline?

```
addiu r1, r2, 1
addiu r2, r3, 1
```
Consider the classic vector-vector add loop over arrays with 64 elements. This loop has a CPI of 1.33 on the canonical single-issue PARCv1 processor. What is the CPI on the canonical dual-issue PARCv1 processor?

```
loop:
    lw   r12, 0(r4)
    lw   r13, 0(r5)
    addu r14, r12, r13
    sw   r14, 0(r6)
    addiu r4, r4, 4
    addiu r5, r5, 4
    addiu r6, r6, 4
    addiu r7, r7, -1
    bne  r7, r0, loop
    jr   r31
```