Dealing with Hazards in Pipelined Processors

Two options

1. Either the control unit can be smart, i.e. it can delay instruction phases to avoid hazards. Processor cost increases.

2. The compiler can be smart, i.e. produce optimized codes either by inserting NOPs or by rearranging instructions. The cost of the compiler goes up.
Instruction Reorganization by Compiler

To avoid data hazards, the control unit can insert bubbles. As an alternative, the compiler can use \texttt{NOP} instructions.

Example: Compute \(a: = b + c; d: = e + f\)
\((a, b, c, d, e, f\) are stored in the memory)
Instruction Reorganization by Compiler

The compiler can further speedup by reorganizing the instruction stream and minimizing the no of NOP’s.

Example: Compute $a = b + c; d = e + f$

<table>
<thead>
<tr>
<th>Original code</th>
<th>Code reorganized by a smart compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>LW R1,b</td>
<td>LW R1,b</td>
</tr>
<tr>
<td>LW R2,c</td>
<td>LW R2,c</td>
</tr>
<tr>
<td>ADD R3, R1, R2</td>
<td>LW R4, e</td>
</tr>
<tr>
<td>SW a, R3</td>
<td>LW R5, f</td>
</tr>
<tr>
<td>LW R1, e</td>
<td>ADD R3,R1,R2</td>
</tr>
<tr>
<td>LW R2,f</td>
<td>NOP</td>
</tr>
<tr>
<td>SUB R3, R1, R2</td>
<td>SW a, R3</td>
</tr>
<tr>
<td>SW d, R3</td>
<td>SUB R6, R5, R4</td>
</tr>
<tr>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td></td>
<td>SW d, R6</td>
</tr>
<tr>
<td></td>
<td>NOP</td>
</tr>
</tbody>
</table>

(Control unit remains unchanged)

Note the reassignment of registers
Another example: delayed branch

```
add $r1, $r2, $r3
beq $r2, $zero, L
<instruction>
L:  <instruction>
L': <instruction>
```

```
add $r1, $r2, $r3
beq $r2, $zero, L'
NOP (delay slot)
<instruction>
<instruction>
L:  <instruction>
L': <instruction>
```

The compiler may try to schedule other instructions in the delay slot for the sake of speed-up. Here is an example:

```
beq $r2, $zero, L
add $r1, $r2, $r3
<instruction>
<instruction>
L:  <instruction>
```

Note that the first two instructions have been swapped. How does it help?
Techniques for speeding up processors

Instruction level parallelism

Thread level parallelism

Superscalar processors