Data and control hazards

- Data hazards:
- Detecting dependencies
- Forwarding
- Stalls
- Detecting branch hazards
- Reducing the delay of branches
Dependencies

- Problem with starting next instruction before first is finished
  - dependencies that “go backward in time” are data hazards
Software Solution

- Have compiler guarantee no hazards
- Where do we insert the “nops”?

  ```
  sub     $2, $1, $3
  and    $12, $2, $5
  or      $13, $6, $2
  add     $14, $2, $2
  sw      $15, 100 ($2)
  ```

- Problem: this really slows us down!
Forwarding

- Use temporary results, don’t wait for them to be written
  - register file forwarding to handle read/write to same register
  - ALU forwarding

<table>
<thead>
<tr>
<th>Time (in clock cycles)</th>
<th>CC 1</th>
<th>CC 2</th>
<th>CC 3</th>
<th>CC 4</th>
<th>CC 5</th>
<th>CC 6</th>
<th>CC 7</th>
<th>CC 8</th>
<th>CC 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of register $2$</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10–20</td>
<td>–20</td>
<td>–20</td>
<td>–20</td>
<td>–20</td>
</tr>
<tr>
<td>Value of EX/MEM</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>–20</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Value of MEM/WB</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>–20</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Program execution order (in instructions)

- sub $2$, $1$, $3$
- and $12$, $2$, $5$
- or $13$, $6$, $2$
- add $14$, $2$, $2$
- sw $15$, 100($2$)
Forwarding
Data hazards and stalls

- Load word can still cause a hazard:
  - an instruction tries to read a register following a load instruction that writes to the same register.

- Thus, we need a hazard detection unit to “stall” the load instruction
Stalling

- We can stall the pipeline by keeping an instruction in the same stage.

Program execution order (in instructions):
- lw $2, 20($1)
- and $4, $2, $5
- or $8, $2, $6
- add $9, $4, $2
- slt $1, $6, $7

Time (in clock cycles):
<table>
<thead>
<tr>
<th>OC 1</th>
<th>OC 2</th>
<th>OC 3</th>
<th>OC 4</th>
<th>OC 5</th>
<th>OC 6</th>
<th>OC 7</th>
<th>OC 8</th>
<th>OC 9</th>
<th>OC 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>IM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
<td>DM</td>
<td>Reg</td>
</tr>
<tr>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
</tr>
<tr>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
</tr>
<tr>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
</tr>
<tr>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
<td>IM</td>
<td>Reg</td>
</tr>
</tbody>
</table>

©1998 Morgan Kaufmann Publishers
Hazard Detection Unit

- Stall by letting an instruction that won’t write anything go forward
and $4, $2, $5

lw $2, 20($1)

before<1>

before<2>

before<3>

Clock 2

or $4, $4, $2

and $4, $2, $5

lw $2, 20($1)

before<1>

before<2>

Clock 3

©1998 Morgan Kaufmann Publishers 25
or $4, $4, $2 and $4, $2, $5 bubble lw $2, . . . before<1>

add $9, $4, $2 or $4, $4, $2 and $4, $2, $5 bubble lw $2, . . .
Branch Hazards

- When we decide to branch, other instructions are in the pipeline!

- We are predicting “branch not taken”
  - need to add hardware for flushing instructions if we are wrong
Flush Instructions
and $12, $2, $5  
beq $1, $3, 7  
sub $10, $4, $8  
before<1>  
before<2>  

Clock 3  
lw $4, 50($7)  
bubble (nop)  
beq $1, $3, 7  
sub $10, . . .  
before<1>  

Clock 4  
IF.Flush  
ID/EX  
EX/MEM  
MEM/WB  
ALU  
Data memory  
Forwarding unit  
Control  
Hazard detection unit  
IF/ID  

bubble (nop)  
beq $1, $3, 7  
sub $10, . . .  
before<1>  

©1998 Morgan Kaufmann Publishers
Advanced Pipelining

- Longer pipelines - Superpipelining
- Replicating components of the datapath - Multiple instruction per cycle (superscalar)
- Dynamic pipeline scheduling - avoid stalls
Dynamic Scheduling

- The hardware performs the “scheduling”
  - hardware tries to find instructions to execute
  - out of order execution is possible
  - speculative execution and dynamic branch prediction
- All modern processors are very complicated
  - DEC Alpha 21264: 9 stage pipeline, 6 instruction issue
  - PowerPC and Pentium: branch history table
  - Compiler technology important