Instruction Pipelining

- Instruction Pipeline

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Decode</th>
<th>Operand</th>
<th>Execute</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Decode</td>
<td>Operand</td>
<td>Execute</td>
<td>Store</td>
</tr>
<tr>
<td>Fetch</td>
<td>Decode</td>
<td>Operand</td>
<td>Execute</td>
<td>Store</td>
</tr>
<tr>
<td>Fetch</td>
<td>Decode</td>
<td>Operand</td>
<td>Execute</td>
<td>Store</td>
</tr>
<tr>
<td>Fetch</td>
<td>Decode</td>
<td>Operand</td>
<td>Execute</td>
<td>Store</td>
</tr>
</tbody>
</table>

- **PC is incremented by** 4 at the Fetch stage to fetch the next instruction
- A delay slot caused by a jmp instruction, why?

<table>
<thead>
<tr>
<th>PC</th>
<th>PC’</th>
<th>instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12</td>
<td>add</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
<td>jmp 40</td>
</tr>
<tr>
<td>16</td>
<td>40</td>
<td>delay slot</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>44</td>
<td>sub</td>
</tr>
</tbody>
</table>

- What happens to an instruction placed in the delay slot?
Delay-Slot Instructions

- If this is a “feature,” it certainly contradicts “normal” expectations
  
  If you think this is confusing, use a `nop` in all delay slots

- Optimizers may be able take advantage of the delay slot

  unoptimized `if (a > b) c = a; else c = b;` takes 7 cycles

  ```
  cmp a,b
  ble L1
  nop
  mov a,c
  ba L2
  nop
  L1: mov b,c
  L2:
  ```

  optimized takes 4 cycles

  ```
  cmp a,b
  ble L1
  mov b,c
  mov a,c
  ```

  L1:
Annul Bit

• What if the branch with an instruction in its delay slot does not take?

• *Annul bit* controls the execution of the delay-slot instruction
  
  bg, a      L1
  mov a, c
  
  the “,a” causes the mov instruction to be executed if the branch is taken,
  and not executed if the branch is not taken

• Exception
  
  ba, a L does *not* execute its delay-slot instruction

• *What is the advantage of this counterintuitive convention?*
Annul Bit, cont’d

- Optimized for \((i = 0; i < n; i++)\) 1; 2; \(\ldots\); n
  
  ```
  clr  i
  ba  L2
  nop
  L1:  1
       2
       \(\ldots\)
       n
  inc  i
  L2:  cmp  i,n
       bl  L1
       nop
  ```

- better code uses delay slots
  
  ```
  clr  i
  ba,a  L2
  L1:  2
       \(\ldots\)
       n
  inc  i
  L2:  cmp  i,n
       bl,a  L1
       1
  ```

  \(n + 4\) instructions/iteration

- Convention of programming
  
  don’t use annul unless absolutely necessary
  
  place nop after control-transfer instructions

- What happens when the delay-slot instruction is a control-transfer instruction?

- See pages 51–55 in the SPARC Architecture Manual for details