• Daily Review of today’s lecture
  ♦ Due tomorrow (11/6) at 8am
• HW #8 due Monday, 11/12 at 5pm
• Program #2 due Friday, 11/16 at 11:59pm
5. Transfer control to OS

- Same as before
Increase Depth – assembly line – build many cars at the same time, but each car is in a different stage of assembly.

Increase Width – multiple assembly lines – build many cars at the same time by building many line, all of which operate simultaneously.

Ford mass produces cars. We want to “mass produce” instructions
Superpipelining
(very deep pipelining)

- Register delays are higher percentage of clock
- Difficult to split stages up evenly
SuperScalars

- Replicate parts of pipeline
- Multiple instructions in same stage at once
Super Scalars

Fetch → Decode → Execute → Memory → WriteBack

Read Values → Write Values
Super Scalars

- Which instructions can execute in parallel?
- Fetching multiple instructions per cycle
SuperScalars

• Which instructions can execute in parallel?
  ◆ Independent instructions

• Fetching multiple instructions per cycle
SuperScalars

- Which instructions can execute in parallel?
  - Independent instructions
- Fetching multiple instructions per cycle
  - Higher ideal throughput
  - Branches are even more punishing
Static Scheduling – VLIW

- Compiler schedules the instructions
- If one instruction stalls, all following instructions stall
SuperScalar MIPS

- Two instructions / cycle
- one alu/branch, one ld/st each cycle
Schedule for SS MIPS

Loop:  lw     $t0, 0($s1)
      addu  $t0, $t0, $s2
      sw    $t0, 0($s1)
      addi  $s1, $s1, -4
      bne   $s1, $zero,Loop

PC    ALU/branch     ld/st
0
8
16
24
32
Schedule for SS MIPS

Loop:  lw     $t0, 0($s1)
      addu   $t0, $t0, $s2
      sw     $t0, 0($s1)
      addi   $s1, $s1, -4
      bne    $s1, $zero,Loop

PC    | ALU/branch       | ld/st
0     | lw $t0, 0($s1)   |
8     |                 |
16    |                 |
24    |                 |
32    |                 |
Schedule for SS MIPS

Loop:

lw    $t0, 0($s1)
addu  $t0, $t0, $s2
sw    $t0, 0($s1)
addi  $s1, $s1, -4
bne   $s1, $zero, Loop

PC    ALU/branch      ld/st
0     lw $t0, 0($s1)  lw $t0, 0($s1)
8     addu $t0, $t0, $s2
16    
24    
32    

Schedule for SS MIPS

Loop:  lw    $t0, 0($s1)
      addu  $t0, $t0, $s2
      sw    $t0, 0($s1)
      addi  $s1, $s1, -4
      bne   $s1, $zero,Loop

PC          ALU/branch        ld/st
0            lw $t0, 0($s1)    lw $t0, 0($s1)
8
16           addu $t0, $t0, $s2
24
32           sw $t0, 0($s1)
Schedule for SS MIPS

Loop:  
lw  $t0, 0($s1)  
addu $t0, $t0, $s2  
sw  $t0, 0($s1)  
addi $s1, $s1, -4  
bne $s1, $zero,Loop

What?!? We can’t reorder the addi and the sw! There is a **WAR** dependency!!!
Schedule for SS MIPS

Loop:  lw  $t0, 0($s1)
       addu  $t0, $t0, $s2
       sw  $t0, 0($s1)
       addi  $s1, $s1, -4
       bne  $s1, $zero,Loop

PC  | ALU/branch | ld/st
---|------------|---
0  |            | lw $t0, 0($s1)
8  | addi  $s1, $s1, -4 |
16 | addu  $t0, $t0, $s2 |
24 | **bne  $s1, $zero,Loop** |
32 |             | sw $t0, 4($s1)
SuperScalars – Static

Fetch → Decode → Execute → Memory → WriteBack

lw

Read Values

Write Values
SuperScalars – Static

Fetch: addi
Decode: lw
Execute: 
Memory: 
WriteBack: 

Read Values
Write Values
SuperScalars – Static

Fetch: addu
Decode: addi
Execute: lw
Memory:
WriteBack:

Read Values
Write Values
SuperScalars – Static

Fetch | Decode | Execute | Memory | WriteBack

bne → addu → addi → lw →

Read Values

sw

Write Values

addu

lw
SuperScalars – Static

Fetch: bne
Decode: addu
Execute: addi
Memory: nop
WriteBack: lw

3 Nops - We can do better!!!!!
Loop Problem

• Problem:

• Solution:
Loop Problem

• Problem:
  ◦ Too many dependencies in loop
  ◦ Not enough instructions to fill in holes

• Solution:
Loop Problem

- **Problem:**
  - Too many dependencies in loop
  - Not enough instructions to fill in holes

- **Solution:**
  - Do two iterations at once
  - More instructions
  - Only one branch
Loop Unrolling

Loop:  lw $t0, 0($s1)
      addi $s1, $s1, -4
      addu $t0, $t0, $s2
      sw $t0, 4($s1)
      bne $s1, $zero, Loop
Loop Unrolling

1. Unroll Loop

Loop:  
lw  $t0, 0($s1)  
addi $s1, $s1, -4  
addu $t0, $t0, $s2  
sw  $t0, 4($s1)  
lw  $t0, 0($s1)  
addi $s1, $s1, -4  
addu $t0, $t0, $s2  
sw  $t0, 4($s1)  
bne $s1, $zero, Loop
Loop Unrolling

Loop:  
\[
\begin{align*}
\text{lw} & \quad \text{\$t0, 0(\$s1)} \\
\text{addi} & \quad \text{\$s1, \$s1, -4} \\
\text{addu} & \quad \text{\$t0, \$t0, \$s2} \\
\text{sw} & \quad \text{\$t0, 4(\$s1)} \\
\text{lw} & \quad \text{\$t0, 0(\$s1)} \\
\text{addi} & \quad \text{\$s1, \$s1, -4} \\
\text{addu} & \quad \text{\$t0, \$t0, \$s2} \\
\text{sw} & \quad \text{\$t0, 4(\$s1)} \\
\text{bne} & \quad \text{\$s1, \$zero, Loop}
\end{align*}
\]

But wait!!! How has this helped? There are tons of dependencies? Whatever are we to do?
Loop:  lw   $t0, 0($s1)
       addi  $s1, $s1, -4
       addu  $t0, $t0, $s2
       sw    $t0, 4($s1)
       lw    $t0, 0($s1)
       addi  $s1, $s1, -4
       addu  $t0, $t0, $s2
       sw    $t0, 4($s1)
       bne   $s1, $zero, Loop

But wait!!!  How has this helped?  There are tons of dependencies?  Whatever are we to do?
Loop Unrolling

2. Rename Registers

Loop:  
  lw   $t0, 0($s1)  
  addi $s1, $s1, -4  
  addu $t0, $t0, $s2  
  sw   $t0, 4($s1)  
  lw   $t1, 0($s1)  
  lw   $t1, 0($s1)  
  addi $s1, $s1, -4  
  addu $t1, $t1, $s2  
  sw   $t1, 4($s1)  
  bne  $s1, $zero,Loop

But wait!!! How has this helped? There are tons of dependencies? Whatever are we to do? **Register Renaming!!!**
Loop Unrolling

3. Reduce Instructions

Move the $s1 decrement to be with the other one
Loop Unrolling

3. Reduce Instructions

Loop: lw $t0, 0($s1)  
     addi $s1, $s1, -4  
     addu $t0, $t0, $s2  
     sw $t0, 4($s1)  
     lw $t1, 0($s1)  
     addi $s1, $s1, -4  
     addu $t1, $t1, $s2  
     sw $t1, 4($s1)  
     bne $s1, $zero,Loop

Loop: lw $t0, 0($s1)  
     addi $s1, $s1, -4  
     addu $t0, $t0, $s2  
     sw $t0, 8($s1)  
     lw $t1, 0($s1)  
     addi $s1, $s1, -4  
     addu $t1, $t1, $s2  
     sw $t1, 4($s1)  
     bne $s1, $zero,Loop

Move the $s1 decrement to be with the other one
### Loop Unrolling

#### 3. Reduce Instructions

<table>
<thead>
<tr>
<th>Original Code</th>
<th>Unrolled Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t0, 0($s1)</td>
<td>$t0, 0($s1)</td>
</tr>
<tr>
<td>addi $s1, $s1, -4</td>
<td>addi $s1, $s1, -8</td>
</tr>
<tr>
<td>addi $s1, $s1, -4</td>
<td>addi $s1, $s1, -8</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>addu $t0, $t0, $s2</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>addu $t0, $t0, $s2</td>
</tr>
<tr>
<td>sw $t0, 8($s1)</td>
<td>sw $t0, 8($s1)</td>
</tr>
<tr>
<td>lw $t1, 4($s1)</td>
<td>lw $t1, 4($s1)</td>
</tr>
<tr>
<td>addu $t1, $t1, $s2</td>
<td>addu $t1, $t1, $s2</td>
</tr>
<tr>
<td>sw $t1, 4($s1)</td>
<td>sw $t1, 4($s1)</td>
</tr>
<tr>
<td>bne $s1, $zero,Loop</td>
<td>bne $s1, $zero,Loop</td>
</tr>
</tbody>
</table>
## Loop Unrolling
### 4. Schedule

<table>
<thead>
<tr>
<th>Loop:</th>
<th>ALU/branch</th>
<th>lw/sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $t0, 0($s1)</td>
<td>addi $s1, $s1, -8</td>
<td>lw $t0, 0($s1)</td>
</tr>
<tr>
<td>addi $s1, $s1, -8</td>
<td>addu $t0, $t0, $s2</td>
<td></td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>sw $t0, 8($s1)</td>
<td></td>
</tr>
<tr>
<td>sw $t0, 8($s1)</td>
<td>lw $t1, 4($s1)</td>
<td></td>
</tr>
<tr>
<td>lw $t1, 4($s1)</td>
<td>addu $t1, $t1, $s2</td>
<td></td>
</tr>
<tr>
<td>addu $t1, $t1, $s2</td>
<td>sw $t1, 4($s1)</td>
<td></td>
</tr>
<tr>
<td>sw $t1, 4($s1)</td>
<td>bne $s1, $zero, Loop</td>
<td></td>
</tr>
<tr>
<td>bne $s1, $zero, Loop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Loop Unrolling
4. Schedule

<table>
<thead>
<tr>
<th>Loop:</th>
<th>ALU/branch</th>
<th>lw/sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>$t0, 0($s1)</td>
<td>lw $t0,0($s1)</td>
</tr>
<tr>
<td>addi</td>
<td>$s1, $s1, -8</td>
<td>addi $s1,$s1,-8</td>
</tr>
<tr>
<td>addu</td>
<td>$t0, $t0, $s2</td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t0, 8($s1)</td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t1, 4($s1)</td>
<td></td>
</tr>
<tr>
<td>addu</td>
<td>$t1, $t1, $s2</td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t1, 4($s1)</td>
<td></td>
</tr>
<tr>
<td>bne</td>
<td>$s1, $zero,Loop</td>
<td></td>
</tr>
</tbody>
</table>
## Loop Unrolling

### 4. Schedule

<table>
<thead>
<tr>
<th>Loop</th>
<th>ALU/branch</th>
<th>lw/sw</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lw</code> $t0, 0($s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>addi</code> $s1, $s1, -8</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>addu</code> $t0, $t0, $s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sw</code> $t0, 8($s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>lw</code> $t1, 4($s1)</td>
<td><code>addu</code> $t0,$t0,$s2</td>
<td></td>
</tr>
<tr>
<td><code>addu</code> $t1, $t1, $s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sw</code> $t1, 4($s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>bne</code> $s1, $zero, Loop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Loop Unrolling

4. Schedule

<table>
<thead>
<tr>
<th>Loop:</th>
<th>ALU/branch</th>
<th>lw/sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $t0, 0($s1)</td>
<td>addi $s1,$s1,-8</td>
<td>lw $t0,0($s1)</td>
</tr>
<tr>
<td>addi $s1, $s1, -8</td>
<td>addi $s1,$s1,-8</td>
<td></td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>addu $t0,$t0,$s2</td>
<td></td>
</tr>
<tr>
<td>sw $t0, 8($s1)</td>
<td>sw $t0,8($s1)</td>
<td></td>
</tr>
<tr>
<td>lw $t1, 4($s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addu $t1, $t1, $s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw $t1, 4($s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bne $s1, $zero,Loop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Loop Unrolling
4. Schedule

<table>
<thead>
<tr>
<th>Loop:</th>
<th>ALU/branch</th>
<th>lw/sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $t0, 0($s1)</td>
<td>addi $s1,$s1,-8</td>
<td>lw $t0,0($s1)</td>
</tr>
<tr>
<td>addi $s1, $s1, -8</td>
<td>addu $t0, $t0, $s2</td>
<td>lw $t1,4($s1)</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>sw $t0, 8($s1)</td>
<td>addu $t0,$t0,$s2</td>
</tr>
<tr>
<td>sw $t0, 8($s1)</td>
<td>lw $t1,4($s1)</td>
<td>sw $t0,8($s1)</td>
</tr>
<tr>
<td>lw $t1, 4($s1)</td>
<td>addu $t1,$t1,$s2</td>
<td></td>
</tr>
<tr>
<td>addu $t1, $t1, $s2</td>
<td>sw $t1, 4($s1)</td>
<td></td>
</tr>
<tr>
<td>sw $t1, 4($s1)</td>
<td>bne $s1, $zero,Loop</td>
<td></td>
</tr>
</tbody>
</table>
### Loop Unrolling

#### 4. Schedule

<table>
<thead>
<tr>
<th>Loop:</th>
<th>ALU/branch</th>
<th>lw/sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $t0, 0($s1)</td>
<td>addi $s1,$s1,-8</td>
<td>lw $t0,0($s1)</td>
</tr>
<tr>
<td>addi $s1, $s1, -8</td>
<td>addu $t0,$t0,$s2</td>
<td>lw $t1,4($s1)</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>sw $t0, 8($s1)</td>
<td>addu $t0,$t0,$s2</td>
</tr>
<tr>
<td>sw $t1, 4($s1)</td>
<td>addu $t1,$t1,$s2</td>
<td>sw $t0,8($s1)</td>
</tr>
<tr>
<td>addu $t1, $t1, $s2</td>
<td>lw $t1,4($s1)</td>
<td></td>
</tr>
<tr>
<td>sw $t1, 4($s1)</td>
<td>lw $t0,0($s1)</td>
<td></td>
</tr>
<tr>
<td>bne $s1, $zero, Loop</td>
<td>addu $t0,$t0,$s2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sw $t0,8($s1)</td>
<td></td>
</tr>
</tbody>
</table>
## Loop Unrolling

### 4. Schedule

<table>
<thead>
<tr>
<th>Loop:</th>
<th>ALU/branch</th>
<th>lw/sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $t0, 0($s1)</td>
<td>addi $s1,$s1,-8</td>
<td>lw $t0,0($s1)</td>
</tr>
<tr>
<td>addi $s1, $s1, -8</td>
<td>addu $t0,$t0,$s2</td>
<td>lw $t1,4($s1)</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>sw $t0, 8($s1)</td>
<td>addu $t0,$t0,$s2</td>
</tr>
<tr>
<td>sw $t0, 8($s1)</td>
<td>addu $t1,$t1,$s2</td>
<td>sw $t0,8($s1)</td>
</tr>
<tr>
<td>lw $t1, 4($s1)</td>
<td>sw $t1, 4($s1)</td>
<td>sw $t1,4($s1)</td>
</tr>
<tr>
<td>addu $t1, $t1, $s2</td>
<td>bne $s1, $zero,Loop</td>
<td></td>
</tr>
<tr>
<td>sw $t1, 4($s1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Loop Unrolling

**4. Schedule**

<table>
<thead>
<tr>
<th>Loop:</th>
<th>ALU/branch</th>
<th>lw/sw</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $t0, 0($s1)</td>
<td>addi $s1,$s1,-8</td>
<td>lw $t0,0($s1)</td>
</tr>
<tr>
<td>addi $s1, $s1, -8</td>
<td>addu $t0,$t0,$s2</td>
<td>lw $t1,4($s1)</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>sw $t0, 8($s1)</td>
<td>alu $t0,$t0,$s2</td>
</tr>
<tr>
<td>sw $t0, 8($s1)</td>
<td>addu $t1,$t1,$s2</td>
<td>sw $t0,8($s1)</td>
</tr>
<tr>
<td>lw $t1, 4($s1)</td>
<td>sw $t1,4($s1)</td>
<td>[ \text{bne } $s1,$zero,Loop \text{ } \text{sw } $t1,4($s1) ]</td>
</tr>
</tbody>
</table>
## Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Unrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU/branch</td>
<td>ld/st</td>
<td>ALU/branch</td>
</tr>
<tr>
<td></td>
<td>lw $t0, 0($s1)</td>
<td>addi $s1,$s1,-8</td>
</tr>
<tr>
<td>addi $s1, $s1, -4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td></td>
<td>addu $t0,$t0,$s2</td>
</tr>
<tr>
<td>bne $s1, $zero,L</td>
<td>sw $t0, 4($s1)</td>
<td>addu $t1,$t1,$s2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bne $s1,$zero,Lo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sw $t0,8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sw $t1,4</td>
</tr>
</tbody>
</table>
## Performance Comparison

<table>
<thead>
<tr>
<th>Original</th>
<th>Unrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU/branch</td>
<td>ALU/branch</td>
</tr>
<tr>
<td>addi $s1, $s1, -4</td>
<td>addi $s1,$s1,-8</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>addu $t0,$t0,$s2</td>
</tr>
<tr>
<td>bne $s1, $zero,L</td>
<td>bne $s1,$zero,Lo</td>
</tr>
<tr>
<td>lw $t0, 0($s1)</td>
<td>lw $t0,0</td>
</tr>
<tr>
<td>sw $t0, 4($s1)</td>
<td>sw $t0,8</td>
</tr>
<tr>
<td>bne $s1, $zero,L</td>
<td>bne $s1,$zero,Lo</td>
</tr>
</tbody>
</table>

4 cycles / iteration
Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Unrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU/branch</td>
<td>ld/st</td>
<td>ALU/branch</td>
</tr>
<tr>
<td></td>
<td>lw $t0, 0($s1)</td>
<td>addi $s1,$s1,-8</td>
</tr>
<tr>
<td>addi $s1, $s1, -4</td>
<td>lw $t0, 0($s1)</td>
<td>addi $s1,$s1,-8</td>
</tr>
<tr>
<td>addu $t0, $t0, $s2</td>
<td>addu $t0,$t0,$s2</td>
<td>addu $t0,$t0,$s2</td>
</tr>
<tr>
<td>bne $s1, $zero, L</td>
<td>addu $t0,$t0,$s2</td>
<td>addu $t1,$t1,$s2</td>
</tr>
<tr>
<td></td>
<td>sw $t0, 4($s1)</td>
<td>sw $t0, 4($s1)</td>
</tr>
<tr>
<td></td>
<td>sw $t0, 4($s1)</td>
<td>sw $t0, 4($s1)</td>
</tr>
<tr>
<td>bne $s1,$zero,Lo</td>
<td>sw $t0, 4($s1)</td>
<td>sw $t0, 4($s1)</td>
</tr>
<tr>
<td></td>
<td>sw $t1,4</td>
<td>sw $t1,4</td>
</tr>
</tbody>
</table>

4 cycles / iteration  5 cycles / 2 iterations  2.5 cycles / iteration
Correct Solution: Execute 4 iterations simultaneously

<table>
<thead>
<tr>
<th>Loop:</th>
<th>ALU or branch instruction</th>
<th>Data transfer instruction</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>addi $s1,$s1,-16</td>
<td>lw $t0, 0($s1)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lw $t1,12($s1)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>addu $t0,$t0,$s2</td>
<td>lw $t2, 8($s1)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>addu $t1,$t1,$s2</td>
<td>lw $t3, 4($s1)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>addu $t2,$t2,$s2</td>
<td>sw $t0, 16($s1)</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>addu $t3,$t3,$s2</td>
<td>sw $t1,12($s1)</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sw $t2, 8($s1)</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>bne $s1,$zero,Loop</td>
<td>sw $t3, 4($s1)</td>
<td>8</td>
</tr>
</tbody>
</table>
Static Scheduling Summary

- Code size increases (because of nops)
- It cannot resolve ld/st dependencies
- If one instruction stalls, all later ones stall
Dynamic Scheduling
Dynamic Scheduling

- Hardware schedules ready instructions
Dynamic Scheduling

- Hardware schedules ready instructions
- Only *dependent* instructions stall
Dynamic Scheduling

- Hardware schedules ready instructions
- Only *dependent* instructions stall
- WAW and WAR resolved in hardware
4-wide Dynamic Superscalar Fetch

Loop:
lw r2, 0(r1)
addu r2, r2, r5
sw r2, 0(r1)
addi r1, r1, -4
bne r1, r7, Loop

KEY
Waiting for value
Reading value

Register Alias Table

Fetch 4 instructions each cycle

Instruction Window

Ld/St

1Add

2Add

3Add

Commit Buffer

Ld/St Queue

sw 1add1, 0(s1)
lw r2, 0(s1)
addu r2, ldst1, r5
addi r1, r1, -4
bne 2add1, r7, Loop

addi r1, r1, -4
sw r2, 0(s1)
addi r1, r1, -4
lw r2, 0(s1)
addu r2, ldst1, r5
addi r1, r1, -4
bne 2add1, r7, Loop

sw r2, 0(s1)
4-wide Dynamic Superscalar

Decode

Register Alias Table

1. Current Register Number (WAW/WAR Register Renaming)

Register Alias Table records

or

Commit Buffer

Ld/St Queue

sw r2, 0(s1)

Ld/St

sw 1add1, 0(s1)

1Add

addu r2, ldst1, r5

2Add

addi r1, r1, -4

3Add

bne 2add1, r7, Loop

addi r1, r1, -4

lw r2, 0(s1)

addu r2, ldst1, r5

lw r2, 0(s1)

sw r2, 0(s1)

lw r2, 0(s1)

sw 1add1, 0(s1)

lw r2, 0(s1)
4-wide Dynamic Superscalar Decode

Register Alias Table records
1. Current Register Number (WAW/WAR Register Renaming)
   or
2. Functional Unit (RAW – result not ready)
Wait until your inputs are ready

lw r2, 0(r1)
addu r2, r2, r5
sw r2, 0(r1)
addi r1, r1, -4
bne r1, r7, Loop

lw r2, 0(s1)
addu r2, ldst1, r5
sw r2, 0(s1)
addi r1, r1, -4
bne 2add1, r7, Loop

lw r2, 0(s1)
addu r2, ldst1, r5
addi r1, r1, -4
bne 2add1, r7, Loop
Execute once they are ready

sw r12, 0(r1)
addi r1, r1, -4
bne r1, r7, Loop

lw r2, 0(r1)
addu r2, r2, r5
sw r2, 0(r1)
addi r1, r1, -4
bne r1, r7, Loop

lw r2, 0(s1)
addu r2, ldst1, r5
sw 1add1, 0(s1)
addi r1, r1, -4
bne 2add1, r7, Loop

lw r2, 0(s1)
addu r2, ldst1, r5
sw r2, 0(s1)
addi r1, r1, -4
bne 2add1, r7, Loop
First calculate the address

4-wide Dynamic Superscalar Memory

Register Alias Table

2add1 1add1 2 3

Register File

Instruction Window

addi r1, r1, -4
sw r2, 0(s1)
addu r2, r2, r5
lw r2, 0(s1)

Ld/St Queue

Ld/St

lw r2, 0(s1)
sw r2, 0(s1)
addu r2, ldst1, r5

Commit Buffer

bne 2add1, r7, Loop
addi r1, r1, -4
addu r2, ldst1, r5

1Add

addi r1, r1, -4
sw r2, 0(s1)
addu r2, ldst1, r5

2Add

2add1

3Add
4-wide Dynamic Superscalar Memory

Loop:

```
lw  r2, 0(r1)  
addu r2, r2, r5  
sw  r2, 0(r1)  
addi r1, r1, -4  
bne  r1, r7, Loop  
```

Ld/St Queue checks memory addresses – out of order lw/sw

Window

Register Alias Table

Ld/St Queue checks memory addresses – out of order lw/sw
4-wide Dynamic Superscalar Commit

Loop:  
- `lw r2, 0(r1)`  
- `addu r2, r2, r5`  
- `sw r2, 0(r1)`  
- `addi r1, r1, -4`  
- `bne r1, r7, Loop`

**KEY**
- Waiting for value
- Reading value

Instructions wait until all previous instructions have completed
Fallacies & Pitfalls

- Pipelining is easy
- Pipelining ideas are good ideas regardless of technology
- Instruction set has no impact on pipelining
Fallacies & Pitfalls

- Pipelining is easy
  - Verification is difficult
- Pipelining ideas are good ideas regardless of technology
- Instruction set has no impact on pipelining
Fallacies & Pitfalls

- Pipelining is easy
  - Verification is difficult
- Pipelining ideas are good ideas regardless of technology
  - Only recently, with extra chip space, has branch prediction become better than delay slots
- Instruction set has no impact on pipelining
Fallacies & Pitfalls

• Pipelining is easy
  - Verification is difficult

• Pipelining ideas are good ideas regardless of technology
  - Only recently, with extra chip space, has branch prediction become better than delay slots

• Instruction set has no impact on pipelining
  - Complicated addressing & variable lengths instructions complicate pipelining immensely