This Unit: Static & Dynamic Scheduling

- Pipelining and superscalar review
- Code scheduling
  - To reduce pipeline stalls
  - To increase ILP (insn level parallelism)
- Two approaches
  - Static scheduling by the compiler
  - Dynamic scheduling by the hardware

Readings

- Textbook (MA:FSPTCM)
  - Sections 3.3.1 – 3.3.4 (but not "Sidebar:" )
  - Sections 5.0-5.2, 5.3.3, 5.4, 5.5
- Paper
  - “Memory Dependence Prediction using Store Sets” by Chrysos & Emer

Pipelining Review

- Increases clock frequency by staging instruction execution
- “Scalar” pipelines have a best-case CPI of 1
- Challenges:
  - Data and control dependencies further worsen CPI
  - Data: With full bypassing, load-to-use stalls
  - Control: use branch prediction to mitigate penalty
- Big win, done by all processors today
- How many stages (depth)?
  - Five stages is pretty good minimum
  - Intel Pentium II/III: 12 stages
  - Intel Pentium 4: 22+ stages
  - Intel Core 2: 14 stages
Pipeline Diagram

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>add $3$,$2$,$1$</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw $4$,$4$,$(3)$</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi $6$,$4$,$1$</td>
<td>F</td>
<td>D</td>
<td>d*</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub $8$,$3$,$1$</td>
<td>F</td>
<td>p*</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Use compiler scheduling to reduce load-use stall frequency

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>add $3$,$2$,$1$</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw $4$,$4$,$(3)$</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi $6$,$4$,$1$</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub $8$,$3$,$1$</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- “d*” is data dependency, “s*” is structural hazard, “p*” is propagation hazard (only n instructions per stage)

Superscalar Pipeline Diagrams - Ideal

<table>
<thead>
<tr>
<th>scalar</th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw 0(r1)</td>
<td>r2</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 4(r1)</td>
<td>r3</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 8(r1)</td>
<td>r4</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r14,r15,r6</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r17,r16,r8</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 0(r18)</td>
<td>r9</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2-way superscalar</th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw 0(r1)</td>
<td>r2</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 4(r1)</td>
<td>r3</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 8(r1)</td>
<td>r4</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r14,r15,r6</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r17,r16,r8</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 0(r18)</td>
<td>r9</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Superscalar Pipeline Diagrams - Realistic

<table>
<thead>
<tr>
<th>scalar</th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw 0(r1)</td>
<td>r2</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 4(r1)</td>
<td>r3</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 8(r1)</td>
<td>r4</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r14,r15,r6</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r17,r16,r8</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 0(r8)</td>
<td>r9</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2-way superscalar</th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw 0(r1)</td>
<td>r2</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 4(r1)</td>
<td>r3</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 8(r1)</td>
<td>r4</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r14,r15,r6</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r17,r16,r8</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw 0(r8)</td>
<td>r9</td>
<td>F</td>
<td>d*</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Code Scheduling

- Scheduling: act of finding independent instructions
  - "Static" done at compile time by the compiler (software)
  - "Dynamic" done at runtime by the processor (hardware)

- Why schedule code?
  - Scalar pipelines: fill in load-to-use delay slots to improve CPI
  - Superscalar: place independent instructions together
    - As above, load-to-use delay slots
    - Allow multiple-issue decode logic to let them execute at the same time

Compiler Scheduling

- Compiler can schedule (move) instructions to reduce stalls
  - Basic pipeline scheduling: eliminate back-to-back load-use pairs
  - Example code sequence: \( a = b + c; \quad d = f - e; \)
    - sp stack pointer, sp+0 is "a", sp+4 is "b", etc...

Compiler Scheduling Requires

- Large scheduling scope
  - Independent instruction to put between load-use pairs
    + Original example: large scope, two independent computations
    - This example: small scope, one computation

- One way to create larger scheduling scopes?
  - Loop unrolling

Before

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld r2,4(sp)</td>
<td>ld r2,4(sp)</td>
<td></td>
</tr>
<tr>
<td>ld r3,8(sp)</td>
<td>ld r3,8(sp)</td>
<td></td>
</tr>
<tr>
<td>add r3,r2,r1</td>
<td>add r3,r2,r1</td>
<td></td>
</tr>
<tr>
<td>st r1,0(sp)</td>
<td>st r1,0(sp)</td>
<td></td>
</tr>
</tbody>
</table>

Wrong!

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld r2,4(sp)</td>
<td>ld r2,4(sp)</td>
<td></td>
</tr>
<tr>
<td>ld r1,8(sp)</td>
<td>ld r1,8(sp)</td>
<td></td>
</tr>
<tr>
<td>add r1,r2,r1</td>
<td>add r1,r2,r1</td>
<td></td>
</tr>
<tr>
<td>st r1,0(sp)</td>
<td>st r1,0(sp)</td>
<td></td>
</tr>
<tr>
<td>sub r2,r1,r1</td>
<td>sub r2,r1,r1</td>
<td></td>
</tr>
<tr>
<td>st r1,12(sp)</td>
<td>st r1,12(sp)</td>
<td></td>
</tr>
</tbody>
</table>
Compiler Scheduling Requires

- **Alias analysis**
  - Ability to tell whether load/store reference same memory locations
  - Effectively, whether load/store can be rearranged
  - Example code: easy, all loads/stores use same base register (sp)
  - New example: can compiler tell that r8 != sp?
  - Must be **conservative**

### Before

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Before</th>
<th>Wrong(?)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld r2,4(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld r3,8(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r3,r2,r1 //stall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>st r1,0(sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld r5,0(r8)</td>
<td></td>
<td>//does r8==sp?</td>
</tr>
<tr>
<td>ld r6,4(r8)</td>
<td></td>
<td>//does r8+4==sp?</td>
</tr>
<tr>
<td>sub r5,r6,r4 //stall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>st r4,8(r8)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Code Example: SAXPY

- **SAXPY** (Single-precision A X Plus Y)
  - Linear algebra routine (used in solving systems of equations)
  - Part of early "Livermore Loops" benchmark suite
  - Uses floating point values in "F" registers
  - Uses floating point version of instructions (ldf, addf, mulf, stf, etc.)

```plaintext
for (i=0;i<N;i++)
    Z[i]=(A*X[i])+Y[i];
```  

<table>
<thead>
<tr>
<th>Instruction</th>
<th>0: ldf X(r1)⇒f1</th>
<th>1: mulf f0,f1⇒f2</th>
<th>2: ldf Y(r1)⇒f3</th>
<th>3: addf f2,f3⇒f4</th>
<th>4: stf f4⇒Z(r1)</th>
<th>5: addi r1,4⇒r1</th>
<th>6: blt r1,r2,0</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>d*</td>
<td>E*</td>
<td>E*</td>
<td>E*</td>
<td>W</td>
<td>E*</td>
</tr>
<tr>
<td>F</td>
<td>p*</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>d*</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>d*</td>
<td>E+</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td>p*</td>
<td>p*</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
</tr>
</tbody>
</table>

### New Metric: Utilization

- **Utilization**: actual performance / peak performance
  - Important metric for performance/cost
  - No point to paying for hardware you will rarely use
  - Adding hardware usually improves performance & reduces utilization
    - Additional hardware can only be exploited some of the time
    - Diminishing marginal returns
  - Compiler can help make better use of existing hardware
    - Important for superscalar

### SAXPY Performance and Utilization

<table>
<thead>
<tr>
<th>Instruction</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldf X(r1)⇒f1</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>d*</td>
<td>E*</td>
<td>E*</td>
<td>E*</td>
<td>E*</td>
<td>W</td>
<td>E*</td>
<td>E*</td>
<td>E*</td>
<td>E*</td>
<td>E*</td>
<td>E*</td>
<td>E*</td>
</tr>
<tr>
<td>mulf f0,f1⇒f2</td>
<td>F</td>
<td>D</td>
<td>d*</td>
<td>E*</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>d*</td>
<td>d*</td>
<td>E+</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>p*</td>
<td>D</td>
</tr>
<tr>
<td>ldf Y(r1)⇒f3</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>d*</td>
<td>d*</td>
<td>E+</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>p*</td>
<td>D</td>
</tr>
<tr>
<td>addf f2,f3⇒f4</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>p*</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>d*</td>
<td>d*</td>
<td>E+</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>p*</td>
</tr>
<tr>
<td>stf f4⇒Z(r1)</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>p*</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>d*</td>
<td>d*</td>
<td>E+</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>p*</td>
</tr>
<tr>
<td>addi r1,4⇒r1</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>p*</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>d*</td>
<td>d*</td>
<td>E+</td>
<td>W</td>
<td>F</td>
<td>p*</td>
<td>p*</td>
</tr>
<tr>
<td>blt r1,r2,0</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
</tr>
</tbody>
</table>

- **Scalar pipeline**
  - Full bypassing, 5-cycle E*, 2-cycle E+, branches predicted taken
  - Single iteration (7 insns) latency: 16–5 = 11 cycles
  - **Performance**: 7 insns / 11 cycles = 0.64 IPC
  - **Utilization**: 0.64 actual IPC / 1 peak IPC = 64%
SAXPY Performance and Utilization

- 2-way superscalar pipeline
  - Any two insns per cycle + split integer and floating point pipelines
  - Performance: 7 insns / 10 cycles = 0.70 IPC
  - Utilization: 0.70 actual IPC / 2 peak IPC = 35%
  - More hazards → more stalls
  - Each stall is more expensive

Static (Compiler) Instruction Scheduling

- Idea: place independent insns between slow ops and uses
  - Otherwise, pipeline stalls while waiting for RAW hazards to resolve
  - Have already seen pipeline scheduling

- To schedule well you need ... independent insns
- Scheduling scope: code region we are scheduling
  - The bigger the better (more independent insns to choose from)
  - Once scope is defined, schedule is pretty obvious
  - Trick is creating a large scope (must schedule across branches)

- Scope enlarging techniques
  - Loop unrolling
  - Others: “superblocks”, “hyperblocks”, “trace scheduling”, etc.

Unrolling SAXPY I: Fuse Iterations

- Combine two (in general K) iterations of loop
  - Fuse loop control: induction variable \( i \) increment + branch
  - Adjust (implicit) induction uses: constants → constants + 4

Unrolling SAXPY II: Schedule two or more iterations together

- Fuse iterations
- Schedule to reduce stalls
- Schedule introduces ordering problems, rename registers to fix

Loop Unrolling SAXPY

- Goal: separate dependent insns from one another
- SAXPY problem: not enough flexibility within one iteration
  - Longest chain of insns is 9 cycles
    - Load (1)
    - Forward to multiply (5)
    - Forward to add (2)
    - Forward to store (1)
      - Can't hide a 9-cycle chain using only 7 insns
    - But how about two 9-cycle chains using 14 insns?
- Loop unrolling: schedule two or more iterations together
  - Fuse iterations
  - Schedule to reduce stalls
  - Schedule introduces ordering problems, rename registers to fix
Unrolling SAXPY II: Pipeline Schedule

- Pipeline schedule to reduce stalls
  - Have already seen this: pipeline scheduling

Unrolled SAXPY Performance/Utilization

- Performance: 12 insn / 13 cycles = 0.92 IPC
- Utilization: 0.92 actual IPC / 1 peak IPC = 92%
- Speedup: (2 * 11 cycles) / 13 cycles = 1.69

Loop Unrolling Shortcomings

- Static code growth → more I$ misses (limits degree of unrolling)
- Needs more registers to hold values (ISA limits this)
- Doesn’t handle non-loops
- Doesn’t handle inter-iteration dependences

for (i=0; i<N; i++)
X[i]=A*X[i-1];

Unrolling SAXPY III: “Rename” Registers

- Pipeline scheduling causes reordering violations
  - Adjust register names to correct

Loop Unrolling Shortcomings

- Two mulf’s are not parallel
- Other (more advanced) techniques help
Another Limitation: Branches

r1 and r2 are inputs
loop:
  jz r1, not_found
  ld [r1+0] -> r3
  sub r2, r3 -> r4
  jz r4, found
  ld [r1+4] -> r1
  jmp loop

Aside: what does this code do?

Legal to move load up past branch?

Can Hardware Overcome These Limits?

- **Dynamically-scheduled processors**
  - Also called "out-of-order" processors
  - Hardware re-schedules insns...
  - ...within a sliding window of VonNeumann insns
  - As with pipelining and superscalar, ISA unchanged
    - Same hardware/software interface, appearance of in-order
- Increases scheduling scope
  - Does loop unrolling transparently
  - Uses branch prediction to "unroll" branches
- Examples:
  - Pentium Pro/II/III (3-wide), Core 2 (4-wide), Alpha 21264 (4-wide), MIPS R10000 (4-wide), Power5 (5-wide)
- Basic overview of approach

Summary: Static Scheduling Limitations

- Limited number of registers (set by ISA)
- Scheduling scope
  - Example: can't generally move memory operations past branches
- Inexact memory aliasing information
  - Often prevents reordering of loads above stores
- Caches misses (or any runtime event) confound scheduling
  - How can the compiler know which loads will miss vs hit?
  - Can impact the compiler's scheduling decisions

The Problem With In-Order Pipelines

* add f 0, f 1 → f 2 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  D  E+  E+  W
  * mulf f 2, f 3 → f 4  D  d*  d*  E+  E+  E+  E+  W
  * subf f 0, f 1 → f 4  F  p*  p*  D  E+  E+  E+  W

- What's happening in cycle 4?
  - mulf stalls due to **data dependence**
    - OK, this is a fundamental problem
  - subf stalls due to **pipeline (propagation) hazard**
    - Why? subf can't proceed into D because mulf is there
    - That is the only reason, and it isn't a fundamental one
    - Maintaining in-order writes to register file
- Why can't subf go into D in cycle 4 and E+ in cycle 5?
Out-of-order Pipeline

- Fetch
- Decode
- Rename
- Dispatch
- Issue
- Reg-read
- Execute
- Writeback
- Commit

In-order front end

Out-of-order execution

**Code Example**

- Code:

  ```
  add r2, r3, r1
  sub r2, r1, r3
  mul r2, r3, r3
  div r1, 4, r1
  ```

- “True” (real) & “False” (artificial) dependencies
- Divide insn independent of subtract and multiply insns
  - Can execute in parallel with subtract
- Many registers re-used
  - Just as in static scheduling, the register names get in the way
  - How does the hardware get around this?
- Approach: (step #1) rename registers, (step #2) schedule

**Step #1: Register Renaming**

- To eliminate register conflicts/hazards
- “Architected” vs “Physical” registers – level of indirection
  - Names: r1, r2, r3
  - Locations: p1, p2, p3, p4, p5, p6, p7
  - Original mapping: r1 -> p1, r2 -> p2, r3 -> p3, p4-p7 are “available”

**MapTable | FreeList | Original insns | Renamed insns**

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
<th>p5</th>
<th>p6</th>
<th>p7</th>
<th>add r2, r3, r1</th>
<th>add p2, p3, p4</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>p2</td>
<td>p3</td>
<td>p4</td>
<td>p5</td>
<td>p6</td>
<td>p7</td>
<td>sub r2, r1, r3</td>
<td>sub p2, p3, p5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>p2</td>
<td>p5</td>
<td>p6</td>
<td>p7</td>
<td>mul r2, r3, r3</td>
<td>mul p2, p6, p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>p2</td>
<td>p6</td>
<td>p7</td>
<td>div r1, 4, r1</td>
<td>div p4, 4, p7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Renaming – conceptually write each register once
  - Removes false dependencies
  - Leaves true dependencies intact!
- When to reuse a physical register? After overwriting insn done

**Step #2: Dynamic Scheduling**

- Instructions fetch/decoded/renamed into Instruction Buffer
  - Also called “instruction window” or “instruction scheduler”
- Instructions (conceptually) check ready bits every cycle
  - Execute when ready
**Register Renaming Algorithm**

- Data structures:
  - `maptable[architectural_reg] ➔ physical_reg`
  - Free list: get/put free register (implemented as a queue)

- Algorithm: at decode for each instruction:
  
  ```
  insn.phys_input1 = maptable[insn.arch_input1]
  insn.phys_input2 = maptable[insn.arch_input2]
  insn.phys_to_free = maptable[arch_output]
  new_reg = get_free_phys_reg()
  maptable[arch_output] = new_reg
  insn.phys_output = new_reg
  ```

- At "commit"
  - Once all older instructions have committed, free register
  ```
  put_free_phys_reg(insn.phys_to_free)
  ```

---

**Renaming example**

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

<table>
<thead>
<tr>
<th>r1</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td>r3</td>
<td>p3</td>
</tr>
<tr>
<td>r4</td>
<td>p4</td>
</tr>
<tr>
<td>r5</td>
<td>p5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Map table

<table>
<thead>
<tr>
<th>p6</th>
</tr>
</thead>
<tbody>
<tr>
<td>p7</td>
</tr>
<tr>
<td>p8</td>
</tr>
<tr>
<td>p9</td>
</tr>
<tr>
<td>p10</td>
</tr>
</tbody>
</table>

Free-list

---

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
xor p1 ^ p2 ->
```

<table>
<thead>
<tr>
<th>r1</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td>r3</td>
<td>p3</td>
</tr>
<tr>
<td>r4</td>
<td>p4</td>
</tr>
<tr>
<td>r5</td>
<td>p5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Map table

<table>
<thead>
<tr>
<th>p6</th>
</tr>
</thead>
<tbody>
<tr>
<td>p7</td>
</tr>
<tr>
<td>p8</td>
</tr>
<tr>
<td>p9</td>
</tr>
<tr>
<td>p10</td>
</tr>
</tbody>
</table>

Free-list
Renaming example

 xor r1 ^ r2 -> r3
 add r3 + r4 -> r4
 sub r5 - r2 -> r3
 addi r3 + 1 -> r1

 xor p1 ^ p2 -> p6

Map table

<table>
<thead>
<tr>
<th>r1</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td>r3</td>
<td>p3</td>
</tr>
<tr>
<td>r4</td>
<td>p4</td>
</tr>
<tr>
<td>r5</td>
<td>p5</td>
</tr>
</tbody>
</table>

Free-list

<table>
<thead>
<tr>
<th>p6</th>
</tr>
</thead>
<tbody>
<tr>
<td>p7</td>
</tr>
<tr>
<td>p8</td>
</tr>
<tr>
<td>p9</td>
</tr>
<tr>
<td>p10</td>
</tr>
</tbody>
</table>

CIS 501 (Martin): Scheduling

37
Renaming example

xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1

xor p1 ^ p2 -> p6
add p6 + p4 -> p7

xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1

xor p1 ^ p2 -> p6
add p6 + p4 -> p7

Map table

r1 p1
r2 p2
r3 p6
r4 p7
r5 p5

Free-list

p8
p9
p10

CIS 501 (Martin): Scheduling

41
Renaming example

\[
\begin{align*}
\text{xor } r1 \land r2 &\rightarrow r3 \\
\text{add } r3 + r4 &\rightarrow r4 \\
\text{sub } r5 - r2 &\rightarrow r3 \\
\text{addi } r3 + 1 &\rightarrow r1 \\
\text{xor } p1 \land p2 &\rightarrow p6 \\
\text{add } p6 + p4 &\rightarrow p7 \\
\text{sub } p5 - p2 &\rightarrow p8 \\
\text{addi } p8 + 1 &\rightarrow p9
\end{align*}
\]

Out-of-order Pipeline

\[
\begin{align*}
\text{xor } r1 \land r2 &\rightarrow r3 \\
\text{add } r3 + r4 &\rightarrow r4 \\
\text{sub } r5 - r2 &\rightarrow r3 \\
\text{addi } r3 + 1 &\rightarrow r1 \\
\text{xor } p1 \land p2 &\rightarrow p6 \\
\text{add } p6 + p4 &\rightarrow p7 \\
\text{sub } p5 - p2 &\rightarrow p8 \\
\text{addi } p8 + 1 &\rightarrow p9
\end{align*}
\]

Have unique register names
Now put into out-of-order execution structures
**Dynamic Scheduling**

- Renamed instructions into out-of-order structures
  - Re-order buffer (ROB)
    - All instruction until commit
  - Issue Queue
    - Un-executed instructions
    - Central piece of scheduling logic
    - Content Addressable Memory (CAM)

**RAM vs CAM**

- Random Access Memory
  - Read/write specific index
  - Get/set value there
- Content Addressable Memory
  - Search for a value (send value to all entries)
  - Find matching indices (use comparator at each entry)
  - Output: one bit per entry (multiple match)
- One structure can have ports of both types

**RAM vs CAM: RAM**

- Read index 4
- RAM: read/write specific index
RAM vs CAM: CAM

Find value “17”

CAM: search for value

17
22
47
17
19
12
13
42

Index 0
Index 3

Issue Queue

- Holds un-executed instructions
- Tracks ready inputs
  - Physical register names + ready bit
  - “AND” bits to tell if ready

Dispatch Steps

- Allocate IQ slot
  - Full? Stall
- Read ready bits of inputs
  - Table 1-bit per physical reg
- Clear ready bit of output in table
  - Instruction has not produced value yet
- Write data in IQ slot

Dispatch Example

xor p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9

Ready bits

p1 y
p2 y
p3 y
p4 y
p5 y
p6 y
p7 y
p8 y
p9 y
### Dispatch Example

**Ready bits**
- p1 y
- p2 y
- p3 y
- p4 y
- p5 y
- p6 n
- p7 y
- p8 y
- p9 y

<table>
<thead>
<tr>
<th>Insn</th>
<th>Inp1</th>
<th>R</th>
<th>Inp2</th>
<th>R</th>
<th>Dst</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor</td>
<td>p1</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p6</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>p6</td>
<td>n</td>
<td>p4</td>
<td>y</td>
<td>p7</td>
<td>1</td>
</tr>
<tr>
<td>sub</td>
<td>p5</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p8</td>
<td>2</td>
</tr>
</tbody>
</table>

**Issue Queue**

- xor p1 ^ p2 -> p6
- add p6 + p4 -> p7
- sub p5 - p2 -> p8
- addi p8 + 1 -> p9

<table>
<thead>
<tr>
<th>Insn</th>
<th>Inp1</th>
<th>R</th>
<th>Inp2</th>
<th>R</th>
<th>Dst</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor</td>
<td>p1</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p6</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>p6</td>
<td>n</td>
<td>p4</td>
<td>y</td>
<td>p7</td>
<td>1</td>
</tr>
<tr>
<td>sub</td>
<td>p5</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p8</td>
<td>2</td>
</tr>
</tbody>
</table>

### Dispatch Example

**Ready bits**
- p1 y
- p2 y
- p3 y
- p4 y
- p5 y
- p6 n
- p7 n
- p8 y
- p9 y

<table>
<thead>
<tr>
<th>Insn</th>
<th>Inp1</th>
<th>R</th>
<th>Inp2</th>
<th>R</th>
<th>Dst</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor</td>
<td>p1</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p6</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>p6</td>
<td>n</td>
<td>p4</td>
<td>y</td>
<td>p7</td>
<td>1</td>
</tr>
<tr>
<td>sub</td>
<td>p5</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p8</td>
<td>2</td>
</tr>
</tbody>
</table>

**Issue Queue**

- xor p1 ^ p2 -> p6
- add p6 + p4 -> p7
- sub p5 - p2 -> p8
- addi p8 + 1 -> p9
Out-of-order pipeline

- Execution (out-of-order) stages
- **Select** ready instructions
  - Send for execution
- **Wakeup** dependents

### Dynamic Scheduling/Issue Algorithm

- **Issue** = Select + Wakeup

#### Data structures:
- Ready table[phys_reg] ➔ yes/no (part of issue queue)

#### Algorithm at “schedule” stage (prior to read registers):
```plaintext
foreach instruction:
  if table[insn.phys_input1] == ready &&
      table[insn.phys_input2] == ready then
    insn is “ready”
  select the oldest “ready” instruction
  table[insn.phys_output] = ready
```

#### Issue = Select + Wakeup

- **Select** N oldest, ready instructions
  - “xor” is the oldest ready instruction below
  - “xor” and “sub” are the two oldest ready instructions below
- **Wakeup** dependent instructions
  - CAM search for Dst in inputs
  - Set ready
  - Also update ready-bit table for future instructions

#### Ready bits

<table>
<thead>
<tr>
<th>Insn</th>
<th>Inp1</th>
<th>R</th>
<th>Inp2</th>
<th>R</th>
<th>Dst</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor</td>
<td>p1</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p6</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>p6</td>
<td>n</td>
<td>p4</td>
<td>y</td>
<td>p7</td>
<td>1</td>
</tr>
<tr>
<td>sub</td>
<td>p5</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p8</td>
<td>2</td>
</tr>
<tr>
<td>addi</td>
<td>p8</td>
<td>n</td>
<td>---</td>
<td>y</td>
<td>p9</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Insn</th>
<th>Inp1</th>
<th>R</th>
<th>Inp2</th>
<th>R</th>
<th>Dst</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor</td>
<td>p1</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p6</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>p6</td>
<td>y</td>
<td>p4</td>
<td>y</td>
<td>p7</td>
<td>1</td>
</tr>
<tr>
<td>sub</td>
<td>p5</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p8</td>
<td>2</td>
</tr>
<tr>
<td>addi</td>
<td>p8</td>
<td>y</td>
<td>---</td>
<td>y</td>
<td>p9</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Insn</th>
<th>Inp1</th>
<th>R</th>
<th>Inp2</th>
<th>R</th>
<th>Dst</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor</td>
<td>p1</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p6</td>
<td>0</td>
</tr>
<tr>
<td>add</td>
<td>p6</td>
<td>y</td>
<td>p4</td>
<td>y</td>
<td>p7</td>
<td>1</td>
</tr>
<tr>
<td>sub</td>
<td>p5</td>
<td>y</td>
<td>p2</td>
<td>y</td>
<td>p8</td>
<td>2</td>
</tr>
<tr>
<td>addi</td>
<td>p8</td>
<td>y</td>
<td>---</td>
<td>y</td>
<td>p9</td>
<td>3</td>
</tr>
</tbody>
</table>
**Issue**

- **Select/Wakeup** one cycle
- Dependents go back to back
  - Next cycle: add/addi are ready:

<table>
<thead>
<tr>
<th>Insn</th>
<th>Inp1</th>
<th>R</th>
<th>Inp2</th>
<th>R</th>
<th>Dst</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>p6</td>
<td>y</td>
<td>p4</td>
<td>y</td>
<td>p7</td>
<td>1</td>
</tr>
<tr>
<td>addi</td>
<td>p8</td>
<td>y</td>
<td>---</td>
<td>y</td>
<td>p9</td>
<td>3</td>
</tr>
</tbody>
</table>

**When Does Register Read Occur?**

- Option #1: after select, right before execute
  - *(Not at decode)*
  - Read **physical** register (renamed)
  - Or get value via bypassing (based on physical register name)
  - This is Pentium 4, MIPS R10k, Alpha 21264 style,
    Intel’s “Sandy Bridge” due out in 2011
  - Physical register file may be large
    - Multi-cycle read

- Option #2: as part of issue, keep **values in Issue Queue**
  - Pentium Pro, Core 2, Core i7

**Renaming review**

Everyone rename this instruction:

```plaintext
mul r4 * r5 -> r1
```

<table>
<thead>
<tr>
<th>r1</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td>r3</td>
<td>p3</td>
</tr>
<tr>
<td>r4</td>
<td>p4</td>
</tr>
<tr>
<td>r5</td>
<td>p5</td>
</tr>
</tbody>
</table>

Map table

<table>
<thead>
<tr>
<th>p6</th>
<th>p7</th>
</tr>
</thead>
<tbody>
<tr>
<td>p8</td>
<td>p9</td>
</tr>
</tbody>
</table>

Free-list

**Dispatch Review**

Everyone dispatch this instruction:

```plaintext
div p7 / p6 -> p1
```

<table>
<thead>
<tr>
<th>Ready bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1 y</td>
</tr>
<tr>
<td>p2 y</td>
</tr>
<tr>
<td>p3 y</td>
</tr>
<tr>
<td>p4 y</td>
</tr>
<tr>
<td>p5 y</td>
</tr>
<tr>
<td>p6 n</td>
</tr>
<tr>
<td>p7 y</td>
</tr>
<tr>
<td>p8 y</td>
</tr>
<tr>
<td>p9 y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Insn</th>
<th>Inp1</th>
<th>R</th>
<th>Inp2</th>
<th>R</th>
<th>Dst</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Determine which instructions are ready.
Which will be issued on a 1-wide machine?
Which will be issued on a 2-wide machine?

What information will change if we issue the add?

OOO execution (2-wide)
add p6 + p4 \rightarrow p7
addi p8 + 1 \rightarrow p9
xor 7 \text{^} 3 \rightarrow p6
sub 6 - 3 \rightarrow p8
add _ + 9 \rightarrow p7
addi _ +1 \rightarrow p9
4 \rightarrow p6
3 \rightarrow p8
13 \rightarrow p7
4 \rightarrow p9
4 \rightarrow p9
p7 13
p8 3
p9 4
**OOO execution (2-wide)**

Note similarity to in-order

<table>
<thead>
<tr>
<th>Instruction</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p6</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p7</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p8</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p9</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Multi-cycle operations**

- Multi-cycle ops (load, fp, multiply, etc)
  - Wakeup deferred a few cycles
    - Structural hazard?
- Cache misses?
  - Speculative wake-up (assume hit)
  - Cancel exec of dependents
  - Re-issue later
  - Details: complicated, not important

**Re-order Buffer (ROB)**

- All instructions in order
- Two purposes
  - Misprediction recovery
  - In-order commit
    - Maintain appearance of in-order execution
    - Freeing of physical registers

**RENAMEING REVISITED**
Renaming revisited

- Overwritten register
- Freed at commit
- Restore in map table on recovery
  - Branch mis-prediction recovery
  - Also must be read at rename

Renaming example

xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1

| r1 | p1 | p6 |
| r2 | p2 | p7 |
| r3 | p3 | p8 |
| r4 | p4 | p9 |
| r5 | p5 | p10 |

Map table | Free-list

Renaming example

xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1

xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1

| r1 | p1 | p6 |
| r2 | p2 | p7 |
| r3 | p3 | p6 |
| r4 | p4 | p9 |
| r5 | p5 | p10 |

Map table | Free-list
### Renaming example

**CIS 501 (Martin): Scheduling**

$xor\ r1 \land r2 \rightarrow r3$

$xor\ p1 \land p2 \rightarrow p6$

$add\ r3 + r4 \rightarrow r4$

$add\ p6 + p4 \rightarrow p7$

$addi\ r3 + 1 \rightarrow r1$

**Map table**

<table>
<thead>
<tr>
<th>r1</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td>r3</td>
<td>p6</td>
</tr>
<tr>
<td>r4</td>
<td>p4</td>
</tr>
<tr>
<td>r5</td>
<td>p5</td>
</tr>
</tbody>
</table>

**Free-list**

<table>
<thead>
<tr>
<th>p7</th>
</tr>
</thead>
<tbody>
<tr>
<td>p8</td>
</tr>
<tr>
<td>p9</td>
</tr>
</tbody>
</table>

### Renaming example

**CIS 501 (Martin): Scheduling**

$xor\ r1 \land r2 \rightarrow r3$

$xor\ p1 \land p2 \rightarrow p6$

$add\ r3 + r4 \rightarrow r4$

$add\ p6 + p4 \rightarrow p7$

$addi\ r3 + 1 \rightarrow r1$

**Map table**

<table>
<thead>
<tr>
<th>r1</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td>r3</td>
<td>p6</td>
</tr>
<tr>
<td>r4</td>
<td>p7</td>
</tr>
<tr>
<td>r5</td>
<td>p5</td>
</tr>
</tbody>
</table>

**Free-list**

<table>
<thead>
<tr>
<th>p8</th>
</tr>
</thead>
<tbody>
<tr>
<td>p9</td>
</tr>
<tr>
<td>p10</td>
</tr>
</tbody>
</table>

### Renaming example

**CIS 501 (Martin): Scheduling**

$xor\ r1 \land r2 \rightarrow r3$

$xor\ p1 \land p2 \rightarrow p6$

$add\ r3 + r4 \rightarrow r4$

$add\ p6 + p4 \rightarrow p7$

$addi\ r3 + 1 \rightarrow r1$

**Map table**

<table>
<thead>
<tr>
<th>r1</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td>r3</td>
<td>p6</td>
</tr>
<tr>
<td>r4</td>
<td>p7</td>
</tr>
<tr>
<td>r5</td>
<td>p5</td>
</tr>
</tbody>
</table>

**Free-list**

<table>
<thead>
<tr>
<th>p8</th>
</tr>
</thead>
<tbody>
<tr>
<td>p9</td>
</tr>
<tr>
<td>p10</td>
</tr>
</tbody>
</table>
## Renaming example

<table>
<thead>
<tr>
<th>r1</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td>r3</td>
<td>p8</td>
</tr>
<tr>
<td>r4</td>
<td>p7</td>
</tr>
<tr>
<td>r5</td>
<td>p5</td>
</tr>
</tbody>
</table>

**Map table**

<table>
<thead>
<tr>
<th>r1</th>
<th>p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td>r3</td>
<td>p8</td>
</tr>
<tr>
<td>r4</td>
<td>p7</td>
</tr>
<tr>
<td>r5</td>
<td>p5</td>
</tr>
</tbody>
</table>

**Free-list**

- xor r1 ^ r2 -> r3
- add r3 + r4 -> r4
- sub r5 - r2 -> r3
- addi r3 + 1 -> r1

- xor p1 ^ p2 -> p6
- add p6 + p4 -> p7
- sub p5 - p2 -> p8
- addi p8 + 1 -> p1

- xor r1 ^ r2 -> r3
- add r3 + r4 -> r4
- sub r5 - r2 -> r3
- addi r3 + 1 -> r1

- xor p1 ^ p2 -> p6
- add p6 + p4 -> p7
- sub p5 - p2 -> p8
- addi p8 + 1 -> p1

### ROB

- ROB entry holds all info for recover/commit
  - Logical register names
  - Physical register names
  - Instruction types
- Dispatch: insert at tail
  - Full? Stall
- Commit: remove from head
  - Not completed? Stall

### Recovery

- Completely remove wrong path instructions
  - Flush from IQ
  - Remove from ROB
  - Restore map table to before misprediction
  - Free destination registers
### Recovery example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Map table</th>
<th>Free-list</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bnz r1 loop</code></td>
<td>r1</td>
<td>p9</td>
</tr>
<tr>
<td><code>xor r1 ^ r2 -&gt; r3</code></td>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td><code>add r3 + r4 -&gt; r4</code></td>
<td>r3</td>
<td>p8</td>
</tr>
<tr>
<td><code>sub r5 - r2 -&gt; r3</code></td>
<td>r4</td>
<td>p7</td>
</tr>
<tr>
<td><code>addi r3 + 1 -&gt; r1</code></td>
<td>r5</td>
<td>p5</td>
</tr>
</tbody>
</table>

### Recovery example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Map table</th>
<th>Free-list</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bnz r1 loop</code></td>
<td>p1</td>
<td>p10</td>
</tr>
<tr>
<td><code>xor r1 ^ r2 -&gt; r3</code></td>
<td>p2</td>
<td></td>
</tr>
<tr>
<td><code>add r3 + r4 -&gt; r4</code></td>
<td>p4</td>
<td></td>
</tr>
<tr>
<td><code>sub r5 - r2 -&gt; r3</code></td>
<td>p8</td>
<td></td>
</tr>
<tr>
<td><code>addi r3 + 1 -&gt; r1</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Recovery example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Map table</th>
<th>Free-list</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bnz r1 loop</code></td>
<td>r1</td>
<td>p1</td>
</tr>
<tr>
<td><code>xor r1 ^ r2 -&gt; r3</code></td>
<td>r2</td>
<td>p2</td>
</tr>
<tr>
<td><code>add r3 + r4 -&gt; r4</code></td>
<td>r3</td>
<td>p6</td>
</tr>
<tr>
<td><code>sub r5 - r2 -&gt; r3</code></td>
<td>r4</td>
<td>p7</td>
</tr>
<tr>
<td><code>addi r3 + 1 -&gt; r1</code></td>
<td>r5</td>
<td>p5</td>
</tr>
</tbody>
</table>

### Recovery example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Map table</th>
<th>Free-list</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bnz r1 loop</code></td>
<td>p1</td>
<td>p8</td>
</tr>
<tr>
<td><code>xor r1 ^ r2 -&gt; r3</code></td>
<td>p2</td>
<td></td>
</tr>
<tr>
<td><code>add r3 + r4 -&gt; r4</code></td>
<td>p4</td>
<td></td>
</tr>
<tr>
<td><code>sub r5 - r2 -&gt; r3</code></td>
<td>p6</td>
<td></td>
</tr>
<tr>
<td><code>addi r3 + 1 -&gt; r1</code></td>
<td>p10</td>
<td></td>
</tr>
</tbody>
</table>

CIS 501 (Martin): Scheduling
Recovery example

bnz r1 loop
xor r1 ^ r2 -> r3

bnz p1, loop
xor p1 ^ p2 -> p6

Map table

Free-list

r1 p1 p6
r2 p2 p7
r3 p3 p8
r4 p4 p9
r5 p5 p10

What about stores

- Stores: Write D$, not registers
  - Can we rename memory?
  - Recover in the cache?

What about stores

- Stores: Write D$, not registers
  - Can we rename memory?
  - Recover in the cache?
- No (at least not easily)
  - Cache writes unrecoverable
  - Stores: only when certain
    - Commit
Commit

- Commit: instruction becomes **architected state**
  - In-order, only when instructions are finished
  - Free over-written register (why?)

Freeing over-written register

- P3 was r3 **before** xor
- P6 is r3 **after** xor
  - Anything older than xor should read p3
  - Anything younger than xor should p6 (until next r3 writing instruction)
- At commit of xor, no older instructions exist

Commit Example

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p9</td>
<td></td>
<td>p2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>p3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r2</td>
<td>p4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p5</td>
<td>p8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Map table

<table>
<thead>
<tr>
<th>p10</th>
</tr>
</thead>
</table>

Free-list

<table>
<thead>
<tr>
<th>[ p3 ]</th>
</tr>
</thead>
</table>

Commit Example

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p9</td>
<td></td>
<td>p2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r2</td>
<td>p4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p5</td>
<td>p8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Map table

<table>
<thead>
<tr>
<th>p10</th>
</tr>
</thead>
</table>

Free-list

<table>
<thead>
<tr>
<th>[ p3 ]</th>
</tr>
</thead>
</table>

Commit Example

<table>
<thead>
<tr>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p9</td>
<td></td>
<td>p2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r2</td>
<td>p4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p5</td>
<td>p8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>p8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Map table

<table>
<thead>
<tr>
<th>p10</th>
</tr>
</thead>
</table>

Free-list

<table>
<thead>
<tr>
<th>[ p3 ]</th>
</tr>
</thead>
</table>
Commit Example

add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1

add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9

[ p4 ]
[ p6 ]
[ p1 ]

r1 p9
r2 p2
r3 p8
r4 p7
r5 p5

[ p4 ]
[ p1 ]

r1 p9
r2 p2
r3 p8
r4 p7
r5 p5

Map table
Free-list

Commit Example

add r3 + 1 -> r1
addi p8 + 1 -> p9

[ p1 ]

r1 p9
r2 p2
r3 p8
r4 p7
r5 p5

[ p1 ]

r1 p9
r2 p2
r3 p8
r4 p7
r5 p5

Map table
Free-list

MORE ON DEPENDENCIES
Dependence types

- **RAW (Read After Write)** = “true dependence”
  
  ```
  mul r0 * r1 -> r2
  ...
  add r2 + r3 -> r4
  ```

- **WAW (Write After Write)** = “output dependence”
  
  ```
  mul r0 * r1 -> r2
  ...
  add r1 + r3 -> r2
  ```

- **WAR (Write After Read)** = “anti-dependence”
  
  ```
  mul r0 * r1 -> r2
  ...
  add r3 + r4 -> r1
  ```

Memory dependences

- If value in “r2” and “r3” is the same...

  ```
  st r1 -> [r2]
  ...
  ld [r3] -> r4
  ```

- **RAW (Read After Write)**
  
  ```
  st r1 -> [r2]
  ...
  ld [r3] -> r4
  ```

- **WAW (Write After Write)**
  
  ```
  st r1 -> [r2]
  ...
  st r4 -> [r3]
  ```

- **WAR (Write After Read)**
  
  ```
  ld [r2] -> r1
  ...
  st r4 -> [r3]
  ```

More on dependences

- **RAW**
  
  - When more than one applies, RAW dominates:
    ```
    add r1 + r2 -> r3
    addi r3 + 1 -> r3
    ```
  - Must be respected: no trick to avoid

- WAR/WAW on registers
  
  - Two things happen to use same name
  - Can be eliminated by renaming

- WAR/WAW on memory
  
  - Can’t rename memory in same way as registers
  - Need to use other tricks (later this lecture)

**MOTIVATING OUT-OF-ORDER EXECUTION**
Limitations of In-Order Pipelines

• In-order pipeline, two-cycle load-use penalty
• 2-wide
• Why not?

Out-of-Order to the Rescue

• Still 2-wide superscalar, but now out-of-order, too
  • Allows instructions to issues when dependences are ready
• Longer pipeline
  • Front end: Fetch, “Dispatch”
  • Execution core: “Issue”, “Reg. Read”, Execute, Memory, Writeback
  • Retirement: “Commit”

OUT-OF-ORDER PIPELINE EXAMPLE
### Out-of-Order Pipeline – Cycle 0

<table>
<thead>
<tr>
<th>Instruction</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ld [r1] -&gt; r2</code></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>add r2 + r3 -&gt; r4</code></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>xor r4 ^ r5 -&gt; r6</code></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ld [r7] -&gt; r4</code></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Map Table**
- r1: p8
- r2: p9
- r3: p6
- r4: p5
- r5: p4
- r6: p3
- r7: p2
- r8: p1

**Ready Table**
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: yes
- p9: yes
- p10: yes
- p11: yes
- p12: yes

**Issue Queue**
- `ld` p6
- `add` p5
- `xor` p4
- `ld` p3
- `add` p2
- `xor` p1

**Reorder Buffer**
- `ld` p8
- `add` p5
- `xor` p4
- `ld` p3
- `add` p2
- `xor` p1

---

### Out-of-Order Pipeline – Cycle 1a

<table>
<thead>
<tr>
<th>Instruction</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ld [r1] -&gt; r2</code></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>add r2 + r3 -&gt; r4</code></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>xor r4 ^ r5 -&gt; r6</code></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ld [r7] -&gt; r4</code></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Map Table**
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: yes
- p9: yes
- p10: yes
- p11: yes
- p12: yes

**Ready Table**
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: yes
- p9: yes
- p10: yes
- p11: yes
- p12: yes

**Issue Queue**
- `ld` p8
- `add` p9
- `xor` p6
- `ld` p4
- `add` p3
- `xor` p2
- `ld` p1

**Reorder Buffer**
- `ld` p7
- `add` p6
- `xor` p5
- `ld` p3
- `add` p2
- `xor` p1

---

### Out-of-Order Pipeline – Cycle 1b

<table>
<thead>
<tr>
<th>Instruction</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ld [r1] -&gt; r2</code></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>add r2 + r3 -&gt; r4</code></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>xor r4 ^ r5 -&gt; r6</code></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ld [r7] -&gt; r4</code></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Map Table**
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: yes
- p9: yes
- p10: yes
- p11: yes
- p12: yes

**Ready Table**
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: yes
- p9: no
- p10: no
- p11: no
- p12: no

**Issue Queue**
- `ld` p8
- `add` p9
- `xor` p6
- `ld` p4
- `add` p3
- `xor` p2
- `ld` p1

**Reorder Buffer**
- `ld` p7
- `add` p6
- `xor` p5
- `ld` p3
- `add` p2
- `xor` p1

---

### Out-of-Order Pipeline – Cycle 1c

<table>
<thead>
<tr>
<th>Instruction</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ld [r1] -&gt; r2</code></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>add r2 + r3 -&gt; r4</code></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>xor r4 ^ r5 -&gt; r6</code></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ld [r7] -&gt; r4</code></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Map Table**
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: yes
- p9: yes
- p10: yes
- p11: yes
- p12: yes

**Ready Table**
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: no
- p9: no
- p10: no
- p11: no
- p12: no

**Issue Queue**
- `ld` p8
- `add` p9
- `xor` p6
- `ld` p4
- `add` p3
- `xor` p2
- `ld` p1

**Reorder Buffer**
- `ld` p7
- `add` p6
- `xor` p5
- `ld` p3
- `add` p2
- `xor` p1

---

CIS 501 (Martin): Scheduling
### Out-of-Order Pipeline – Cycle 4

<table>
<thead>
<tr>
<th>Map Table</th>
<th>Ready Table</th>
<th>Reorder Buffer</th>
<th>Insns</th>
<th>To Free</th>
<th>Done?</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>p8</td>
<td>yes</td>
<td>ld</td>
<td>p7</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>r2</td>
<td>p9</td>
<td>yes</td>
<td>add</td>
<td>p5</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td>p6</td>
<td>yes</td>
<td>xor</td>
<td>p3</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td>p12</td>
<td>yes</td>
<td>ld</td>
<td>p10</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>r5</td>
<td>p4</td>
<td>yes</td>
<td>add</td>
<td>p9</td>
<td>yes</td>
<td>p6</td>
</tr>
<tr>
<td>r6</td>
<td>p11</td>
<td>yes</td>
<td>xor</td>
<td>p10</td>
<td>no</td>
<td>p4</td>
</tr>
<tr>
<td>r7</td>
<td>p2</td>
<td>no</td>
<td>ld</td>
<td>p2</td>
<td>yes</td>
<td>p12</td>
</tr>
<tr>
<td>r8</td>
<td>p1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Out-of-Order Pipeline – Cycle 5b

<table>
<thead>
<tr>
<th>Map Table</th>
<th>Ready Table</th>
<th>Reorder Buffer</th>
<th>Insns</th>
<th>To Free</th>
<th>Done?</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>p8</td>
<td>yes</td>
<td>ld</td>
<td>p7</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>r2</td>
<td>p9</td>
<td>yes</td>
<td>add</td>
<td>p5</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td>p6</td>
<td>yes</td>
<td>xor</td>
<td>p3</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td>p12</td>
<td>yes</td>
<td>ld</td>
<td>p10</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>r5</td>
<td>p4</td>
<td>yes</td>
<td>add</td>
<td>p9</td>
<td>yes</td>
<td>p6</td>
</tr>
<tr>
<td>r6</td>
<td>p11</td>
<td>yes</td>
<td>xor</td>
<td>p10</td>
<td>yes</td>
<td>p11</td>
</tr>
<tr>
<td>r7</td>
<td>p2</td>
<td>yes</td>
<td>ld</td>
<td>p2</td>
<td>yes</td>
<td>p12</td>
</tr>
</tbody>
</table>

### Out-of-Order Pipeline – Cycle 6

<table>
<thead>
<tr>
<th>Map Table</th>
<th>Ready Table</th>
<th>Reorder Buffer</th>
<th>Insns</th>
<th>To Free</th>
<th>Done?</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>p8</td>
<td>yes</td>
<td>ld</td>
<td>p7</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>r2</td>
<td>p9</td>
<td>yes</td>
<td>add</td>
<td>p5</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td>p6</td>
<td>yes</td>
<td>xor</td>
<td>p3</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td>p12</td>
<td>yes</td>
<td>ld</td>
<td>p10</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>r5</td>
<td>p4</td>
<td>yes</td>
<td>add</td>
<td>p9</td>
<td>yes</td>
<td>p6</td>
</tr>
<tr>
<td>r6</td>
<td>p11</td>
<td>yes</td>
<td>xor</td>
<td>p10</td>
<td>yes</td>
<td>p11</td>
</tr>
<tr>
<td>r7</td>
<td>p2</td>
<td>yes</td>
<td>ld</td>
<td>p2</td>
<td>yes</td>
<td>p12</td>
</tr>
</tbody>
</table>

---

**CIS 501 (Martin): Scheduling**
### Out-of-Order Pipeline – Cycle 7

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>id [r1] -&gt; r2</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>M</td>
<td>j</td>
<td>M</td>
<td>j</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r2 + r3 -&gt; r4</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xor r4 + r5 -&gt; r6</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld [r7] -&gt; r4</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>M</td>
<td>j</td>
<td>M</td>
<td>j</td>
<td>W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Map Table
- r1: p8
- r2: p9
- r3: p6
- r4: p12
- r5: p7
- r6: p11
- r7: p2
- r8: p1

#### Ready Table
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: yes
- p9: yes
- p10: yes
- p11: yes
- p12: yes

#### Issue Queue
- Insn: ld
- Src1: p7
- Src2: p12
- R7: yes
- R8: no
- Dest: p9
- Age: 0

#### Reorder Buffer
- Insn: ld
- To Free: p7
- Done?: yes

### Out-of-Order Pipeline – Cycle 8a

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>id [r1] -&gt; r2</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>M</td>
<td>j</td>
<td>M</td>
<td>j</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r2 + r3 -&gt; r4</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xor r4 + r5 -&gt; r6</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld [r7] -&gt; r4</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>M</td>
<td>j</td>
<td>M</td>
<td>j</td>
<td>W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Map Table
- r1: p8
- r2: p9
- r3: p6
- r4: p12
- r5: p7
- r6: p11
- r7: p2
- r8: p1

#### Ready Table
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: yes
- p9: yes
- p10: yes
- p11: yes
- p12: yes

#### Issue Queue
- Insn: ld
- Src1: p7
- Src2: p12
- R7: yes
- R8: no
- Dest: p9
- Age: 0

#### Reorder Buffer
- Insn: ld
- To Free: p7
- Done?: yes

### Out-of-Order Pipeline – Cycle 8b

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>id [r1] -&gt; r2</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>M</td>
<td>j</td>
<td>M</td>
<td>j</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add r2 + r3 -&gt; r4</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xor r4 + r5 -&gt; r6</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld [r7] -&gt; r4</td>
<td>F</td>
<td>D</td>
<td>i</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>M</td>
<td>j</td>
<td>M</td>
<td>j</td>
<td>W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Map Table
- r1: p8
- r2: p9
- r3: p6
- r4: p12
- r5: p7
- r6: p11
- r7: p2
- r8: p1

#### Ready Table
- p1: yes
- p2: yes
- p3: yes
- p4: yes
- p5: yes
- p6: yes
- p7: yes
- p8: yes
- p9: yes
- p10: yes
- p11: yes
- p12: yes

#### Issue Queue
- Insn: ld
- Src1: p7
- Src2: p12
- R7: yes
- R8: no
- Dest: p9
- Age: 0

#### Reorder Buffer
- Insn: ld
- To Free: p7
- Done?: yes
Out-of-Order Pipeline – Cycle 9b

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ready Table</td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p6</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p9</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p10</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p11</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CIS 501 (Martin): Scheduling

Out-of-Order Pipeline – Cycle 9b

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ready Table</td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p6</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p9</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p10</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p11</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CIS 501 (Martin): Scheduling

Out-of-Order Pipeline – Done!

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ready Table</td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p6</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p9</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p10</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p11</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CIS 501 (Martin): Scheduling

Out-of-Order Pipeline – Done!

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ready Table</td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p6</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p9</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p10</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p11</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CIS 501 (Martin): Scheduling

Out-of-Order Pipeline – Cycle 10

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ready Table</td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p6</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p9</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p10</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p11</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CIS 501 (Martin): Scheduling

Out-of-Order Pipeline – Cycle 10

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ready Table</td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p3</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p6</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p7</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p9</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p10</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p11</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CIS 501 (Martin): Scheduling

HANDLING MEMORY OPERATIONS

CIS 501 (Martin): Scheduling 136
Handling Stores

- Can "st p4 -> [p6+8]" issue and begin execution?
  - Its registers inputs are ready...
  - Why or why not?

Problem #1: Out-of-Order Stores

- Can "st p4 -> [p6+8]" write the cache in cycle 6?
  - "st p5 -> [p3+4]" has not yet executed
- What if "p3+4 == p6+8"
  - The two stores write the same address! WAW dependency!
  - Not known until their "X" stages (cycle 5 & 8)
- Unappealing solution: all stores execute in-order
- We can do better...

Store Queue (SQ)

- Two problems
  - Speculative stores
  - Out-of-order stores
- Solution: Store Queue (SQ)
  - When dispatch, each store is given a slot in the Store Queue
  - First-in-first-out (FIFO) queue
  - Each entry contains: "address", "value", and "age"
- Operation:
  - Dispatch (in-order): allocate entry in SQ (stall if full)
  - Execute (out-of-order): write store value into store queue
  - Commit (in-order): read value from SQ and write into data cache
  - Branch recovery: remove entries from the store queue
- Address the above two problems, plus more...

Problem #2: Speculative Stores

- Can "st p4 -> [p6+8]" write the cache in cycle 6?
  - Store is still "speculative" at this point
- What if "jump-not-zero" is mis-predicted?
  - Not known until its "X" stage (cycle 8)
- How does it "undo" the store once it hits the cache?
  - Answer: it can't; stores write the cache only at commit
  - Guaranteed to be non-speculative at that point
Memory Forwarding

- Stores write cache at commit
  - Why? Allows stores to be “undone” on branch mis-predictions, etc.
  - Commit is in-order, delayed until all prior instructions are done

- Loads read cache
  - Early execution of loads is critical

- Forwarding
  - Allow store to load communication before store commit
  - Conceptually like register bypassing, but different implementation
    - Why? Addresses unknown until execute

Can “ld [p7] -> p8” issue and begin execution?
  - Why or why not?
  - If the load reads from either of the store’s addresses...
    - The load must get the value, but it isn’t written to the cache until commit...

Solution: “memory forwarding”
  - Loads also read from the Store Queue (in parallel with the cache)
Problem #3: WAR Hazards

- What if “p3+4 == p6 + 8”?
  - Then load and store access same memory location
- Need to make sure that load doesn’t read store’s result
  - Need to get values based on “program order” not “execution order”
- Bad solution: require all stores/loads to execute in-order
- Good solution: add “age” fields to store queue (SQ)
  - Loads read matching address that is “earlier” (or “older”) than it
  - Another reason the SQ is a FIFO queue

### Store Queue (SQ)
- On load execution, select the store that is:
  - To same address as load
  - Older than the load (before the load in program order)
- Of these, select the youngest store
  - The store to the same address that immediately precedes the load

### Memory Forwarding via Store Queue
- Store Queue (SQ)
  - Holds all in-flight stores
  - CAM: searchable by address
  - Age logic: determine youngest matching store older than load
- Store rename/dispatch
  - Allocate entry in SQ
- Store execution
  - Update SQ
    - Address + Data
- Load execution
  - Search SQ identify youngest older matching store
    - Match? Read SQ
    - No Match? Read cache

### When Can Loads Execute?
- Can “ld [p6+8] -> p7” issue in cycle 3
  - Why or why not?
When Can Loads Execute?

- **Aliasing**! Does \( p3+4 \) == \( p6+8 \)?
  - If no, load should get value from memory
  - **Can it start to execute?**
    - If yes, load should get value from store
      - By reading the store queue?
  - But the value isn't put into the store queue until cycle 9
- **Key challenge**: don’t know addresses until execution!
  - One solution: require all loads to wait for all earlier (prior) stores

### Load scheduling

- **Store->Load Forwarding:**
  - Get value from executed (but not committed) store to load
- **Load Scheduling:**
  - Determine when load can execute with regard to older stores

- **Conservative load scheduling:**
  - All older stores have executed
    - Some architectures: split store address / store data
      - Only requires knowing addresses (not the store values)
    - Advantage: always safe
    - Disadvantage: performance (limits out-of-orderness)

### Conservative Load Scheduling

<table>
<thead>
<tr>
<th>Instruction</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>mul p1 * p2 -&gt; p3</td>
<td>F</td>
<td>Di</td>
<td>I</td>
<td>RR</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>W</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>jump-not-zero p3</td>
<td>F</td>
<td>Di</td>
<td>I</td>
<td>RR</td>
<td>X</td>
<td>W</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>st p5 -&gt; [p3+4]</td>
<td>F</td>
<td>Di</td>
<td>I</td>
<td>RR</td>
<td>X</td>
<td>W</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld [p6+8] -&gt; p7</td>
<td>F</td>
<td>Di</td>
<td>I?</td>
<td>RR</td>
<td>X</td>
<td>M1</td>
<td>M2</td>
<td>W</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Conservative load scheduling: can’t issue ld [p1+4] until cycle 7!**

Might as well be an in-order machine on this example

- Can we do better? How?

### Dynamically Scheduling Memory Ops

- **Compilers must schedule memory ops conservatively**
- **Options for hardware:**
  - Don’t execute any load until all prior stores execute (conservative)
  - Execute loads as soon as possible, detect violations (optimistic)
    - When a store executes, it checks if any later loads executed too early (to same address). If so, flush pipeline
  - Learn violations over time, selectively reorder (predictive)

<table>
<thead>
<tr>
<th>Before</th>
<th>Wrong(?)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld r2, 4(sp)</td>
<td>ld r2, 4(sp)</td>
</tr>
<tr>
<td>ld r3, 8(sp)</td>
<td>ld r3, 8(sp)</td>
</tr>
<tr>
<td>add r3, r2, r1 //stall</td>
<td>ld r5, 0(r8) //does r8==sp?</td>
</tr>
<tr>
<td>st r1, 0(sp)</td>
<td>add r3, r2, r1</td>
</tr>
<tr>
<td>ld r5, 0(r8)</td>
<td>ld r6, 4(r8) //does r8+4==sp?</td>
</tr>
<tr>
<td>ld r6, 4(r8)</td>
<td>st r1, 0(sp)</td>
</tr>
<tr>
<td>sub r5, r6, r4 //stall</td>
<td>sub r5, r6, r4</td>
</tr>
<tr>
<td>st r4, 8(r8)</td>
<td>st r4, 8(r8)</td>
</tr>
<tr>
<td>st r4, 8(r8)</td>
<td>st r4, 8(r8)</td>
</tr>
</tbody>
</table>
**Optimistic Load Scheduling**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld [p1] -&gt; p4</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>R</td>
<td>X</td>
<td>M₁</td>
<td>M₂</td>
<td>W</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld [p2] -&gt; p5</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>R</td>
<td>X</td>
<td>M₁</td>
<td>M₂</td>
<td>W</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add p4, p5 -&gt; p6</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>R</td>
<td>X</td>
<td>W</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>st p6 -&gt; [p3]</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>R</td>
<td>X</td>
<td>S</td>
<td>Q</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld [p1+4] -&gt; p7</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>R</td>
<td>X</td>
<td>M₁</td>
<td>M₂</td>
<td>W</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ld [p2+4] -&gt; p8</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>R</td>
<td>X</td>
<td>M₁</td>
<td>M₂</td>
<td>W</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add p7, p8 -&gt; p9</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>R</td>
<td>X</td>
<td>W</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>st p9 -&gt; [p3+4]</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>R</td>
<td>X</td>
<td>S</td>
<td>Q</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Optimistic load scheduling: can actually benefit from out-of-order!

But how do we know when out speculation (optimism) fails?

---

**Load Speculation**

- Speculation requires two things.....
  - Detection of mis-speculations
  - How can we do this?

- Recovery from mis-speculations
  - Squash from offending load
  - Saw how to squash from branches: same method

---

**Load Queue**

- Detects load ordering violations
- Load execution: Write address into LQ
  - Also note any store forwarded from
- Store execution: Search LQ
  - Younger load with same addr?
  - Didn't forward from younger store? (optimization for full renaming)

---

**Store Queue + Load Queue**

- Store Queue: handles forwarding
  - Written by stores (@ execute)
  - Searched by loads (@ execute)
  - Read from to write data cache (@ commit)

- Load Queue: detects ordering violations
  - Written by loads (@ execute)
  - Searched by stores (@ execute)

- Both together
  - Allows aggressive load scheduling
  - Stores don't constrain load execution
Optimistic Load Scheduling

- Allows loads to issue before older stores
  - Increases out-of-orderness
    + When no conflict, increases performance
    - Conflict => squash => worse performance than waiting
- Some loads might forward from stores
  - Always aggressive will squash a lot
- Can we have our cake AND eat it too?

Predictive Load Scheduling

- Predict which loads must wait for stores
  - Fool me once, shame on you-- fool me twice?
    - Loads default to aggressive
    - Keep table of load PCs that have been caused squashes
      - Schedule these conservatively
      + Simple predictor
      - Makes “bad” loads wait for all older stores is not so great
- More complex predictors used in practice
  - Predict which stores loads should wait for
  - “Store Sets” paper for next time

Challenges for Out-of-Order Cores

- Design complexity
  - More complicated than in-order? Certainly!
  - But, we have managed to overcome the design complexity
- Clock frequency
  - Can we build a “high ILP” machine at high clock frequency?
    - Yep, with some additional pipe stages, clever design
- Limits to (efficiently) scaling the window and ILP
  - Large physical register file
  - Fast register renaming/wakeup/select
  - Branch & memory depend. prediction (limits effective window size)
  - Plus all the issues of build “wide” in-order superscalar
- Power efficiency
  - Today, mobile phone chips are still in-order cores
Out of Order: Window Size

- Scheduling scope = out-of-order window size
  - Larger = better
  - Constrained by physical registers (#preg)
    - Window limited by #preg = ROB size + #logical registers
    - Big register file = hard/slow
- Constrained by issue queue
  - Limits number of un-executed instructions
  - CAM = can't make big (power + area)
  - Constrained by load + store queues
    - Limit number of loads/stores
    - CAMs
- Active area of research: scaling window sizes
  - Usefulness of large window: limited by branch prediction
    - 95% branch mis-prediction rate: 1 in 20 branches, or 1 in 100 insn.

Reprise: Static vs Dynamic Scheduling

- If we can do this in software...
- ...why build complex (slow-clock, high-power) hardware?
  + Performance portability
    - Don't want to recompile for new machines
  + More information available
    - Memory addresses, branch directions, cache misses
  + More registers available
    - Compiler may not have enough to schedule well
  + Speculative memory operation re-ordering
    - Compiler must be conservative, hardware can speculate
      - But compiler has a larger scope
        - Compiler does as much as it can (not much)
        - Hardware does the rest

Out of Order: Benefits

- Allows speculative re-ordering
  - Loads / stores
  - Branch prediction to look past branches
- Schedule can change due to cache misses
  - Different schedule optimal from on cache hit
- Done by hardware
  - Compiler may want different schedule for different hw configs
  - Hardware has only its own configuration to deal with

Recap: Dynamic Scheduling

- Dynamic scheduling
  - Totally in the hardware
  - Also called “out-of-order execution” (OoO)
- Fetch many instructions into instruction window
  - Use branch prediction to speculate past (multiple) branches
  - Flush pipeline on branch misprediction
- Rename to avoid false dependencies
- Execute instructions as soon as possible
  - Register dependencies are known
  - Handling memory dependencies more tricky
- “Commit” instructions in order
  - Anything strange happens before commit, just flush the pipeline
- Current machines: 100+ instruction scheduling window
Out of Order: Top 5 Things to Know

- Register renaming
  - How to perform is and how to recover it
- Commit
  - Precise state (ROB)
  - How/when registers are freed
- Issue/Select
  - Wakeup
  - Choose N oldest ready instructions
- Stores
  - Write at commit
  - Forward to loads via LQ
- Loads
  - Conservative/optimistic/predictive scheduling
  - Violation detection

LOAD/STORE QUEUE EXAMPLES

Summary: Scheduling

- Pipelining and superscalar review
- Code scheduling
  - To reduce pipeline stalls
  - To increase ILP (insn level parallelism)
- Two approaches
  - Static scheduling by the compiler
  - Dynamic scheduling by the hardware
- Up next: multicore

Initial State
(All same address)

1. St p1 -> [p2]
2. St p3 -> [p4]

<table>
<thead>
<tr>
<th>RegFile</th>
<th>Load Queue</th>
<th>RegFile</th>
<th>Load Queue</th>
<th>RegFile</th>
<th>Load Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>Age</td>
<td>p1</td>
<td>Age</td>
<td>p1</td>
<td>Age</td>
</tr>
<tr>
<td>p2</td>
<td>100</td>
<td>p2</td>
<td>100</td>
<td>p2</td>
<td>100</td>
</tr>
<tr>
<td>p3</td>
<td>9</td>
<td>p3</td>
<td>9</td>
<td>p3</td>
<td>9</td>
</tr>
<tr>
<td>p4</td>
<td>200</td>
<td>p4</td>
<td>100</td>
<td>p4</td>
<td>100</td>
</tr>
<tr>
<td>p5</td>
<td>100</td>
<td>p5</td>
<td>100</td>
<td>p5</td>
<td>100</td>
</tr>
<tr>
<td>p6</td>
<td>----</td>
<td>p6</td>
<td>----</td>
<td>p6</td>
<td>----</td>
</tr>
<tr>
<td>p7</td>
<td>----</td>
<td>p7</td>
<td>----</td>
<td>p7</td>
<td>----</td>
</tr>
<tr>
<td>p8</td>
<td>----</td>
<td>p8</td>
<td>----</td>
<td>p8</td>
<td>----</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Store Queue</th>
<th>Age</th>
<th>Addr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>100</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td>100</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>100</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>100</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>

Cache

<table>
<thead>
<tr>
<th>Addr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td>200</td>
<td>17</td>
</tr>
</tbody>
</table>

(All same address)
### Good Interleaving

(Shows importance of address check)

1. St p1 -> [p2]
2. St p3 -> [p4]

### Different Initial State

(Different addresses)

1. St p1 -> [p2]
2. St p3 -> [p4]

### Good Interleaving

(Program Order)

1. St p1 -> [p2]
2. St p3 -> [p4]

### Bad Interleaving #1

(Load reads the cache)

1. St p1 -> [p2]
2. St p3 -> [p4]
**Bad Interleaving #2**

(Load gets value from wrong store)

1. St p1 -> [p2]
2. St p3 -> [p4]

<table>
<thead>
<tr>
<th>RegFile</th>
<th>Load Queue</th>
<th>Store Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>p2</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>p5</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>p6</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>p7</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>p8</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
<th>Addr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>17</td>
</tr>
</tbody>
</table>

1. St p1 -> [p2]
2. St p3 -> [p4]

---

**Bad/Good Interleaving**

(Load gets value from correct store, but does it work?)

1. St p1 -> [p2]
2. St p3 -> [p4]

<table>
<thead>
<tr>
<th>RegFile</th>
<th>Load Queue</th>
<th>Store Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>p2</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>p5</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>p6</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>p7</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>p8</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
<th>Addr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>17</td>
</tr>
</tbody>
</table>

1. St p1 -> [p2]
2. St p3 -> [p4]

---

1. St p1 -> [p2]
2. St p3 -> [p4]

<table>
<thead>
<tr>
<th>RegFile</th>
<th>Load Queue</th>
<th>Store Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>p2</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>p4</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>p5</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>p6</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>p7</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>p8</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache</th>
<th>Addr</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>17</td>
</tr>
</tbody>
</table>