SW Techniques Instruction-level
Parallelism
Compiler Perspectives on Code Movement

- dependencies are a property of code, whether or not it is a HW hazard depends on the given pipeline.

- Compiler must respect (True) Data dependencies (RAW)
  - Instruction i produces a result used by instruction j, or
  - Instruction j is data dependent on instruction k, and instruction k is data dependent on instruction i.
Compiler Perspectives on Code Movement

- Other kinds of dependence also called *name (false) dependence*: two instructions use same *name* but don’t exchange data
- *Antidependence* (WAR dependence)
  - Instruction j writes a register or memory location that instruction i reads from and instruction i is executed first
- *Output dependence* (WAW dependence)
  - Instruction i and instruction j write the same register or memory location; ordering between instructions must be preserved.
Control Dependence

• Control Dependence
  • Example
    
    ```
    if (c1)
        I1;
    if (c2)
        I2;
    
    I1 is control dependent on c1 and I2 is control dependent on c2 but not on c1.
    ```
A sample loop

Loop:

- **LD** $F0,0(R1)$ ; $F0$=array element, $R1=X[]$
- **MULD** $F4,F0,F2$ ; multiply scalar in $F2$
- **SD** $F4, 0(R1)$ ; store result
- **ADDI** $R1,R1,8$ ; increment pointer 8B (DW)
- **SEQ** $R3, R1, R2$ ; $R2 = &X[1001]$
- **BNEZ** $R3, Loop$ ; branch $R3!=zero$
- **NOP** ; delayed branch slot

Where are the dependencies and stalls?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency (stalls)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>FP Mult</em></td>
<td>6 (5)</td>
</tr>
<tr>
<td><em>LD</em></td>
<td>2 (1)</td>
</tr>
<tr>
<td><em>Int ALU</em></td>
<td>1 (0)</td>
</tr>
</tbody>
</table>
Instruction Scheduling

Loop:
- LD   F0,0(R1)
- MULD F4,F0,F2
- SD   0(R1),F4
- ADDI R1,R1,8
- SEQ  R3, R1, R2
- BNEZ R3,Loop
- NOP

Number of cycle per iteration?
### Instruction Scheduling

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Instruction</th>
<th>Destination</th>
<th>Source 1</th>
<th>Source 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F0,0(R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULD</td>
<td>F4,F0,F2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>0(R1),F4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI</td>
<td>R1,R1,8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ</td>
<td>R3, R1, R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNEZ</td>
<td>R3,Loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Instruction</th>
<th>Destination</th>
<th>Source 1</th>
<th>Source 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F0,0(R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI</td>
<td>R1,R1,8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULD</td>
<td>F4,F0,F2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ</td>
<td>R3, R1, R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNEZ</td>
<td>R3,Loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>-8(R1),F4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Cycles/iteration?**
Loop Unrolling

Loop:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Destination</th>
<th>Source 1</th>
<th>Source 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F0,0(R1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDI</td>
<td>R1,R1,8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULD</td>
<td>F4,F0,F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ</td>
<td>R3, R1, R2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNEZ</td>
<td>R3,Loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>-8(R1),F4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Can extract more parallelism
Loop Unrolling

What is the problem here?
Loop Unrolling

Unnecessary instructions and redundant instructions
Loop Unrolling

Loop:
- LD F0,0(R1)
- ADDI R1,R1,8
- MULD F4,F0,F2
- SEQ R3, R1, R2
- BNEZ R3, Loop
- SD -8(R1),F4

Loop:
- LD F0,0(R1)
- MULD F4,F0,F2
- SD 0(R1),F4
- LD F0,8(R1)
- ADDI R1,R1,16
- MULD F4,F0,F2
- SEQ R3, R1, R2
- BNEZ R3,Loop
- SD -8(R1),F4

Still problems with scheduling?

Hint
Register Renaming

Let’s schedule now
### Register Renaming

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Instruction</th>
<th></th>
<th>Loop:</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LD</td>
<td>F0,0(R1)</td>
<td></td>
<td>LD</td>
</tr>
<tr>
<td></td>
<td>ADDI</td>
<td>R1,R1,8</td>
<td></td>
<td>LD</td>
</tr>
<tr>
<td></td>
<td>MULD</td>
<td>F4,F0,F2</td>
<td></td>
<td>MULD</td>
</tr>
<tr>
<td></td>
<td>SEQ</td>
<td>R3, R1, R2</td>
<td></td>
<td>MULD</td>
</tr>
<tr>
<td></td>
<td>BNEZ</td>
<td>R3,Loop</td>
<td></td>
<td>ADDI</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>-8(R1),F4</td>
<td></td>
<td>SEQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BNEZ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SD</td>
</tr>
</tbody>
</table>

**Cycles/iteration?**
Software Pipeline?!
Software Pipeline

Original iter 1
Original iter 2
Original iter 3
Original iter 4

New iter 1
New iter 2
New iter 3
New iter 4
Software Pipelining

---

**Loop:**

- L.D    F0, 0(R1)
- ADD.D  F4, F0, F2
- S.D    F4, 0(R1)
- DADDUI R1, R1,# -8
- BNE    R1, R2, Loop

**Loop:**

- S.D    F4, 16(R1)
- ADD.D  F4, F0, F2
- L.D    F0, 0(R1)
- DADDUI R1, R1,# -8
- BNE    R1, R2, Loop

---

**Advantages:**
- Achieves nearly the same effect as loop unrolling, but without the code expansion – an unrolled loop may have inefficiencies at the start and end of each iteration, while a sw-pipelined loop is almost always in steady state – a sw-pipelined loop can also be unrolled to reduce loop overhead

**Disadvantages:**
- Does not reduce loop overhead, may require more registers