LECTURE - 11
Instruction Level Parallelism

- Pipelining achieves Instruction Level Parallelism (ILP)
  - Multiple instructions in parallel
- But, problems with pipeline hazards
  - CPI = Ideal CPI + stalls/instruction
  - Stalls = Structural + Data (RAW/WAW/WAR) + Control
- How to reduce stalls?
  - That is, how to increase ILP?
Techniques for Improving ILP

- Loop unrolling
- Basic pipeline scheduling
- Dynamic scheduling, scoreboard, register renaming
- Dynamic memory disambiguation
- Dynamic branch prediction
- Multiple instruction issue per cycle
  - Software and hardware techniques
Loop-Level Parallelism

- **Basic block**: straight-line code w/o branches
- Fraction of branches: 0.15
- ILP is limited!
  - Average basic-block size is 6-7 instructions
  - And, these may be dependent
- Hence, look for parallelism beyond a basic block
- *Loop-level parallelism* is a simple example of this
Loop-Level Parallelism: An Example

• Consider the loop:

```java
for(int i = 1000; i >= 1; i = i-1) {
    x[i] = x[i] + C; // FP
}
```

  - Each iteration of the loop is independent of other iterations
  - Loop-level parallelism

• To convert it into ILP:

  - Loop unrolling (static, dynamic)
  - Vector instructions
The Loop, in DLX

In DLX, the loop looks like:

```
Loop:  LD    F0, 0(R1) // F0 is array element
      ADDD  F4, F0, F2 // F2 has the scalar 'C'
      SD    0(R1), F4 // Stored result
      SUBI  R1, R1, 8 // For next iteration
      BNEZ  R1, Loop // More iterations?
```

Assume:
- R1 is the initial address
- F2 has the scalar value 'C'
- Lowest address in array is '8'
## How Many Cycles per Loop?

<table>
<thead>
<tr>
<th>CC1</th>
<th>Loop: LD F0, 0(R1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC2</td>
<td>stall</td>
</tr>
<tr>
<td>CC3</td>
<td>ADDD F4, F0, F2</td>
</tr>
<tr>
<td>CC4</td>
<td>stall</td>
</tr>
<tr>
<td>CC5</td>
<td>stall</td>
</tr>
<tr>
<td>CC6</td>
<td>SD 0(R1), F4</td>
</tr>
<tr>
<td>CC7</td>
<td>SUBI R1, R1, 8</td>
</tr>
<tr>
<td>CC8</td>
<td>stall</td>
</tr>
<tr>
<td>CC9</td>
<td>BNEZ R1, Loop</td>
</tr>
<tr>
<td>CC10</td>
<td>stall</td>
</tr>
</tbody>
</table>
Reducing Stalls by Scheduling

CC1  Loop:  LD   F0, 0(R1)
CC2  SUBI   R1, R1, 8
CC3  ADDD  F4, F0, F2
CC4  stall
CC5  BNEZ   R1, Loop
CC6  SD  8(R1), F4

- Realizing that SUBI and SD can be swapped is non-trivial!

- Overhead versus actual work:
  - 3 cycles of work, 3 cycles of overhead
Unrolling the Loop

Loop:  

- LD  F0, 0(R1)
- ADDD F4, F0, F2
- SD  0(R1), F4 // No SUBI, BNEZ
- LD  F6, -8(R1) // Note diff FP reg, new offset
- ADDD F8, F6, F2
- SD  -8(R1), F8
- LD  F10, -16(R1) // Note diff FP reg, new offset
- ADDD F12, F10, F2
- SD  -16(R1), F8
- LD  F14, -24(R1) // Note diff FP reg, new offset
- ADDD F16, F14, F2
- SD  -24(R1), F16
- SUBI R1, R1, 32
How Many Cycles per Loop?

Loop:

- LD F0, 0(R1)  // 1 stall
- ADDD F4, F0, F2  // 2 stalls
- SD 0(R1), F4
- LD F6, -8(R1)  // 1 stall
- ADDD F8, F6, F2  // 2 stalls
- SD -8(R1), F8
- LD F10, -16(R1)  // 1 stall
- ADDD F12, F10, F2  // 2 stalls
- SD -16(R1), F8
- LD F14, -24(R1)  // 1 stall
- ADDD F16, F14, F2  // 2 stalls
- SD -24(R1), F16
- SUBI R1, R1, 32  // 1 stall

28 cycles per unrolled loop

==

7 cycles per original loop
Scheduling the Unrolled Loop

Loop:  
LD  F0, 0(R1)  
LD  F6, -8(R1)  
LD  F10, -16(R1)  
LD  F14, -24(R1)  
ADDD  F4, F0, F2  
ADDD  F8, F6, F2  
ADDD  F12, F10, F2  
ADDD  F16, F14, F2  
SD  0(R1), F4  
SD  -8(R1), F8  
SUBI  R1, R1, 32  
SD  16(R1), F8  
BNEZ  R1, Loop

14 cycles per unrolled loop

==

3.5 cycles per original loop
Observations and Requirements

- Gain from scheduling is even higher for unrolled loop!
  - More parallelism is exposed on unrolling
- Need to know that 1000 is a multiple of 4
- Requirements:
  - Determine that loop can be unrolled
  - Use different registers to avoid conflicts
  - Determine that SD can be moved after SUBI, and find the offset adjustment
- Understand *dependencies*
Dependences

- Dependent instructions ==> cannot be in parallel
- Three kinds of dependences:
  - Data dependence (RAW)
  - Name dependence (WAW and WAR)
  - Control dependence
Dependences (continued)

- Dependences are properties of programs
- Stalls are properties of the pipeline
- Two possibilities:
  - Maintain dependence, but avoid stalls
  - Eliminate dependence by code transformation
Data Dependence

- Data dependence represents data flow from one instruction to another
  - One instruction uses the result of another
  - Take transitive closure

In our example:

Loop: 

- LD F0, 0(R1)
- ADDD F4, F0, F2
- SD 0(R1), F4
- SUBI R1, R1, 8

Note: dependence in memory is hard to detect. 100(R4) and 80(R6) may be the same. 20(R1) and 20(R1) may be different at different times.
Name Dependence

- Two instructions use the same register/memory (name), but there is no flow of data
  - Anti-dependence: WAR hazard
  - Output dependence: WAW hazard
- Can do register renaming statically, or dynamically
Name Dependence in our Example

Loop: LD F0, 0(R1)
ADDD F4, F0, F2
SD 0(R1), F4
LD F0, -8(R1)
ADDD F4, F0, F2
SD -8(R1), F4
LD F0, -16(R1)
ADDD F4, F0, F2
SD -16(R1), F4
LD F0, -24(R1)
ADDD F4, F0, F2
SD -24(R1), F4
SUBI R1, R1, 32

Loop: LD F0, 0(R1)
ADDD F4, F0, F2
SD 0(R1), F4
LD F6, -8(R1)
ADDD F8, F6, F2
SD -8(R1), F8
LD F10, -16(R1)
ADDD F12, F10, F2
SD -16(R1), F8
LD F14, -24(R1)
ADDD F16, F14, F2
SD -24(R1), F16
SUBI R1, R1, 32

Register renaming
Control Dependence

• An example:
  
  ```
  T1;
  if p1 {
    S1;
  }
  ```

• Statement S1 is *control-dependent* on p1, but T1 is not

• What this means for execution
  - S1 cannot be moved before p1
  - T1 cannot be moved after p1
Control Dependence in our Example

<table>
<thead>
<tr>
<th>Loop:</th>
<th>LD F0, 0(R1)</th>
<th>Loop:</th>
<th>LD F0, 0(R1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDD F4, F0, F2</td>
<td>ADDD F4, F0, F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD 0(R1), F4</td>
<td>SD 0(R1), F4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBI R1, R1, 8</td>
<td>LD F6, -8(R1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQZ R1, exit</td>
<td>ADDD F8, F6, F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F6, 0(R1)</td>
<td>SD -8(R1), F8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDD F8, F6, F2</td>
<td>LD F10, -16(R1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD 0(R1), F8</td>
<td>ADDD F12, F10, F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBI R1, R1, 8</td>
<td>SD -16(R1), F8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQZ R1, exit</td>
<td>LD F14, -24(R1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>// Two more such...</td>
<td>ADDD F16, F14, F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBI R1, R1, 8</td>
<td>SD -24(R1), F16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNEZ R1, Loop</td>
<td>SUBI R1, R1, 32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>