Plan for Today

• Questions

• Today’s discussion
  – Project logistics
  – Static Code Scheduling
  – Loop Unrolling
  – VLIW

• Assignment
Project Logistics

• Groups of three

• Three major deadlines
  – Design due Oct 19
  – Project Demo Nov 23
  – Report due Dec 7

• Today: PM1 assigned

### Project Timeline

<table>
<thead>
<tr>
<th>Task</th>
<th>Due</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Milestone 1</td>
<td>Oct 19</td>
<td>Processor Design</td>
</tr>
<tr>
<td>Project Milestone 2</td>
<td>Nov 2</td>
<td>Detailed Design &amp; VHDL</td>
</tr>
<tr>
<td>Design Presentations</td>
<td>Nov 4</td>
<td>Presentation of detailed design</td>
</tr>
<tr>
<td>Project Demo</td>
<td>Nov 23</td>
<td>Demo working VHDL model</td>
</tr>
<tr>
<td>Final Report</td>
<td>Dec 7</td>
<td>Final report due</td>
</tr>
</tbody>
</table>
# Project Groups

<table>
<thead>
<tr>
<th>Group</th>
<th>Member 1</th>
<th>Member 2</th>
<th>Member 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>David Lu</td>
<td>Michael Schultz</td>
<td>Austin Abrams</td>
</tr>
<tr>
<td>2</td>
<td>Yong Fu</td>
<td>Greg Galloway</td>
<td>Christopher Thomas</td>
</tr>
<tr>
<td>3</td>
<td>Timothy York</td>
<td>Haowei Yuan</td>
<td>Stephen Schuh</td>
</tr>
<tr>
<td>4</td>
<td>Yu-Ying Liang</td>
<td>Abu Sayeed Saifullah</td>
<td>Raphael Njuguna</td>
</tr>
<tr>
<td>5</td>
<td>Cory Flanagan</td>
<td>Jessica Schupp</td>
<td></td>
</tr>
</tbody>
</table>
Compiler Goals for Exposing ILP

• It is best to fill the pipeline with independent instructions

• In general, separate dependent instructions from their sources by $n$ instructions
  – where $n$ is the latency of the source instructions
Sample Code

For \( i=1000; \ i>0; \ i=i-1 \)

\[
x[i] = x[i] + s
\]

Loop:

\[
\text{L.D} \quad F0,0(R1)\\
\text{ADD.D} \quad F4,F0,F2\\
\text{S.D} \quad F4,0(R1)\\
\text{DADDUI} \quad R1,R1,#-8\\
\text{BNE} \quad R1,R2,Loop
\]
Machine Characteristics

• 5 stage integer pipeline
  – Branches have 1 cycle delay (branch delay slot)

• Floating point latencies
  – F-P Op → F-P Op: 3
  – F-P Op → S.D: 2
  – L.D → F-P Op: 1
  – L.D → S.D: 0
Original Loop Schedule

- Questions:
  - Stall causes?
  - Better schedule?

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cycle issued</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>1</td>
</tr>
<tr>
<td>ADD.D</td>
<td>2</td>
</tr>
<tr>
<td>F4,F0,F2</td>
<td>3</td>
</tr>
<tr>
<td>F4,0(R1)</td>
<td>4</td>
</tr>
<tr>
<td>F4,0(R1)</td>
<td>5</td>
</tr>
<tr>
<td>R1,R1,#-8</td>
<td>6</td>
</tr>
<tr>
<td>R1,R2,Loop</td>
<td>7</td>
</tr>
<tr>
<td>Stall</td>
<td>8</td>
</tr>
<tr>
<td>Stall</td>
<td>9</td>
</tr>
<tr>
<td>Stall</td>
<td>10</td>
</tr>
</tbody>
</table>

Data

RAW

Control

RAW

Stall

RAW

Stall

RAW

Stall

Data

Loop:
Better Loop Schedule

Loop: L.D  
DADDUI  
ADD.D  
stall  
BNE  
S.D  
F0,0(R1)  
R1,R1,#-8  
F4,F0,F2  
R1,R2,Loop  
F4,8(R1)  
• Only 1 stall remains  
• Iteration time reduced from 10 to 6 cycles  
• Needed to  
  – Change store address
Critical Instructions

Loop:

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

- This dependence chain determines loop latency
- Other three constitute loop overhead (what % ?)

Questions:
- Can we eliminate the stall?
- Can we reduce loop overhead?
Loop Unrolling

Loop:  

L.D  F0,0(R1)
ADD.D F4,F0,F2
S.D  F4,0(R1)
L.D  F6,-8(R1)
ADD.D F8,F6,F2
S.D  F8,-8(R1)
DADDUI R1,R1,#-16
BNE R1,R2,Loop

- Amortize loop overhead
- Update load/store addresses
- Update array bound check

Questions:
- What is the loop duration?
- Faster/slower than scheduled version?
- What if odd # of iterations?
- How should we schedule this?
Loop Unrolling & Scheduling

Loop:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Register(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0, 0(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F6, -8(R1)</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F4, F0, F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F8, F6, F2</td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1, R1, #−16</td>
</tr>
<tr>
<td>S.D</td>
<td>F4, 16(R1)</td>
</tr>
<tr>
<td>BNE</td>
<td>R1, R2, Loop</td>
</tr>
<tr>
<td>S.D</td>
<td>F8, 8(R1)</td>
</tr>
</tbody>
</table>

- **What is the loop duration?**
- **What is the loop overhead?**
Limits and Tradeoffs

• Diminishing returns from loop unrolling

• Code size increases with loop unrolling
  – Embedded systems
  – Instruction cache performance

• Register allocation
  – Each unrolled loop iteration contributes live values, creates register pressure
Summarized Compiler/Programmer Activities

1. Determine that loop iterations are independent
2. Use different registers for each iteration
3. Remove extra loop termination instructions
4. Schedule the code according to instruction latencies while preserving dependences
5. Re-order loads and stores from different iterations (memory alias analysis)
6. Update index calculations and load and store offsets
Static Multiple Issue

<table>
<thead>
<tr>
<th>Integer</th>
<th>F-P</th>
<th>Clock Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F0,0(R1)</td>
<td>1</td>
</tr>
<tr>
<td>L.D</td>
<td>F6,-8(R1)</td>
<td>2</td>
</tr>
<tr>
<td>L.D</td>
<td>F10,-16(R1)</td>
<td>3</td>
</tr>
<tr>
<td>L.D</td>
<td>F14,-24(R1)</td>
<td>4</td>
</tr>
<tr>
<td>L.D</td>
<td>F18,-32(R1)</td>
<td>5</td>
</tr>
<tr>
<td>S.D</td>
<td>F4,0(R1)</td>
<td>6</td>
</tr>
<tr>
<td>S.D</td>
<td>F8,-8(R1)</td>
<td>7</td>
</tr>
<tr>
<td>S.D</td>
<td>F12,-16(R1)</td>
<td>8</td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1,R1,#-40</td>
<td>9</td>
</tr>
<tr>
<td>S.D</td>
<td>F12,-16(R1)</td>
<td>10</td>
</tr>
<tr>
<td>BNE</td>
<td>R1,R2,Loop</td>
<td>11</td>
</tr>
<tr>
<td>S.D</td>
<td>F12,-16(R1)</td>
<td>12</td>
</tr>
</tbody>
</table>

- Technique can be used on a statically-scheduled superscalar as well
- Unrolled 5 times
- Loop duration is 12 cycles
Using Static Branch Prediction

- Fill stall with an instruction
  - Branch target?
  - Fall-through?

- Slot can be utilized if branch outcome biased in one direction
  - Taken, 34% avg but wide range
  - FTBNT, ≤40%
  - Profile-based
Profile-driven Static Prediction

Misprediction rate

Benchmark

© 2003 Elsevier Science (USA). All rights reserved.
VLIW Approach to ILP

• Organize program into groups of independent instructions
  – All instructions in a group can issue at the same time
  – No need for dynamic issue/scheduling decisions

• Emphasis on
  – Simple hardware
  – Sophisticated compiler
  – Wide-issue machines
    • where superscalar hardware overhead is greatest
Sources of Parallelism

- Instructions come from a traditional code sequences
  - programmers still use traditional programming languages

- Use loop unrolling and code scheduling
  - Local scheduling used between branches
  - Global scheduling used across branches (we’ll look at `trace scheduling` later)
Sample VLIW Machine

- Each ‘wide’ instruction has 5 operations
  - 1 integer (or branch)
  - 2 f-p
  - 2 memory references

- Instruction size
  - Example: 3*32 + 2*24 = 144 bits
Instruction Scheduling Example

Unrolled 7 times to eliminate all stalls

Loop duration is 9 cycles

Is this good resource utilization?
Assignment

• PM1 assigned today, due Oct 19

• Readings
  – For Monday
    • Commentary: The Microarchitecture of Superscalar Processors
  – For Wednesday
    • No assigned readings