Topics

• Out-of-order execution (O-O-O)
  – Tanenbaum 4.5.3

• Register renaming
  – Tanenbaum 4.5.3

• Speculative execution
  – Tanenbaum 4.5.4

• Example architectures
  – Tanenbaum 4.6

Modern CPU Design

• Pipelined – instructions execute in stages
• Superscalar – contains multiple functional units
In-Order Execution

• Issue instructions in program order

• Retire (complete) instructions in program order

• May not give optimal performance because of instruction dependencies
  – Created whenever there is a dependence between instructions where pipelining overlap changes the order of access
    • RAW
    • WAW
    • WAR

Data Hazards

• Read After Write (RAW)
  – True data dependence (most common type)
  – Must preserve program order to ensure correct execution

• Write After Read (WAR)
  – Output dependence
  – Only present in pipelines that
    • write in more than one pipe stage
    • allow instructions to proceed after previous instructions stalls

• Write After Write (WAW)
  – Anti-dependence
  – Can occur when instructions are reordered
Example: Pipelined, Superscalar CPU

- 2-way superscalar
  - Can issue up to 2 instructions per cycle

- For instructions decoded in cycle $n$
  - Execution starts in cycle $n + 1$
  - ADD/SUB completes in cycle $n + 2$
  - MUL completes in cycle $n + 3$

Rules for Issuing Instructions

- **RAW dependence**
  - Don’t issue if any operand is being written

- **WAR dependence**
  - Don’t issue if result register is being read

- **WAW dependence**
  - Don’t issue if result register is being written
Scoreboarding: In-Order Issue/Completion

- I4 has a RAW dependence from I2
  - Stall until R4 is available
Scoreboarding: In-Order Issue/Completion

- I2 actually completes during cycle 3
  - Must retire instructions in order

Figure 4-43. Operation of a superscalar CPU with in-order issue and in-order completion.
Scoreboarding: In-Order Issue/Completion

- I4 has a RAW dependence from I2
  - Stall until R4 is available

Scoreboarding: In-Order Issue/Completion

- I2 actually completes during cycle 3
  - Must retire instructions in order
Out-of-Order (O-O-O) Execution

• What if we skipped the dependent instructions?
  – Potentially find independent instructions to execute

• What about the program’s result?
  – Must still guarantee the same result as in-order execution

Scoreboarding: O-O-O Issue/Completion

• Skip I4 to issue I5
• Allow I2 to retire to issue I4
### Scoreboarding: O-O-O Issue/Completion

<table>
<thead>
<tr>
<th>Cy</th>
<th>#</th>
<th>Decoded</th>
<th>Iss</th>
<th>Ref</th>
<th>Registers being read</th>
<th>Registers being written</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>R3=R0+R1</td>
<td>1</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>R4=R0+R2</td>
<td>2</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>R5=R0+R1</td>
<td>3</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>R6=R0+R2</td>
<td>4</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>R7=R0+R3</td>
<td>5</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>R8=R0+R4</td>
<td>6</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Register renaming in I6 & I7 to eliminate WAR hazard

### Scoreboarding: O-O-O Issue/Completion

<table>
<thead>
<tr>
<th>Cy</th>
<th>#</th>
<th>Decoded</th>
<th>Iss</th>
<th>Ref</th>
<th>Registers being read</th>
<th>Registers being written</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>R3=R0+R1</td>
<td>1</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>R4=R0+R2</td>
<td>2</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>R5=R0+R1</td>
<td>3</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>R6=R0+R2</td>
<td>4</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>R7=R0+R3</td>
<td>5</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>R8=R0+R4</td>
<td>6</td>
<td></td>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Register renaming in I8 to eliminate WAR hazard
Precise Interrupt

• Need the capability to store the CPU state

• With out-of-order completion
  – If an interrupt occurred, it would be difficult to save current state
  – Not possible to say all instructions up to some address had been executed and all instructions beyond it had not

• In-order completion ensures precise interrupts

Speculative Execution

• Basic block
  – Linear sequence of code with one entry and one exit
  – No control (branch) instructions

• Insufficient parallelism in basic blocks
  – Allow reordering across basic blocks
  – Hoisting - moving instructions upward over a branch
    • Effective for potentially slow instructions like LOAD

• By hoisting, you do not know if that code would have been executed
  – Typically a compiler does the scheduling
Problem with Speculative Execution

• Speculative execution could cause exceptions
  – Ex: LOAD could cause a cache miss
    • If the data was needed, then the exception is OK
    • If the data was not needed, then the exception is bad
  – Some architectures include SPECULATIVE-LOAD
    • Only tries to retrieve data from cache

• Speculative execution could cause correct programs to fail
  – Ex: if \((x > 0)\) \(z = y / x\)
    • If the DIV is hoisted, the program could cause a divide by zero error despite the condition check
    • Use a poison bit for speculative instructions that fail

Example Microarchitectures

• IA-32 (Pentium II)
  – CISC

• Version 9 SPARC (UltraSPARC II)
  – RISC

• Java Virtual Machine (picoJava II)
  – Stack machine
Microarchitecture for IA-32 ISA

Figure 4-46. The Pentium II microarchitecture.

Microarchitecture for IA-32 ISA

Figure 4-47. Internal structure of the Fetch/Decode unit (simplified).
Microarchitecture for IA-32 ISA

Figure 4-48. The Dispatch/Execute unit.

Microarchitecture for Version 9 SPARC ISA

Figure 4-49. The UltraSPARC II microarchitecture.
Microarchitecture for Version 9 SPARC ISA

Figure 4-50. The UltraSPARC II’s pipeline.

Microarchitecture for JVM ISA

Figure 4-51. The block diagram of the picolava II with both level 1 caches and the floating-point unit. This is configuration of the microjava 701.
Microarchitecture for JVM ISA

Figure 4-52. The picoJava II has a six-stage pipeline.

Instruction Folding

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF</td>
<td>Nonfoldable instructions</td>
<td>GOTO</td>
</tr>
<tr>
<td>LV</td>
<td>Pushing a word onto the stack</td>
<td>ILOAD</td>
</tr>
<tr>
<td>MEM</td>
<td>Popping a word and storing it in memory</td>
<td>ISTORE</td>
</tr>
<tr>
<td>BG1</td>
<td>Operations using one stack operand</td>
<td>IFEQ</td>
</tr>
<tr>
<td>BG2</td>
<td>Operations using two stack operands</td>
<td>IF_CMPEQ</td>
</tr>
<tr>
<td>OP</td>
<td>Computations on two operands with one result</td>
<td>IADD</td>
</tr>
</tbody>
</table>

Figure 4-54. JVM instruction groups for folding purposes.

<table>
<thead>
<tr>
<th>Instruction sequence</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>LV LV OP MEM</td>
<td>ILOAD, ILOAD, IADD, ISTORE</td>
</tr>
<tr>
<td>LV LV OP MEM</td>
<td>ILOAD, ILOAD, IADD</td>
</tr>
<tr>
<td>LV LV BG2</td>
<td>ILOAD, ILOAD, IF_CMPEQ</td>
</tr>
<tr>
<td>LV BG1</td>
<td>ILOAD, IFEQ</td>
</tr>
<tr>
<td>LV BG2</td>
<td>ILOAD, IF_CMPEQ</td>
</tr>
<tr>
<td>LV MEM</td>
<td>ILOAD, ISTORE</td>
</tr>
<tr>
<td>OP MEM</td>
<td>IADD, ISTORE</td>
</tr>
</tbody>
</table>

Figure 4-55. Some of the JVM instruction sequences that can be folded.
Example Architectures

- Pentium II
  - CISC
- UltraSPARC II
  - RISC
- picoJava II
  - Stack machine

Similarity Among Architectures

**Implementation of the execution unit**
- Use an opcode, 2 source registers, and a destination register (3-register format)
- Execute a micro-instruction in one cycle
- Deep pipelines and branch prediction
- Split I-cache and D-cache
Difference Among Architectures

How the instructions get to the execution unit

- Pentium II has to break CISC instructions into 3-register format
- picoJava II must fold instructions into 3-register format
- UltraSPARC II already has the 3-register format