Dynamic Scheduling

Or
dynamic scheduling

CDC 6600 scoreboard
- Instruction storage added to each functional execution unit
- Instructions issue to FU when no structural hazards, begin execution when dependencies satisfied. Thus, instructions issued to different FUs can execute out of order.
- “scoreboard” tracks RAW, WAR, WAW hazards, tells each instruction when to proceed.
- No forwarding
- No register renaming

Tomasulo (IBM 360/91)

Instruction Queue (MIPS R10000, Alpha 21264, …)

Tomasulo Algorithm

For IBM 360/91 about 3 years after CDC 6600
Goal: High Performance without special compilers
Differences between IBM 360 & CDC 6600 ISA
- IBM has only 2 register specifiers/instr vs. 3 in CDC 6600
- IBM has 4 FP registers vs. 8 in CDC 6600
- Implications?

Differences between Tomasulo Algorithm & Scoreboard

Control & buffers distributed with Function Units vs. centralized in scoreboard; called “reservation stations”
- instrs schedule themselves
- Registers in instructions replaced by pointers to reservation station buffer
- scoreboard => registers primary operand storage
- Tomasulo => reservation stations as operand storage
- HW renaming of registers to avoid WAR, WAW hazards
- Scoreboard => both source registers read together
- Tomasulo => each register read as soon as available.
- Common Data Bus broadcasts results to all FUs
- RS’s (FU’s), registers, etc. responsible for collecting own data off CDB
- Load and Store Queues treated as FUs as well
Three Stages of Tomasulo Algorithm

1. Issue—get instruction from FP Inst Queue
   If reservation station free, the IQ issues instr & sends operands (renames registers).
2. Execution—operate on operands (EX)
   When both operands ready then execute; if not ready, watch CDB for result
3. Write result—finish execution (WB)
   Write on Common Data Bus to all waiting units; mark reservation station available.

Tomasulo Example

ADDD F4, F2, F0
MULD F8, F4, F2
ADDD F6, F8, F6
SUBD F8, F2, F0
ADDD F2, F8, F0

Multiply takes 10 clocks, add/sub take 4
Tomasulo – cycle 0

- ADDD F4, F2, F0
- MULD F8, F4, F2
- ADDD F6, F8, F6
- SUBD F8, F2, F0
- ADDD F2, F8, F0

Instruction Queue:

<table>
<thead>
<tr>
<th>Stage</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>2.0</td>
<td>4.0</td>
<td>6.0</td>
<td>8.0</td>
</tr>
</tbody>
</table>

FP adders  
FP mult's

Tomasulo – cycle 1

- ADDD F4, F2, F0
- MULD F8, F4, F2
- ADDD F6, F8, F6
- SUBD F8, F2, F0
- ADDD F2, F8, F0

Instruction Queue:

<table>
<thead>
<tr>
<th>Stage</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>add1</td>
</tr>
</tbody>
</table>

FP adders  
FP mult's

Tomasulo – cycle 2

- ADDD F4, F2, F0
- MULD F8, F4, F2
- ADDD F6, F8, F6
- SUBD F8, F2, F0
- ADDD F2, F8, F0

Instruction Queue:

<table>
<thead>
<tr>
<th>Stage</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>add1</td>
</tr>
</tbody>
</table>

FP adders  
FP mult's

Tomasulo – cycle 3

- ADDD F4, F2, F0
- MULD F8, F4, F2
- ADDD F6, F8, F6
- SUBD F8, F2, F0
- ADDD F2, F8, F0

Instruction Queue:

<table>
<thead>
<tr>
<th>Stage</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>add2</td>
</tr>
</tbody>
</table>

FP adders  
FP mult's
Tomasulo – cycle 19

- ADDD F4, F2, F0
- MULD F8, F4, F2
- ADDD F6, F8, F6
- SUBD F8, F2, F0
- ADDD F2, F8, F0

Instruction Queue

1. ADDD 4.0, 6.0
2. FP adders
3. FP mult’s

10.0 (add2 result)

Tomasulo Summary

- Prevents Register as bottleneck
- Avoids WAR, WAW hazards of Scoreboard
- Allows loop unrolling in HW
- Not limited to basic blocks (provided branch prediction)
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming (in what way does the register name change?)
  - Load/store disambiguation

Modern Architectures

- Alpha 21264+, MIPS R10K+, Pentium 4 use an instruction queue.
- They use explicit register renaming. Registers are not read until instruction dispatches (begins execution). Register renaming ensures no conflicts.

MIPS R10000, some detail

- Register Map
- Instruction Queue
- Active List

Active list – maintains original instruction order, determines when a physical register can be freed.
Dynamic Scheduling Key Points

- Dynamic scheduling is code motion in HW.
- Dynamic scheduling can do things SW scheduling (static scheduling) cannot.
- Register renaming eliminates WAW, WAR dependencies.
- To get cross-iteration parallelism, we need to eliminate WAW, WAR dependencies.