Another Dynamic Algorithm: 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 (thus one could not be overwritten while we wait for the other).
  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

Tomasulo Organization

Reservation Station Components

Op—Operation to perform in the unit (e.g., + or –)
Qj, Qk—Reservation stations producing source registers
Vj, Vk—Value of Source operands
Rj, Rk—Flags indicating when Vj, Vk are ready
Busy—Indicates reservation station is busy

Register result status—Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.
Three Stages of Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue
   If reservation station free, the scoreboard 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.

CSE 240A Dean Tullsen

Tomasulo Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Q Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDD F4, F2, F0</td>
<td></td>
</tr>
<tr>
<td>MULD F8, F4, F2</td>
<td></td>
</tr>
<tr>
<td>ADDD F6, F8, F6</td>
<td></td>
</tr>
<tr>
<td>SUBD F8, F2, F0</td>
<td></td>
</tr>
<tr>
<td>ADDD F2, F8, F0</td>
<td></td>
</tr>
</tbody>
</table>

Multiply takes 10 clocks, add/sub take 4

CSE 240A Dean Tullsen

Tomasulo – cycle 0

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDD F4, F2, F0</td>
<td>F0: 0.0</td>
</tr>
<tr>
<td>MULD F8, F4, F2</td>
<td>F2: 2.0</td>
</tr>
<tr>
<td>ADDD F6, F8, F6</td>
<td>F4: 4.0</td>
</tr>
<tr>
<td>SUBD F8, F2, F0</td>
<td>F6: 6.0</td>
</tr>
<tr>
<td>ADDD F2, F8, F0</td>
<td>F8: 8.0</td>
</tr>
</tbody>
</table>

CSE 240A Dean Tullsen

Tomasulo – cycle 1

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDD F4, F2, F0</td>
<td>F0: 0.0</td>
</tr>
<tr>
<td>MULD F8, F4, F2</td>
<td>F2: 2.0</td>
</tr>
<tr>
<td>ADDD F6, F8, F6</td>
<td>F4: 4.0</td>
</tr>
<tr>
<td>SUBD F8, F2, F0</td>
<td>F6: 6.0</td>
</tr>
<tr>
<td>ADDD F2, F8, F0</td>
<td>F8: 8.0</td>
</tr>
</tbody>
</table>

CSE 240A Dean Tullsen
Tomasulo – cycle 12

1. ADDD F4, F2, F0
2. ADDD F6, F8, F6
3. SUBD F8, F2, F0
4. ADDD F2, F8, F0

Instruction Queue:
- F0: 0.0
- F2: 2.0
- F4: 2.0
- F6: 6.0 (add2)
- F8: 2.0

CSE 240A

Tomasulo – cycle 15

1. ADDD F4, F2, F0
2. MULD F8, F4, F2
3. ADDD F6, F8, F6
4. SUBD F8, F2, F0
5. ADDD F2, F8, F0

Instruction Queue:
- F0: 0.0
- F2: 2.0
- F4: 2.0
- F6: 6.0 (add2)
- F8: 2.0

CSE 240A

Tomasulo – cycle 16

1. ADDD F4, F2, F0
2. MULD F8, F4, F2
3. ADDD F6, F8, F6
4. SUBD F8, F2, F0
5. ADDD F2, F8, F0

Instruction Queue:
- F0: 0.0
- F2: 2.0
- F4: 2.0
- F6: 6.0 (add2)
- F8: 2.0

CSE 240A

Tomasulo – cycle 19

1. ADDD F4, F2, F0
2. MULD F8, F4, F2
3. ADDD F6, F8, F6
4. SUBD F8, F2, F0
5. ADDD F2, F8, F0

Instruction Queue:
- F0: 0.0
- F2: 2.0
- F4: 2.0
- F6: 6.0 (add2)
- F8: 2.0

CSE 240A
**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

**Scoreboard vs. Tomasulo, the score**

<table>
<thead>
<tr>
<th></th>
<th>Scoreboard</th>
<th>Tomasulo</th>
</tr>
</thead>
<tbody>
<tr>
<td>issue</td>
<td>when FU free</td>
<td>when RS free</td>
</tr>
<tr>
<td>read operands</td>
<td>from reg file</td>
<td>from reg file, CDB</td>
</tr>
<tr>
<td>write operands</td>
<td>to reg file</td>
<td>to CDB</td>
</tr>
<tr>
<td>structural hazards</td>
<td>functional units</td>
<td>reservation stations</td>
</tr>
<tr>
<td>WAW, WAR hazards</td>
<td>problem</td>
<td>no</td>
</tr>
<tr>
<td>register renaming</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>instructions completing</td>
<td>no limit</td>
<td>1 / cycle (per CDB)</td>
</tr>
<tr>
<td>instructions beginning ex.</td>
<td>1 (per set of read ports)</td>
<td>no limit</td>
</tr>
</tbody>
</table>

**Modern Architectures**

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

**MIPS R10000, some detail**

```
Div R5, R4, R2
Add R7, R5, R1
Sub R5, R3, R2
Lw R7, 1000(R5)

Div PR37, PR45, PR2
Add PR4, PR37, PR23
Sub PR42, PR17, PR2
Lw PR19, 1000(PR42)
```
MIPS R10000, some detail

Register Map

I1: Div R5, R4, R2
I2: Add R7, R5, R1
I3: Sub R5, R3, R2
I4: Lw R7, 1000(R5)

Instruction Queue

Active List

Head

Tail

Register Free List

PR37, PR4, PR42, PR19, …

MIPS R10000, some detail

Register Map

I1: Div R5, R4, R2
I2: Add R7, R5, R1
I3: Sub R5, R3, R2
I4: Lw R7, 1000(R5)

Instruction Queue

Active List

Head

Tail

Register Free List

PR42, PR19, …

Dynamic Scheduling Key Points

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