Dynamic Scheduling

Or

dynamic scheduling

CSE 240A  Dean Tullsen

Dynamic Scheduling
(or out-of-order execution)

- CDC 6600 scoreboard
  - Instruction storage added to each functional execution unit
  - Instructions issue to FU when no structural hazards, begin execution when dependences 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?

CSE 240A  Dean Tullsen

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

CSE 240A  Dean Tullsen
**Tomasulo Organization**

Reserveation 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 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

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

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

Instruction Queue

1 2 3

FP adders
FP mult's

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

CSE 240A
Dean Tullsen

Tomasulo – cycle 1

Instruction Queue

1 2 3

FP adders
FP mult's

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

CSE 240A
Dean Tullsen

Tomasulo – cycle 2

Instruction Queue

1 2 3

FP adders
FP mult's

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

CSE 240A
Dean Tullsen

Tomasulo – cycle 2

Instruction Queue

1 2 3

FP adders
FP mult's

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

CSE 240A
Dean Tullsen
**Tomasulo – cycle 15**

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

**Instruction Queue**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>0.0</td>
</tr>
<tr>
<td>F2</td>
<td>2.0</td>
</tr>
<tr>
<td>F4</td>
<td>2.0</td>
</tr>
<tr>
<td>F6</td>
<td>6.0 add2</td>
</tr>
<tr>
<td>F8</td>
<td>2.0</td>
</tr>
</tbody>
</table>

**FP adders**

4.0 (mult1 result)

**FP mult's**

**Tomasulo – cycle 16**

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

**Instruction Queue**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>0.0</td>
</tr>
<tr>
<td>F2</td>
<td>2.0</td>
</tr>
<tr>
<td>F4</td>
<td>2.0</td>
</tr>
<tr>
<td>F6</td>
<td>6.0 add2</td>
</tr>
<tr>
<td>F8</td>
<td>2.0</td>
</tr>
</tbody>
</table>

**FP adders**

**FP mult's**

**Tomasulo – cycle 19**

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

**Instruction Queue**

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>0.0</td>
</tr>
<tr>
<td>F2</td>
<td>2.0</td>
</tr>
<tr>
<td>F4</td>
<td>2.0</td>
</tr>
<tr>
<td>F6</td>
<td>10.0</td>
</tr>
<tr>
<td>F8</td>
<td>2.0</td>
</tr>
</tbody>
</table>

**FP adders**

10.0 (add2 result)

**FP mult's**

**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

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

Instruction Queue

Active List

Register Free List

Active list – maintains original instruction order, determines when a physical register can be freed.
I1: Div R5, R4, R2
I2: Add R7, R5, R1
I3: Sub R5, R3, R2
I4: Lw R7, 1000(R5)

I1: PR13
I2: PR30
I3: PR37
I4: PR4

I1, producing register 37, completes, broadcasts a completion signal to IQ
I2, producing register 4, completes, broadcasts a completion signal to IQ

I3, producing register 42, completes, broadcasts a completion signal to IQ
I4, producing register 19, completes, broadcasts a completion signal to IQ

I1 commits.
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.