**Chapter 3 – Instruction-Level Parallelism and its Exploitation (Part 1)**

ILP vs. Parallel Computers
Dynamic Scheduling (Section 3.4, 3.5)
Dynamic Branch Prediction (Section 3.3)
Hardware Speculation and Precise Interrupts (Section 3.6)
Multiple Issue (Section 3.7)
Static Techniques (Section 3.2, Appendix H)
Limitations of ILP
Multithreading (Section 3.11)
Putting it Together (Mini-projects)

---

**ILP vs. Parallel Computers**

- **Instruction-Level Parallelism (ILP)**
  Instructions of single process (or thread) executed in parallel
  Parallel components must appear to execute in sequential program order

- **Parallel Computers or Multiprocessors**
  Program divided into multiple processes (or threads)
  Instructions of multiple threads executed in parallel
  Typically also involves ILP within each thread
  No a priori sequential order between parallel threads

---

**Dynamic Scheduling - Basics**

The situation:
```
DIV.D F0, F2, F4
ADD.D F10, F0, F8
MULT.D F6, F6, F14
```

The problem:
- ADD stalls due to RAW hazard
- MULT stalls because ADD stalls

Example
```
1 2 3 4 5 6 7 8
DIV.D IF ID E/ E/ E/ E/ MEM WB
ADD.D IF ID ** ** ** E+ E+
MULT.D IF ** ** ** ID E* why stall?
```

In-order execution limits performance

---

**Dynamic Scheduling - Basics (Cont.)**

Solutions
- Static Scheduling
- Dynamic Scheduling

Static Scheduling (Software)
  - Compiler reorganizes instructions
  +
  + (Will see more later)

Dynamic Scheduling (Hardware)
  - Hardware reorganizes instructions
  +
  +
Dynamic Scheduling - Basics (Cont.)

In-order execution - Static
- Instructions sent to execution units sequentially
- Stall instruction $i + 1$ if instruction $i$ stalls for lack of operands

Out-of-order execution - Dynamic
- Send independent instructions to execution units as soon as possible

Dynamic Scheduling Basics (Cont.)

Original simple pipeline
- ID – decode, check all hazards, read operands
- EX – execute

Dynamic pipeline
- Split ID (“issue to execution unit”) into two parts
  - Check for structural hazards
  - Wait for data dependences
- New organization (conceptual):
  - Issue – decode, check structural hazards, read ready operands
  - ReadOps – wait until data hazards clear, read operands, begin execution
- Issue stays in-order; ReadOps/beginning of EX is out-of-order

Dynamic Scheduling Basics (Cont.)

Dynamic scheduling can create WAW, WAR hazards, and imprecise exceptions

WAW hazards with dynamic scheduling
- DIV.D F0, F2, F4
- ADD.D F10, F0, F8
- MUL.D F10, F8, F14

WAR hazards with dynamic scheduling
- DIV.D F0, F2, F4
- ADD.D F10, F0, F8
- MUL.D F8, F8, F14

Can always stall,
- but more aggressive solution with register renaming

Register Renaming - Tomasulo’s Algorithm

Registers are Names for data values
- Think of register specifiers as tags
- NOT storage locations
- Tomasulo’s algorithm exploited above in IBM 360/91

WAW hazards:
- DIV.D F0, F2, F4
- ADD.D F10, F0, F8
- MUL.D F10, F8, F14

WAR hazards:
- DIV.D F0, F2, F4
- ADD.D F10, F0, F8
- MUL.D F8, F8, F14
**Some History - IBM 360/91**

- Fast 360 for scientific code
- Completed in 1967
- Predates cache memories
- Pipelined, rather than multiple, functional units (FU)
- We will assume multiple functional units
- 360 had register memory instructions, we don’t

---

**Register Renaming - Tomasulo’s Algorithm**

- Tomasulo’s algm uses reservation stations for register renaming
- Instruction is “issued” to a reservation station
- A pending operand is designated via a tag
- Tag = reservation station that will provide the operand
- Reservation station with pending instruction fetches and buffers the operand when it becomes available
- All FUs place output on the common data bus (CDB) with tag
- Waiting reservation station gets the data from the CDB (register bypass)

---

**Tomasulo’s Algorithm - Implementation**

- Extend simple pipeline as example for Tomasulo’s algorithm
- Assume multiple FUs

---

**Our Tomasulo Pipeline**

- 3-stage Execution (ignore IF and MEM)
  - **Issue**: Get instruction from queue
    - ALU Op: Check for available reservation station
    - Load/Store: Check for available load/store buffer
    - If not, stall due to structural hazard
  - **Execute**: If operands available, execute operation
    - If not, monitor CDB for operand
  - **Write**: If CDB available, write it on CDB
    - If not, stall
Our Tomasulo Pipeline, cont

Reservation Stations
Handle distributed hazard detection and instruction control

Everything, except store buffers, has a tag

4-bit tag specifies reservation station or load buffer
Specifies which FU will produce result

Register specifier is used to assign tags

THEN IT'S DISCARDED!
Register specifiers are ONLY used in ISSUE

---

Our Tomasulo Pipeline, cont

Reservation Stations
Op Opcode
Qj, Qk Tag Fields
Vj, Vk Operand values
Busy Currently in use

Register File and Store Buffer
Qj Tag Field
Busy Currently in use

Load and Store Buffers
Busy Currently in use
A Address

Latencies: FP+ = 2, FP* = 10, FP/ = 40, Load/int = 1

---

Tomasulo Example

Example code

L.D F6,34(R2)
L.D F2,45(R3)
MULT.D F0,F2,F4
SUB.D F8,F6,F2
DIV.D F10,F0,F6
ADD.D F6,F8,F2

---

Tomasulo Example

Instruction Status (For illustration ONLY)

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Issue</th>
<th>Execute</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F6,34(R2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F2,45(R3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT.D</td>
<td>F0,F2,F4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB.D</td>
<td>F8,F6,F2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV.D</td>
<td>F10,F0,F6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F6,F8,F2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FU Name Busy Op Vj Vk Qj Qk
1 Add1
2 Add2
3 Add3
4 Mult1
5 Mult2

Register Result Status

<table>
<thead>
<tr>
<th>Register Result Status</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>...</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Busy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Tomasulo Example

**Instruction Status (For illustration ONLY)**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue</th>
<th>Execute</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0,48(R0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F2,36(R3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULT.D</td>
<td>F0,F2,F4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV.D</td>
<td>F10,F0,F6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F8,F9,F2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FU Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Add2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Add3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Mult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Mult</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register Result Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
</tr>
<tr>
<td>QI</td>
</tr>
</tbody>
</table>

### Tomasulo, cont.

Out-of-order loads and stores?

CDB is a bottleneck
Could duplicate
Increases the required hardware
Complex implementation

### Tomasulo, cont.

Advantages
- Distribution of hazard detection
- Elimination of WAR and WAW stalls

Common Data Bus
- Broadcasts results to multiple instructions, bypasses registers
- Central bottleneck
  - Could duplicate (increases required hardware)

Register Renaming
- Eliminates WAR and WAW Hazards
- Allows dynamic loop unrolling
  - Especially important with only 4 registers
  - Requires many associative lookups
Loops with Tomasulo’s Algorithm

Consider the following example:

**FORTRAN:**
DO I = 1, N
    C[I] = A[I] + s * B[I]

**ASSEMBLY:**
L.D F0, A(R1)
L.D F2, B(R1)
MUL.D F2, F2, F4 /* s in F4 */
ADD.D F2, F2, F0
S.D C(R1), F2
Branch code

What would Tomasulo’s algorithm do?