Dynamic Scheduling I

- basic pipeline started with single, in-order issue, single-cycle operations
- have extended this basic pipeline with
  - multi-cycle operations
  - multiple issue
- now: dynamic scheduling (out-of-order issue)
  - Scoreboard: OoO without solving WAW/WAR
  - Tomasulo's algorithm: OoO + register renaming to fix WAR/WAW
- next half unit: dynamic scheduling II
  - dynamic scheduling + precise state + speculation
  - advanced topic: dynamic load scheduling

Dynamic Scheduling: Motivation

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>divf</td>
<td>F</td>
<td>D</td>
<td>E/</td>
<td>E/</td>
<td>E/</td>
<td>E/</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addf</td>
<td>F</td>
<td>D</td>
<td>d*</td>
<td>d*</td>
<td>d*</td>
<td>E+</td>
<td>E+</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mulf</td>
<td>F</td>
<td>p*</td>
<td>p*</td>
<td>p*</td>
<td>D</td>
<td>E*</td>
<td>E*</td>
<td>W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- cycle4: addf stalls due to RAW hazard
  - OK, fundamental problem
- also cycle4: mulf stalls due to pipeline hazard (addf stalls)
  - why? mulf can't proceed into ID because addf is there
  - but that's the only reason ⇒ not good enough!
- why can't we decode mulf in cycle 4 and execute it in c5?
  - no fundamental reason why we can't do this!

Dynamic Scheduling

dynamic scheduling (out-of-order execution)

- execute instructions in non-sequential (non-vonNeumann) order
  - reduce stalls
  - improve functional unit utilization
  - enable parallel execution (not in-order ⇒ can be in parallel)
- make it appear like sequential execution: precise interrupts
  - very important
    - but hard
  - next unit of this course

Readings

H+P
- chapter 2

Recent Research Papers (can read these soon)
- Pentium4
- Complexity-Effective Superscalar
- Checkpoint Processing and Recovery
Scheduling

scheduling: re-arranging instructions to maximize performance
  • requires knowledge about structure of processor
  • requires knowledge about latencies and dependences

two options for who should schedule instructions
  • static scheduling: by compiler
  • dynamic scheduling: by hardware

Before We Start

why build complicated hardware if we can do this in software?
  + performance portability
    • don't want to recompile for new machines
  + more information available to hardware
    • addresses, branch directions, cache misses unknown to compiler
  + more resources available to hardware
    • may not have enough architectural registers to fix WAR/WAW
  + easier to speculate in hardware
    • easier to recover from mis-speculation
  – but compiler can look at more instructions
    • it's possible to do combination of both
    • compiler does as much as it can (not much), hardware does rest

The Problem with In-Order Pipelines

in-order pipeline
  • often written as IF,ID, EX (multiple cycle, includes M) , WB
  • structural hazard: 1 instruction register (latch) per stage
    • 1 instruction per stage per cycle (unless pipe is replicated)
    • younger instruction can’t pass older without “killing” it

out-of-order pipeline
  • must implement “passing” functionality

Instruction Buffer

trick: instruction buffer (many names for this buffer)
  • basically: a bunch of latches for holding instructions
    • this is the scope of instructions that the scheduler can see
  • split ID into two pieces
    • accumulate decoded instructions in buffer in-order
    • buffer sends instructions down rest of pipe out-of-order
Dispatch and Issue

- **Dispatch (DS):** first part of ID
  - Allocate resources in instruction buffer
    - New kind of structural hazard (instruction buffer could be full)
    - Dispatch is in-order, and stall propagates to younger instructions

- **Issue (IS):** second part of ID
  - Send instructions from instruction buffer to execution units
  - Out-of-order, wait does NOT propagate to younger instructions

DS Method #1: Scoreboarding

- **Instruction buffer ⇒ scoreboard**
  - **Centralized** control scheme
  - No bypassing
  - No elimination of WAR/WAW hazards

- First implementation: CDC6600 [1964]
  - 16 separate non-pipelined functional units
  - 4 FP, 5 memory, 7 integer

- Our example: Simple Scoreboard
  - 5 functional units: 1 ALU, 1 load, 1 store, 2 FP (3-cycle, pipelined)
  - For simplicity, assume 1-wide pipeline (not superscalar)

Scoreboard Data Structures

- **Instruction status:** 1 entry per “active” instruction
  - Which stage instruction is in (presence in scoreboard implies DS)

- **Functional unit (FU) status:** 1 entry per FU
  - **Busy:** FU is busy, **Op:** current operation
  - R1, R2, R: source and destination registers
  - T1, T2: tags of FUs producing source registers
  - T: tag of FU producing destination register

- **Register status:** 1 entry per architectural register
  - T: tag of FU (if any) that will write the register

- **Tag fields interpreted as “ready bits” (conversely “busy bits”)**
  - Tag == 0: register value is ready (in register file)
  - Tag != 0: register value is not ready (will be supplied by [tag])

Simple Scoreboard

- Instruction fields and status bits
  - **Tags**
  - **Values**
Scoreboard Pipeline

new pipeline structure: IF, DS, IS, EX, WB

• **DS (dispatch)** from fetch to the scoreboard
  • (no scoreboard entry/structural hazard/WAW) ? (stall) : (allocate)

• **IS (issue)** to the functional units
  • (RAW hazard) ? (wait) : (read registers, go directly to execute)

• **EX (execute)**
  • execute operation, notify scoreboard when done

• **WB (writeback)**
  • (WAR hazard) ? (wait) : (write register, free scoreboard entry)

• assume
  • WB and RAW-dependent IS can take place in same cycle
  • WB and structural-dependent DS can take place in same cycle

Scoreboard: Dispatch (DS)

Scoreboard: Issue (IS)

Scoreboard: Execute (EX)
Scoreboard: Writeback (WB)

- wait for WAR hazards, but otherwise:
  - writeback result
  - compare tags with waiting instructions
  - on match: clear tag (set input to “ready”)

Scoreboard Data Structures

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction</td>
<td>register</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>f0</td>
</tr>
<tr>
<td>mulf f4,f0,f2</td>
<td>f2</td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td>f4</td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>r1</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td></td>
</tr>
<tr>
<td>mulf f4,f0,f2</td>
<td></td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td></td>
</tr>
</tbody>
</table>

Functional unit status

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Running Example

SAX: simplified SAXPY

\[ \text{DO } I = 1, N \]

\[ Z[I] = A \times X[I] \]

assembly code:

\[
\text{loop:}
\]

\[
\text{ldf } f_0, X(r_1) \quad \text{ // } f_0 = X[I] \text{, assume } I \text{ in } r_1
\]

\[
\text{mulf } f_4, f_0, f_2 \quad \text{ // assume } A \text{ in } f_2
\]

\[
\text{stf } f_4, Z(r_1) \quad \text{ // } Z[I] = A \times X[I]
\]

\[
\text{add } r_1, r_1, #4 \quad \text{ // } I = I+4
\]

\[
\text{ble } r_1, r_2, \text{loop} \quad \text{ // assume } 4N \text{ in } r_2
\]

consider two iterations, ignore branch

Scoreboard Example: Cycle 1

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction</td>
<td>register</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c1</td>
</tr>
<tr>
<td>mulf f4,f0,f2</td>
<td>f2</td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td>f4</td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>r1</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td></td>
</tr>
<tr>
<td>mulf f4,f0,f2</td>
<td></td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td></td>
</tr>
</tbody>
</table>

Functional unit status

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>Yes</td>
<td>idf</td>
<td>T0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

allocate
### Scoreboard Example: Cycle 2

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ldf f0, X(r1)</code></td>
<td><code>f0</code> load</td>
</tr>
<tr>
<td><code>mulf f4, f0, f2</code></td>
<td><code>f2</code></td>
</tr>
<tr>
<td><code>stf f4, Z(r1)</code></td>
<td><code>f4</code> FP1</td>
</tr>
<tr>
<td><code>add r1, r1, #8</code></td>
<td><code>r1</code> ALU</td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>Yes</td>
<td>ldf</td>
<td>f0</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>Yes</td>
<td>mulf</td>
<td>f4</td>
<td>f0</td>
<td>f2</td>
<td>load</td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **allocate**

### Scoreboard Example: Cycle 3

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ldf f0, X(r1)</code></td>
<td><code>f0</code> load</td>
</tr>
<tr>
<td><code>mulf f4, f0, f2</code></td>
<td><code>f2</code></td>
</tr>
<tr>
<td><code>stf f4, Z(r1)</code></td>
<td><code>f4</code> FP1</td>
</tr>
<tr>
<td><code>add r1, r1, #8</code></td>
<td><code>r1</code> ALU</td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>Yes</td>
<td>ldf</td>
<td>f0</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>f4</td>
<td>r1</td>
<td>FP1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>Yes</td>
<td>mulf</td>
<td>f4</td>
<td>f0</td>
<td>f2</td>
<td>load</td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **allocate**
- **stalled on RAW**

### Scoreboard Example: Cycle 4

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ldf f0, X(r1)</code></td>
<td><code>f0</code> load</td>
</tr>
<tr>
<td><code>mulf f4, f0, f2</code></td>
<td><code>f2</code></td>
</tr>
<tr>
<td><code>stf f4, Z(r1)</code></td>
<td><code>f4</code> FP1</td>
</tr>
<tr>
<td><code>add r1, r1, #8</code></td>
<td><code>r1</code> ALU</td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Yes</td>
<td>add</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>f4</td>
<td>r1</td>
<td>FP1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>Yes</td>
<td>mulf</td>
<td>f4</td>
<td>f0</td>
<td>f2</td>
<td>load</td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **allocate**
- **result written, clear status**

### Scoreboard Example: Cycle 5

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ldf f0, X(r1)</code></td>
<td><code>f0</code> load</td>
</tr>
<tr>
<td><code>mulf f4, f0, f2</code></td>
<td><code>f2</code></td>
</tr>
<tr>
<td><code>stf f4, Z(r1)</code></td>
<td><code>f4</code> FP1</td>
</tr>
<tr>
<td><code>add r1, r1, #8</code></td>
<td><code>r1</code> ALU</td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Yes</td>
<td>add</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>Yes</td>
<td>ldf</td>
<td>f0</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>f4</td>
<td>r1</td>
<td>FP1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP1</td>
<td>Yes</td>
<td>mulf</td>
<td>f4</td>
<td>f0</td>
<td>f2</td>
<td>load</td>
<td></td>
</tr>
<tr>
<td>FP2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **allocate**

---

© 2007 by Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

ECE 252 / CPS 220 Lecture Notes
Dynamic Scheduling I
### Scoreboard Example: Cycle 6

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ldf f0,X(r1)</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>mul f4,f0,f2</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>stf f4,Z(r1)</code></td>
<td><code>T</code></td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Yes</td>
<td>add</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>Yes</td>
<td>ldf</td>
<td>f0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>f4</td>
<td>r1</td>
<td></td>
<td></td>
<td>FP1</td>
</tr>
</tbody>
</table>

### Scoreboard Example: Cycle 7

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ldf f0,X(r1)</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>mul f4,f0,f2</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>stf f4,Z(r1)</code></td>
<td><code>T</code></td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Yes</td>
<td>add</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>Yes</td>
<td>ldf</td>
<td>f0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>f4</td>
<td>r1</td>
<td></td>
<td></td>
<td>FP1</td>
</tr>
</tbody>
</table>

### Scoreboard Example: Cycle 8

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ldf f0,X(r1)</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>mul f4,f0,f2</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>add r1,r1,8</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>ldf f0,X(r1)</code></td>
<td><code>T</code></td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Yes</td>
<td>add</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>Yes</td>
<td>ldf</td>
<td>f0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>f4</td>
<td>r1</td>
<td></td>
<td></td>
<td>FP1</td>
</tr>
<tr>
<td>FP1</td>
<td>Yes</td>
<td>mul</td>
<td>f4</td>
<td>f0</td>
<td>f2</td>
<td>load</td>
<td></td>
</tr>
</tbody>
</table>

**First mul f4 (FP1) is finished**

**WB stall due to WAR hazard**

### Scoreboard Example: Cycle 9

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ldf f0,X(r1)</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>mul f4,f0,f2</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>add r1,r1,8</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>ldf f0,X(r1)</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>stf f4,Z(r1)</code></td>
<td><code>T</code></td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Yes</td>
<td>add</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>Yes</td>
<td>ldf</td>
<td>f0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>f4</td>
<td>r1</td>
<td></td>
<td></td>
<td>FP1</td>
</tr>
<tr>
<td>FP1</td>
<td>Yes</td>
<td>mul</td>
<td>f4</td>
<td>f0</td>
<td>f2</td>
<td>load</td>
<td></td>
</tr>
</tbody>
</table>

**First mul f4 (FP1) is finished**

**WB stall due to WAR hazard**

**DS stall due to structural hazard**

### Scoreboard Example: Cycle 10

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ldf f0,X(r1)</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>mul f4,f0,f2</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>add r1,r1,8</code></td>
<td><code>T</code></td>
</tr>
<tr>
<td><code>ldf f0,X(r1)</code></td>
<td><code>T</code></td>
</tr>
</tbody>
</table>

**Functional unit status**

<table>
<thead>
<tr>
<th>T</th>
<th>busy</th>
<th>op</th>
<th>R</th>
<th>R1</th>
<th>R2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Yes</td>
<td>add</td>
<td>r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>load</td>
<td>Yes</td>
<td>ldf</td>
<td>f0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>f4</td>
<td>r1</td>
<td></td>
<td></td>
<td>FP1</td>
</tr>
<tr>
<td>FP1</td>
<td>Yes</td>
<td>mul</td>
<td>f4</td>
<td>f0</td>
<td>f2</td>
<td>load</td>
<td></td>
</tr>
</tbody>
</table>

**f4 is ready**

**allocate**

**free entry**

**r1 is ready**
Scoreboard Example: Cycle 10

<table>
<thead>
<tr>
<th>Instruction Status</th>
<th>Register Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>Register</td>
</tr>
<tr>
<td>op</td>
<td>DS</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>f0</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c2</td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td>c3</td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>c4</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c5</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c8</td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td>c10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional unit status</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>ALU</td>
</tr>
<tr>
<td>load</td>
</tr>
<tr>
<td>store</td>
</tr>
<tr>
<td>FP1</td>
</tr>
<tr>
<td>FP2</td>
</tr>
</tbody>
</table>

Register Renaming

- change register names to eliminate WAR/WAW hazards
- one of the most elegant concepts in computer architecture

key: think of architectural registers as names, not locations
- can have more locations than names
- dynamically map names to locations
- map table holds the current mappings (name→location)
  - write: allocate new location and record it in map table
  - read: find location of most recent write by name lookup in map table
  - minor detail: must de-allocate locations appropriately

Scoreboard Redux

+ cheap hardware
  - scoreboard is cheap (~1 FU in area)
+ pretty good performance
  - 1.7X for FORTRAN programs
  - 2.5X for hand-coded assembly (how would a compiler do?)
- no bypassing
  - RAW dependences handled through registers
- limited scheduling scope
  - WAW/structural hazards force in-order dispatch
  - WAR hazards delay writeback and issue of dependent operations
  - can solve these problems with register renaming!

Register Renaming Example

- names: r1, r2, r3, locations: 11, 12, 13, 14, 15, 16, 17
- original mapping: r1→11, r2→12, r3→13 (14–17 “free”)

raw instructions  | map table | free locations  | renamed instructions |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>r1</td>
<td>r2</td>
<td>r3</td>
</tr>
<tr>
<td>add r1,r2,r3</td>
<td>14</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>sub r2,r1,r3</td>
<td>14</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>mul r2,r2,r3</td>
<td>16</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>div r2,r1,r3</td>
<td>16</td>
<td>17</td>
<td>15</td>
</tr>
</tbody>
</table>

- renaming removes WAW/WAR, leaves RAW intact!!
  - Tomasulo’s algorithm (next) implements this concept “in principle”
DS Method #2: Tomasulo's Algorithm

instruction buffer ⇒ reservation stations (RS)
  - **distributed** control scheme (Scoreboard was centralized)
  - uses data bypassing
  - **common data bus (CDB)** broadcasts results to RS
  - **register renaming** eliminates WAR/WAW hazards

- first implementation: IBM 360/91 [1967]
  - dynamic scheduling for **FP units only**
- our example: Simple Tomasulo
  - dynamic scheduling for **everything**
  - load/store buffers replaced by reservation stations
  - no bypassing (for comparison with Scoreboard)
  - 5 RS: 1 ALU, 1 load, 1 store, 2 FP (3 cycle, pipelined)

Tomasulo Data Structures

- reservation stations
  - **busy**, **FU**, **op**, **R**: destination register name
  - **T1, T2**: source register tag (RS# that will produce the value)
  - **T**: destination register tag (RS# of this RS)
  - **V1, V2**: source register value

- register table
  - **T**: tag (RS# that will write register)

- CDB: common data bus
  - broadcasts <value, tag> of completed instructions
  - tags interpreted as (more sophisticated) ready bits
    - **tag == 0? value is ready (somewhere)**
    - **tag != 0? value is not ready, wait until CDB broadcasts this tag**

Simple Tomasulo

- instruction fields and status bits
  - **tags**
  - **values**

Scoreboard vs. Tomasulo

- what about Tomasulo implements register renaming?
  - value copies in reservation stations (RS)
  - instruction holds correct input values in its own RS
  - future instructions can overwrite RF master copy, won’t matter!
Tomasulo Pipeline

new pipeline structure: IF, DS, IS, EX, WB

- **DS (dispatch)**
  - (available RS)?
    - (allocate RS, copy ready values, non-ready tags to RS) : (stall)
- **IS (issue)**
  - (operands ready)? (execute) : (wait, monitor CDB)
- **WB (writeback)**
  - (CDB available)? (broadcast result, write reg, free RS) : (wait)
- assume
  - WB and RAW dependent IS can go in same cycle
  - WB and structural dependent DS can go in same cycle

Tomasulo: Dispatch (DS)

Tomasulo: Issue (IS)

Tomasulo: Execute (EX)
Tomasulo: Writeback (WB)

- wait for free CDB
  - broadcast result (value+tag) on CDB
  - write result to register + clear reg status (if tag matches)
  - compare with RS input tags (match? clear tag + copy value)

Tomasulo Data Structures

<table>
<thead>
<tr>
<th>Instruction Status (illustration only)</th>
<th>Reg. Status</th>
<th>CDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldf $f0, X(r1)$</td>
<td>reg $f0$</td>
<td>$r1$</td>
</tr>
<tr>
<td>mulf $f4, f0, f2$</td>
<td>$f2$</td>
<td></td>
</tr>
<tr>
<td>stf $f4, Z(r1)$</td>
<td>$f4$</td>
<td></td>
</tr>
<tr>
<td>add $r1, r1, #4$</td>
<td>$r1$</td>
<td></td>
</tr>
<tr>
<td>ldf $f0, X(r1)$</td>
<td>reg $f0$</td>
<td></td>
</tr>
<tr>
<td>mulf $f4, f0, f2$</td>
<td>$f2$</td>
<td></td>
</tr>
<tr>
<td>stf $f4, Z(r1)$</td>
<td>$f4$</td>
<td></td>
</tr>
</tbody>
</table>

Reservation Stations & Load/Store Buffers

<table>
<thead>
<tr>
<th>T</th>
<th>FU</th>
<th>busy</th>
<th>op</th>
<th>V1</th>
<th>V2</th>
<th>T1</th>
<th>T2</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALU</td>
<td>No</td>
<td>V1</td>
<td>V2</td>
<td>T1</td>
<td>T2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>load</td>
<td>No</td>
<td>V1</td>
<td>V2</td>
<td>T1</td>
<td>T2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>store</td>
<td>No</td>
<td>V1</td>
<td>V2</td>
<td>T1</td>
<td>T2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FP1</td>
<td>No</td>
<td>V1</td>
<td>V2</td>
<td>T1</td>
<td>T2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FP2</td>
<td>No</td>
<td>V1</td>
<td>V2</td>
<td>T1</td>
<td>T2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tomasulo-Style Register Renaming

names: architectural registers

locations: registers in register file AND reservation stations (RS)
  - values can (and do) exist in both!
    - copies eliminate WAR hazards
  - called “value-based” or “copy-based” renaming

locations referred to internally by tags
  - register table translates names to tags
    - tag == 0 means “in register file”
    - tag != 0 means “in RS#tag”
  - CDB broadcasts values with tags attached
    - so instructions know what value they are looking at
Tomasulo Example: Cycle 1

Instruction Status (illustration only) | Reg. Status | CDB
--- | --- | ---
ldf f0,X(r1) | c1 | f0 RS#2
mulf f4,f0,f2 | c2 | f2
stf f4,Z(r1) | c3 | f4 RS#4
add r1,r1,#4 | | r1
ldf f0,X(r1) | | 
mulf f4,f0,f2 | | 
stf f4,Z(r1) | | 

Reservation Stations & Load/Store Buffers

| T | FU | busy | op | V1 | V2 | T1 | T2 | addr |
--- | --- | --- | --- | --- | --- | --- | --- | --- |
1 | ALU | No | | | | | | |
2 | load | Yes | ld | REG[r1] | &X[0] | | | |
3 | store | No | | | | | | |
4 | FP1 | No | | | | | | |
5 | FP2 | No | | | | | | |

Tomasulo Example: Cycle 2

Instruction Status (illustration only) | Reg. Status | CDB
--- | --- | ---
ldf f0,X(r1) | c1 | f0 RS#2
mulf f4,f0,f2 | c2 | f2
stf f4,Z(r1) | c3 | f4 RS#4
add r1,r1,#4 | | r1
ldf f0,X(r1) | | 
mulf f4,f0,f2 | | 
stf f4,Z(r1) | | 

Reservation Stations & Load/Store Buffers

| T | FU | busy | op | V1 | V2 | T1 | T2 | addr |
--- | --- | --- | --- | --- | --- | --- | --- | --- |
1 | ALU | No | | | | | | |
2 | load | Yes | ld | REG[r1] | &X[0] | | | |
3 | store | No | | | | | | |
4 | FP1 | Yes | mulf | REG[f2] | RS#2 | | | |
5 | FP2 | No | | | | | | |

allocate RS, set reg. status

Tomasulo Example: Cycle 3

Instruction Status (illustration only) | Reg. Status | CDB
--- | --- | ---
ldf f0,X(r1) | c1 | f0 RS#2
mulf f4,f0,f2 | c2 | f2
stf f4,Z(r1) | c3 | f4 RS#4
add r1,r1,#4 | | r1
ldf f0,X(r1) | | 
mulf f4,f0,f2 | | 
stf f4,Z(r1) | | 

Reservation Stations & Load/Store Buffers

| T | FU | busy | op | V1 | V2 | T1 | T2 | addr |
--- | --- | --- | --- | --- | --- | --- | --- | --- |
1 | ALU | No | | | | | | |
2 | load | Yes | ld | REG[r1] | &X[0] | | | |
3 | store | Yes | stf | REG[r1] | RS#4 &Z[0] | | | |
4 | FP1 | Yes | mulf | REG[f2] | RS#2 | | | |
5 | FP2 | No | | | | | | |

allocate RS, no reg. status (store)

Tomasulo Example: Cycle 4

Instruction Status (illustration only) | Reg. Status | CDB
--- | --- | ---
ldf f0,X(r1) | c1 | f0 RS#2
mulf f4,f0,f2 | c2 | f2
stf f4,Z(r1) | c3 | f4 RS#4
add r1,r1,#4 | | r1
ldf f0,X(r1) | | 
mulf f4,f0,f2 | | 
stf f4,Z(r1) | | 

Reservation Stations & Load/Store Buffers

| T | FU | busy | op | V1 | V2 | T1 | T2 | addr |
--- | --- | --- | --- | --- | --- | --- | --- | --- |
1 | ALU | Yes | add | REG[r1] | | | | |
2 | load | No | | | | | | |
3 | store | Yes | stf | REG[r1] | RS#4 &Z[0] | | | |
4 | FP1 | Yes | mulf | CDB.V REG[f2] | RS#2 | | | |
5 | FP2 | No | | | | | | |

allocate RS, set reg. status

ldf finished
1. write/clear status
2. CDB broadcast

allocate RS, grab from CDB

f0 is ready
Tomasulo Example: Cycle 5

<table>
<thead>
<tr>
<th>Instruction Status (illustration only)</th>
<th>Reg. Status</th>
<th>CDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction</td>
<td>reg</td>
<td>T</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c1</td>
<td>c2</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c2</td>
<td>c4</td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td>c3</td>
<td></td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>c4</td>
<td>c5</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c5</td>
<td></td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reservation Stations & Load/Store Buffers

<table>
<thead>
<tr>
<th>T</th>
<th>FU</th>
<th>busy</th>
<th>op</th>
<th>V1</th>
<th>V2</th>
<th>T1</th>
<th>T2</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALU</td>
<td>Yes</td>
<td>add</td>
<td>REG[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>load</td>
<td>ldf</td>
<td>REG[1]</td>
<td>&amp;X[1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>REG[1]</td>
<td>&amp;Z[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FP1</td>
<td>Yes</td>
<td>mulf</td>
<td>REG[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FP2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tomasulo Example: Cycle 6

<table>
<thead>
<tr>
<th>Instruction Status (illustration only)</th>
<th>Reg. Status</th>
<th>CDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction</td>
<td>reg</td>
<td>T</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c1</td>
<td>c2</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c2</td>
<td>c4</td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td>c3</td>
<td></td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>c4</td>
<td>c5</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c5</td>
<td></td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reservation Stations & Load/Store Buffers

<table>
<thead>
<tr>
<th>T</th>
<th>FU</th>
<th>busy</th>
<th>op</th>
<th>V1</th>
<th>V2</th>
<th>T1</th>
<th>T2</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALU</td>
<td>Yes</td>
<td>add</td>
<td>REG[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>load</td>
<td>ldf</td>
<td>REG[1]</td>
<td>&amp;X[1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>REG[1]</td>
<td>&amp;Z[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FP1</td>
<td>Yes</td>
<td>mulf</td>
<td>REG[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FP2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tomasulo Example: Cycle 7

<table>
<thead>
<tr>
<th>Instruction Status (illustration only)</th>
<th>Reg. Status</th>
<th>CDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction</td>
<td>reg</td>
<td>T</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c1</td>
<td>c2</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c2</td>
<td>c4</td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td>c3</td>
<td></td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>c4</td>
<td>c5</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c5</td>
<td></td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reservation Stations & Load/Store Buffers

<table>
<thead>
<tr>
<th>T</th>
<th>FU</th>
<th>busy</th>
<th>op</th>
<th>V1</th>
<th>V2</th>
<th>T1</th>
<th>T2</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALU</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>load</td>
<td>ldf</td>
<td>CDB.V</td>
<td>&amp;X[1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>REG[1]</td>
<td>&amp;Z[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FP1</td>
<td>Yes</td>
<td>mulf</td>
<td>REG[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FP2</td>
<td>Yes</td>
<td>mulf</td>
<td>REG[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tomasulo Example: Cycle 8

<table>
<thead>
<tr>
<th>Instruction Status (illustration only)</th>
<th>Reg. Status</th>
<th>CDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction</td>
<td>reg</td>
<td>T</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c1</td>
<td>c2</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c2</td>
<td>c4</td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td>c3</td>
<td></td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>c4</td>
<td>c5</td>
</tr>
<tr>
<td>ldf f0,X(r1)</td>
<td>c5</td>
<td></td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stf f4,Z(r1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reservation Stations & Load/Store Buffers

<table>
<thead>
<tr>
<th>T</th>
<th>FU</th>
<th>busy</th>
<th>op</th>
<th>V1</th>
<th>V2</th>
<th>T1</th>
<th>T2</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALU</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>load</td>
<td>ldf</td>
<td>CDB.V</td>
<td>&amp;X[1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>store</td>
<td>Yes</td>
<td>stf</td>
<td>REG[1]</td>
<td>&amp;Z[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FP1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FP2</td>
<td>Yes</td>
<td>mulf</td>
<td>REG[2]</td>
<td>RS#2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The images show the progression of instruction processing through cycles 5 to 8, illustrating the Tomasulo algorithm's operation with respect to instruction status, reservation stations, and load/store buffers.
Tomasulo Example: Cycle 9

<table>
<thead>
<tr>
<th>Instruction Status (illustration only)</th>
<th>Reg. Status</th>
<th>CDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>DS</td>
<td>IS</td>
</tr>
<tr>
<td>load f4,0,f2</td>
<td>c2</td>
<td>c4</td>
</tr>
<tr>
<td>mul f4,z(r1)</td>
<td>c3</td>
<td>c8</td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>c4</td>
<td>c5</td>
</tr>
<tr>
<td>load f0,x(r1)</td>
<td>c5</td>
<td>c7</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c6</td>
<td>c9</td>
</tr>
<tr>
<td>stf f4,z(r1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FU</th>
<th>busy</th>
<th>op</th>
<th>V1</th>
<th>V2</th>
<th>T1</th>
<th>T2</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALU</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>load</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>store</td>
<td>stf</td>
<td>REG[f1]</td>
<td>&amp;Z[0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FP1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FP2</td>
<td>Yes</td>
<td>mul</td>
<td>CDB.V</td>
<td>REG[f2]</td>
<td>RS#2</td>
<td></td>
</tr>
</tbody>
</table>

Tomasulo Example: Cycle 10

<table>
<thead>
<tr>
<th>Instruction Status (illustration only)</th>
<th>Reg. Status</th>
<th>CDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>DS</td>
<td>IS</td>
</tr>
<tr>
<td>load f0,x(r1)</td>
<td>c1</td>
<td>c2</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c2</td>
<td>c4</td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>c4</td>
<td>c5</td>
</tr>
<tr>
<td>load f0,x(r1)</td>
<td>c5</td>
<td>c7</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c8</td>
<td></td>
</tr>
<tr>
<td>stf f4,z(r1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FU</th>
<th>busy</th>
<th>op</th>
<th>V1</th>
<th>V2</th>
<th>T1</th>
<th>T2</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALU</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>load</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>store</td>
<td>stf</td>
<td>REG[f1]</td>
<td>RS#5</td>
<td>&amp;Z[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FP1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FP2</td>
<td>Yes</td>
<td>mul</td>
<td>CDB.V</td>
<td>REG[f2]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scoreboard vs. Tomasulo

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Scoreboard</th>
<th>Tomasulo</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldf f0,x(r1)</td>
<td>c1</td>
<td>c2</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c2</td>
<td>c4</td>
</tr>
<tr>
<td>add r1,r1,#4</td>
<td>c4</td>
<td>c5</td>
</tr>
<tr>
<td>ldf f0,x(r1)</td>
<td>c5</td>
<td>c7</td>
</tr>
<tr>
<td>mul f4,f0,f2</td>
<td>c8</td>
<td>c10</td>
</tr>
<tr>
<td>stf f4,z(r1)</td>
<td>c10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>hazard</th>
<th>Scoreboard</th>
<th>Tomasulo</th>
</tr>
</thead>
<tbody>
<tr>
<td>insn buffer</td>
<td>stall in IS</td>
<td>stall in IS</td>
</tr>
<tr>
<td>FU</td>
<td>wait in IS</td>
<td>wait in IS</td>
</tr>
<tr>
<td>RAW</td>
<td>wait in IS</td>
<td>wait in IS</td>
</tr>
<tr>
<td>WAR</td>
<td>wait in WB</td>
<td>none</td>
</tr>
<tr>
<td>WAW</td>
<td>wait in DS</td>
<td>none</td>
</tr>
</tbody>
</table>

Tomasulo Redux

- what about out-of-order loads and stores?
  - compare load address against the addresses in store buffers
  - wait on a match
  - much more on this tough problem later ...

- RS
  - distributed hazard detection
  - register renaming eliminates WAW hazards
  - copying values to RS eliminates WAR hazards
  - CDB tag matches require many associative compares
RS Implementation and Design

- tag/CDB part is called “instruction window” or “scheduler”
  - tag/CDB are universal, values not necessarily (later)
  - two components (can be pipelined)
- (1) wakeup: CDB tag broadcast and match
  - long bus, many comparators
- (2) select: choose instructions to issue this cycle
  - $M \rightarrow N$ priority encoder
- design: split (separate for each FU) vs. unified (shared)
  - split: +faster, +simpler logic $C(N,1)$, unified: +utilization
- design: FIFO vs. RAM
  - FIFO: +simple (only with split), RAM: +high performance

Superscalar (Wide) Tomasulo

- dynamic scheduling and multiple issue are orthogonal
  - modern processors have both
  - two dimensions
  - $N$: superscalar width (number of parallel operations)
  - $W$: window size (number of reservation stations)
    - sometimes called “max parallelism” and “max reordering”
- what do we need for $N$-by-$W$ Tomasulo?
  - DS: $N$ RS write ports, $2N$ RegStatus read ports, $N$ RSt write ports,
    $2N$ RS value write ports, $2N$ RF read ports
  - IS: $N$ RS read ports
  - WB: $N$ CDBs, $2NW$ comparators, $2N$ RS value write ports
    - it’s complicated!

Dynamic Scheduling Summary

- dynamic scheduling: out-of-order execution
  - higher utilization, improved performance
  - easier in hardware (we’ll see why later in course)
- single F/D latch: structural impediment to OOO execution
  - instruction buffer: multiple F/D latches
  - split ID into in-order DS and out-of-order IS
- DS implementations
  - Scoreboard: out-of-order without renaming
  - Tomasulo: out-of-order with renaming

next up: DS + precise state + speculation