Last Time in Lecture 5

- Decoupled execution
- Simple out-of-order scoreboard for CDC6600
- Tomasulo algorithm for register renaming
IBM 360/91 Floating-Point Unit
R. M. Tomasulo, 1967

Load buffers (from memory)

Adder

Mult

Common bus ensures that data is made available immediately to all the instructions waiting for it. Match tag, if equal, copy value & set presence “p”.

Distribute reservation stations to functional units

Store buffers (to memory)

Floating-Point Regfile
Out-of-Order Fades into Background

Out-of-order processing implemented commercially in 1960s, but disappeared again until 1990s as two major problems had to be solved:

- Precise traps
  - Imprecise traps complicate debugging and OS code
  - Note, precise interrupts are relatively easy to provide

- Branch prediction
  - Amount of exploitable instruction-level parallelism (ILP) limited by control hazards

Also, simpler machine designs in new technology beat complicated machines in old technology

- Big advantage to fit processor & caches on one chip
- Microprocessors had era of 1%/week performance scaling
Separating Completion from Commit

- Re-order buffer holds register results from completion until commit
  - Entries allocated in program order during decode
  - Buffers completed values and exception state until in-order commit point
  - Completed values can be used by dependents before committed (bypassing)
  - Each entry holds program counter, instruction type, destination register specifier and value if any, and exception status (info often compressed to save hardware)

- Memory reordering needs special data structures
  - Speculative store address and data buffers
  - Speculative load address and data buffers
In-Order Commit for Precise Traps

- In-order instruction fetch and decode, and dispatch to reservation stations inside reorder buffer
- Instructions issue from reservation stations out-of-order
- Out-of-order completion, values stored in temporary buffers
- Commit is in-order, checks for traps, and if none updates architectural state
Phases of Instruction Execution

**Fetch:** Instruction bits retrieved from instruction cache.

**Decode:** Instructions dispatched to appropriate issue buffer

**Execute:** Instructions and operands issued to functional units. When execution completes, all results and exception flags are available.

**Commit:** Instruction irrevocably updates architectural state (aka “graduation”), or takes precise trap/interrupt.
In-Order versus Out-of-Order Phases

- Instruction fetch/decode/rename always in-order
  - Need to parse ISA sequentially to get correct semantics
  - Proposals for speculative OoO instruction fetch, e.g., Multiscalar. Predict control flow and data dependencies across sequential program segments fetched/decoded/executed in parallel, fixup if prediction wrong
- Dispatch (place instruction into machine buffers to wait for issue) also always in-order
  - Dispatch sometimes used to mean issue, but not in these lectures
In-Order Versus Out-of-Order Issue

- **In-order issue:**
  - Issue stalls on RAW dependencies or structural hazards, or possibly WAR/WAW hazards
  - Instruction cannot issue to execution units unless all preceding instructions have issued to execution units

- **Out-of-order issue:**
  - Instructions dispatched in program order to reservation stations (or other forms of instruction buffer) to wait for operands to arrive, or other hazards to clear
  - While earlier instructions wait in issue buffers, following instructions can be dispatched and issued out-of-order
In-Order versus Out-of-Order Completion

- All but the simplest machines have out-of-order completion, due to different latencies of functional units and desire to bypass values as soon as available.
- Classic RISC 5-stage integer pipeline just barely has in-order completion
  - Load takes two cycles, but following one-cycle integer op completes at same time, not earlier
  - Adding pipelined FPU immediately brings OoO completion
In-Order versus Out-of-Order Commit

- In-order commit supports precise traps, standard today
  - Some proposals to reduce the cost of in-order commit by retiring some instructions early to compact reorder buffer, but this is just an optimized in-order commit
- Out-of-order commit was effectively what early OoO machines implemented (imprecise traps) as completion irrevocably changed machine state
OoO Design Choices

- Where are reservation stations?
  - Part of reorder buffer, or in separate issue window?
  - Distributed by functional units, or centralized?

- How is register renaming performed?
  - Tags and data held in reservation stations, with separate architectural register file
  - Tags only in reservation stations, data held in unified physical register file
**“Data-in-ROB” Design**
*(HP PA8000, Pentium Pro, Core2Duo, Nehalem)*

- Managed as circular buffer in program order, new instructions dispatched to free slots, oldest instruction committed/reclaimed when done (“p” bit set on result)
- Tag is given by index in ROB (Free pointer value)
- In dispatch, non-busy source operands read from architectural register file and copied to Src1 and Src2 with presence bit “p” set. Busy operands copy tag of producer and clear “p” bit.
- Set valid bit “v” on dispatch, set issued bit “i” on issue
- On completion, search source tags, set “p” bit and copy data into src on tag match. Write result and exception flags to ROB.
- On commit, check exception status, and copy result into architectural register file if no trap.
- On trap, flush machine and ROB, set free=oldest, jump to handler

<table>
<thead>
<tr>
<th>Oldest</th>
<th>v</th>
<th>i</th>
<th>Opcode</th>
<th>p</th>
<th>Tag</th>
<th>Src1</th>
<th>p</th>
<th>Tag</th>
<th>Src2</th>
<th>p</th>
<th>Reg</th>
<th>Result</th>
<th>Except?</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>i</td>
<td></td>
<td>Opcode</td>
<td>p</td>
<td>Tag</td>
<td>Src1</td>
<td>p</td>
<td>Tag</td>
<td>Src2</td>
<td>p</td>
<td>Reg</td>
<td>Result</td>
<td>Except?</td>
</tr>
<tr>
<td>v</td>
<td>i</td>
<td></td>
<td>Opcode</td>
<td>p</td>
<td>Tag</td>
<td>Src1</td>
<td>p</td>
<td>Tag</td>
<td>Src2</td>
<td>p</td>
<td>Reg</td>
<td>Result</td>
<td>Except?</td>
</tr>
<tr>
<td>v</td>
<td>i</td>
<td></td>
<td>Opcode</td>
<td>p</td>
<td>Tag</td>
<td>Src1</td>
<td>p</td>
<td>Tag</td>
<td>Src2</td>
<td>p</td>
<td>Reg</td>
<td>Result</td>
<td>Except?</td>
</tr>
<tr>
<td>v</td>
<td>i</td>
<td></td>
<td>Opcode</td>
<td>p</td>
<td>Tag</td>
<td>Src1</td>
<td>p</td>
<td>Tag</td>
<td>Src2</td>
<td>p</td>
<td>Reg</td>
<td>Result</td>
<td>Except?</td>
</tr>
</tbody>
</table>

Table showing the process of managing instructions in ROB with tags and flags.
Managing Rename for Data-in-ROB

Rename table associated with architectural registers, managed in decode/dispatch

<table>
<thead>
<tr>
<th>p</th>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>Tag</td>
<td>Value</td>
</tr>
<tr>
<td>p</td>
<td>Tag</td>
<td>Value</td>
</tr>
<tr>
<td>p</td>
<td>Tag</td>
<td>Value</td>
</tr>
</tbody>
</table>

- If “p” bit set, then use value in architectural register file
- Else, tag field indicates instruction that will/has produced value
- For dispatch, read source operands <p,tag,value> from arch. regfile, and also read <p,result> from producing instruction in ROB, bypassing as needed. Copy to ROB
- Write destination arch. register entry with <0,Free,_>, to assign tag to ROB index of this instruction
- On commit, update arch. regfile with <1,_,Result>
- On trap, reset table (All p=1)
Data Movement in Data-in-ROB Design

Architectural Register File

Write results at commit

Read operands during decode

Read results for commit

Write sources in dispatch

Bypass newer values at dispatch

ROB

Source Operands

Result Data

Write results at completion

Read operands at issue

Functional Units
Unified Physical Register File

(MIPS R10K, Alpha 21264, Intel Pentium 4 & Sandy/Ivy Bridge)

- Rename all architectural registers into a single *physical* register file during decode, no register values read
- Functional units read and write from single unified register file holding committed and temporary registers in execute
- Commit only updates mapping of architectural register to physical register, no data movement
Lifetime of Physical Registers

- Physical regfile holds committed and speculative values
- Physical registers decoupled from ROB entries (*no data in ROB*)

```
ld x1, (x3)
addi x3, x1, #4
sub x6, x7, x9
add x3, x3, x6
ld x6, (x1)
add x6, x6, x3
sd x6, (x1)
ld x6, (x11)
```

```
ld P1, (Px)
addi P2, P1, #4
sub P3, Py, Pz
add P4, P2, P3
ld P5, (P1)
add P6, P5, P4
sd P6, (P1)
ld P7, (Pw)
```

When can we reuse a physical register?

*When next writer of same architectural register commits*
Physical Register Management

<table>
<thead>
<tr>
<th>Rename Table</th>
<th>Physical Regs</th>
<th>Free List</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>P0</td>
<td>P0</td>
</tr>
<tr>
<td>x1</td>
<td>P8</td>
<td>P1</td>
</tr>
<tr>
<td>x2</td>
<td>P2</td>
<td>P2</td>
</tr>
<tr>
<td>x3</td>
<td>P7</td>
<td>P3</td>
</tr>
<tr>
<td>x4</td>
<td>P4</td>
<td>P4</td>
</tr>
<tr>
<td>x5</td>
<td>P5</td>
<td></td>
</tr>
<tr>
<td>x6</td>
<td>P5</td>
<td></td>
</tr>
<tr>
<td>x7</td>
<td>P6</td>
<td></td>
</tr>
</tbody>
</table>

### ROB

<table>
<thead>
<tr>
<th>use</th>
<th>ex</th>
<th>op</th>
<th>p1</th>
<th>PR1</th>
<th>p2</th>
<th>PR2</th>
<th>Rd</th>
<th>LPRd</th>
<th>PRd</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

example:

- ld x1, 0(x3)
- addi x3, x1, #4
- sub x6, x7, x6
- add x3, x3, x6
- ld x6, 0(x1)

(LPRd requires third read port on Rename Table for each instruction)
Physical Register Management

**Rename Table**

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P8</td>
<td></td>
<td></td>
<td>P7</td>
<td></td>
<td></td>
<td>P6</td>
</tr>
</tbody>
</table>

**Physical Regs**

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x6&gt;</td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x7&gt;</td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x3&gt;</td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x1&gt;</td>
<td>p</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Free List**

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ROB**

<table>
<thead>
<tr>
<th>use</th>
<th>ex</th>
<th>op</th>
<th>p1</th>
<th>PR1</th>
<th>p2</th>
<th>PR2</th>
<th>Rd</th>
<th>LPRd</th>
<th>PRd</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>ld</td>
<td>p</td>
<td>P7</td>
<td></td>
<td>P2</td>
<td></td>
<td>x1</td>
<td>P8</td>
<td>P0</td>
</tr>
</tbody>
</table>

- `ld x1, 0(x3)`
- `addi x3, x1, #4`
- `sub x6, x7, x6`
- `add x3, x3, x6`
- `ld x6, 0(x1)`
Physical Register Management

### Rename Table

- `x0` is renamed to `P0`
- `x1` is renamed to `P1`
- `x3` is renamed to `P7`
- `x6` is renamed to `P5`
- `x7` is renamed to `P6`

### Physical Regs

- `P0` is not used
- `P1` is not used
- `P2` is not used
- `P3` is not used
- `P4` is not used
- `P5` has `<x6>`
- `P6` has `<x7>`
- `P7` has `<x3>`
- `P8` has `<R1>`

### Free List

- `P8` is free
- `P1` is free
- `P3` is free
- `P2` is free
- `P4` is free

### ROB

<table>
<thead>
<tr>
<th>use</th>
<th>ex</th>
<th>op</th>
<th>p1</th>
<th>PR1</th>
<th>p2</th>
<th>PR2</th>
<th>Rd</th>
<th>LPRd</th>
<th>PRd</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td><code>ld</code></td>
<td>p</td>
<td>P7</td>
<td></td>
<td></td>
<td>x1</td>
<td>P8</td>
<td>P0</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td><code>addi</code></td>
<td>P0</td>
<td></td>
<td></td>
<td></td>
<td>x3</td>
<td>P7</td>
<td>P1</td>
</tr>
</tbody>
</table>

- `ld x1, 0(x3)`
- `addi x3, x1, #4`
- `sub x6, x7, x6`
- `add x3, x3, x6`
- `ld x6, 0(x1)`
Physical Register Management

- **Rename Table**
  - Rename Table
  - Rename Table
  - Rename Table
  - Rename Table
  - Rename Table
  - Rename Table
  - Rename Table
  - Rename Table

- **Physical Regs**
  - Physical Regs
  - Physical Regs
  - Physical Regs
  - Physical Regs
  - Physical Regs
  - Physical Regs
  - Physical Regs
  - Physical Regs

- **Free List**
  - Free List
  - Free List
  - Free List
  - Free List
  - Free List
  - Free List
  - Free List
  - Free List

- **ROB**
  - ROB
  - ROB
  - ROB
  - ROB
  - ROB
  - ROB
  - ROB
  - ROB

- **Instructions**
  - ld x1, 0(x3)
  - addi x3, x1, #4
  - sub x6, x7, x6
  - add x3, x3, x6
  - ld x6, 0(x1)
Physical Register Management

### Rename Table

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P0</td>
<td></td>
<td>P1</td>
<td></td>
<td>P3</td>
<td>P5</td>
<td>P6</td>
</tr>
</tbody>
</table>

### Physical Regs

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x6&gt;</td>
<td></td>
<td></td>
<td>&lt;x1&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Free List

- ld x1, 0(x3)
- addi x3, x1, #4
- sub x6, x7, x6
- add x3, x3, x6
- ld x6, 0(x1)

### ROB

<table>
<thead>
<tr>
<th>use</th>
<th>ex</th>
<th>op</th>
<th>p1</th>
<th>PR1</th>
<th>p2</th>
<th>PR2</th>
<th>Rd</th>
<th>LPRd</th>
<th>PRd</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td>ld</td>
<td>p</td>
<td>P7</td>
<td></td>
<td></td>
<td>x1</td>
<td>P8</td>
<td>P0</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>addi</td>
<td>P0</td>
<td></td>
<td></td>
<td></td>
<td>x3</td>
<td>P7</td>
<td>P1</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>sub</td>
<td>P6</td>
<td>p</td>
<td>P5</td>
<td>x6</td>
<td>P5</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>add</td>
<td>P1</td>
<td>P3</td>
<td>x3</td>
<td></td>
<td>P1</td>
<td>P2</td>
<td></td>
</tr>
</tbody>
</table>

---

CS252, Spring 2014, Lecture 6 © Krste Asanovic, 2014
Physical Register Management

### Rename Table

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x1</td>
<td></td>
<td>x3</td>
<td></td>
<td></td>
<td>P6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P3</td>
<td>x4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P4</td>
<td>x5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P3</td>
<td>x6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P4</td>
<td>x7</td>
</tr>
</tbody>
</table>

### Physical Regs

<table>
<thead>
<tr>
<th>PR0</th>
<th>PR1</th>
<th>PR2</th>
<th>Rd</th>
<th>LPRd</th>
<th>PRd</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>P7</td>
<td></td>
<td>x1</td>
<td>P8</td>
<td>P0</td>
</tr>
<tr>
<td>P1</td>
<td>P0</td>
<td></td>
<td>x3</td>
<td>P7</td>
<td>P1</td>
</tr>
<tr>
<td>P2</td>
<td></td>
<td></td>
<td>x6</td>
<td>P5</td>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
<td>x3</td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>x6</td>
<td>P3</td>
<td>P4</td>
</tr>
</tbody>
</table>

### Free List

- ld x1, 0(x3)
- addi x3, x1, #4
- sub x6, x7, x6
- add x3, x3, x6
- ld x6, 0(x1)
Physical Register Management

**Rename Table**

<table>
<thead>
<tr>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x1</td>
<td></td>
<td>x2</td>
<td>x3</td>
<td>x5</td>
<td>x6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>x3</td>
<td></td>
<td></td>
<td>x6</td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td>P0</td>
<td></td>
<td>P2</td>
<td></td>
<td>P4</td>
<td>P4</td>
<td></td>
</tr>
</tbody>
</table>

**Physical Regs**

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P3</th>
<th>P2</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;x1&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x6&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Free List**

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ROB**

<table>
<thead>
<tr>
<th>use</th>
<th>ex</th>
<th>op</th>
<th>p1</th>
<th>PR1</th>
<th>p2</th>
<th>PR2</th>
<th>Rd</th>
<th>LPRd</th>
<th>PRd</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>ld</td>
<td>p</td>
<td>P7</td>
<td></td>
<td></td>
<td>x1</td>
<td>P8</td>
<td>P0</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>addi</td>
<td></td>
<td>P0</td>
<td></td>
<td></td>
<td>x3</td>
<td>P7</td>
<td>P1</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>sub</td>
<td>p</td>
<td>P6</td>
<td>p</td>
<td>P5</td>
<td>x6</td>
<td>P5</td>
<td>P3</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>add</td>
<td></td>
<td>P1</td>
<td></td>
<td>P3</td>
<td>x3</td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>ld</td>
<td>p</td>
<td>P0</td>
<td></td>
<td></td>
<td>x6</td>
<td>P3</td>
<td>P4</td>
</tr>
</tbody>
</table>

**Execute & Commit**

- ld x1, 0(x3)
- addi x3, x1, #4
- sub x6, x7, x6
- add x3, x3, x6
- ld x6, 0(x1)
Physical Register Management

```
ld x1, 0(x3)
addi x3, x1, #4
sub x6, x7, x6
add x3, x3, x6
ld x6, 0(x1)
```

```
<table>
<thead>
<tr>
<th>use</th>
<th>ex</th>
<th>op</th>
<th>p1</th>
<th>PR1</th>
<th>p2</th>
<th>PR2</th>
<th>Rd</th>
<th>LPRd</th>
<th>PRd</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>ld</td>
<td>p</td>
<td>P7</td>
<td></td>
<td></td>
<td>x1</td>
<td>P8</td>
<td>P0</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>addi</td>
<td>p</td>
<td>P0</td>
<td></td>
<td></td>
<td>x3</td>
<td>P7</td>
<td>P1</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>sub</td>
<td>p</td>
<td>P6</td>
<td>p</td>
<td>P5</td>
<td>x6</td>
<td>P5</td>
<td>P3</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>add</td>
<td>p</td>
<td>P1</td>
<td>P3</td>
<td>x3</td>
<td>P1</td>
<td>P2</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>ld</td>
<td>p</td>
<td>P0</td>
<td></td>
<td></td>
<td>x6</td>
<td>P3</td>
<td>P4</td>
</tr>
</tbody>
</table>
```

Rename Table

Physical Regs

Free List
MIPS R10K Trap Handling

- Rename table is repaired by unrenaming instructions in reverse order using the PRd/LPRd fields
- The Alpha 21264 had similar physical register file scheme, but kept complete rename table snapshots for each instruction in ROB (80 snapshots total)
  - Flash copy all bits from snapshot to active table in one cycle
Reorder Buffer Holds Active Instructions (Decoded but not Committed)

```
ld x1, (x3)
add x3, x1, x2
sub x6, x7, x9
add x3, x3, x6
ld x6, (x1)
add x6, x6, x3
sd x6, (x1)
ld x6, (x1)
```

... (Older instructions)

Cycle \( t \)

```
add x3, x1, x2
sub x6, x7, x9
add x3, x3, x6
ld x6, (x1)
add x6, x6, x3
sd x6, (x1)
ld x6, (x1)
```

... (Newer instructions)

Cycle \( t + 1 \)
Separate Issue Window from ROB

The issue window holds only instructions that have been decoded and renamed but not issued into execution. Has register tags and presence bits, and pointer to ROB entry.

Reorder buffer used to hold exception information for commit.

ROB is usually several times larger than issue window – why?
Superscalar Register Renaming

- During decode, instructions allocated new physical destination register
- Source operands renamed to physical register with newest value
- Execution unit only sees physical register numbers

Does this work?
Superscalar Register Renaming

Must check for RAW hazards between instructions issuing in same cycle. Can be done in parallel with rename lookup.

MIPS R10K renames 4 serially-RAW-dependent insts/cycle
Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)