CS 152 Computer Architecture and Engineering

Lecture 12 - Advanced Out-of-Order Superscalars

Krste Asanovic
Electrical Engineering and Computer Sciences
University of California at Berkeley

http://www.eecs.berkeley.edu/~krste
http://inst.eecs.berkeley.edu/~cs152
Last time in Lecture 11

- Register renaming removes WAR, WAW hazards
- In-order fetch/decode, out-of-order execute, in-order commit gives high performance and precise exceptions
- Dynamic branch predictors can be quite accurate (>95%) and avoid most control hazards
- Branch History Tables (BHTs) just predict direction (later in pipeline)
  - Just need a few bits per entry (2 bits gives hysteresis)
  - Need to decode instruction bits to determine whether this is a branch and what the target address is
- Branch Target Buffers (BTBs) predict direction and target earlier in pipeline, but bigger entries
- Return Address Stack predicts subroutine returns
Branch Mispredict Recovery

In-order execution machines:
  – Assume no instruction issued after branch can write-back before branch resolves
  – Kill all instructions in pipeline behind mispredicted branch

Out-of-order execution?
  – Multiple instructions following branch in program order can complete before branch resolves
In-Order Commit for Precise Exceptions

- Instructions fetched and decoded into instruction reorder buffer in-order
- Execution is out-of-order (⇒ out-of-order completion)
- *Commit* (write-back to architectural state, i.e., regfile & memory, is in-order)

Temporary storage needed in ROB to hold results before commit
Branch Misprediction in Pipeline

- Can have multiple unresolved branches in ROB
- Can resolve branches out-of-order by killing all the instructions in ROB that follow a mispredicted branch
- Must also kill instructions in-flight in execution pipelines

Inject correct PC

Kill

Complete
Take snapshot of register rename table at each predicted branch, recover earlier snapshot if branch mispredicted.
“Data-in-ROB” Design  
(HP PA8000, Intel Pentium Pro, Core2 Duo & Nehalem)

- On dispatch into ROB, ready sources can be in regfile or in ROB dest (copied into src1/src2 if ready before dispatch)
- On completion, write to dest field and broadcast to src fields.
- On issue, read from ROB src fields
Data Movement in Data-in-ROB Design

- Read operands during decode
- Write sources after decode
- Read operands at issue
- Write results at commit
- Read results at commit
- Write results at completion

Diagram:

- Architectural Register File
- Reorder Buffer
- Functional Units
Unified Physical Register File  
(*MIPS R10K, Alpha 21264, Intel Pentium 4 & Sandy 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.

```
+-----------------------------------------------+      +-----------------------------------------------+      +-----------------------------------------------+
| Decode Stage                                  |      | Unified Physical Register File                |      | Committed Register Mapping                    |
| Register Mapping                              |      |                                              |      |                                              |
| Read operands at issue                        |      | Functional Units                             |      | Write results at completion                   |
```
Pipeline Design with Physical Regfile

- **Fetch**
- **Decode & Rename**
- **Reorder Buffer**
- **Commit**
- **Branch Prediction**
- **Physical Reg. File**
- **In-Order**
- **Out-of-Order**

- **Branch Resolution**
- **ALU**
- **MEM**
- **Store Buffer**
- **D$**

- **PC**
- **Branch Unit**

- **Execute**

- **Kill**
Lifetime of Physical Registers

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

```
ld r1, (r3)
add r3, r1, #4
sub r6, r7, r9
add r3, r3, r6
ld r6, (r1)
add r6, r6, r3
st r6, (r1)
ld r6, (r11)
```

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

When can we reuse a physical register?

*When next write of same architectural register commits*
## Physical Register Management

### Rename Table

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P8</td>
<td></td>
<td>P7</td>
<td></td>
<td></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;R6&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;R7&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;R3&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;R1&gt;</td>
<td></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></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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ld r1, 0(r3)
- add r3, r1, #4
- sub r6, r7, r6
- add r3, r3, r6
- ld r6, 0(r1)

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

**Rename Table**

- R0
- R1
- R2
- R3
- R4
- R5
- R6
- R7

**Physical Regs**

- P0
- P1
- P2
- P3
- P4
- P5
- P6
- P7
- P8

**Free List**

- P0
- P1
- P2
- P3
- P4
- P5
- P6
- P7
- P8

**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>r1</td>
<td>P8</td>
<td>P0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- ld r1, 0(r3)
- add r3, r1, #4
- sub r6, r7, r6
- add r3, r3, r6
- ld r6, 0(r1)
Physical Register Management

**Rename Table**

- R0
- R1
- R2
- R3
- R4
- R5
- R6
- R7

**Physical Regs**

- P0
- P1
- P2
- P3
- P4
- P5
- P6
- P7
- P8

**Free List**

- P0
- P1
- P2
- P3
- P4

**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></td>
<td></td>
<td>P7</td>
<td></td>
<td></td>
<td>r1</td>
<td>P8</td>
<td>P0</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>P0</td>
<td></td>
<td></td>
<td>r3</td>
<td>P7</td>
<td>P1</td>
</tr>
</tbody>
</table>

- ld r1, 0(r3)
- add r3, r1, #4
- sub r6, r7, r6
- add r3, r3, r6
- ld r6, 0(r1)
Physical Register Management

```
 ld r1, 0(r3)
 add r3, r1, #4
 sub r6, r7, r6
 add r3, r3, r6
 ld r6, 0(r1)
```
Physical Register Management

Rename Table

Physical Regs

Free List

ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

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>r1</td>
<td>P8</td>
<td>P0</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>add</td>
<td></td>
<td>P0</td>
<td>p</td>
<td></td>
<td>r3</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>r6</td>
<td>P5</td>
<td>P3</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>add</td>
<td></td>
<td>P1</td>
<td>p</td>
<td></td>
<td>r3</td>
<td>P1</td>
<td>P2</td>
</tr>
</tbody>
</table>

March 9, 2011 CS152, Spring 2011
Physical Register Management

**Rename Table**

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P0</td>
<td></td>
<td>P1</td>
<td></td>
<td></td>
<td>P3</td>
<td></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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;R6&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;R7&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;R3&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;R1&gt;</td>
</tr>
</tbody>
</table>

**Free List**

- P6
- P4
- P3
- P2
- P1
- P0

**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>r1</td>
<td>P8</td>
<td>P0</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>add</td>
<td></td>
<td>P0</td>
<td></td>
<td></td>
<td>r3</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>r6</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>r3</td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>ld</td>
<td></td>
<td>P0</td>
<td></td>
<td></td>
<td>r6</td>
<td>P3</td>
<td>P4</td>
</tr>
</tbody>
</table>

- ld r1, 0(r3)
- add r3, r1, #4
- sub r6, r7, r6
- add r3, r3, r6
- ld r6, 0(r1)
Physical Register Management

**Rename Table**

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>P5</td>
<td>P4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>&lt;R1&gt;</td>
<td>p</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
<th>P7</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<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>x</td>
<td>ld</td>
<td>p</td>
<td>P7</td>
<td>p</td>
<td></td>
<td>r1</td>
<td>P8</td>
<td>P0</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>add</td>
<td>P0</td>
<td></td>
<td></td>
<td></td>
<td>r3</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>r6</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>r3</td>
<td>P1</td>
<td>P2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>ld</td>
<td>P0</td>
<td>r6</td>
<td>P3</td>
<td>P4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `ld r1, 0(r3)`
- `add r3, r1, #4`
- `sub r6, r7, r6`
- `add r3, r3, r6`
- `ld r6, 0(r1)`

Execute & Commit
Physical Register Management

Rename Table

Physical Regs

Free List

ROB

ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)
CS152 Administrivia
Separate Pending Instruction Window from ROB

The instruction window holds 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 instruction window – why?
Reorder Buffer Holds Active Instructions
(Decoded but not Committed)

... (Older instructions)

ld r1, (r3)
add r3, r1, r2
sub r6, r7, r9
add r3, r3, r6
ld r6, (r1)
add r6, r6, r3
st r6, (r1)
ld r6, (r1)
... (Newer instructions)

Cycle $t$

Commit

ld r1, (r3)
add r3, r1, r2
sub r6, r7, r9
add r3, r3, r6
ld r6, (r1)
add r6, r6, r3
st r6, (r1)
ld r6, (r1)
... (Newer instructions)

Cycle $t + 1$

Execute

Fetch
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

Update Mapping

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
Memory Dependencies

\[ \text{st r1, (r2)} \]
\[ \text{ld r3, (r4)} \]

When can we execute the load?
In-Order Memory Queue

• Execute all loads and stores in program order

=> Load and store cannot leave ROB for execution until all previous loads and stores have completed execution

• Can still execute loads and stores speculatively, and out-of-order with respect to other instructions

• Need a structure to handle memory ordering…
Conservative O-o-O Load Execution

\[ \text{st r1, (r2)} \]
\[ \text{ld r3, (r4)} \]

- Split execution of store instruction into two phases: address calculation and data write
- Can execute load before store, if addresses known and r4 \(!=\) r2
- Each load address compared with addresses of all previous uncommitted stores \((\text{can use partial conservative check i.e., bottom 12 bits of address})\)
- Don’t execute load if any previous store address not known

\((\text{MIPS R10K, 16 entry address queue})\)
Address Speculation

\[ \text{st } r1, (r2) \]
\[ \text{ld } r3, (r4) \]

• Guess that \( r4 \neq r2 \)

• Execute load before store address known

• Need to hold all completed but uncommitted load/store addresses in program order

• If subsequently find \( r4 == r2 \), squash load and all following instructions

=> Large penalty for inaccurate address speculation
Memory Dependence Prediction
(Alpha 21264)

\begin{verbatim}
st r1, (r2)
ld r3, (r4)
\end{verbatim}

- Guess that $r4 \neq r2$ and execute load before store
- If later find $r4 = r2$, squash load and all following instructions, but mark load instruction as \textit{store-wait}
- Subsequent executions of the same load instruction will wait for all previous stores to complete
- Periodically clear \textit{store-wait} bits
Speculative Loads / Stores

Just like register updates, stores should not modify the memory until after the instruction is committed.

- A speculative store buffer is a structure introduced to hold speculative store data.
Speculative Store Buffer

- On store execute:
  - mark entry valid and speculative, and save data and tag of instruction.
- On store commit:
  - clear speculative bit and eventually move data to cache
- On store abort:
  - clear valid bit
Speculative Store Buffer

- If data in both store buffer and cache, which should we use?
  Speculative store buffer
- If same address in store buffer twice, which should we use?
  Youngest store older than load
Datapath: Branch Prediction and Speculative Execution

Branch Prediction

Branch Resolution

PC → Fetch → Decode & Rename → Reorder Buffer → Commit

Branch Unit → ALU → MEM → Store Buffer → D$

Execute

Reg. File

March 9, 2011
Acknowledgements

• These slides contain material developed and copyright by:
  – Arvind (MIT)
  – Krste Asanovic (MIT/UCB)
  – Joel Emer (Intel/MIT)
  – James Hoe (CMU)
  – John Kubiatowicz (UCB)
  – David Patterson (UCB)

• MIT material derived from course 6.823
• UCB material derived from course CS252