CS 152 Computer Architecture and Engineering

Lecture 11 - Out-of-Order Issue, Register Renaming, & Branch Prediction

John Wawrzynek
Electrical Engineering and Computer Sciences
University of California at Berkeley

http://www.eecs.berkeley.edu/~johnw
http://inst.eecs.berkeley.edu/~cs152
CS152 Administrivia

- Quiz 2 Results posted

- PS3 posted
Last time in Lecture 12

- Pipelining is complicated by multiple and/or variable latency functional units
- Out-of-order and/or pipelined execution requires tracking of dependencies
  - RAW
  - WAR
  - WAW
- Dynamic issue logic can support out-of-order execution to improve performance
  - Last time, looked at simple scoreboard to track out-of-order completion
- Hardware register renaming can further improve performance by removing hazards.
Register Renaming

- Decode does register renaming and adds instructions to the issue-stage instruction reorder buffer (ROB)
  \[\Rightarrow\text{renaming makes WAR or WAW hazards impossible}\]

- Any instruction in ROB whose RAW hazards have been satisfied can be issued.
  \[\Rightarrow\text{Out-of-order or dataflow execution}\]
Renaming Structures

Renaming table & regfile

Reorder buffer

Reordering buffer

Replacing the tag by its value is an expensive operation

- Instruction template (i.e., tag $t$) is allocated by the Decode stage, which also associates tag with register in regfile
- When an instruction completes, its tag is deallocated
Reorder Buffer Management

Instruction slot is candidate for execution when:

- It holds a valid instruction ("use" bit is set)
- It has not already started execution ("exec" bit is clear)
- Both operands are available (p1 and p2 are set)

ROB managed circularly

- "exec" bit is set when instruction begins execution
- When an instruction completes its "use" bit is marked free
- ptr2 is incremented only if the "use" bit is marked free

Instruction slot is marked free when:

- Destination registers are renamed to the instruction’s slot tag next to deallocate

next available

ptr₁

next to deallocate

ptr₂

next to deallocate

\[ t_1 \]

\[ t_2 \]

\[ t_n \]
**IBM 360/91 Floating-Point Unit**

*R. M. Tomasulo, 1967*

- **Load Buffers (from memory)**
- **Instructions**
- **Regfile**
- **Floating-Point Unit**

**Distribute Instruction Templates by Functional Units**

**Store Buffers (to memory)**

**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”**
Effectiveness?

Renaming and Out-of-order execution was first implemented in 1969 in IBM 360/91 but did not show up in the subsequent models until mid-Nineties.

Why?

Reasons

1. Complex logic / cost
2. Memory latency a much bigger problem
3. Exceptions not precise!

One more problem needed to be solved

Control transfers
Precise Interrupts

It must appear as if an interrupt is taken between two instructions (say $I_i$ and $I_{i+1}$)

- the effect of all instructions up to and including $I_i$ is totally complete
- no effect of any instruction after $I_i$ has taken place

The interrupt handler either aborts the program or restarts it at $I_{i+1}$. 
Effect on Interrupts

Out-of-order Completion

\[
\begin{align*}
I_1 &: \text{DIVD} & f_6, & f_6, & f_4 \\
I_2 &: \text{LD} & f_2, & 45(r3) \\
I_3 &: \text{MULTD} & f_0, & f_2, & f_4 \\
I_4 &: \text{DIVD} & f_8, & f_6, & f_2 \\
I_5 &: \text{SUBD} & f_{10}, & f_0, & f_6 \\
I_6 &: \text{ADDD} & f_6, & f_8, & f_2
\end{align*}
\]

\text{out-of-order comp} \quad 1 \quad 2 \quad 2 \quad 3 \quad 1 \quad 4 \quad 3 \quad 5 \quad 5 \quad 4 \quad 6 \quad 6

Consider interrupts

Precise interrupts are difficult to implement at high speed
- want to start execution of later instructions before exception checks finished on earlier instructions
Exception Handling
(In-Order Five-Stage Pipeline)

- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions
- Inject external interrupts at commit point (override others)
- If exception at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage
Phases of Instruction Execution

- **Fetch**: Instruction bits retrieved from cache.
- **Decode**: Instructions dispatched to appropriate issue-stage buffer.
- **Execute**: Instructions and operands issued to execution units. When execution completes, all results and exception flags are available.
- **Commit**: Instruction irrevocably updates architectural state (aka “graduation”).

Diagram:

1. PC
2. I-cache
3. Fetch Buffer
4. Decode/Rename
5. Issue Buffer
6. Functional Units
7. Result Buffer
8. Commit
9. Architectural State

10/11/2016
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 to hold results before commit (shadow registers and store buffers)
## Extensions for Precise Exceptions

- Add `<pd, dest, data, cause>` fields in the instruction template.
- Commit instructions to reg file and memory in program order ⇒ buffers can be maintained circularly.
- On exception, clear reorder buffer by resetting $ptr_1 = ptr_2$.

*Stores must wait for commit before updating memory.*

### Reorder buffer

<table>
<thead>
<tr>
<th>Inst#</th>
<th>use</th>
<th>exec</th>
<th>op</th>
<th>p1</th>
<th>src1</th>
<th>p2</th>
<th>src2</th>
<th>pd</th>
<th>dest</th>
<th>data</th>
<th>cause</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>
<td></td>
<td></td>
</tr>
<tr>
<td>ptr₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>next to commit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ptr₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>next available</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10/11/2016  CS152, Fall 2016
Register file does not contain renaming tags any more.

How does the decode stage find the tag of a source register?

Search the “dest” field in the reorder buffer.
Renaming table is a cache to speed up register name look up. It needs to be cleared after each exception taken.

When else are valid bits cleared? **Control transfers**
Modern processors may have > 10 pipeline stages between next PC calculation and branch resolution!

How much work is lost if pipeline doesn’t follow correct instruction flow?

~ Loop length x pipeline width
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
Recovering ROB/Renaming Table

Take snapshot of register rename table at each predicted branch, recover earlier snapshot if branch mispredicted.
Important Points on OOO Execution

- Precise interrupts are provided by committing (graduating) instructions in-order.

- Any instruction not yet committed is considered “speculative” because it might be killed because of:
  - Interrupt on an older instruction
  - Branch misprediction

- Instructions (and their result values) are held in ROB waiting to commit
“Data-in-ROB” Design
(HP PA8000, Pentium Pro, Core2Duo, Nehalem)

- On dispatch into ROB, ready sources can be in register file 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
- Instructions are committed to the register file in order
- Instruction slots (i.e. tags) are recycled after commit
Data Movement in Data-in-ROB Design

Architectural Register File

Src Operands

Result Data

ROB

Functional Units

Read operands during decode

Write sources after decode

Read operands at issue

Write results at completion

Read results at commit

Bypass newer values at decode
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 the mapping of architectural register to physical register, no data movement
Pipeline Design with Physical Regfile

Branch Prediction

Branch Resolution

Fetch

Decode & Rename

Reorder Buffer

PC

Branch

Resolution

In-Order

Out-of-Order

Commit

Physical Reg. File

Branch Unit

ALU

MEM

Store Buffer

D$

Execute
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 write of same architectural register commits*
### 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>P7</td>
<td></td>
<td></td>
<td>P6</td>
<td></td>
</tr>
</tbody>
</table>

#### Physical Regs

<table>
<thead>
<tr>
<th>PR</th>
<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>
<th>Pn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x6&gt;</td>
<td>&lt;x7&gt;</td>
<td>&lt;x3&gt;</td>
<td>&lt;x1&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>p</td>
<td>p</td>
<td>p</td>
<td>p</td>
<td></td>
</tr>
</tbody>
</table>

#### Free List

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

- \text{ld x1, 0(x3)}
- \text{addi x3, x1, #4}
- \text{sub x6, x7, x6}
- \text{add x3, x3, x6}
- \text{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>P0</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;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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>
</tbody>
</table>

---

10/11/2016

CS152, Fall 2016
Physical Register Management

**Rename Table**

<table>
<thead>
<tr>
<th>x0</th>
<th>P0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>P1</td>
</tr>
<tr>
<td>x2</td>
<td>P2</td>
</tr>
<tr>
<td>x3</td>
<td>P3</td>
</tr>
<tr>
<td>x4</td>
<td>P4</td>
</tr>
<tr>
<td>x5</td>
<td>P5</td>
</tr>
<tr>
<td>x6</td>
<td>P6</td>
</tr>
<tr>
<td>x7</td>
<td>P7</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></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x7&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x3&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>
<tr>
<td>Pn</td>
<td></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>
</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></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>x3</td>
<td>P7</td>
<td>P1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>P1</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>
<th>Pn</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></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>
<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>
<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>
<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></td>
<td>x6</td>
<td>P5</td>
<td>P3</td>
</tr>
</tbody>
</table>
Physical Register Management

**Rename Table**

- x0: P0
- x1: P1
- x2: P2
- x3: P3
- x4: P4
- x5: P5
- x6: P6
- x7: P7

**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>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>P1</td>
<td>P3</td>
<td>x3</td>
<td>P1</td>
<td></td>
<td></td>
<td></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

<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>P8</td>
<td>P0</td>
<td></td>
<td>P1</td>
<td></td>
<td>P3</td>
<td>P4</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x6&gt;</td>
<td></td>
<td>&lt;x7&gt;</td>
<td>p</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x6&gt;</td>
<td></td>
<td>&lt;x3&gt;</td>
<td>p</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x1&gt;</td>
<td></td>
<td></td>
<td>p</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>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>P1</td>
<td>P3</td>
<td>x3</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></td>
<td></td>
<td>x6</td>
<td>P3</td>
<td>P4</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>

10/11/2016
**Physical Register Management**

**Rename Table**

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

<table>
<thead>
<tr>
<th></th>
<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>&lt;x1&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;x6&gt;</td>
<td>&lt;x7&gt;</td>
<td>&lt;x3&gt;</td>
<td>&lt;x1&gt;</td>
</tr>
<tr>
<td></td>
<td>Pn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical Regs**

**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>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>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></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></td>
<td>P0</td>
<td></td>
<td></td>
<td>x6</td>
<td>P3</td>
<td>P4</td>
</tr>
</tbody>
</table>

**Execute & Commit**
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></td>
<td></td>
<td>P2</td>
<td></td>
<td>P4</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>&lt;x1&gt;</td>
<td>&lt;x3&gt;</td>
<td></td>
<td></td>
<td></td>
<td>&lt;x6&gt;</td>
<td>&lt;x7&gt;</td>
<td>&lt;x3&gt;</td>
<td></td>
</tr>
</tbody>
</table>

**Free List**

| P6 | P3 | P2 | P1 | 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>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></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>

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

**Execute & Commit**
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