# CS 152 Computer Architecture and Engineering
# CS252 Graduate Computer Architecture

## Lecture 11 – Out-of-Order Execution

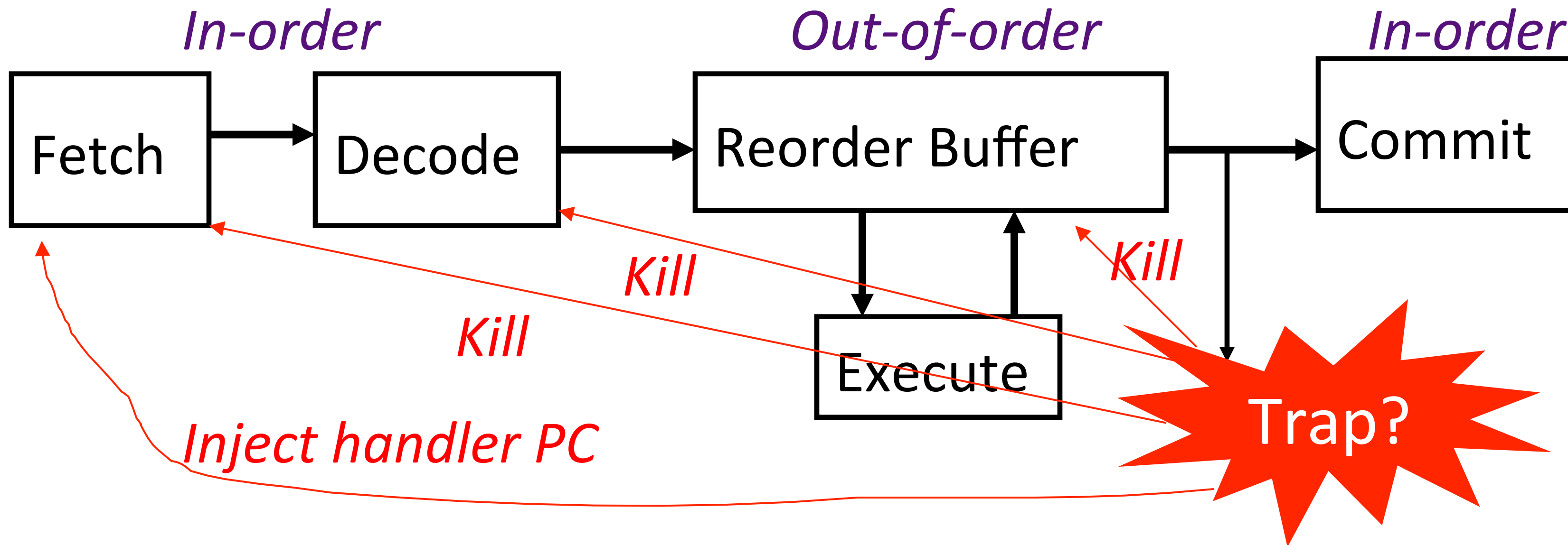
Krste Asanovic
Electrical Engineering and Computer Sciences
University of California at Berkeley

**http://www.eecs.berkeley.edu/~krste**
**http://inst.eecs.berkeley.edu/~cs152**

# Last Time in Lecture 10

- 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)
- OoO issue limited by WAR and WAW hazards caused by reuse of architectural register names, removed by register renaming
- OoO issue and register renaming invented in mid-1960s but disappeared in practice until 1990s, as simpler architecture approaches (pipelining, caches) could more easily take advantage of technology scaling
- Also, two important problems had to be solved:
  - Control hazards
  - Precise traps and interrupts

# In-Order Commit for Precise Traps

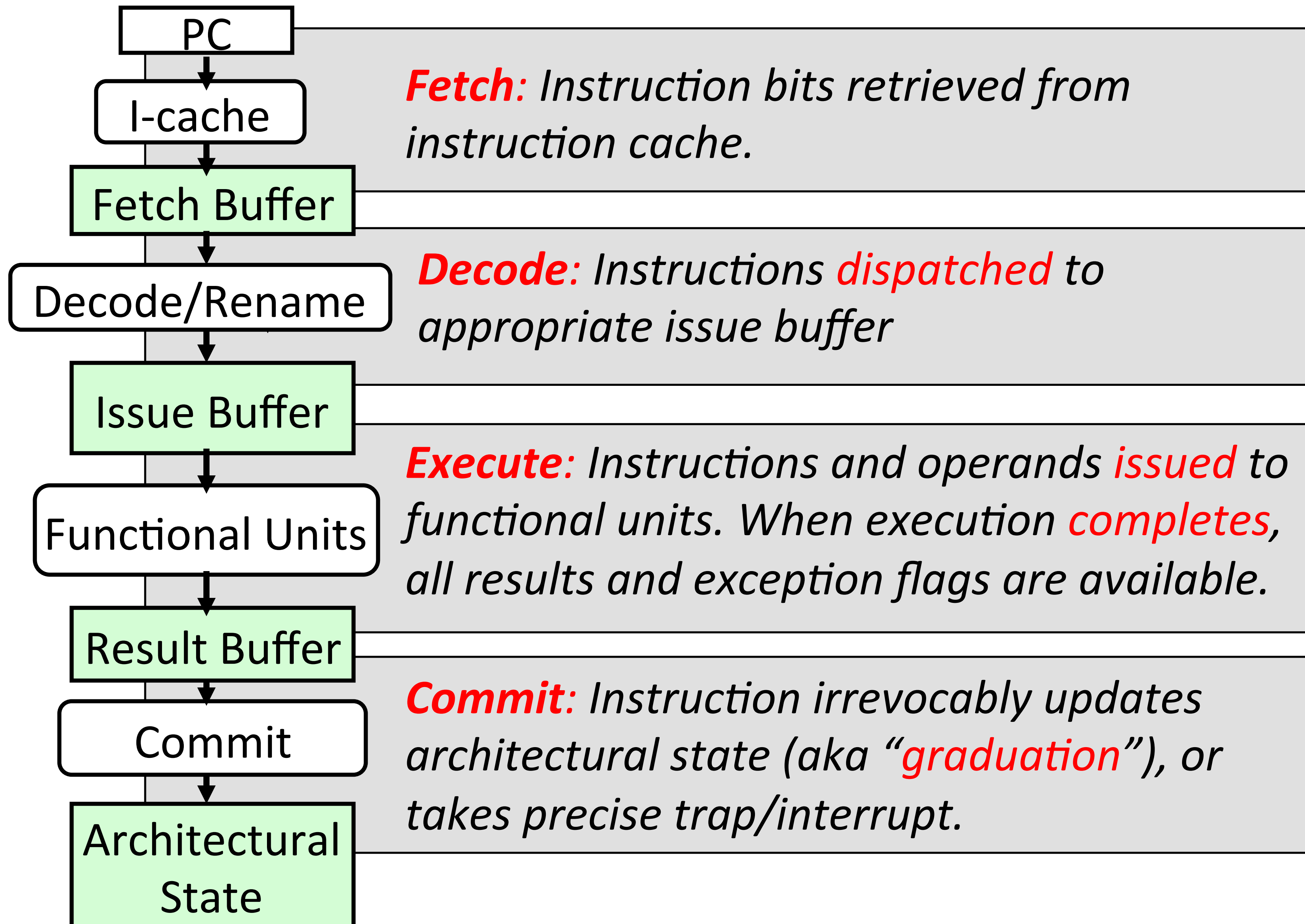



- In-order instruction fetch and decode, and dispatch to reservation stations inside reorder buffer
- Instructions issue from reservation stations out-of-order
- Out-of-order completion, values stored in temporary buffers
- Commit is in-order, checks for traps, and if none updates architectural state

# 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

# Phases of Instruction Execution



# In-Order versus Out-of-Order Phases

- Instruction fetch/decode/rename always in-order
  - Need to parse ISA sequentially to get correct semantics
  - *CS252: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
  - Some use "Dispatch" to mean "Issue", but not in these lectures

# In-Order Versus Out-of-Order Issue

- In-order (InO) 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 (OoO) 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 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

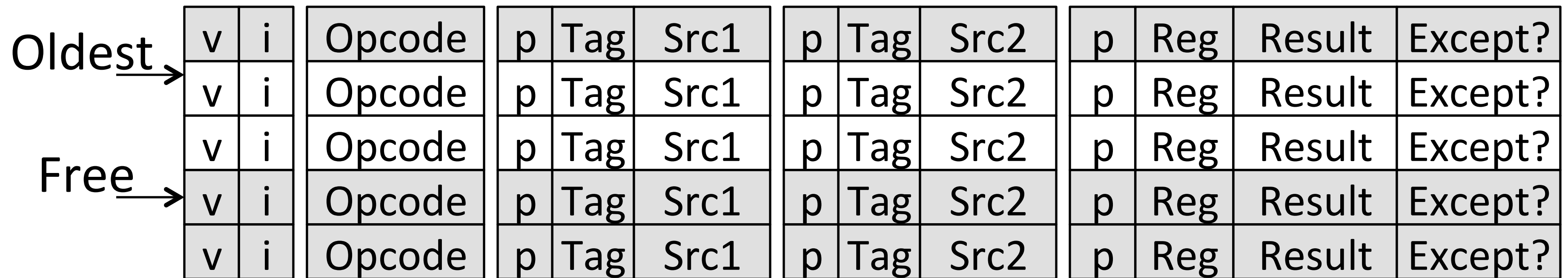
- In-order commit supports precise traps, standard today
  - *CS252: Some proposals to reduce the cost of in-order commit by retiring some instructions early to compact reorder buffer, but this is just an optimized in-order commit*
- Out-of-order commit was effectively what early OoO machines implemented (imprecise traps) as completion irrevocably changed machine state
  - i.e., complete == commit in these machines

# 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

# Managing Rename for Data-in-ROB

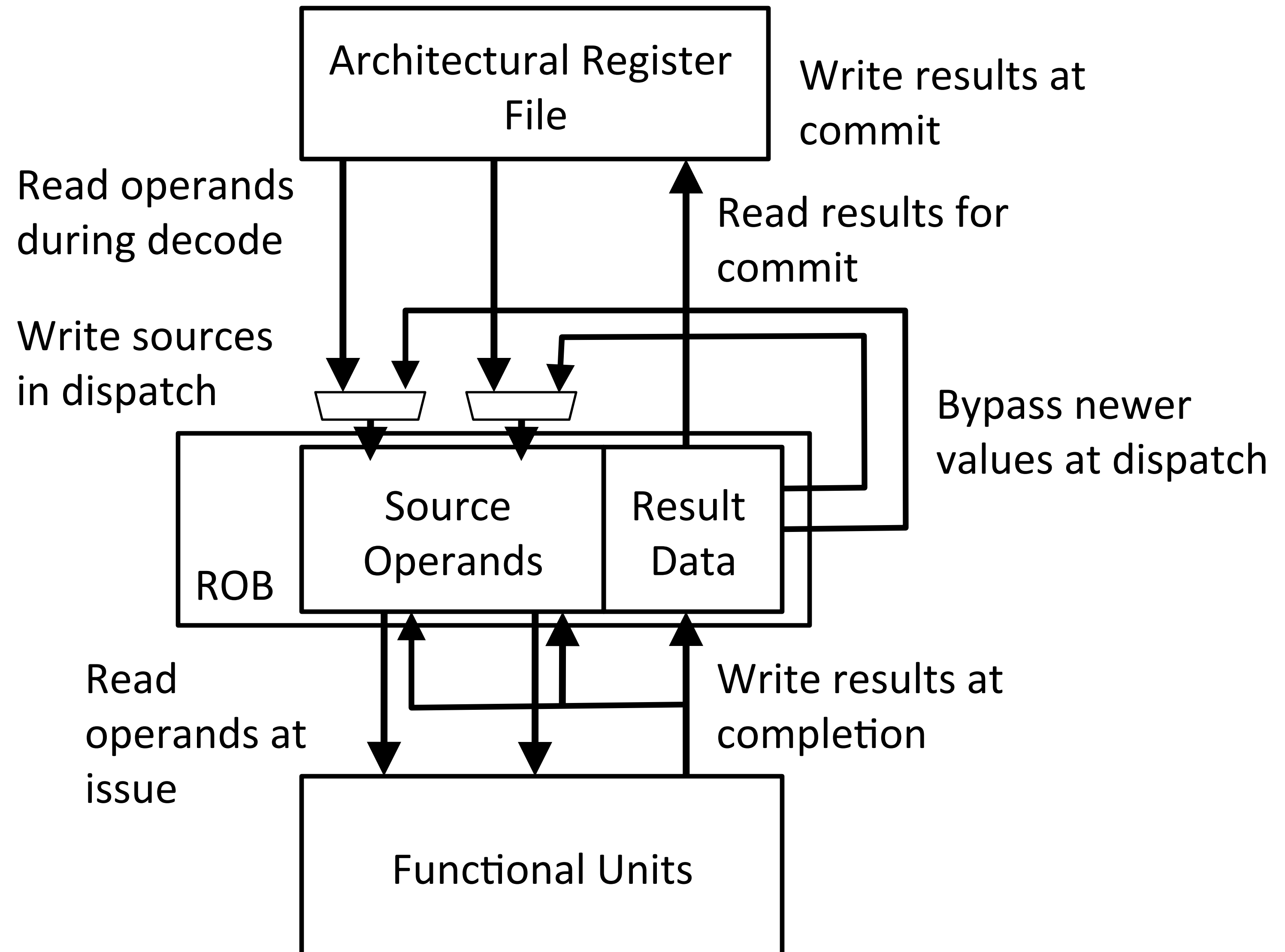
Rename table associated with architectural registers, managed in decode/dispatch

| p | Tag | Value |
|---|---|---|
| p | Tag | Value |
| p | Tag | Value |
| p | Tag | Value |
| p | Tag | Value |

One entry per arch. register

- 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, then also read <p,result> from producing instruction in ROB at tag index, bypassing as needed. Copy operands 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)

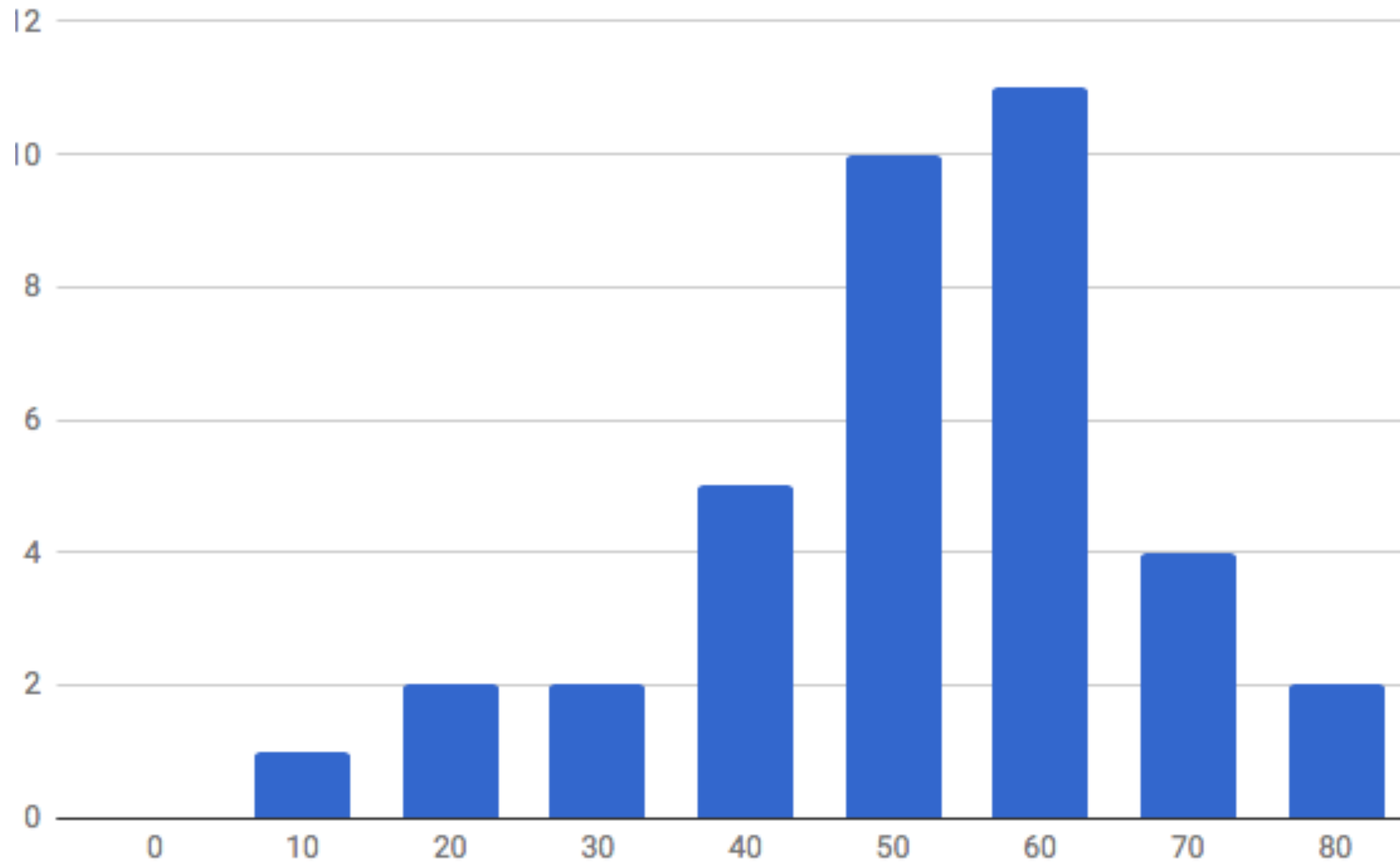
# Data Movement in Data-in-ROB Design



# CS152 Administrivia

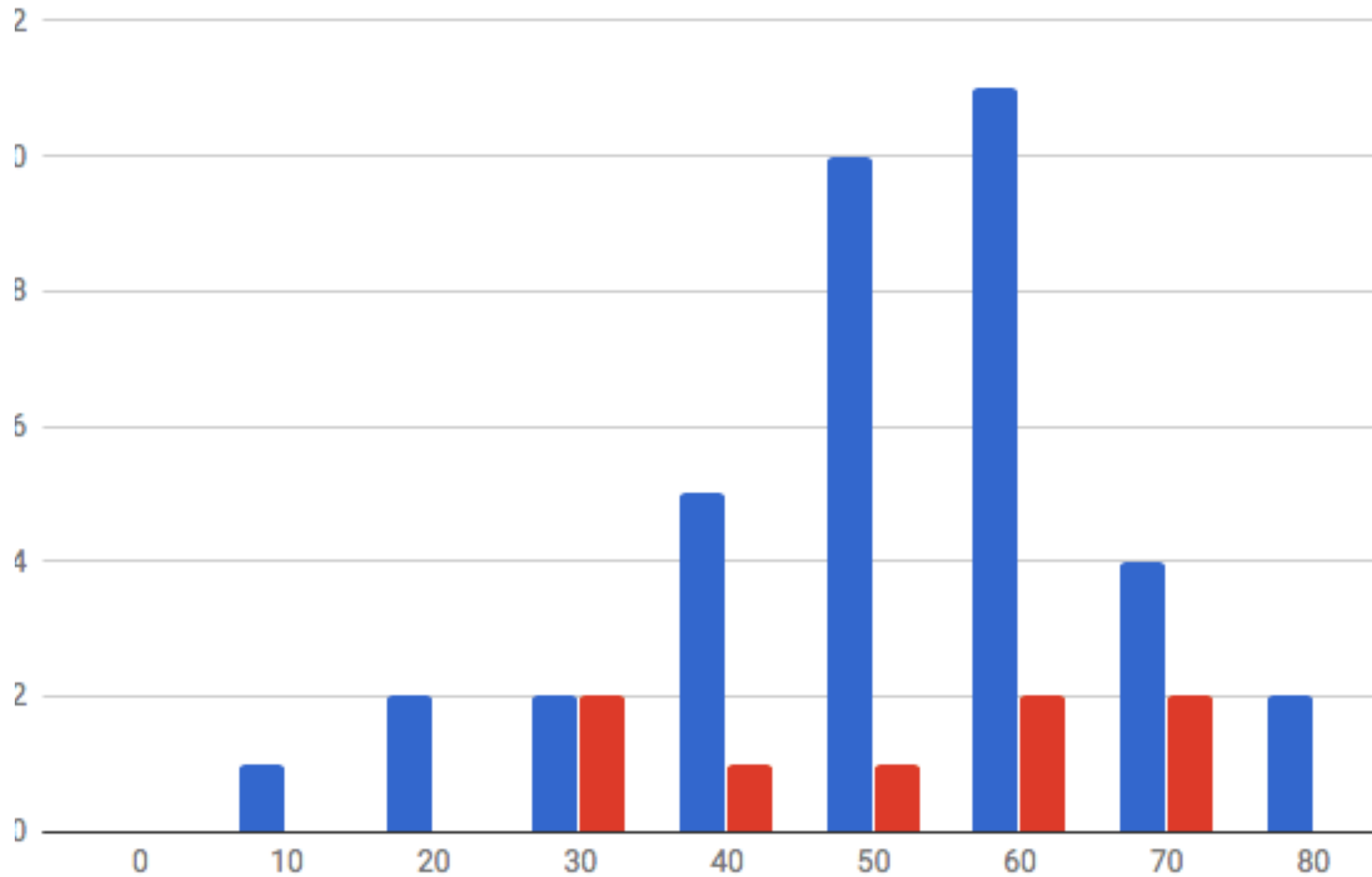
- PS 3 out today, due Monday March 12
- Lab 2 due Monday
  - Lab 2 takes time to run, so please get started ASAP
  - Don't wait till Sunday!
- Lab 3 out Friday, due Monday March 19
- Exams handed back on Friday in Section
  - One week to submit regrade requests (score might go up or down with regrade requests)

# CS152 Midterm 1 Histogram




CS152 Average 56.3

# CS252/CS152 Midterm 1 Histogram



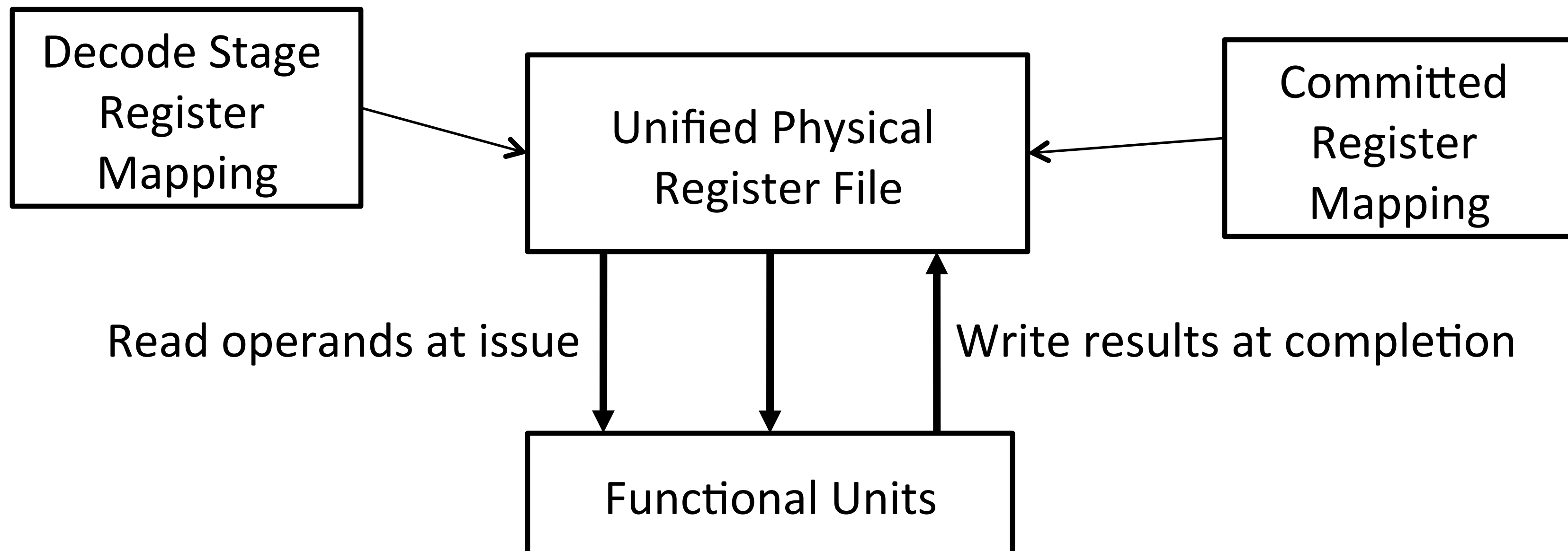
# CS252 Administrivia

- Project proposal due Monday March 5th
  - Mail PDF of proposal to instructors
  - Give a <5-minute presentation in class in discussion section time on March 5th

# Unified Physical Register File

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

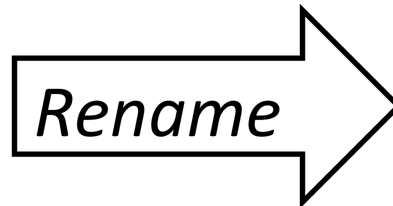
- Rename all architectural registers into a single *physical* register file during decode, no register values read
- Functional units read and write from single unified register file holding committed and temporary registers in execute
- Commit only updates mapping of architectural register to physical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

*Rename Table*

| | |
|---|---|
| x0 | |
| x1 | P8 |
| x2 | |
| x3 | P7 |
| x4 | |
| x5 | |
| x6 | P5 |
| x7 | P6 |

*Physical Regs*

| | | |
|---|---|---|
| P0 | | |
| P1 | | |
| P2 | | |
| P3 | | |
| P4 | | |
| P5 | <x6> | p |
| P6 | <x7> | p |
| P7 | <x3> | p |
| P8 | <x1> | p |
| ... | | |
| Pn | | |

*Free List*

| |
|---|
| P0 |
| P1 |
| P3 |
| P2 |
| P4 |
| |
| |
| |
| ... |
| |

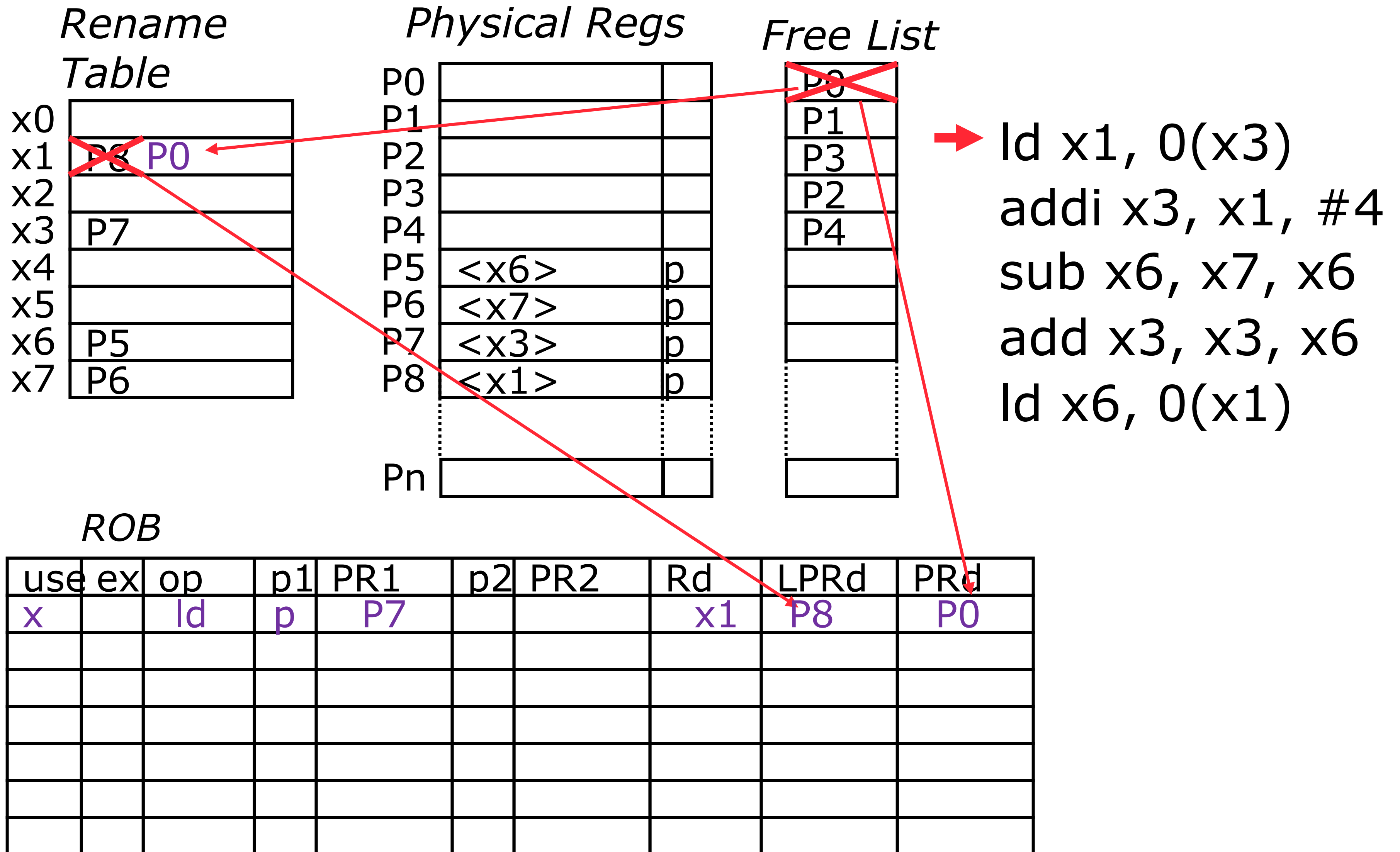
```
ld x1, 0(x3)
addi x3, x1, #4
sub x6, x7, x6
add x3, x3, x6
ld x6, 0(x1)
```

*ROB*

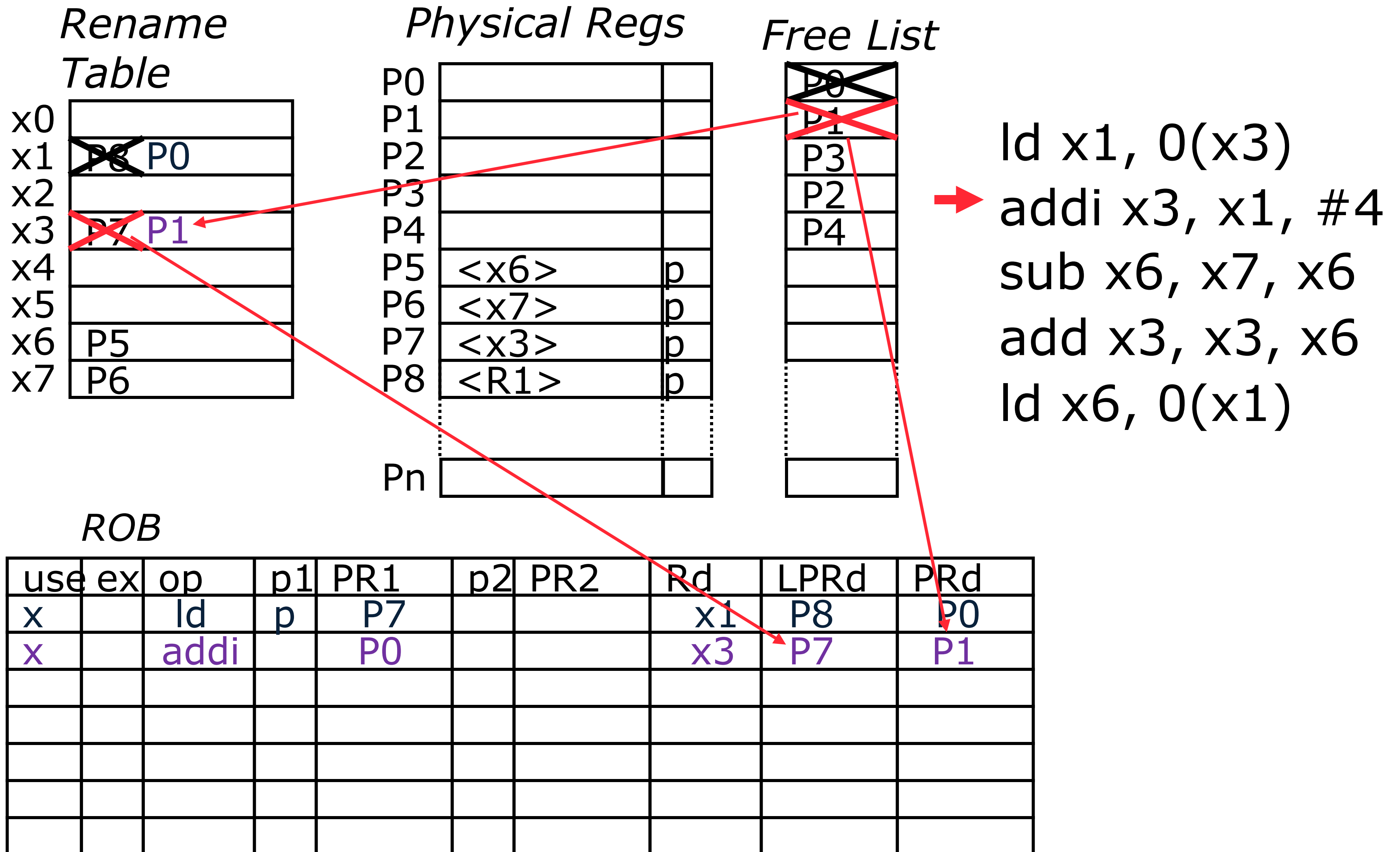
| use | ex | op | p1 | PR1 | p2 | PR2 | Rd | LPRd | PRd |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

*(LPRd requires third read port on Rename Table for each instruction)*

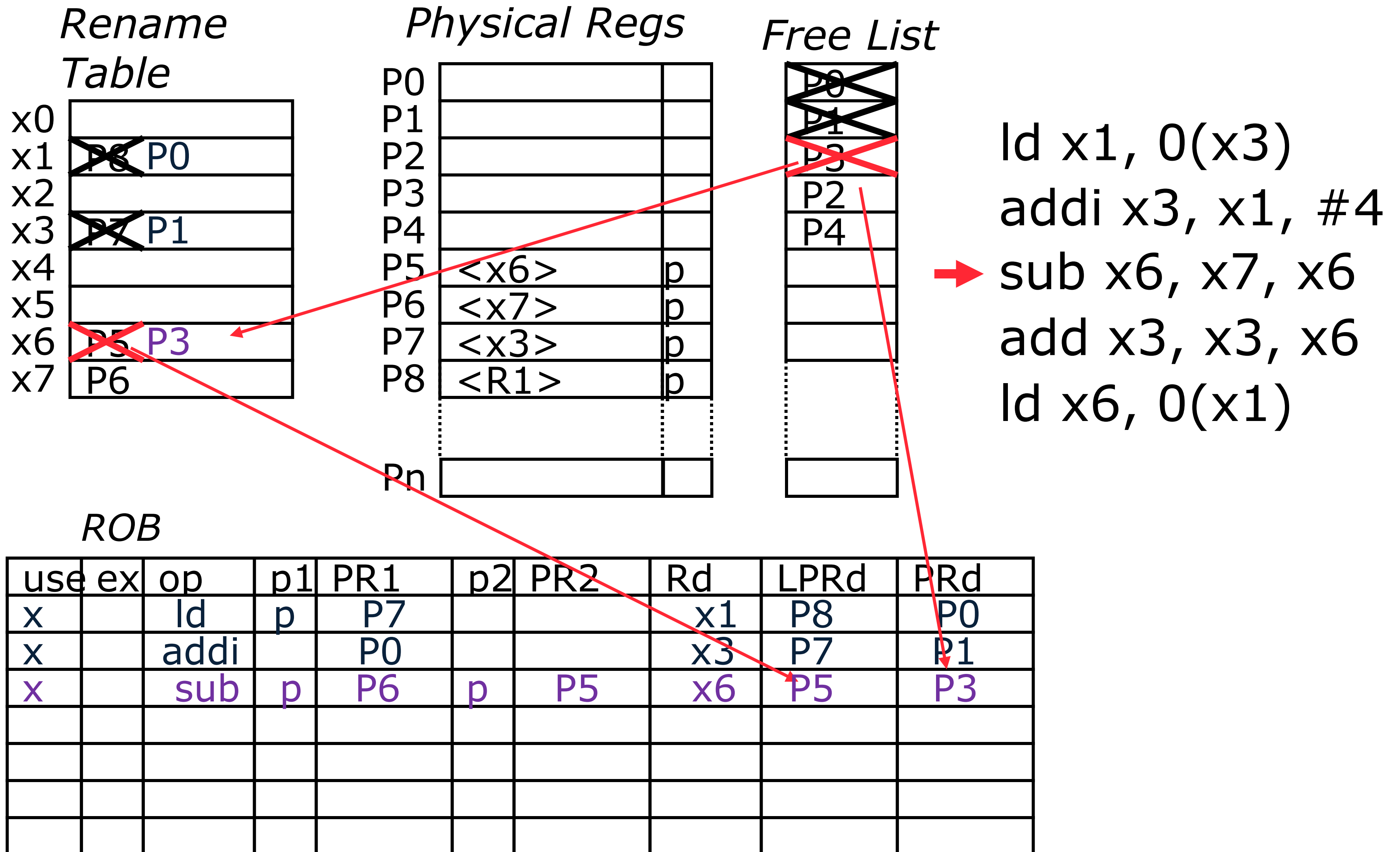
# Physical Register Management



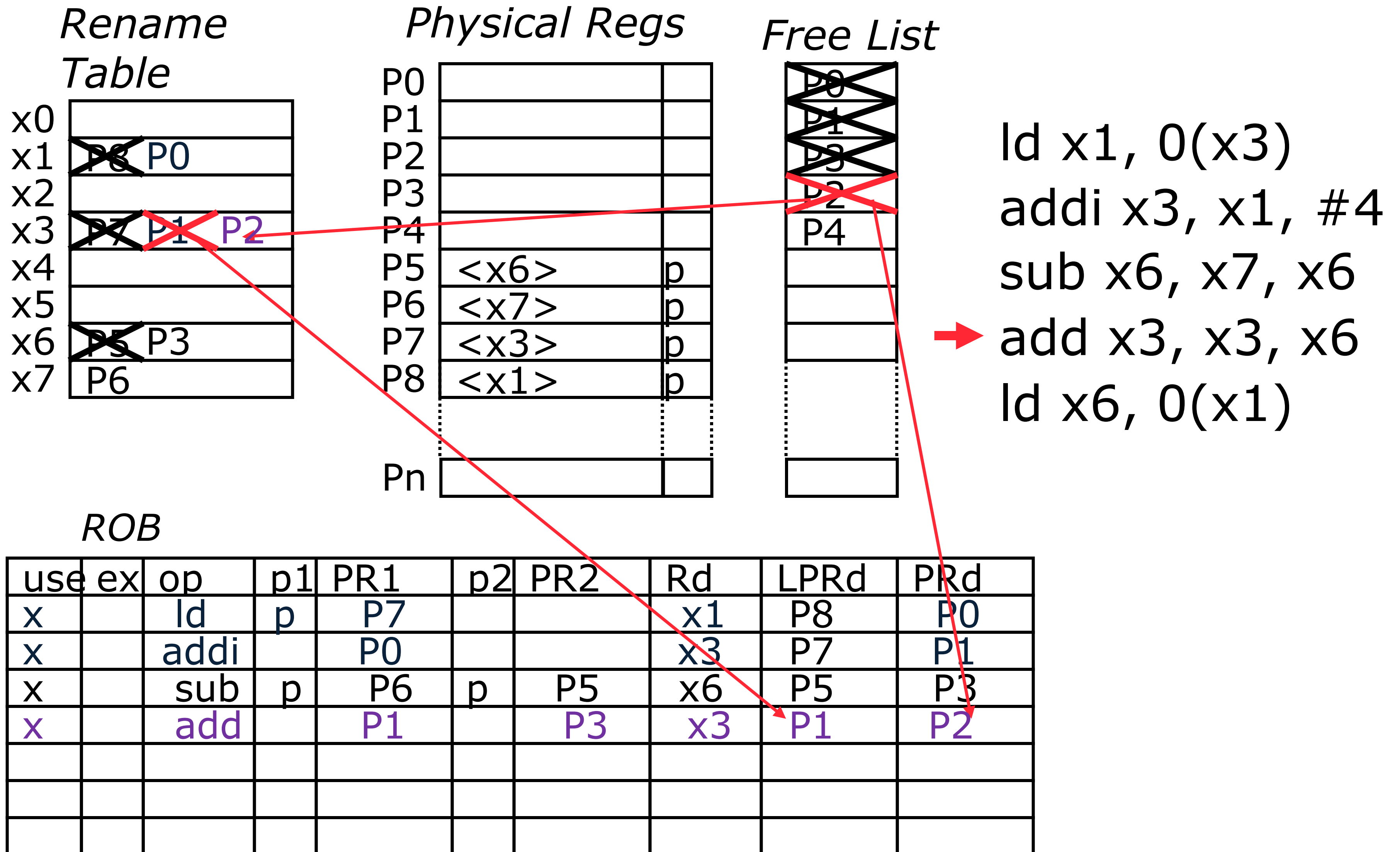
# Physical Register Management



# Physical Register Management

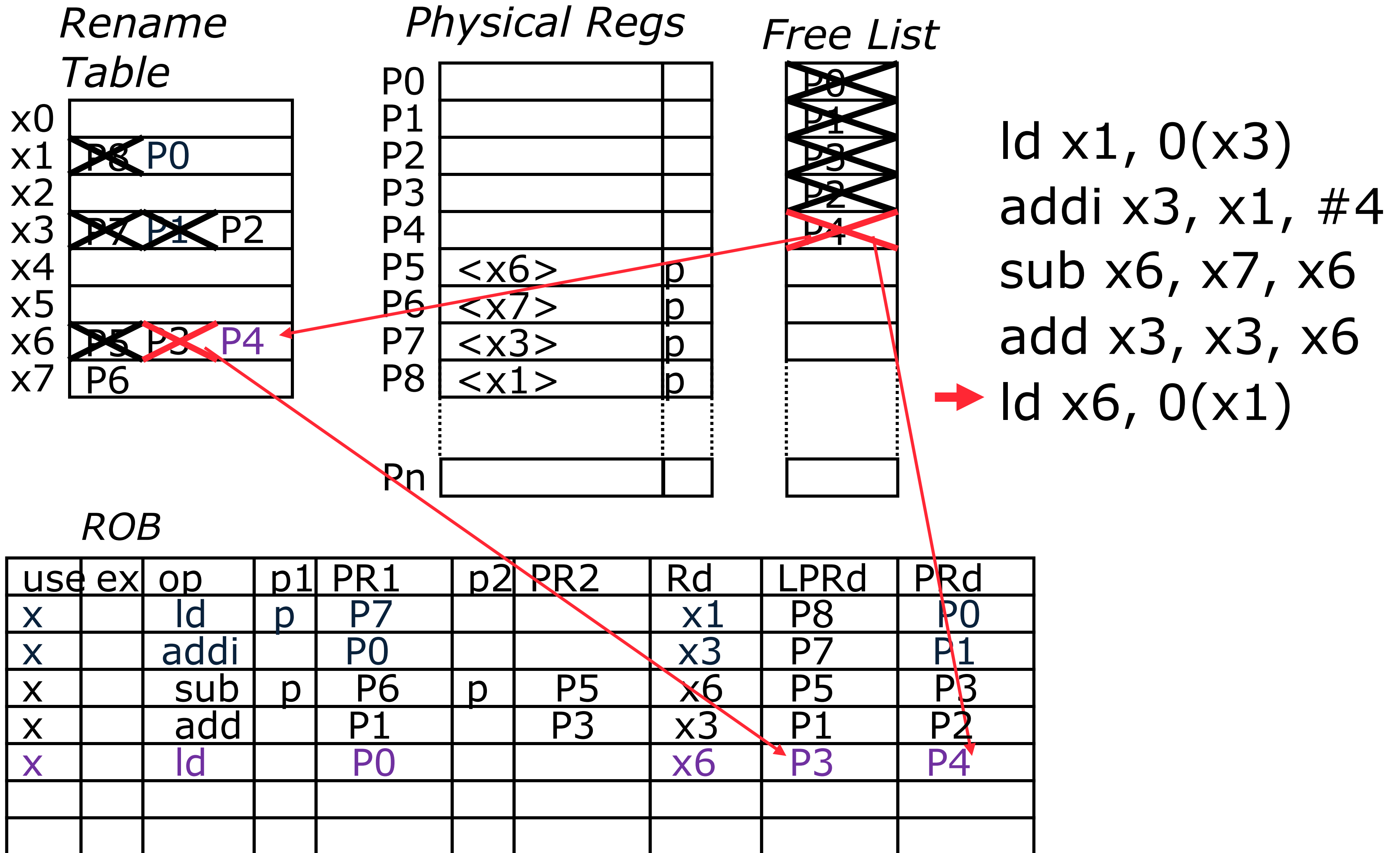


# Physical Register Management

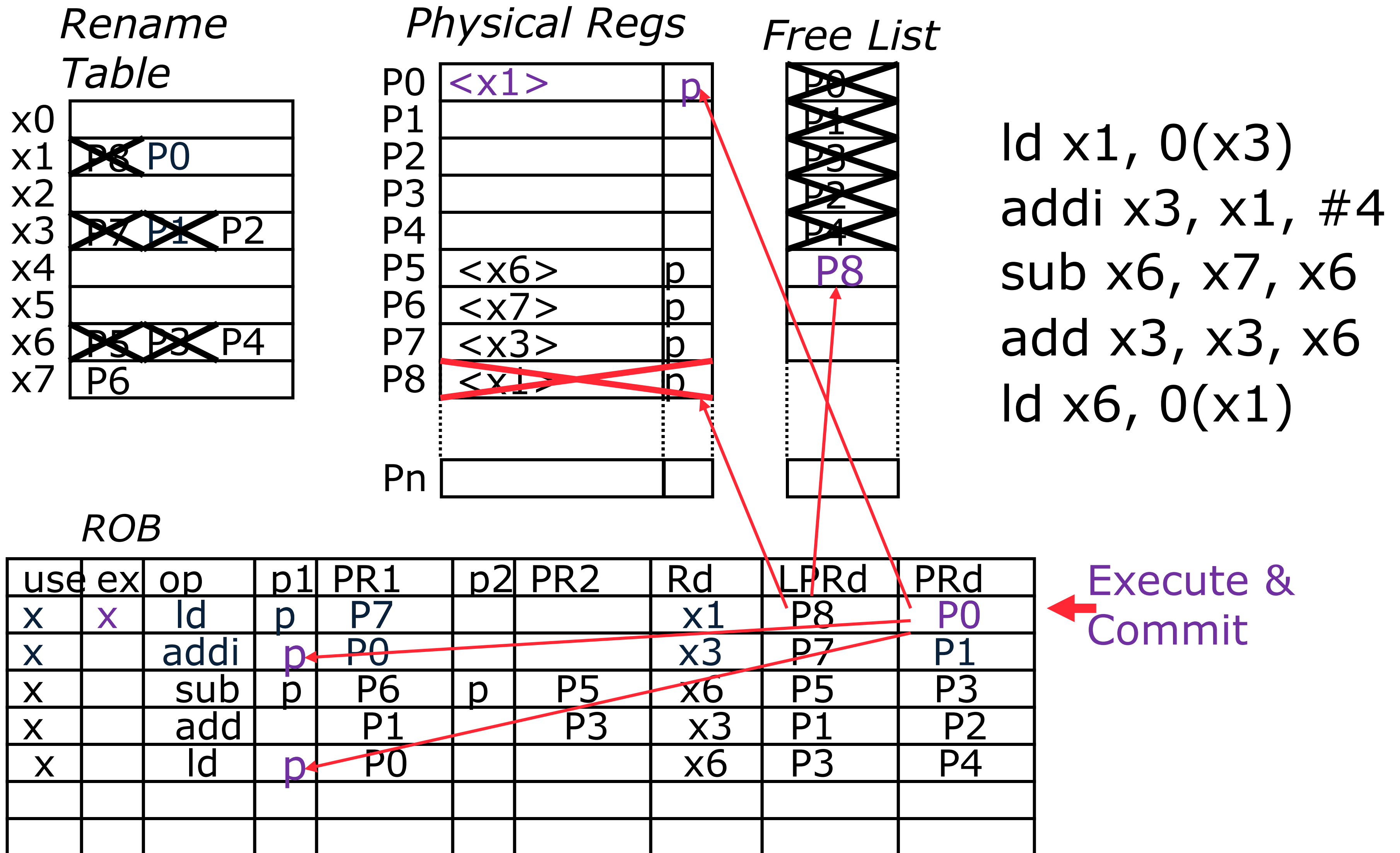



| use | ex | op | p1 | PR1 | p2 | PR2 | Rd | LPRd | PRd |
|---|---|---|---|---|---|---|---|---|---|
| x | | ld | p | P7 | | | x1 | P8 | P0 |
| x | | addi | | P0 | | | x3 | P7 | P1 |
| x | | sub | p | P6 | p | P5 | x6 | P5 | P3 |
| x | | add | | P1 | | P3 | x3 | P1 | P2 |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

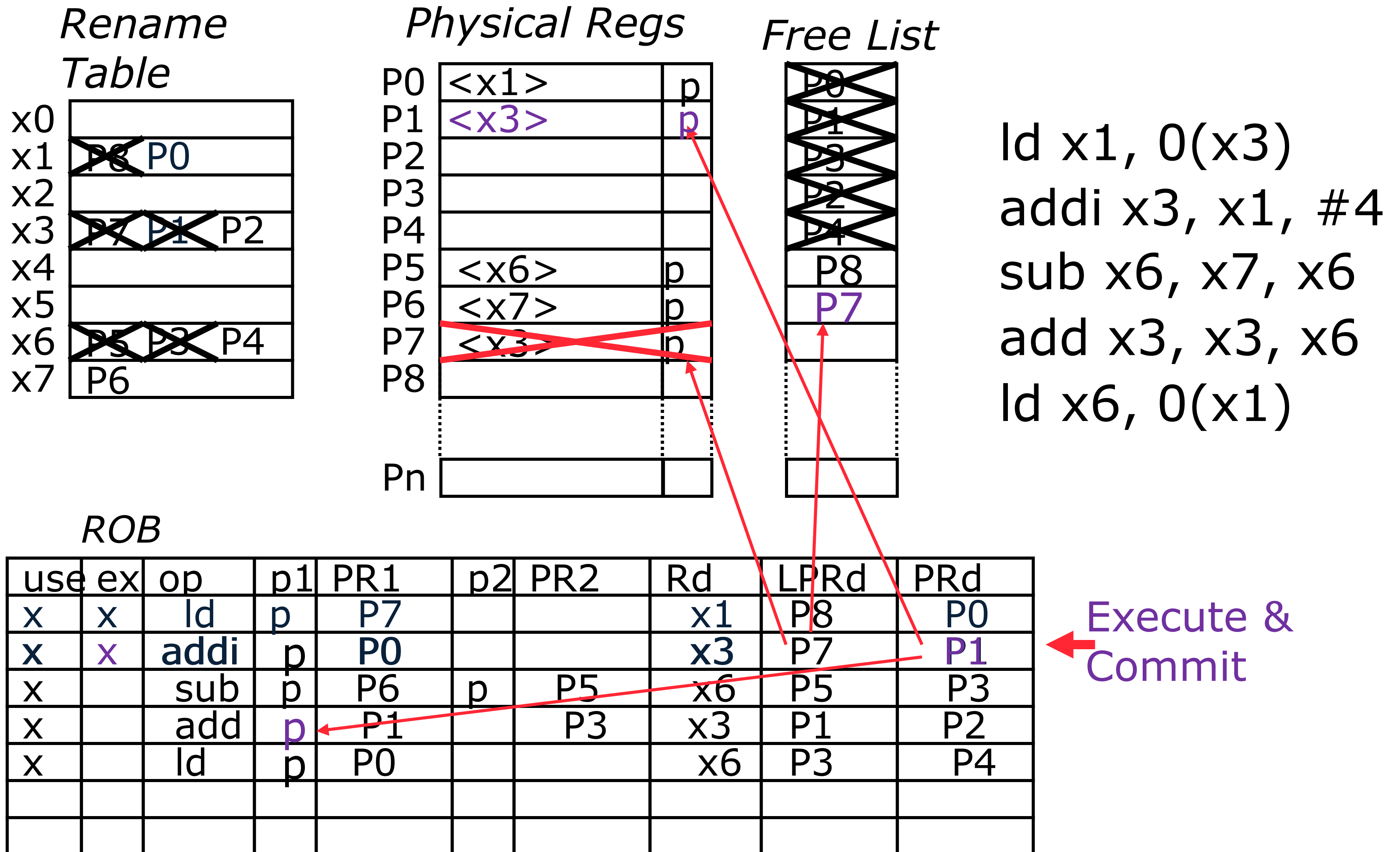
# Physical Register Management



# Physical Register Management



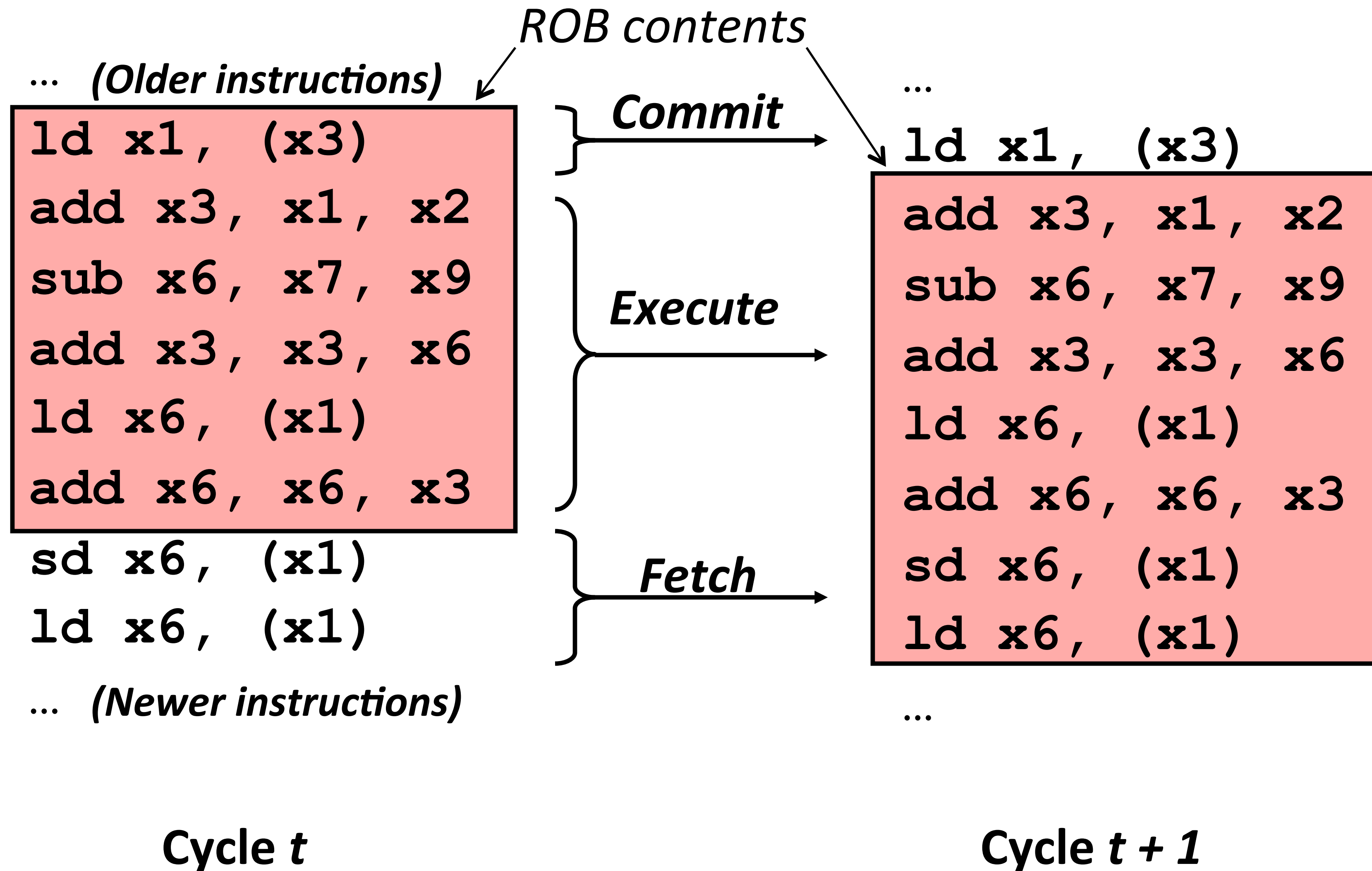
# Physical Register Management



# 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)



# 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.

| use | ex | op | p1 | PR1 | p2 | PR2 | PRd | ROB# |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

Reorder buffer used to hold exception information for commit.

Oldest →

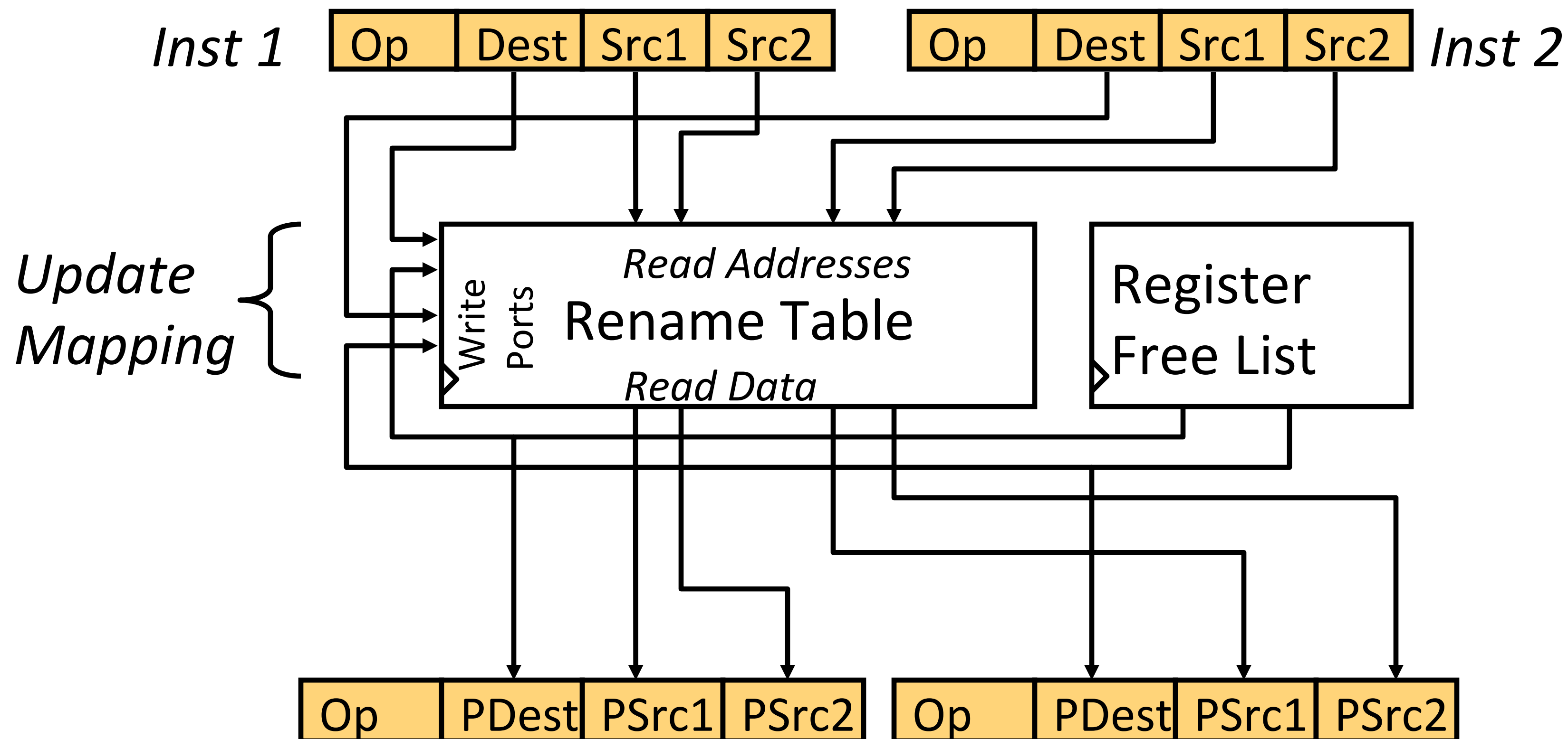
| Done? | Rd | LPRd | PC | Except? |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Free →

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

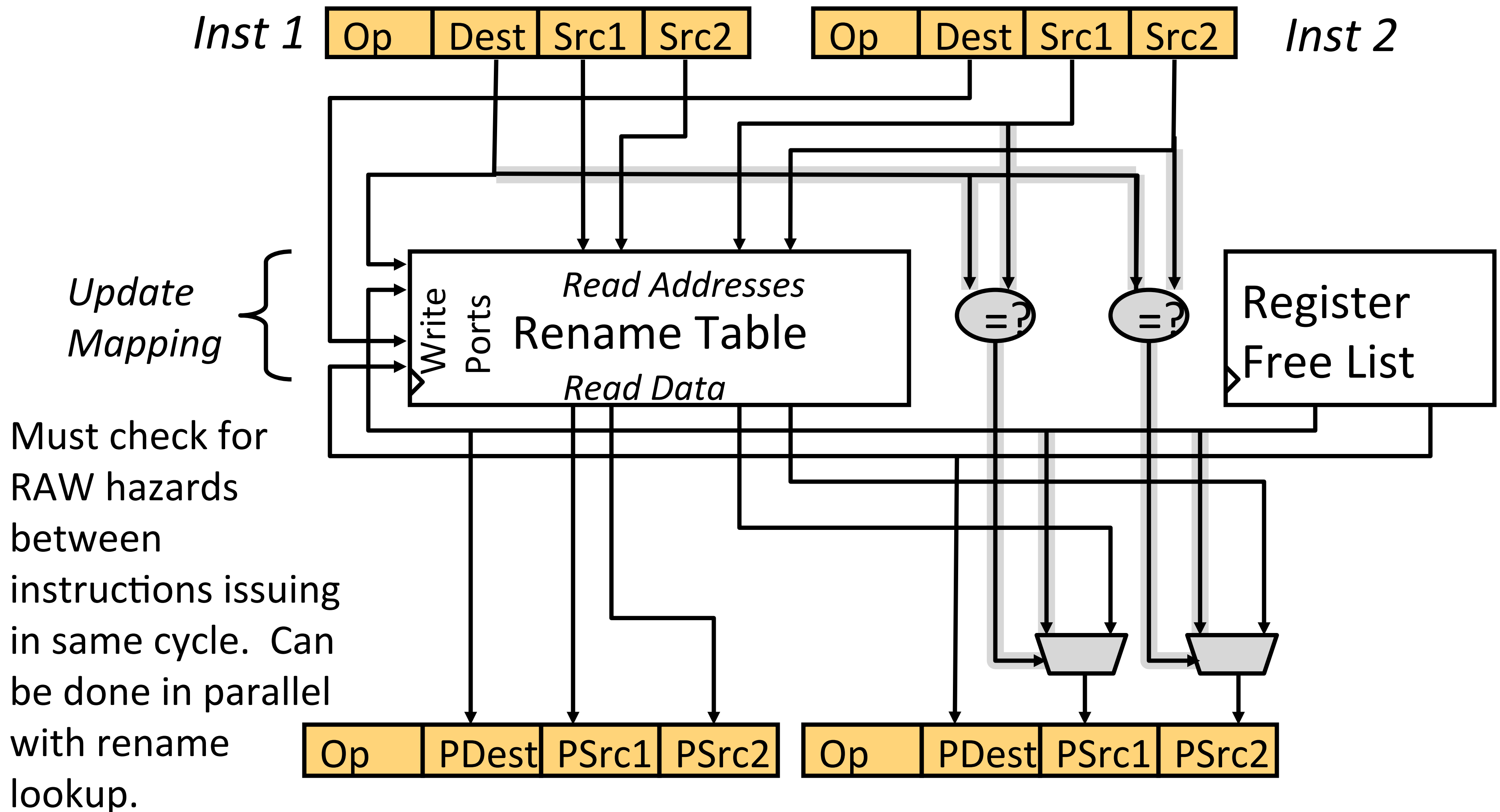
# Superscalar Register Renaming

- 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




*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)