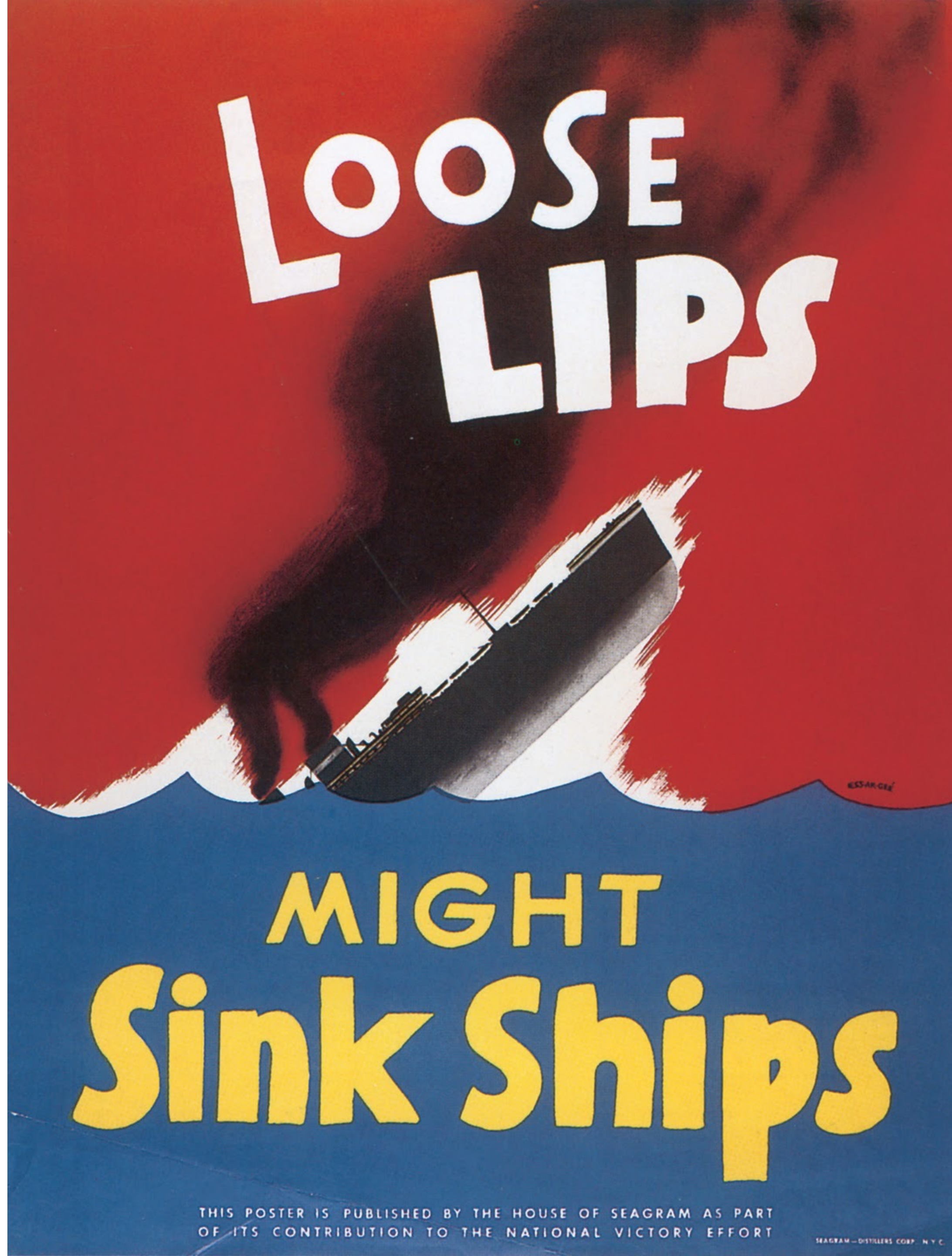
Speculative Execution

Joel Emer
Computer Science and Artificial Intelligence Laboratory
M.I.T.

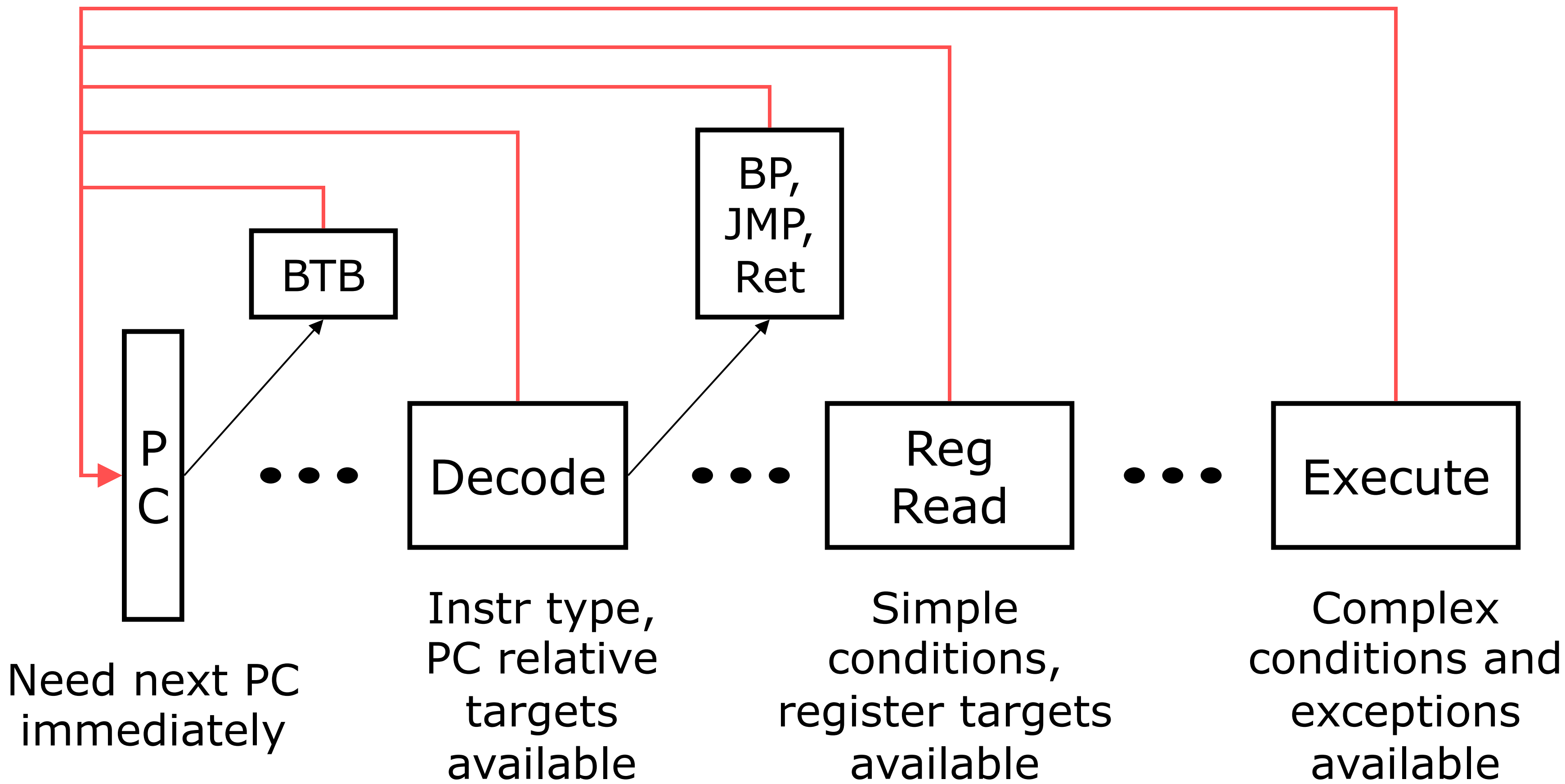
What does this WW2 poster have in common with pipelined processors?



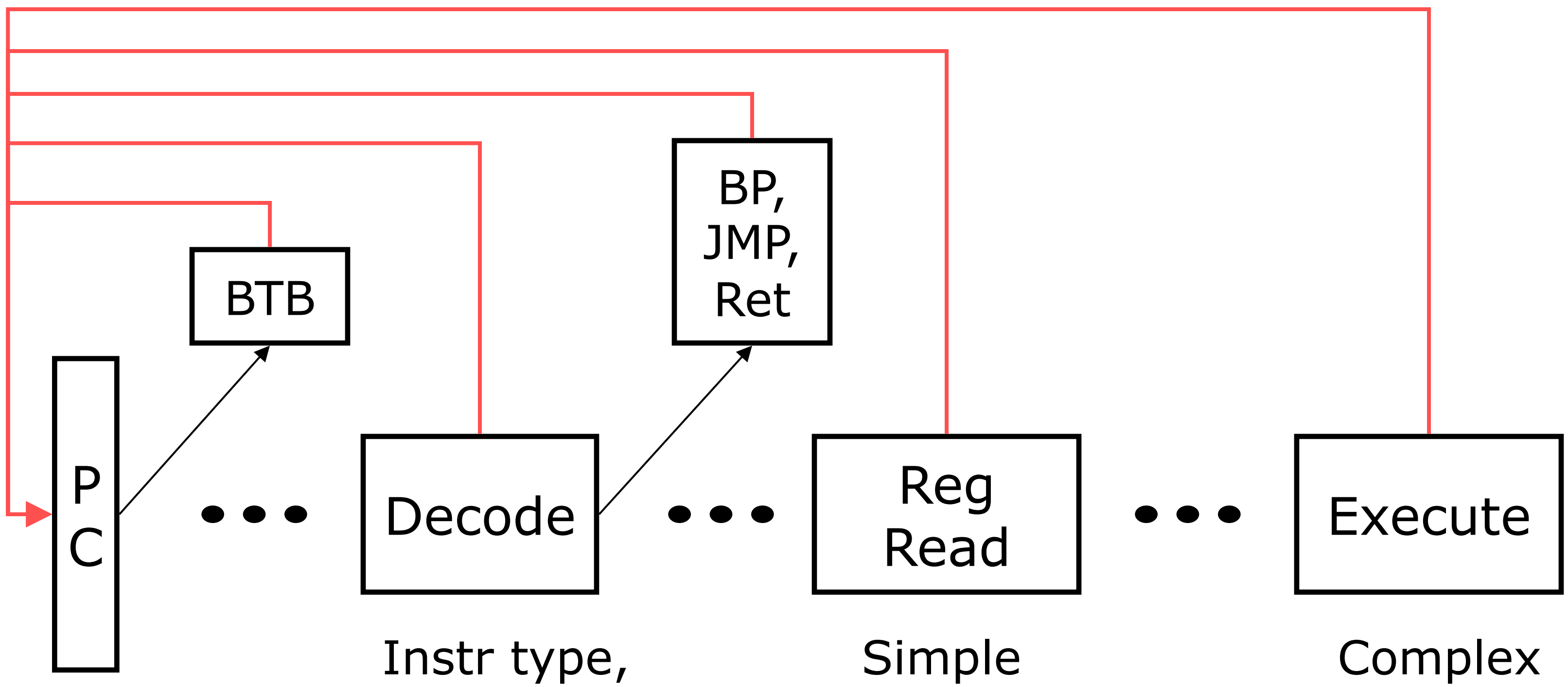
What does this WW2 poster have in common with pipelined processors?

What does this WW2 poster have in common with pipelined processors?

Overview of branch prediction



Overview of branch prediction



Need next PC immediately

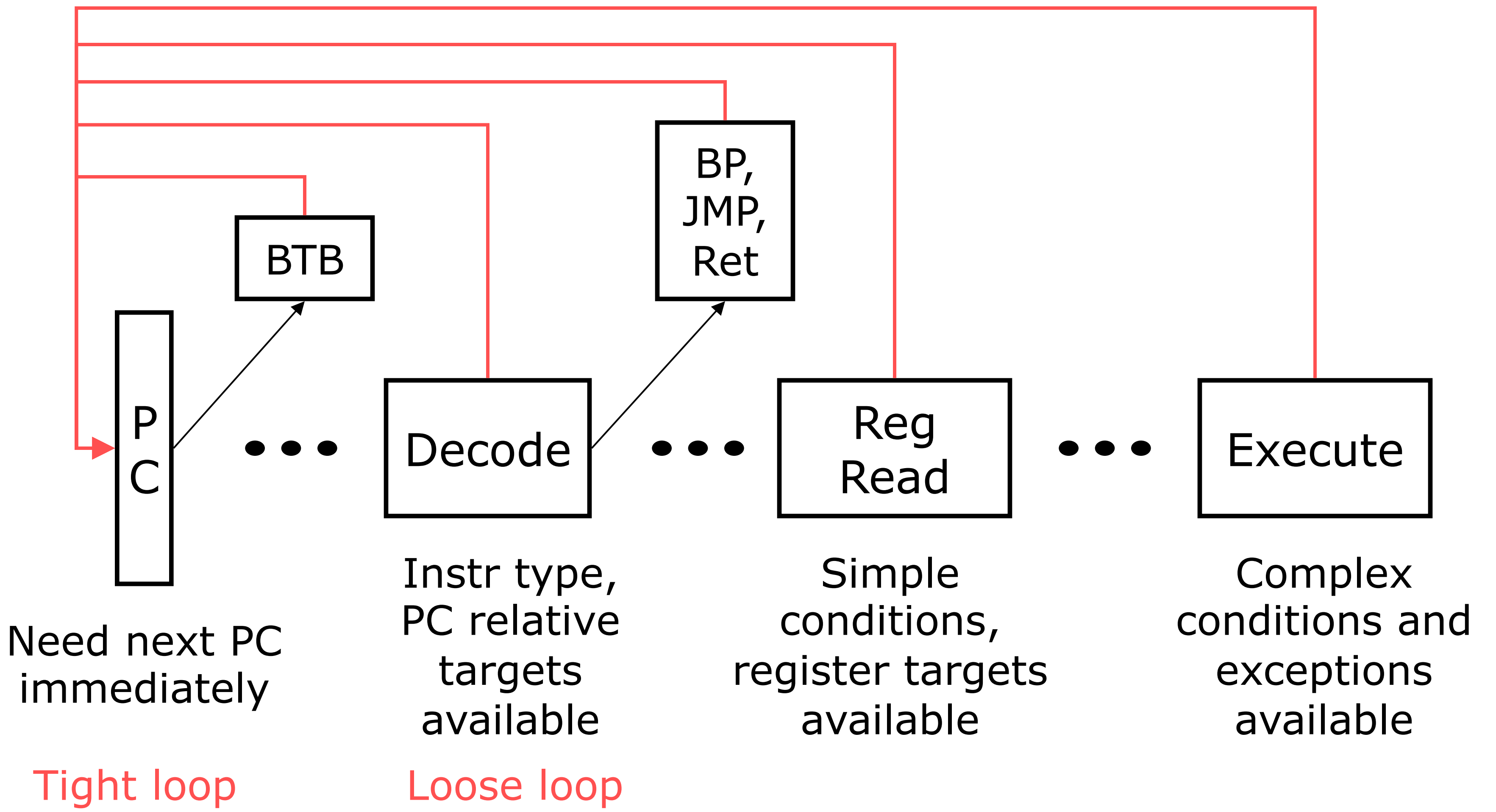
Instr type, PC relative targets available

Simple conditions, register targets available

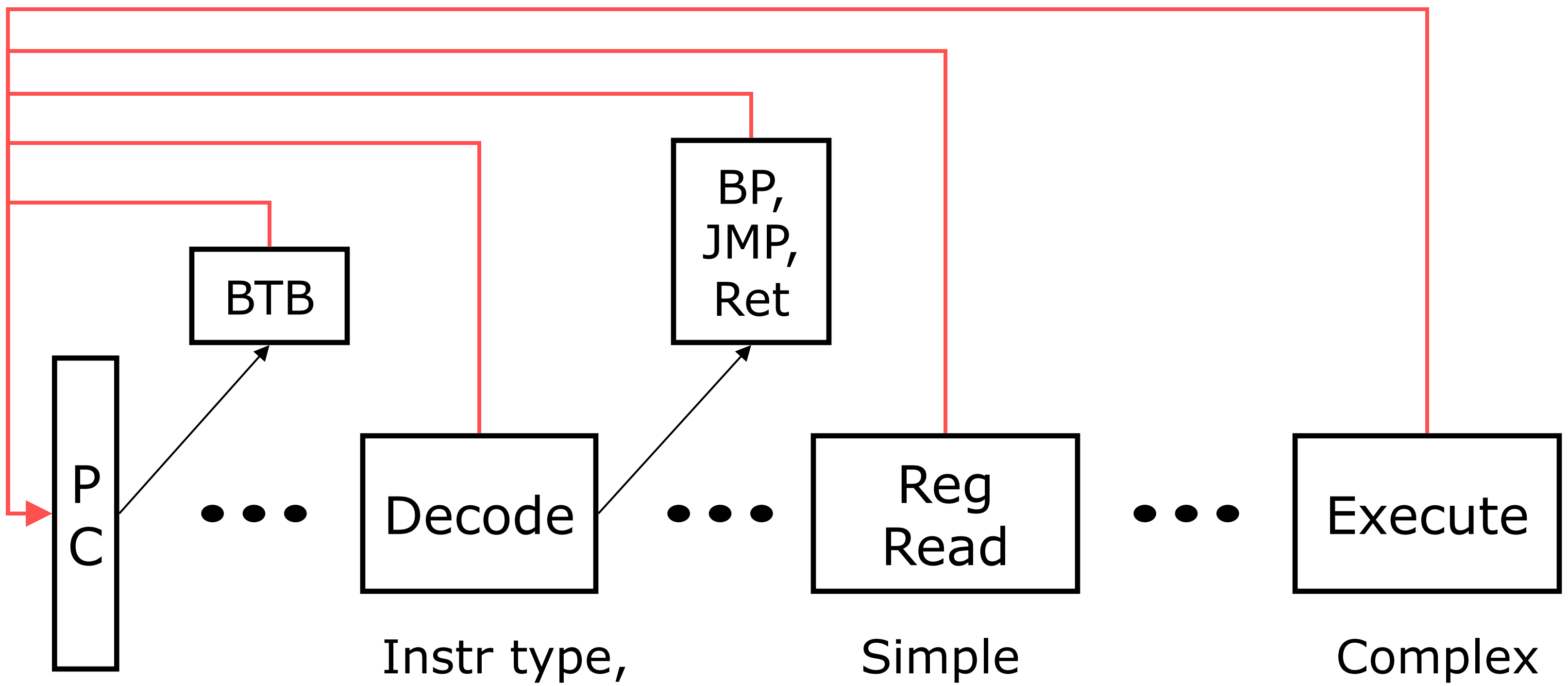
Complex conditions and exceptions available

Tight loop

Overview of branch prediction



Overview of branch prediction



Need next PC immediately

Tight loop

Instr type, PC relative targets available

Loose loop

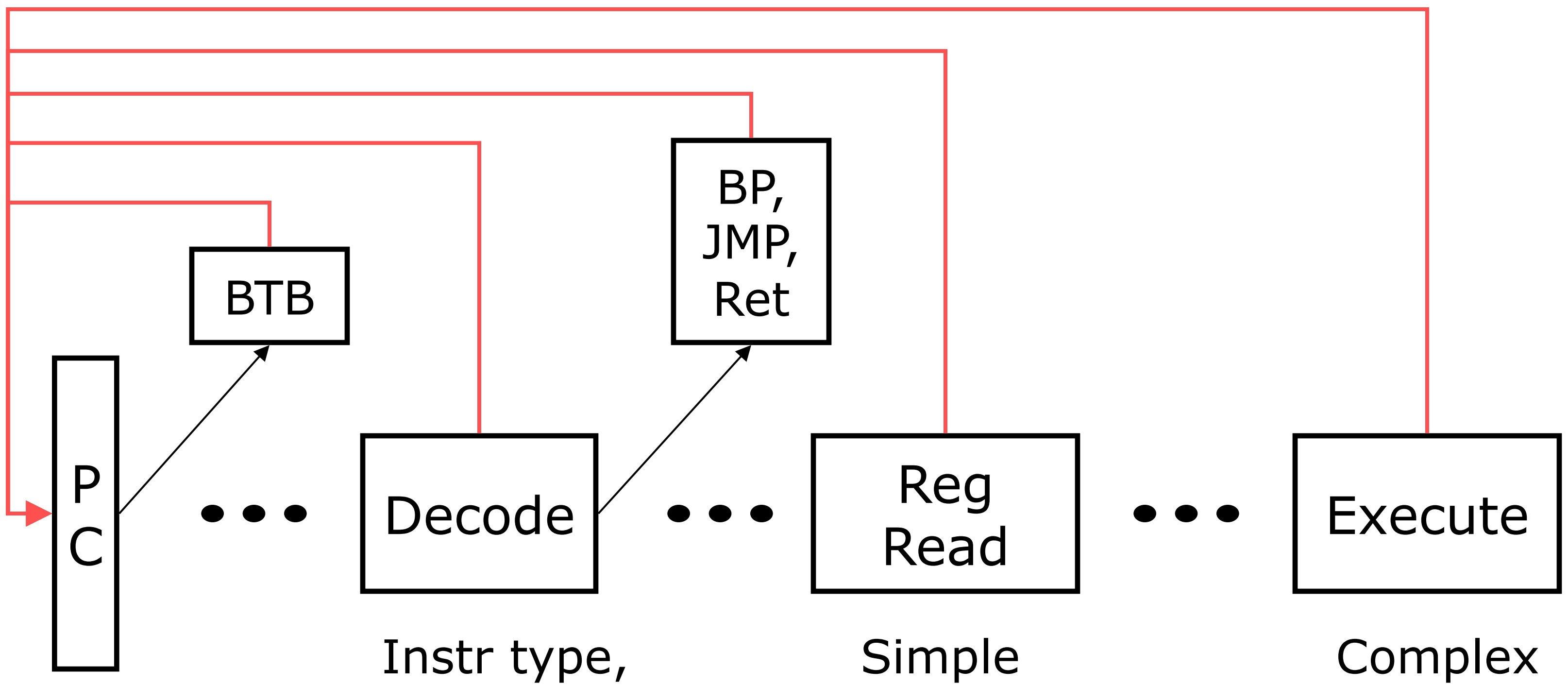
Simple conditions, register targets available

Loose loop

Complex conditions and exceptions available

Loose loop

Overview of branch prediction



Need next PC immediately

Instr type, PC relative targets available

Simple conditions, register targets available

Complex conditions and exceptions available

Tight loop

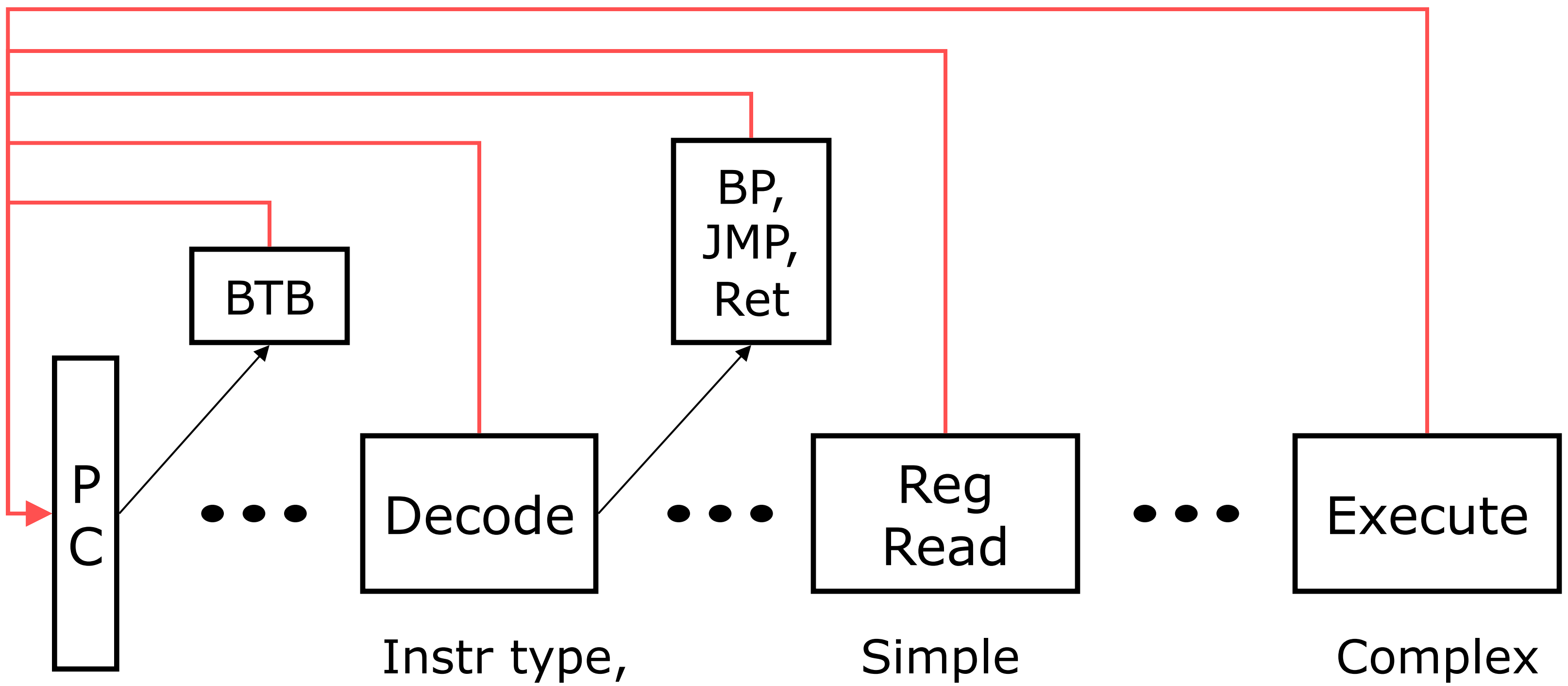
Loose loop

Loose loop

Loose loop

Must speculation check always be correct?

Overview of branch prediction



Need next PC immediately	Instr type, PC relative targets available	Simple conditions, register targets available	Complex conditions and exceptions available
Tight loop	Loose loop	Loose loop	Loose loop

Must speculation check always be correct? No…

Speculative Execution Recipe

1. Proceed ahead despite unresolved dependencies using a prediction for an architectural or micro-architectural value

2. Maintain both old and new values on updates to architectural (and often micro-architectural) state

3. After sure that there was no mis-speculation and there will be no more uses of the old values, discard old values and just use new values

OR

3. In event of mis-speculation, dispose of all new values, restore old values, and re-execute from point before mis-speculation

Why might one use old values?

Speculative Execution Recipe

1. Proceed ahead despite unresolved dependencies using a prediction for an architectural or micro-architectural value

2. Maintain both old and new values on updates to architectural (and often micro-architectural) state

3. After sure that there was no mis-speculation and there will be no more uses of the old values, discard old values and just use new values

OR

3. In event of mis-speculation, dispose of all new values, restore old values, and re-execute from point before mis-speculation

Why might one use old values? O-O-O WAR hazards

Value Management Strategies

Value Management Strategies

Greedy (or Eager) Update:

– Update value in place, and

Value Management Strategies

Greedy (or Eager) Update:

- Update value in place, and
- Provide means to reconstruct old values for recovery

Value Management Strategies

Greedy (or Eager) Update:

- Update value in place, and
- Provide means to reconstruct old values for recovery
  - often this is a log of old values

Value Management Strategies

Greedy (or Eager) Update:

- Update value in place, and
- Provide means to reconstruct old values for recovery
  - often this is a log of old values

Lazy Update:

- Buffer new value, leaving old value in place

Value Management Strategies

Greedy (or Eager) Update:

- Update value in place, and
- Provide means to reconstruct old values for recovery
  - often this is a log of old values

Lazy Update:

- Buffer new value, leaving old value in place
- Replace old value only at 'commit' time

Value Management Strategies

Greedy (or Eager) Update:

- Update value in place, and
- Provide means to reconstruct old values for recovery
  - often this is a log of old values

Lazy Update:

- Buffer new value, leaving old value in place
- Replace old value only at ‘commit’ time

Why leave an old value in place?

Value Management Strategies

Greedy (or Eager) Update:

- Update value in place, and
- Provide means to reconstruct old values for recovery
  - often this is a log of old values

Lazy Update:

- Buffer new value, leaving old value in place
- Replace old value only at ‘commit’ time

Why leave an old value in place?

- When there will be limited use of new value
- To make it easy to use old value after new value is generated
- To simplify recovery

Exceptions

Exceptions

- Exceptions create a dependence on the value of the next PC

Exceptions

- Exceptions create a dependence on the value of the next PC
- Options for handling this dependence:

Exceptions

- Exceptions create a dependence on the value of the next PC
- Options for handling this dependence:
  - Stall

Exceptions

- Exceptions create a dependence on the value of the next PC
- Options for handling this dependence:
  - Stall — No

Exceptions

- Exceptions create a dependence on the value of the next PC
- Options for handling this dependence:
  - Stall — No
  - Bypass

Exceptions

- Exceptions create a dependence on the value of the next PC

- Options for handling this dependence:

  - Stall — No
  - Bypass — No

Exceptions

- Exceptions create a dependence on the value of the next PC

- Options for handling this dependence:

  - Stall — No
  - Bypass — No
  - Find something else to do

Exceptions

- Exceptions create a dependence on the value of the next PC

- Options for handling this dependence:

  - Stall — No
  - Bypass — No
  - Find something else to do — No

Exceptions

- Exceptions create a dependence on the value of the next PC

- Options for handling this dependence:

  - Stall — No
  - Bypass — No
  - Find something else to do — No
  - Change the architecture

Exceptions

- Exceptions create a dependence on the value of the next PC

- Options for handling this dependence:

• Stall	No
• Bypass	No
• Find something else to do	No
• Change the architecture	Sometimes: Alpha, Multiflow

Exceptions

- Exceptions create a dependence on the value of the next PC

- Options for handling this dependence:

• Stall	No
• Bypass	No
• Find something else to do	No
• Change the architecture	Sometimes: Alpha, Multiflow
• Speculate!	

Exceptions

- Exceptions create a dependence on the value of the next PC

- Options for handling this dependence:

• Stall	No
• Bypass	No
• Find something else to do	No
• Change the architecture	Sometimes: Alpha, Multiflow
• Speculate!	Most common approach!

Exceptions

- Exceptions create a dependence on the value of the next PC
- Options for handling this dependence:

• Stall	No
• Bypass	No
• Find something else to do	No
• Change the architecture	Sometimes: Alpha, Multiflow
• Speculate!	Most common approach!

- How can we handle rollback on mis-speculation?

Exceptions

- Exceptions create a dependence on the value of the next PC
- Options for handling this dependence:

• Stall	No
• Bypass	No
• Find something else to do	No
• Change the architecture	Sometimes: Alpha, Multiflow
• Speculate!	Most common approach!

- How can we handle rollback on mis-speculation?

Delay state update until commit on speculated instructions

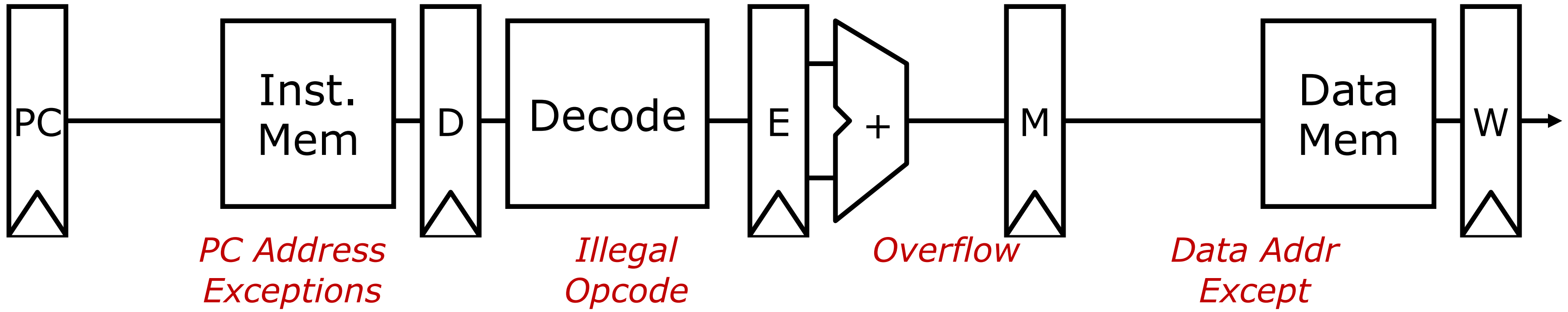
Exceptions

- Exceptions create a dependence on the value of the next PC
- Options for handling this dependence:
  - Stall — No
  - Bypass — No
  - Find something else to do — No
  - Change the architecture — Sometimes: Alpha, Multiflow
  - Speculate! — Most common approach!
- How can we handle rollback on mis-speculation?

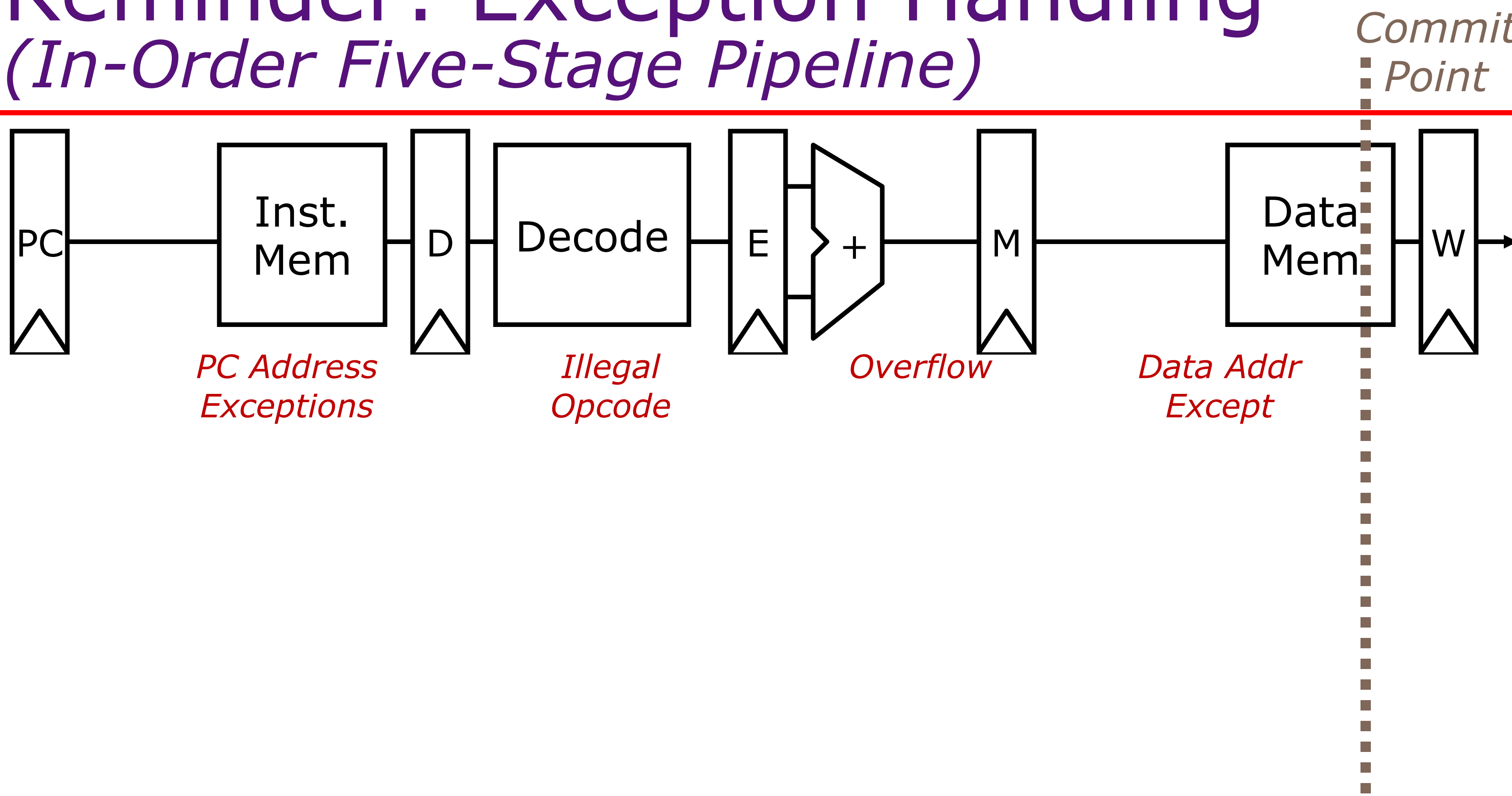
  Delay state update until commit on speculated instructions

- Note: earlier exceptions must override later ones

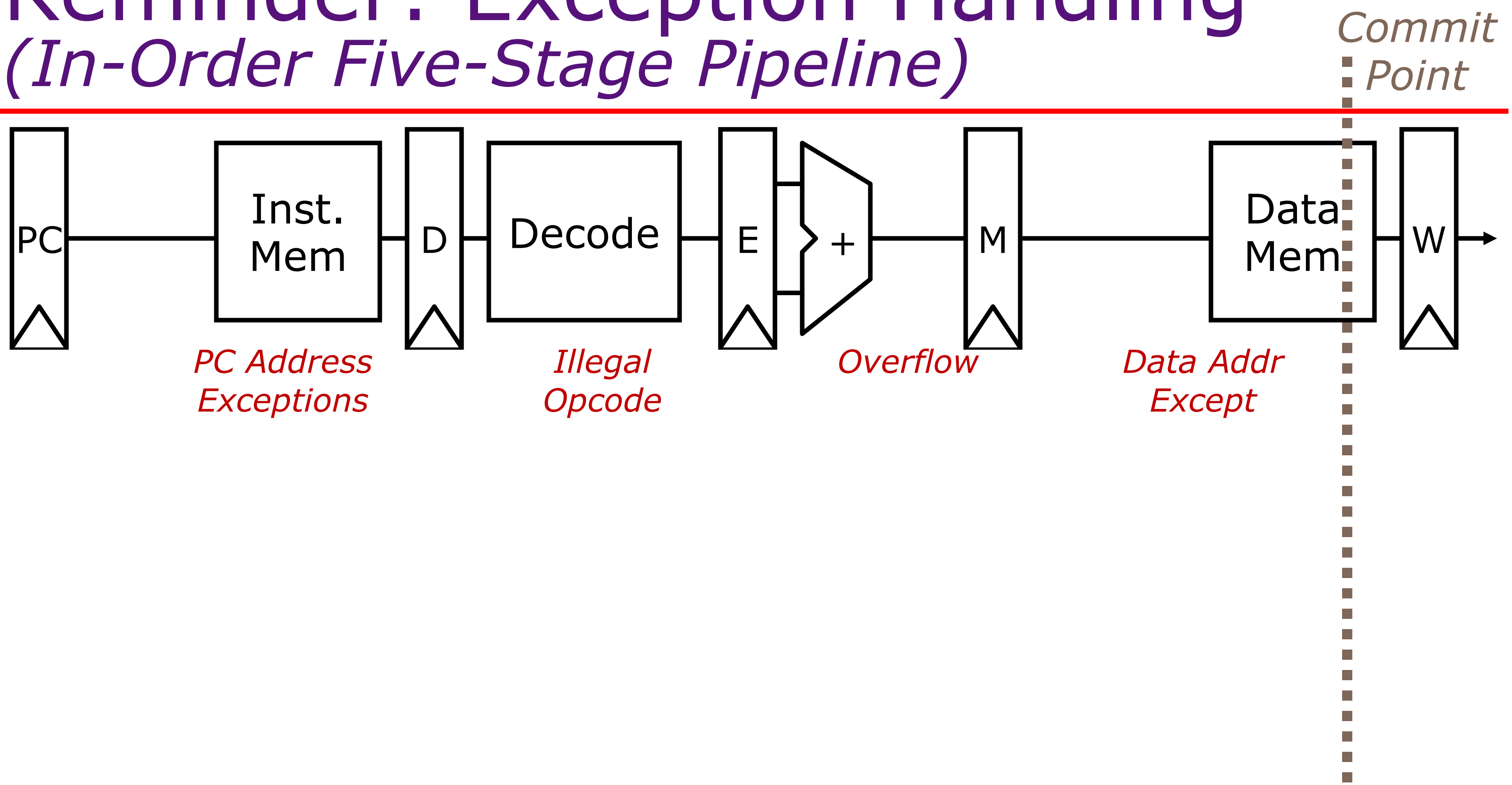
Reminder: Exception Handling
(In-Order Five-Stage Pipeline)



Reminder: Exception Handling
(In-Order Five-Stage Pipeline)

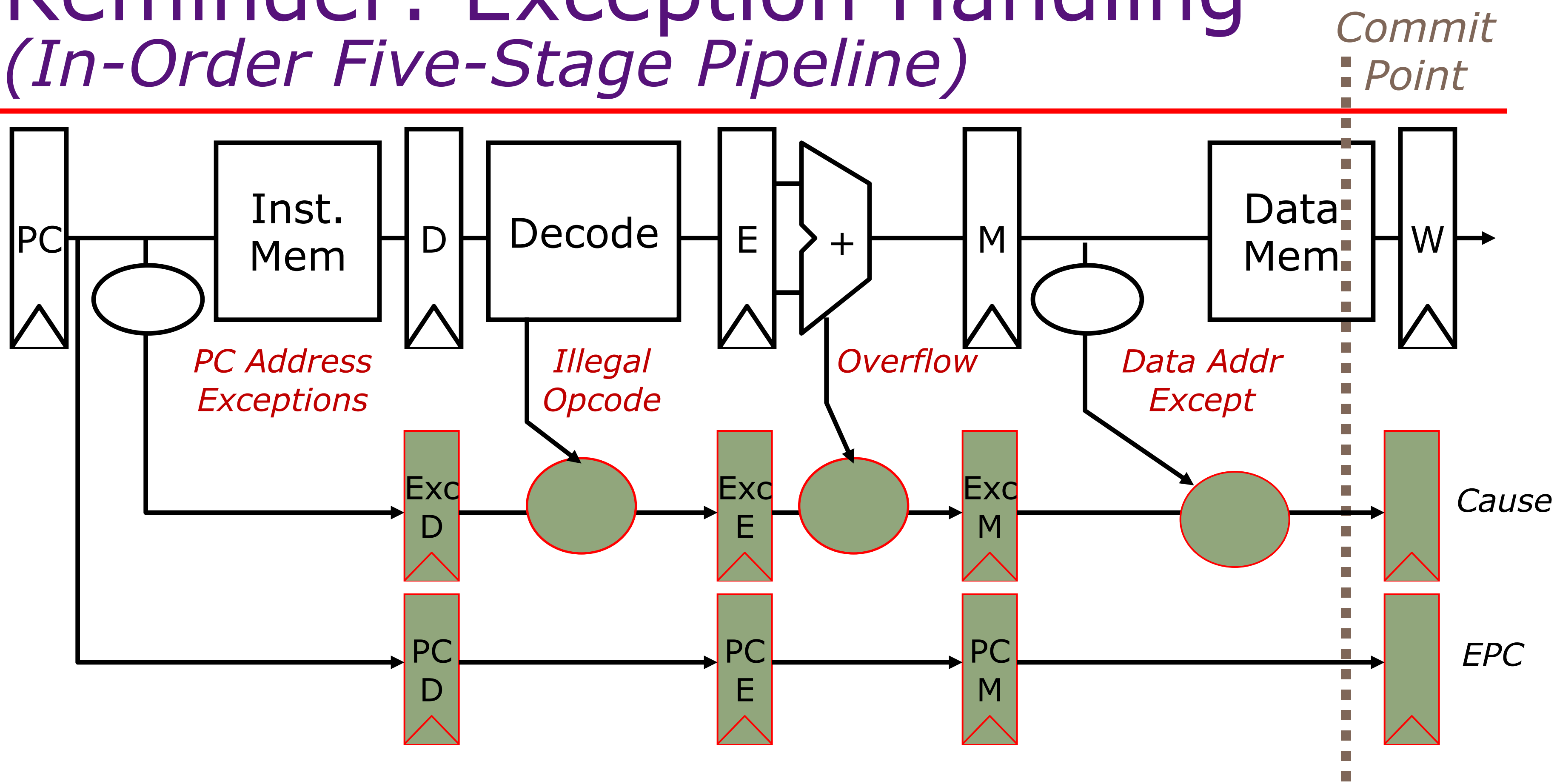


Reminder: Exception Handling
(In-Order Five-Stage Pipeline)



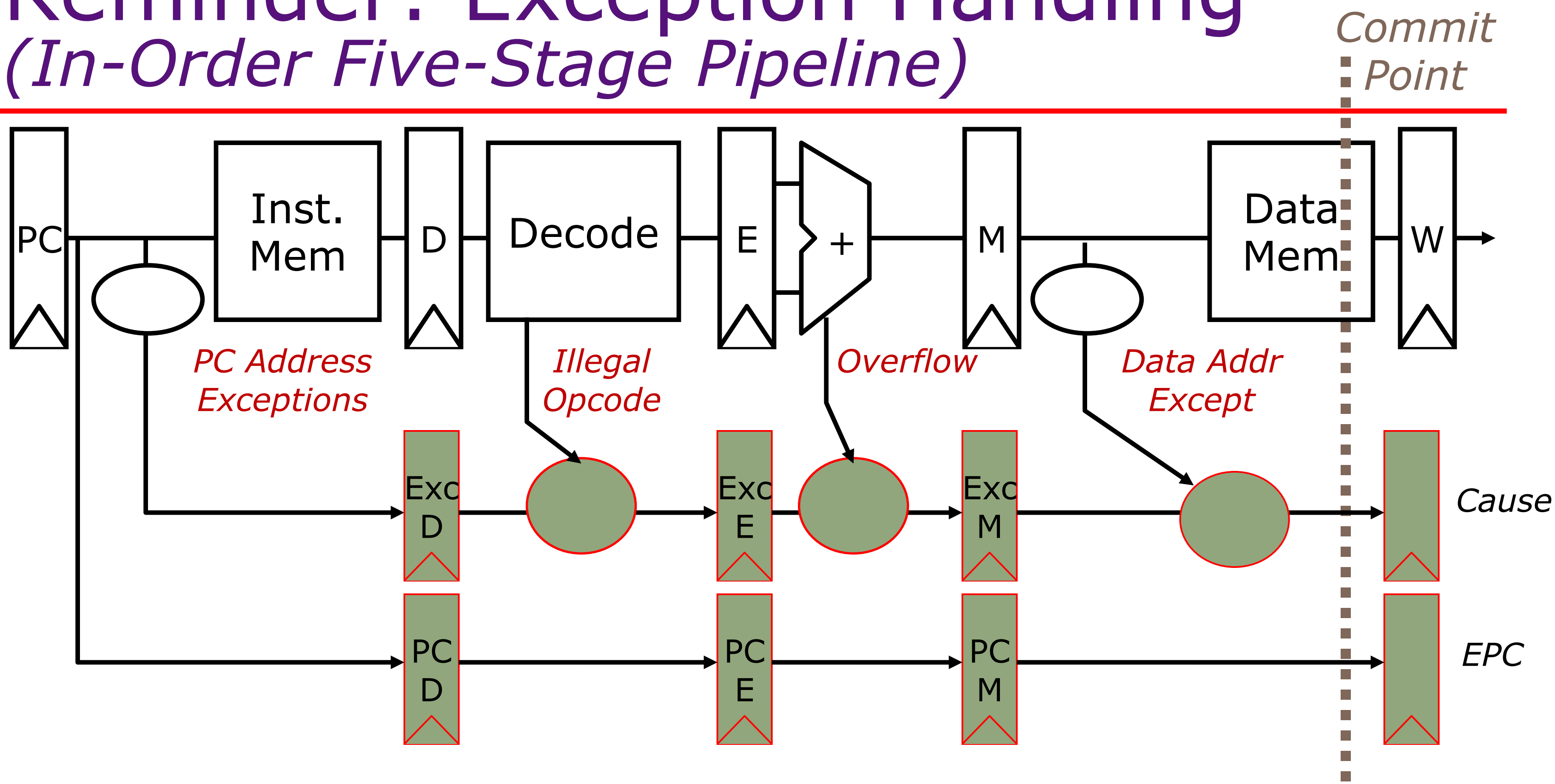
Hold exception flags in pipeline until commit point (M stage)

Reminder: Exception Handling
(In-Order Five-Stage Pipeline)



Hold exception flags in pipeline until commit point (M stage)

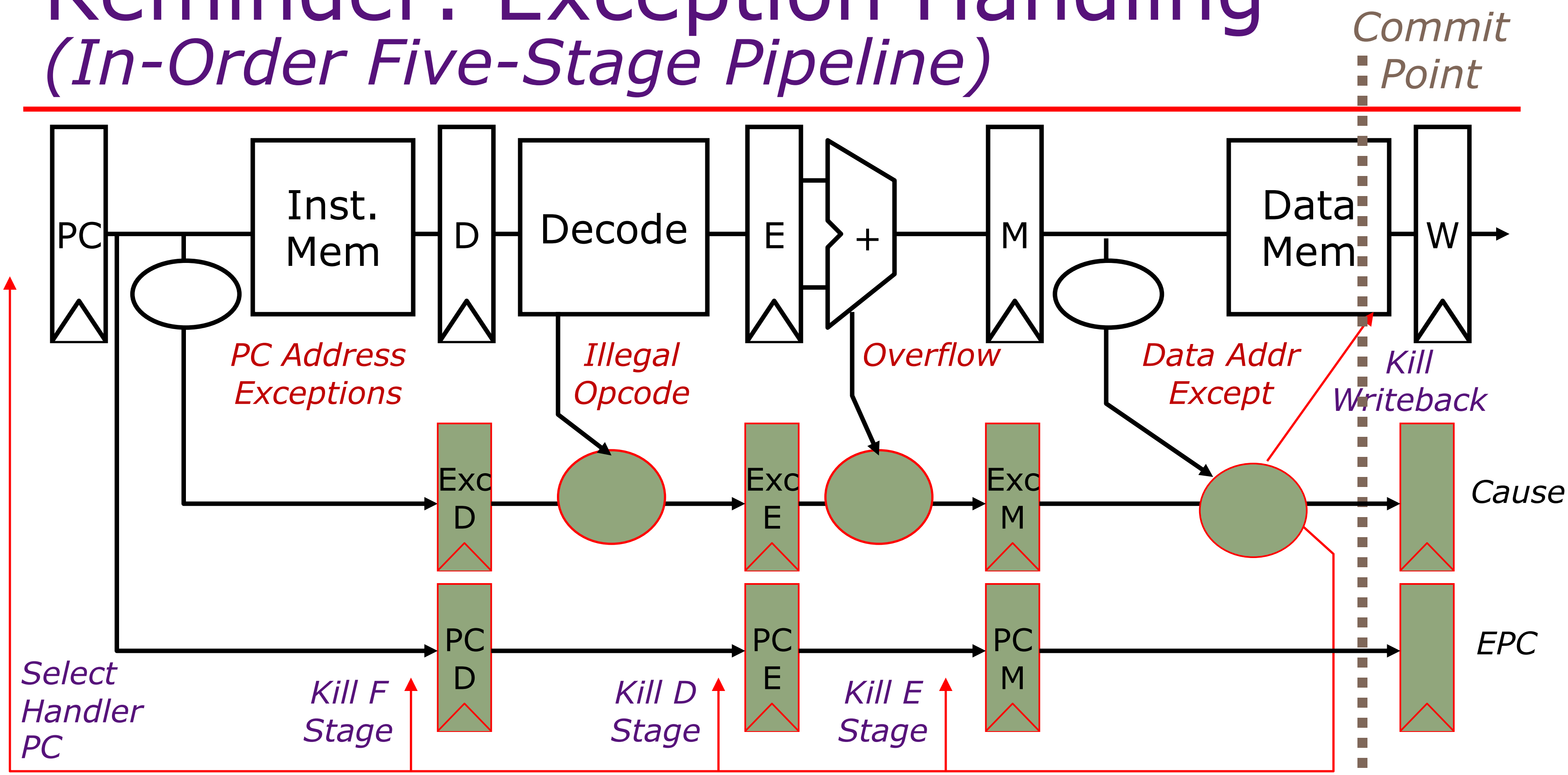
Reminder: Exception Handling
(In-Order Five-Stage Pipeline)



Hold exception flags in pipeline until commit point (M stage)

- If exception at commit:
  - update Cause/EPC registers
  - kill all stages
  - fetch at handler PC

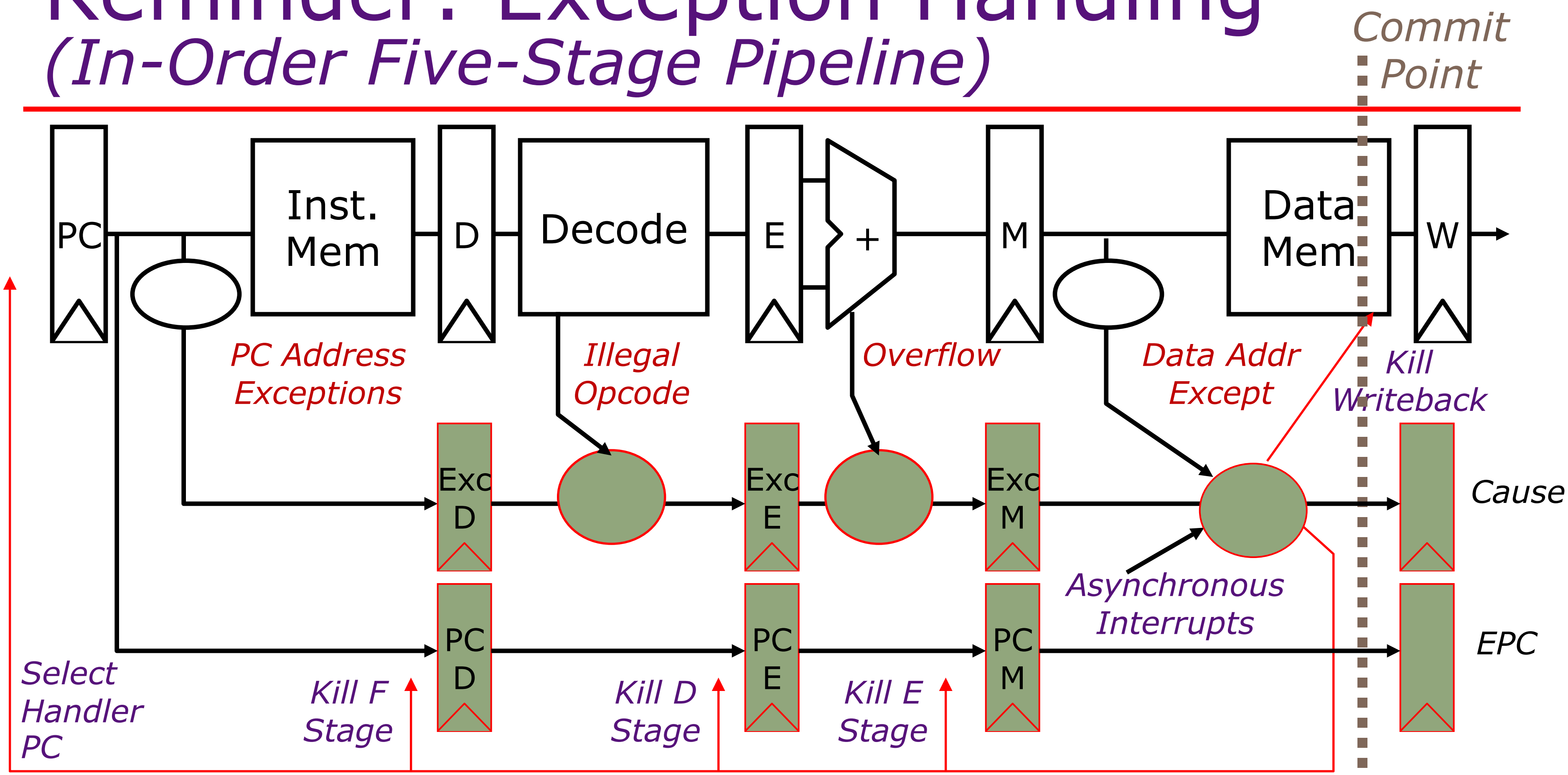
Reminder: Exception Handling
(In-Order Five-Stage Pipeline)



Hold exception flags in pipeline until commit point (M stage)

- If exception at commit:
  - update Cause/EPC registers
  - kill all stages
  - fetch at handler PC

Reminder: Exception Handling
(In-Order Five-Stage Pipeline)

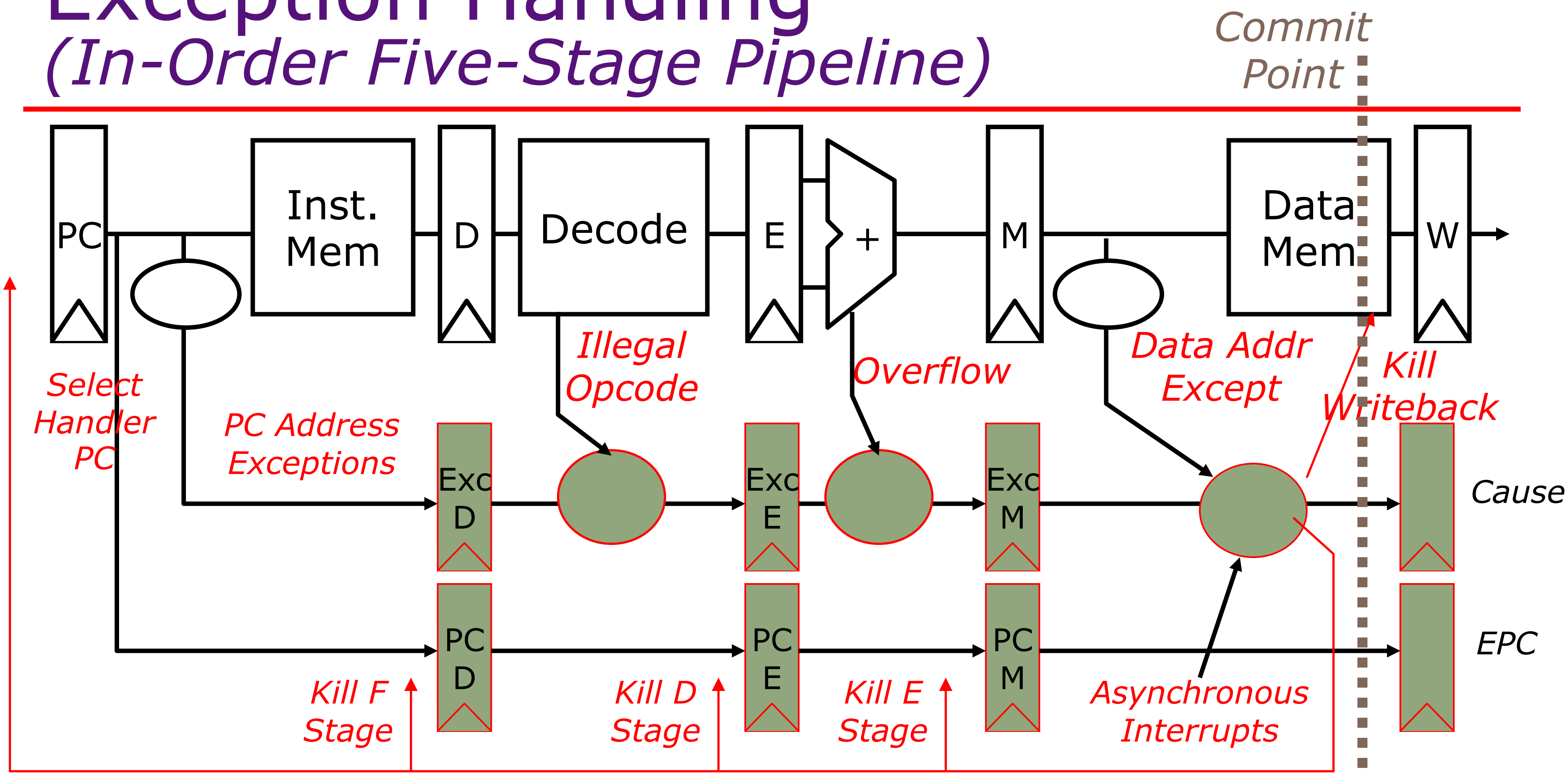


Hold exception flags in pipeline until commit point (M stage)

- If exception at commit:
  - update Cause/EPC registers
  - kill all stages
  - fetch at handler PC

Inject external interrupts at commit point

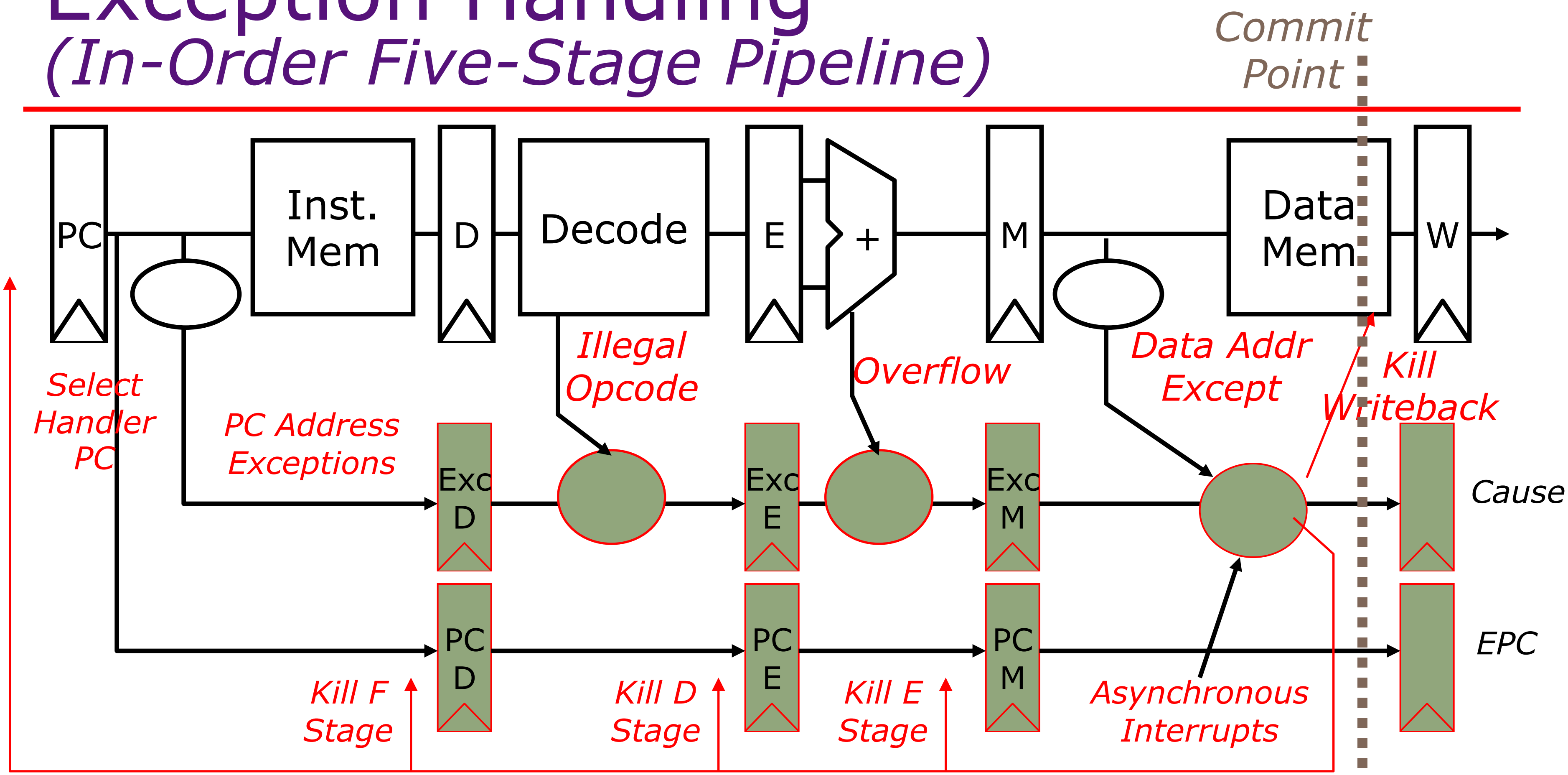
Exception Handling
(In-Order Five-Stage Pipeline)



Strategy for Registers?

Strategy for PC?

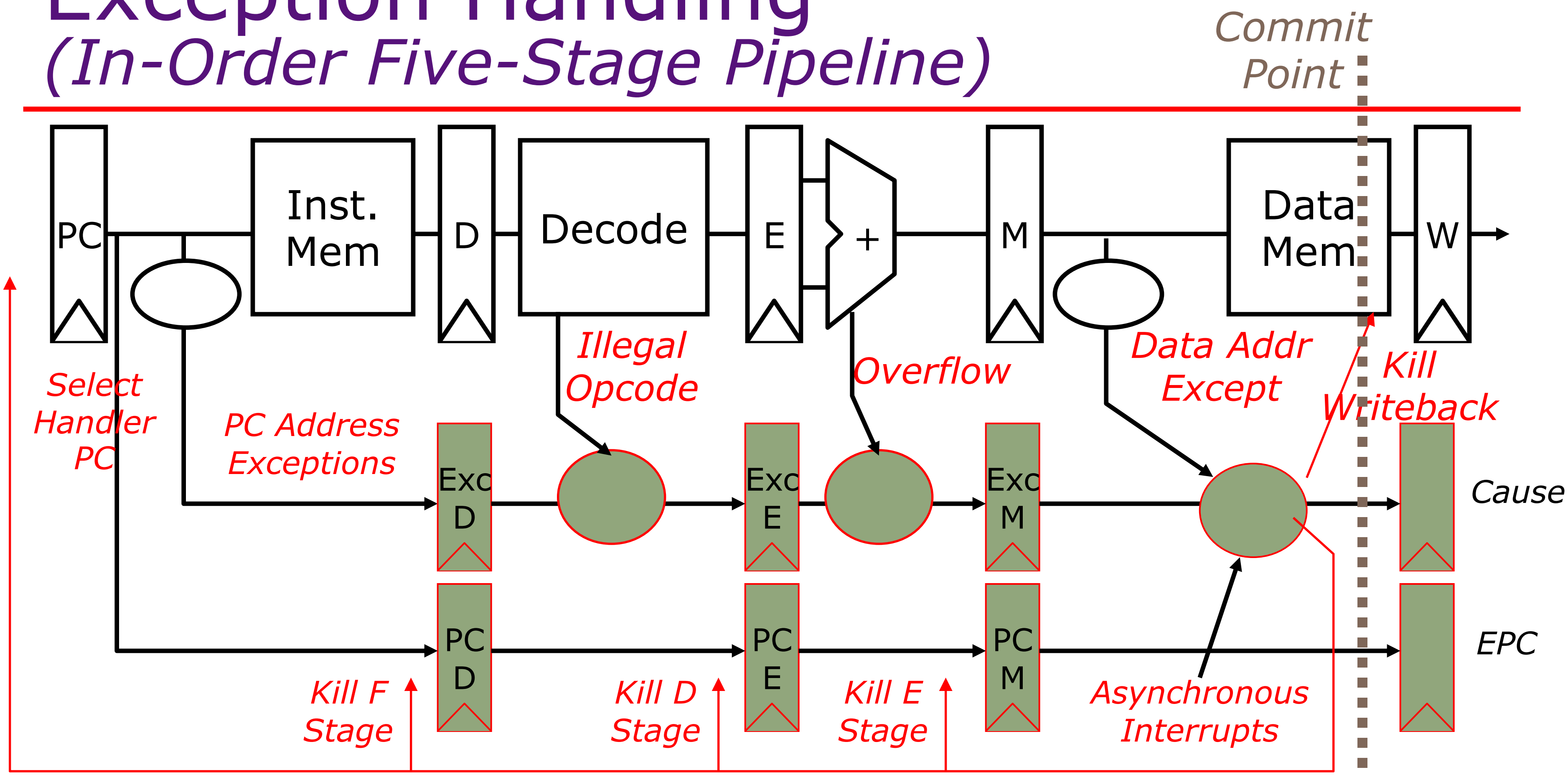
Exception Handling
(In-Order Five-Stage Pipeline)



Strategy for Registers? Lazy – update at commit

Strategy for PC?

Exception Handling
(In-Order Five-Stage Pipeline)

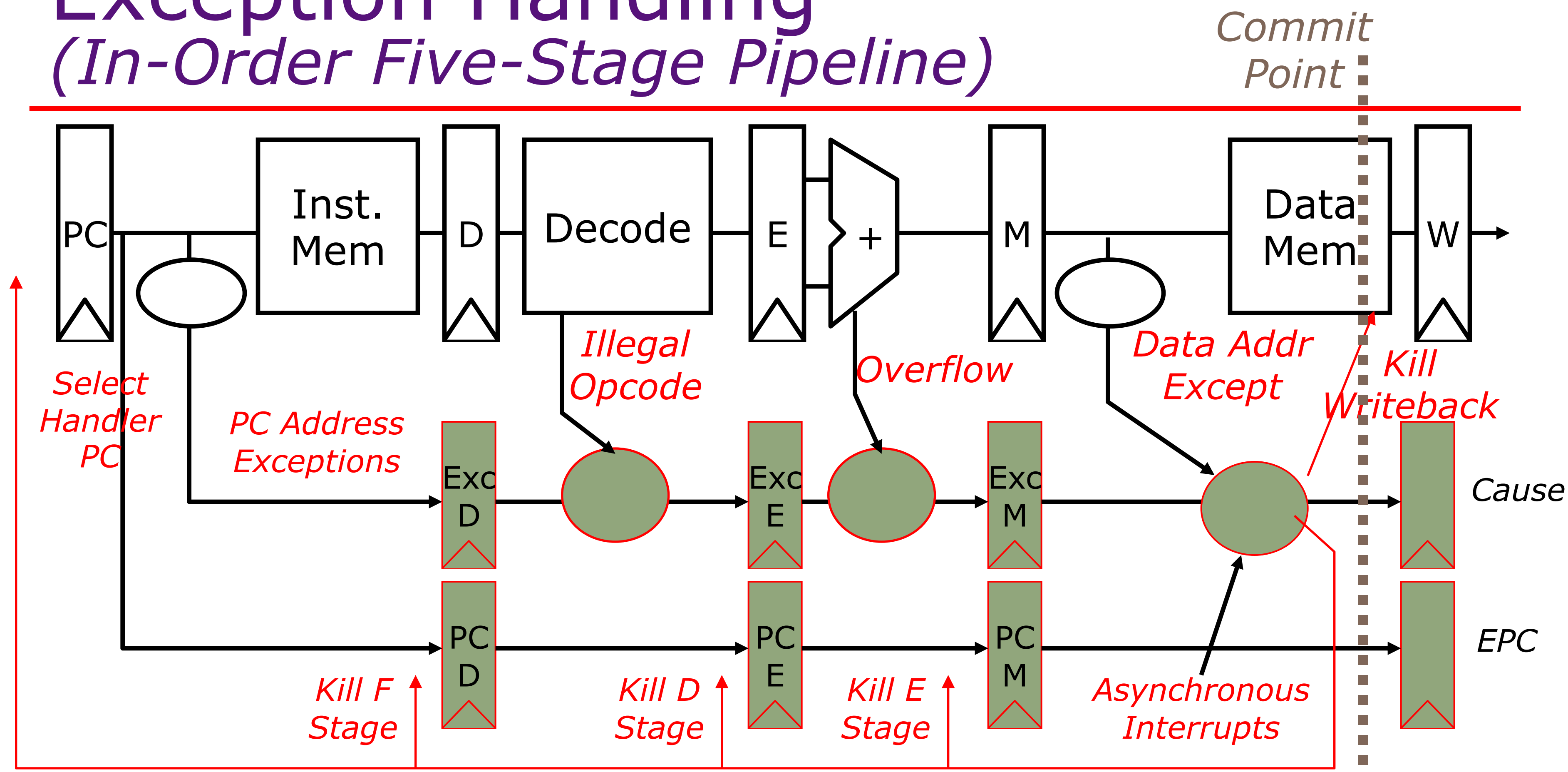


Strategy for Registers? Lazy – update at commit

Where are 'new' values?

Strategy for PC?

Exception Handling
(In-Order Five-Stage Pipeline)



Strategy for Registers?

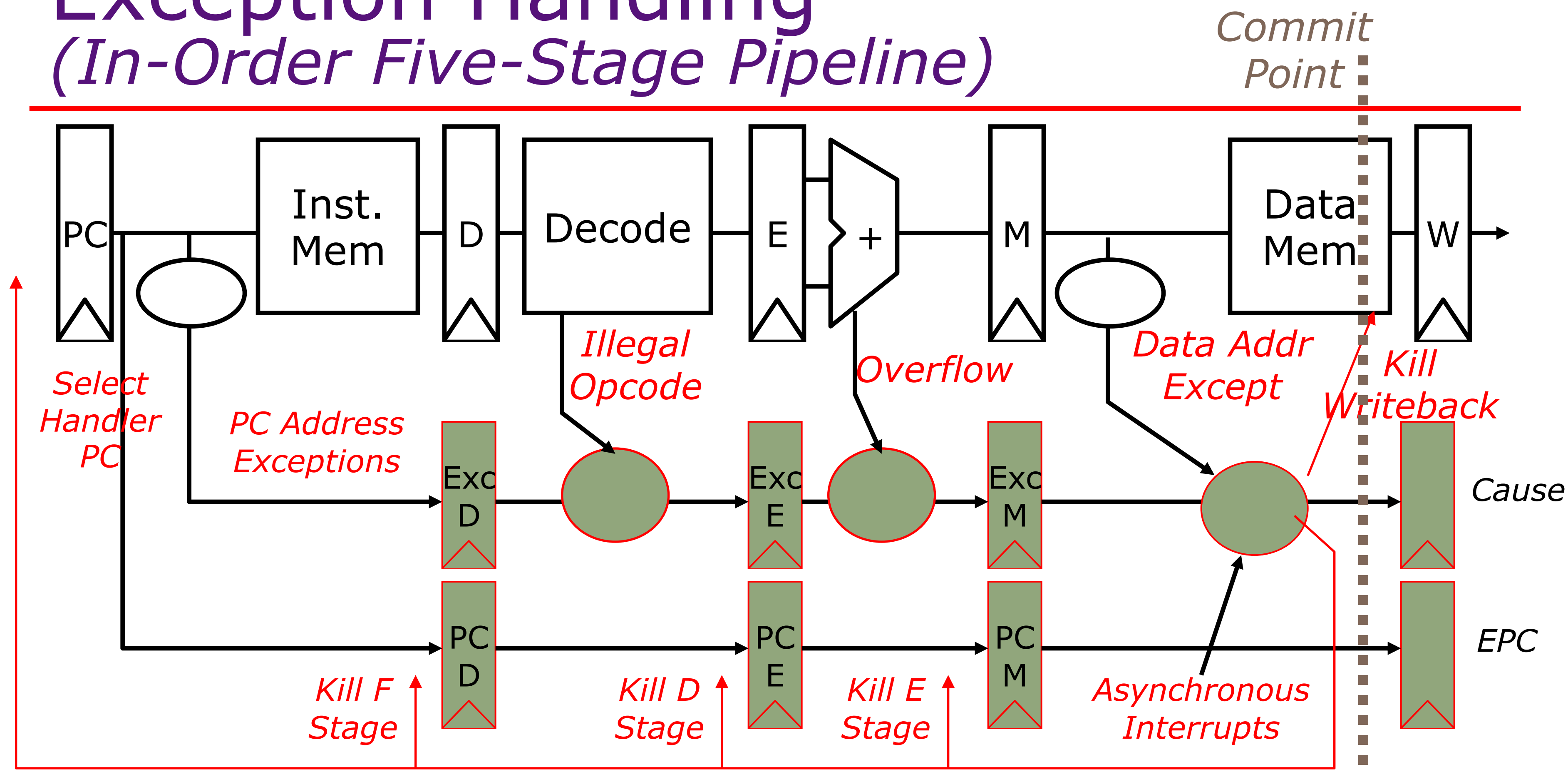
Where are 'new' values?

Strategy for PC?

Lazy – update at commit

In execution pipeline

Exception Handling
(In-Order Five-Stage Pipeline)

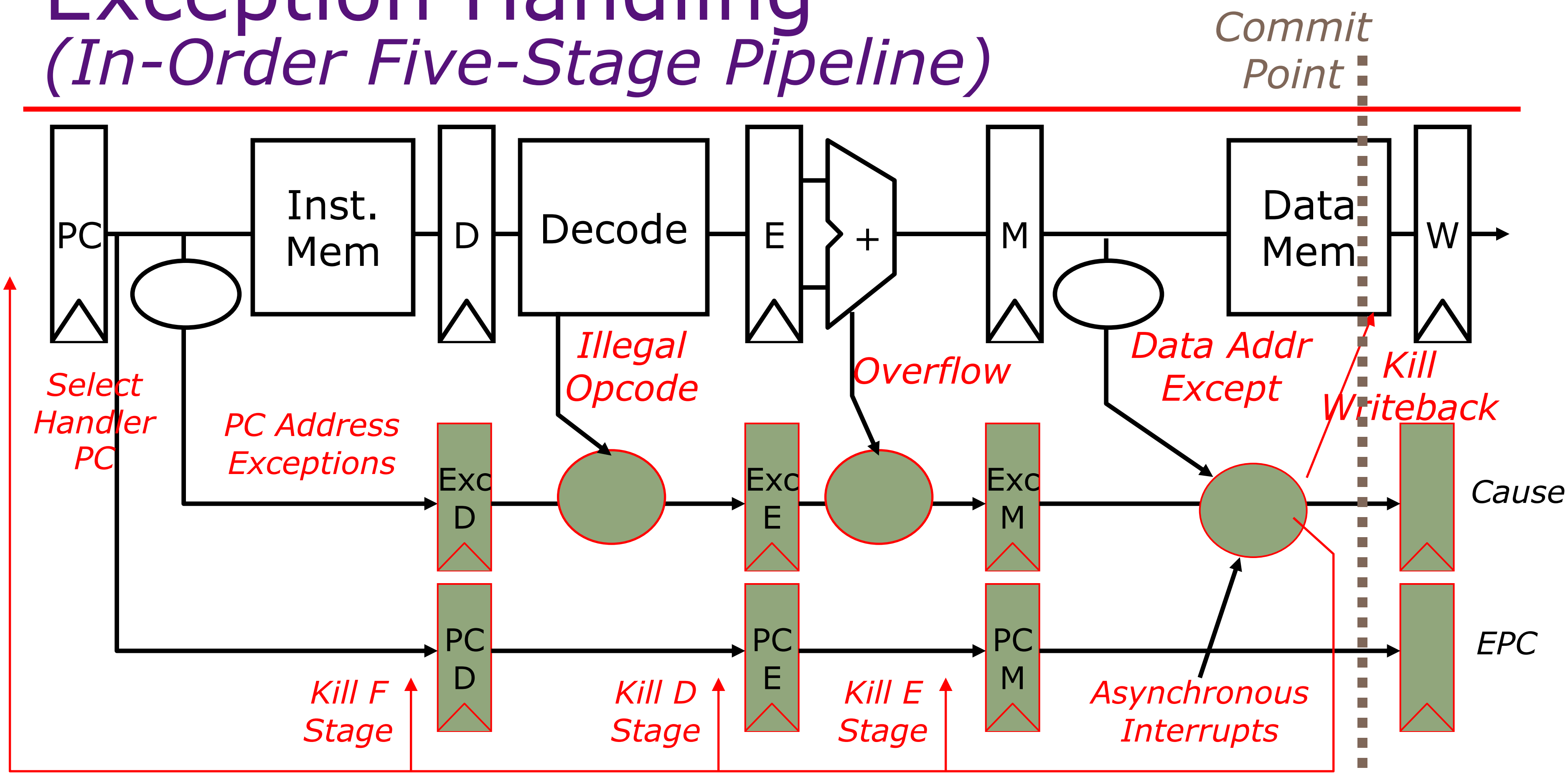


Strategy for Registers? Lazy – update at commit

Where are 'new' values? In execution pipeline

Strategy for PC? Greedy – update immediately

Exception Handling
(In-Order Five-Stage Pipeline)



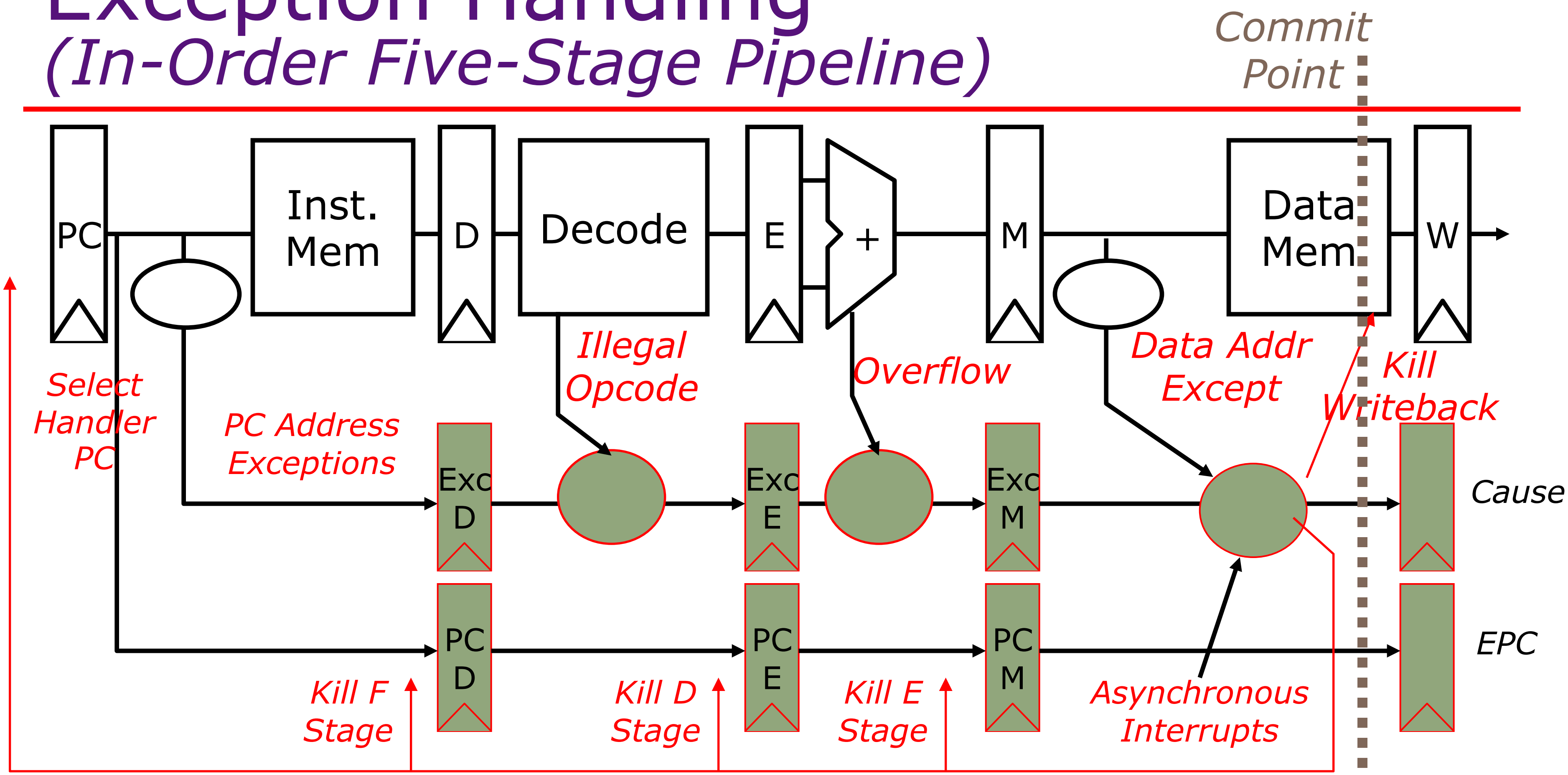
Strategy for Registers?
Where are 'new' values?

Lazy – update at commit
In execution pipeline

Strategy for PC?
Where is 'log'?

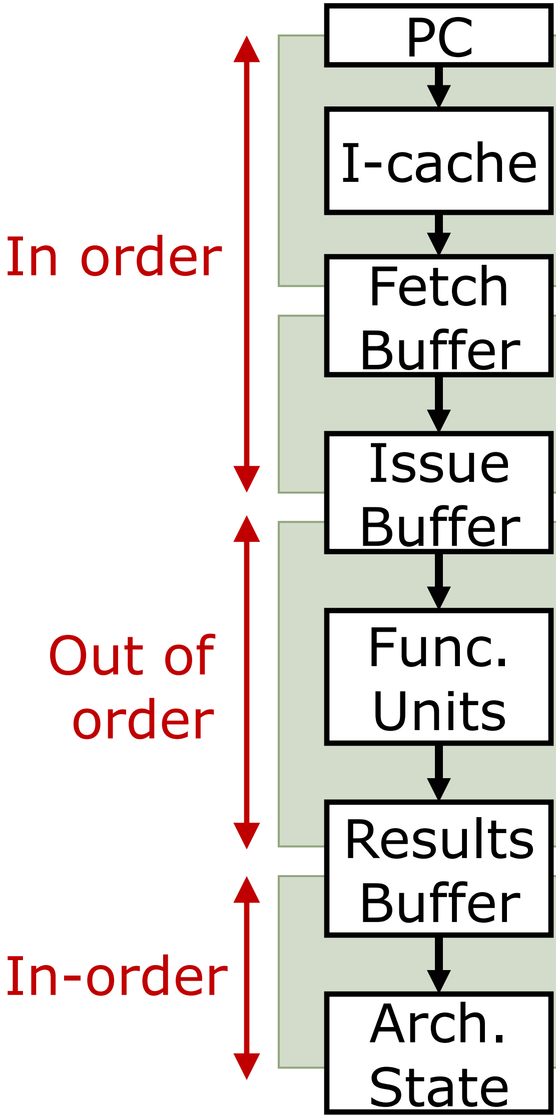
Greedy – update immediately

Exception Handling
(In-Order Five-Stage Pipeline)



Strategy for Registers?	Lazy – update at commit
Where are 'new' values?	In execution pipeline
Strategy for PC?	Greedy – update immediately
Where is 'log'?	In pipeline of PC latches

Phases of Instruction Execution



Fetch: Instruction bits retrieved from cache.

Decode: Instructions placed in appropriate issue (aka "dispatch") stage buffer

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

Commit: Instruction irrevocably updates architectural state (aka "graduation" or "completion").

Misprediction Recovery

In-order execution machines:

- Guarantee no instruction issued after branch can write-back before branch resolves by keeping values in the pipeline

- Kill all values from all instructions in pipeline behind mispredicted branch

Misprediction Recovery

In-order execution machines:

- Guarantee no instruction issued after branch can write-back before branch resolves by keeping values in the pipeline

- Kill all values from all instructions in pipeline behind mispredicted branch

Out-of-order execution?

Misprediction Recovery

In-order execution machines:

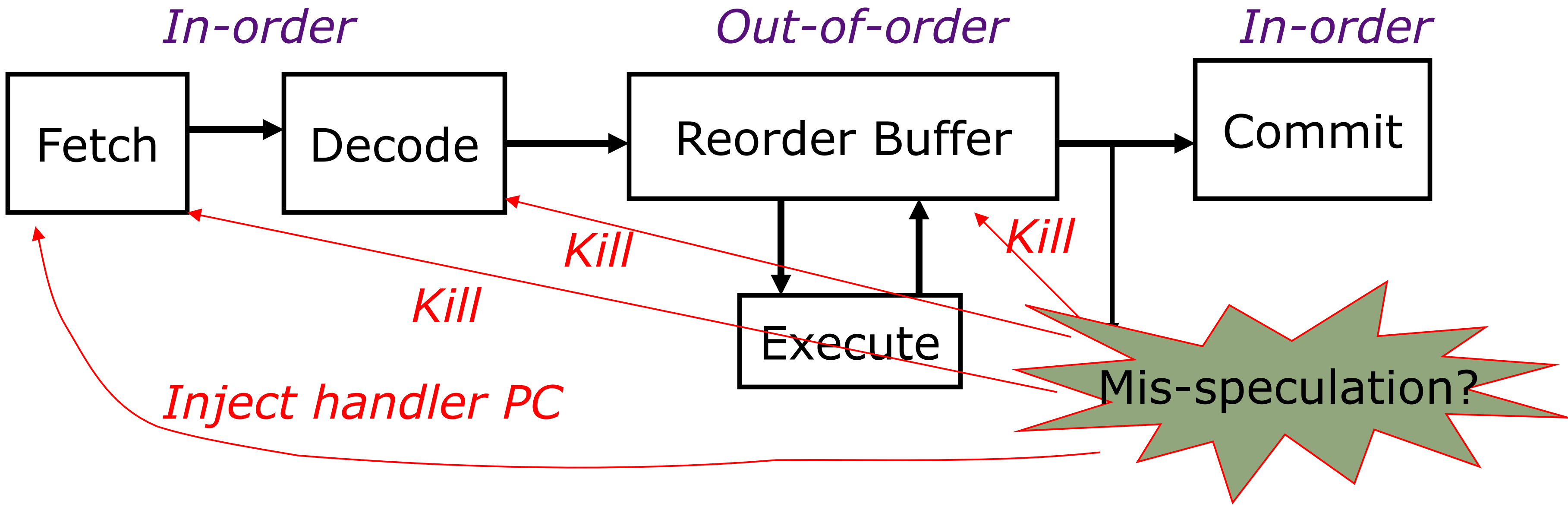
- Guarantee no instruction issued after branch can write-back before branch resolves by keeping values in the pipeline

- Kill all values from all instructions in pipeline behind mispredicted branch

Out-of-order execution?

- Multiple instructions following exception in program order can generate new values before exception resolves

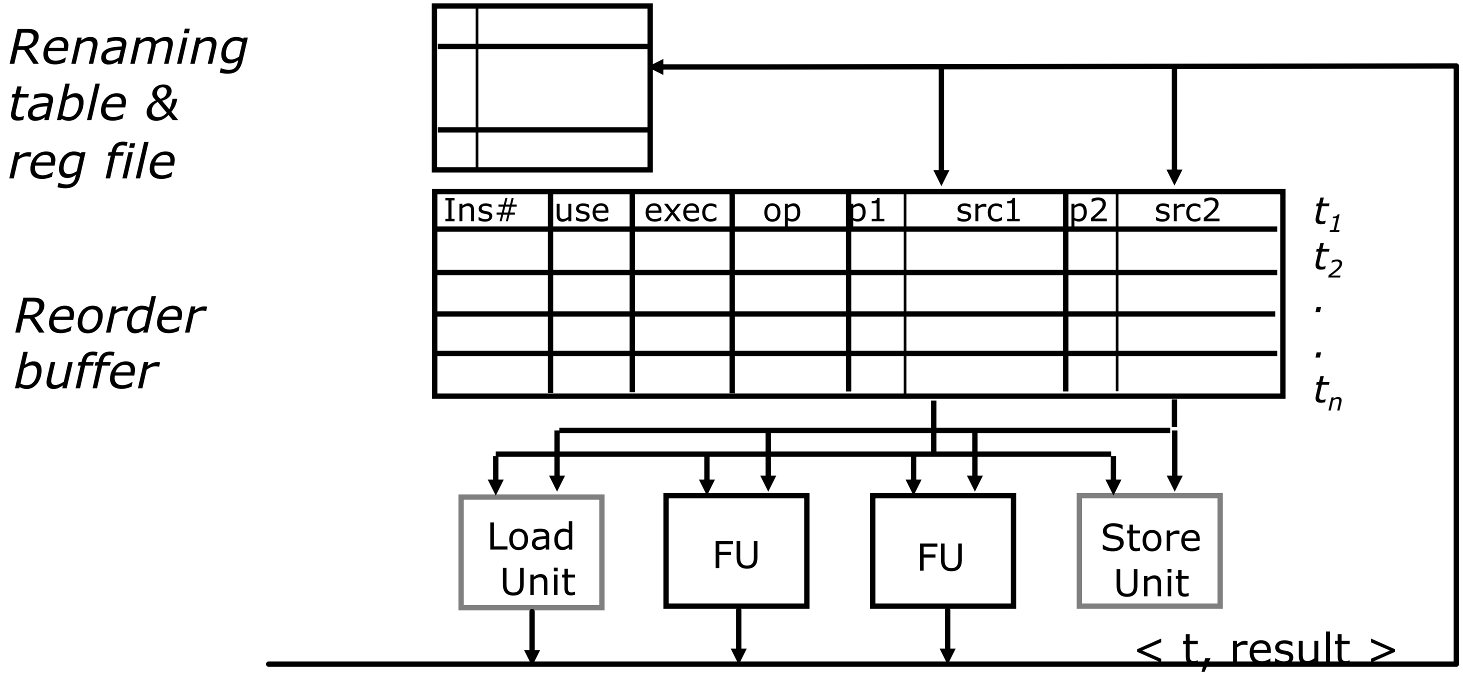
In-Order Commit for Mis-speculation Recovery



- Instructions fetched and decoded into instruction reorder buffer in-order
- Execution is out-of-order ($\Rightarrow$ 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)

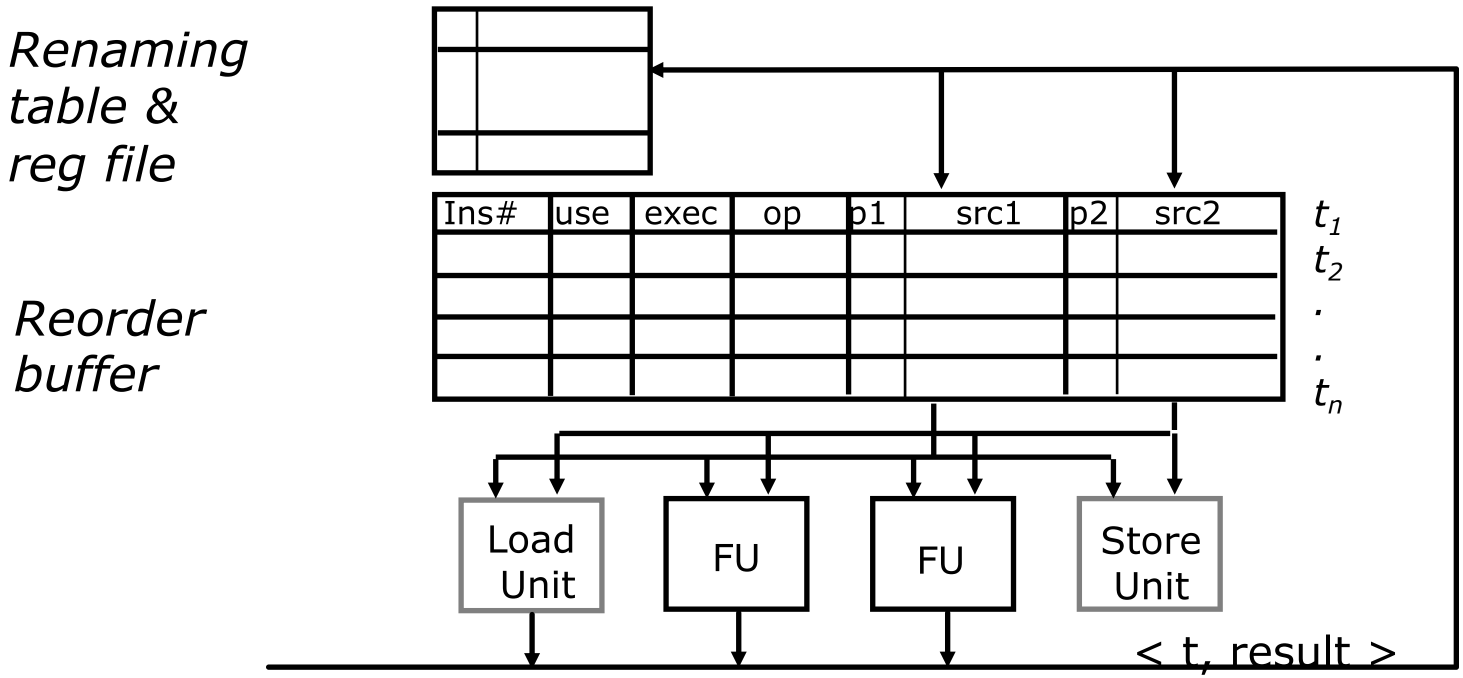
Data-Driven Execution



Basic Operation:

- Enter op and tag or data (if known) for each source
- Replace tag with data as it becomes available
- Issue instruction when all sources are available
- Save dest data when operation finishes

Data-Driven Execution

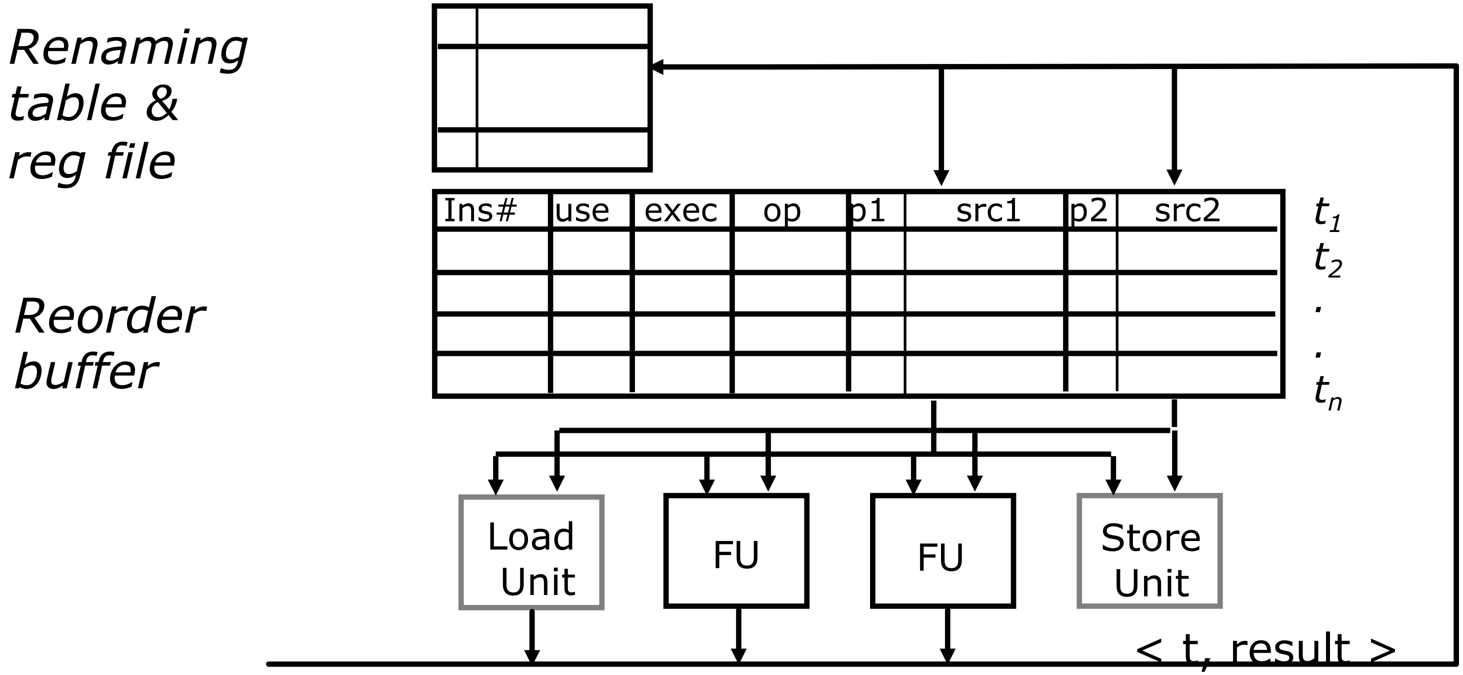


Basic Operation:

- Enter op and tag or data (if known) for each source
- Replace tag with data as it becomes available
- Issue instruction when all sources are available
- Save dest data when operation finishes

Update strategy?

Data-Driven Execution

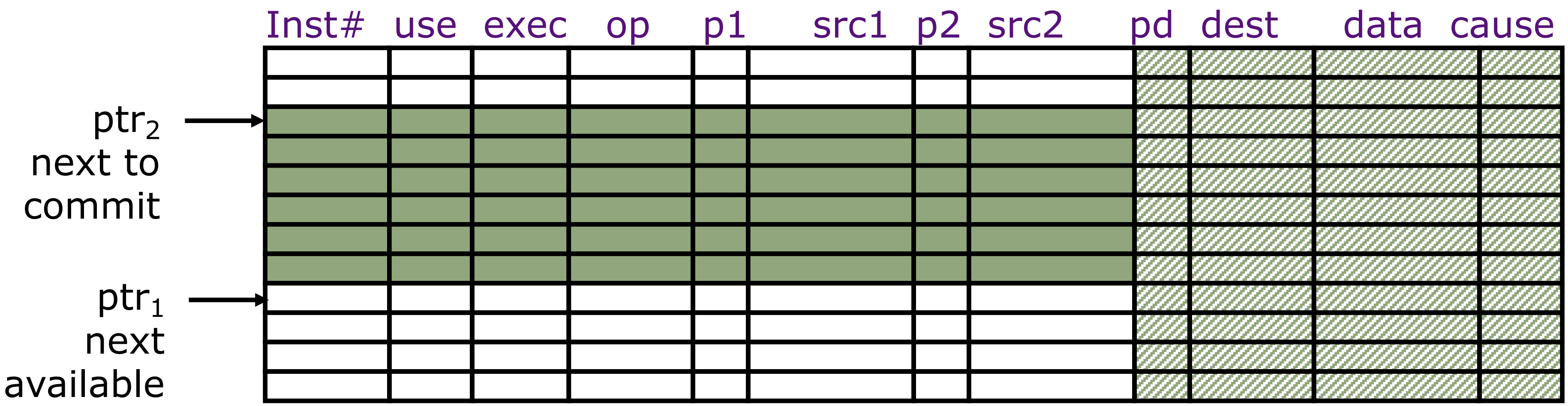


Basic Operation:

- Enter op and tag or data (if known) for each source
- Replace tag with data as it becomes available
- Issue instruction when all sources are available
- Save dest data when operation finishes

Update strategy? Greedy – update at execute

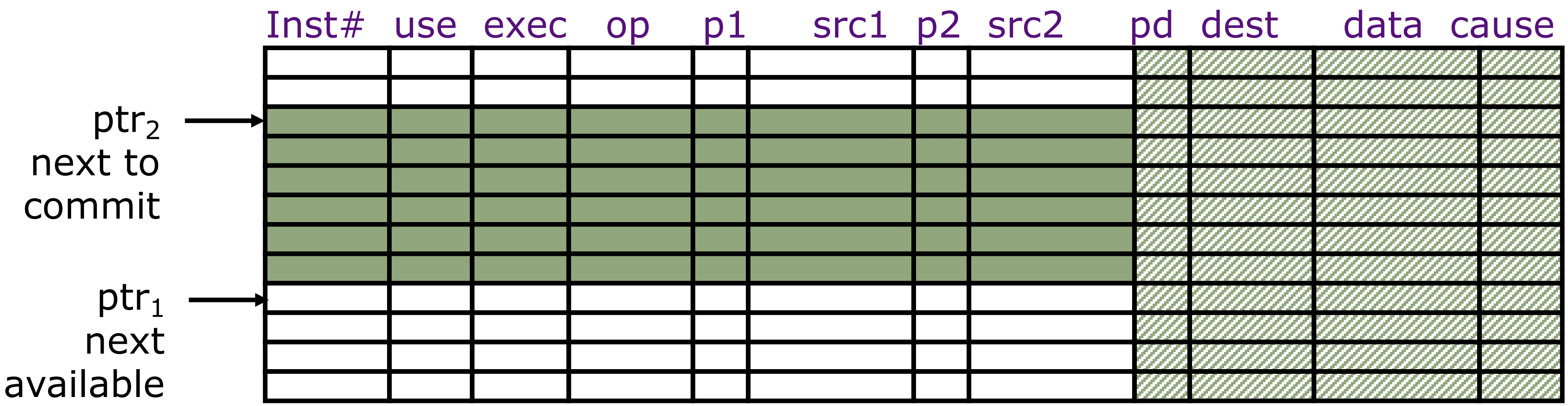
Extensions for Mis-speculation Recovery



Reorder buffer

- 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)

Extensions for Mis-speculation Recovery



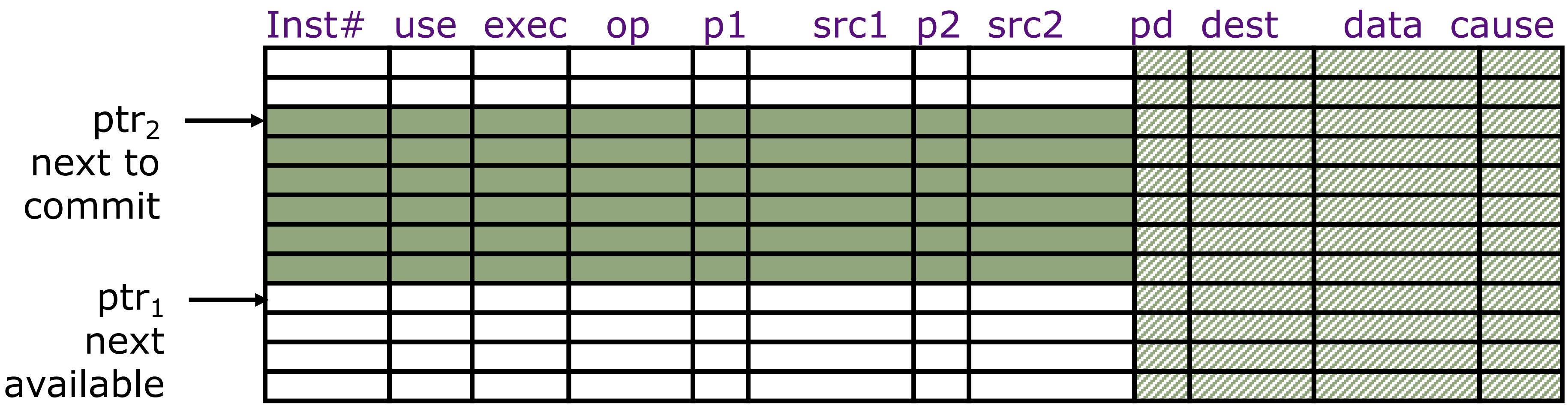
Reorder buffer

- 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)

What is the update policy of registers?

Extensions for Mis-speculation Recovery



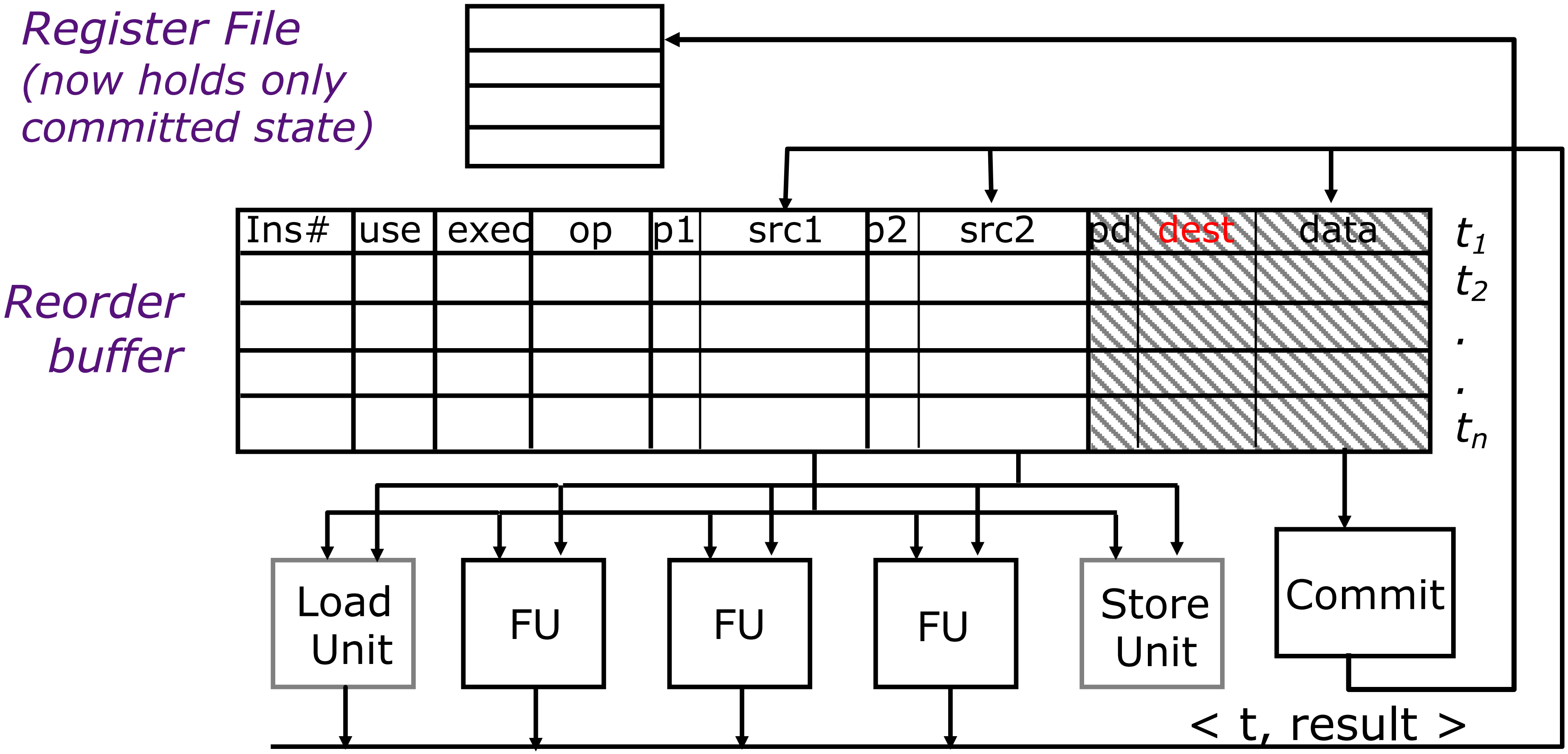
Reorder buffer

- 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)

What is the update policy of registers? *Lazy*

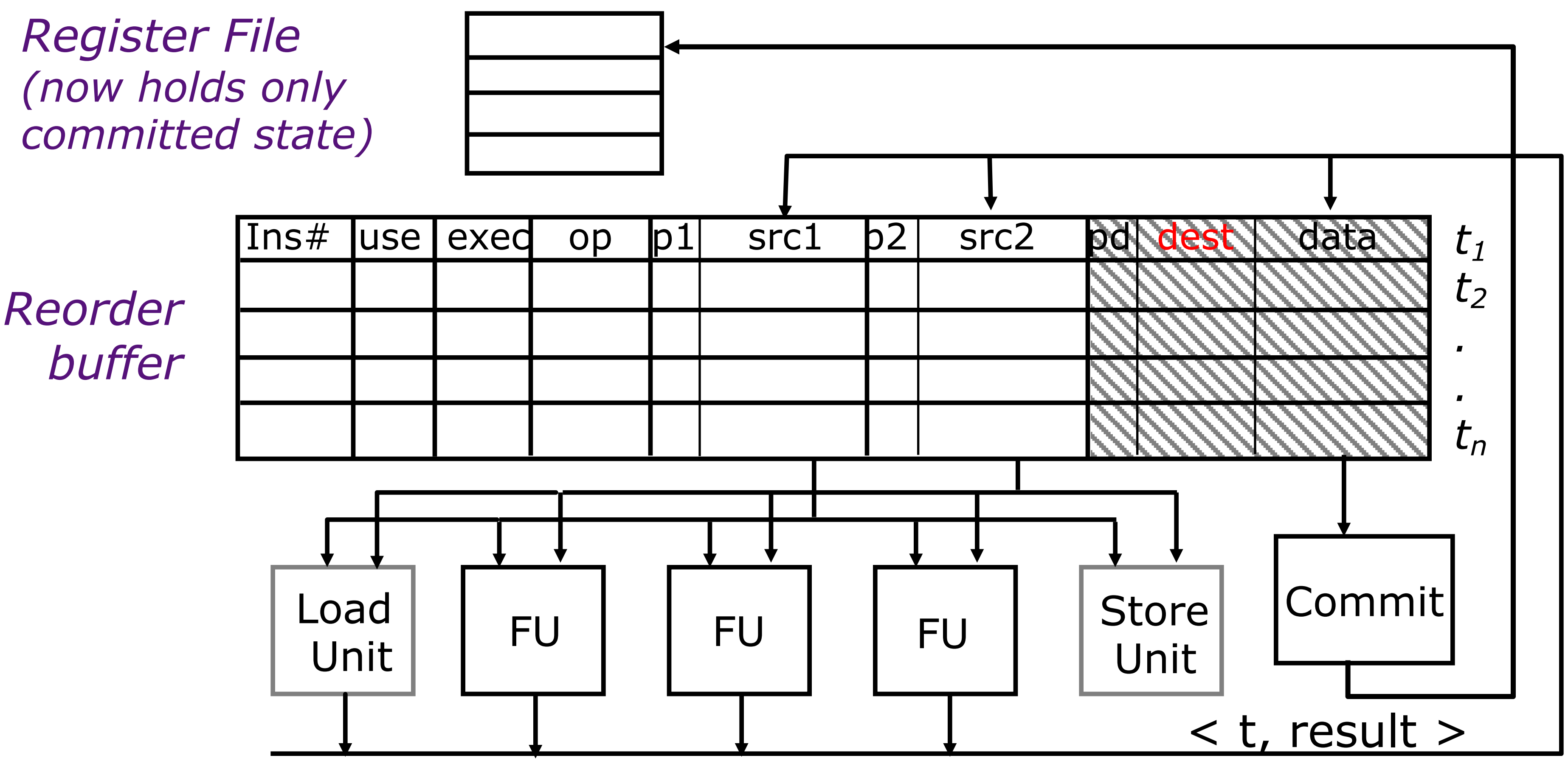
Rollback and Renaming



Convert to lazy by holding data in ROB.

But how do we find values before they are committed?

Rollback and Renaming



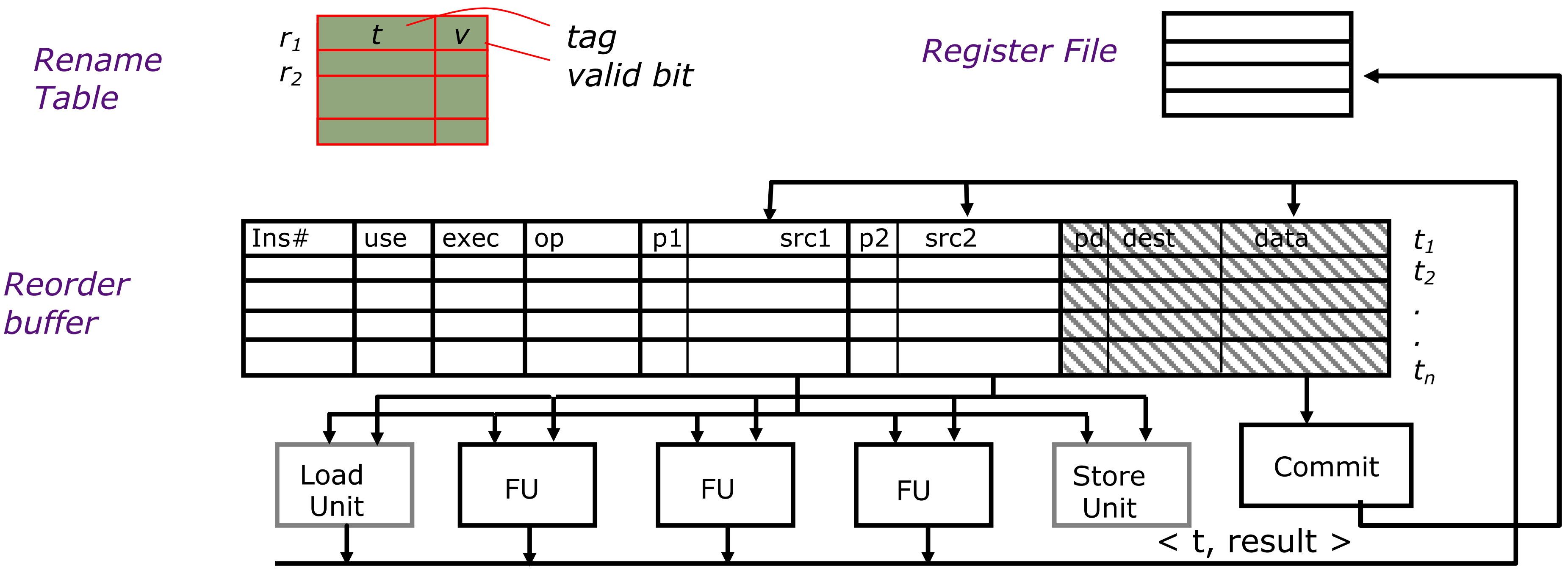
Convert to lazy by holding data in ROB.

But how do we find values before they are committed?

Search the "dest" field in the reorder buffer

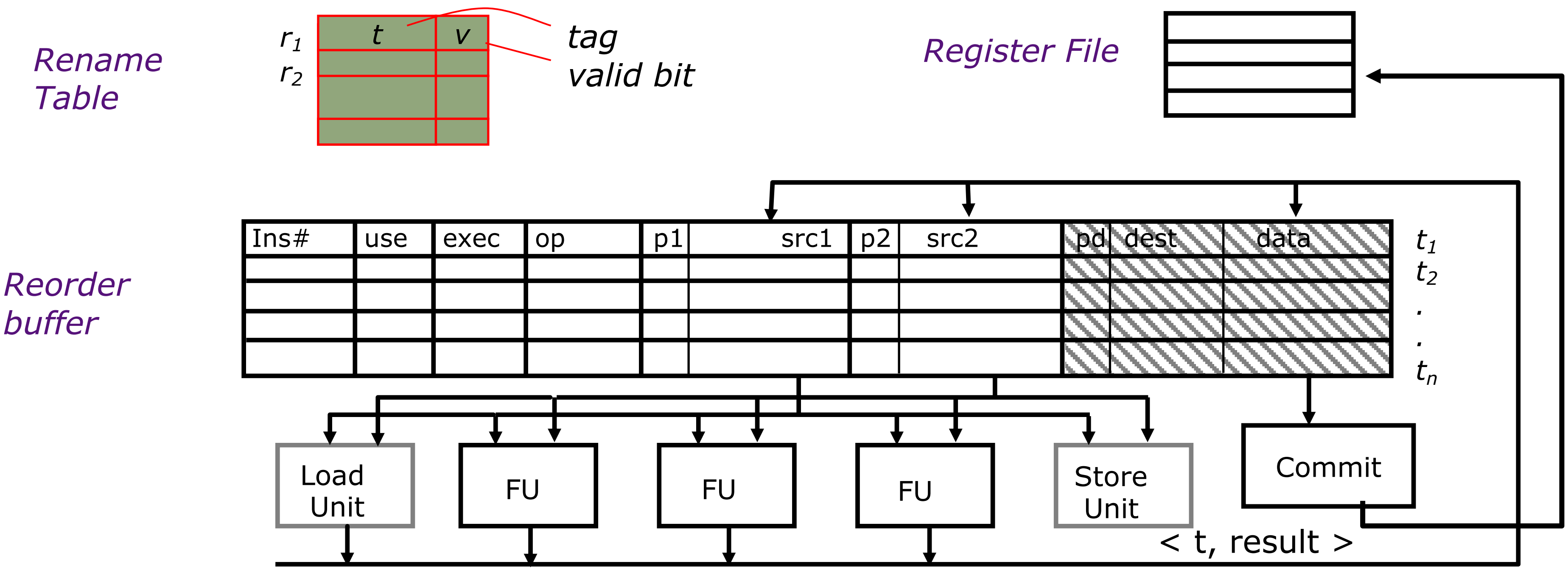
Renaming Table

Micro-architectural speculative cache to speed up tag look up.



Renaming Table

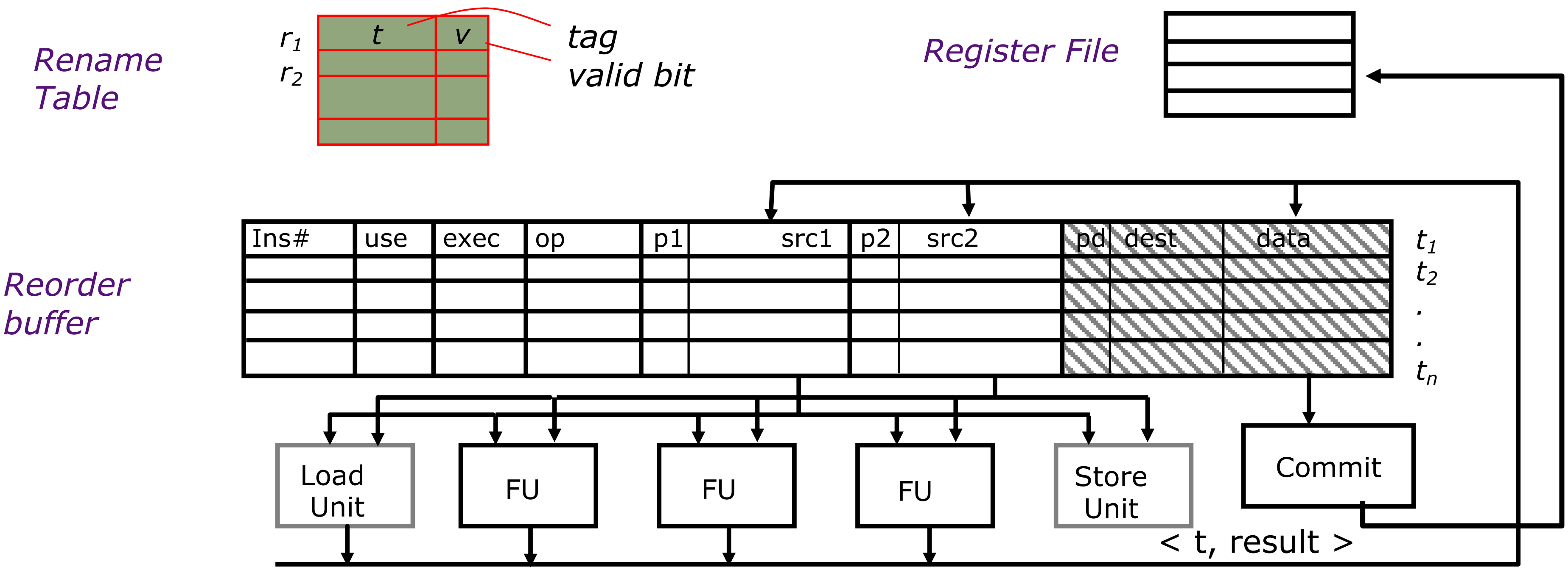
Micro-architectural speculative cache to speed up tag look up.



What is the update policy of rename table?

Renaming Table

Micro-architectural speculative cache to speed up tag look up.

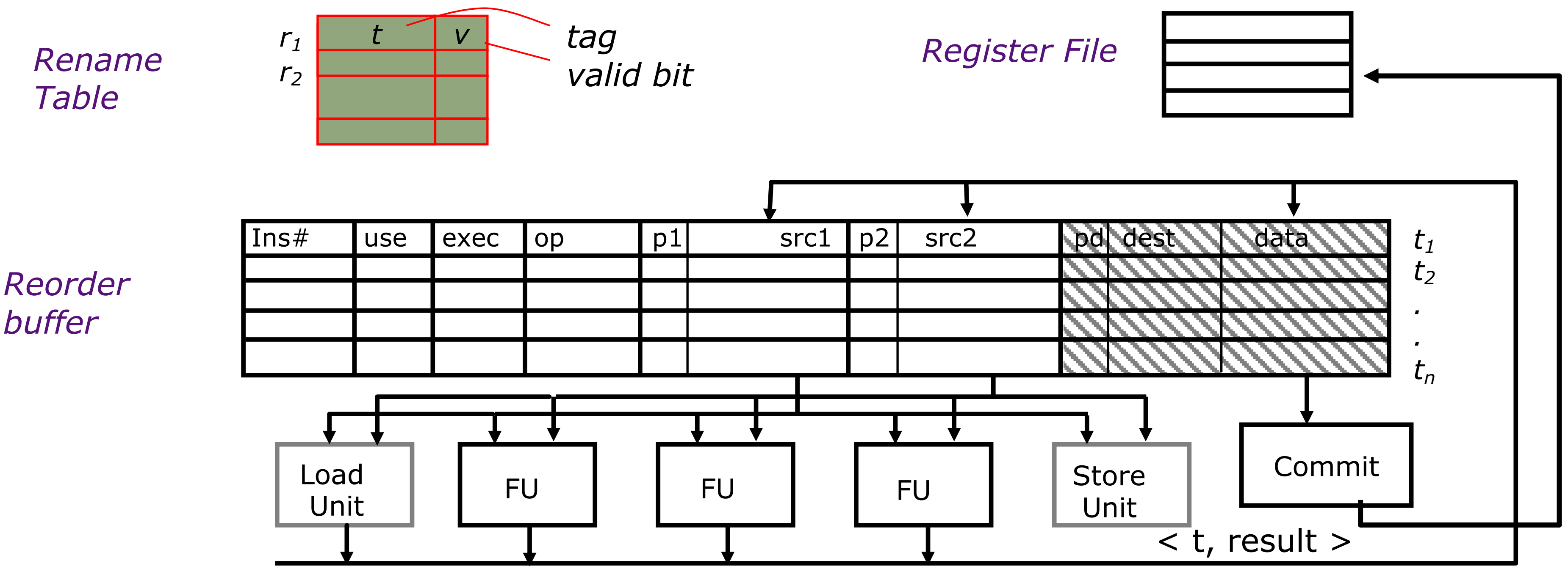


What is the update policy of rename table?

Greedy

Renaming Table

Micro-architectural speculative cache to speed up tag look up.

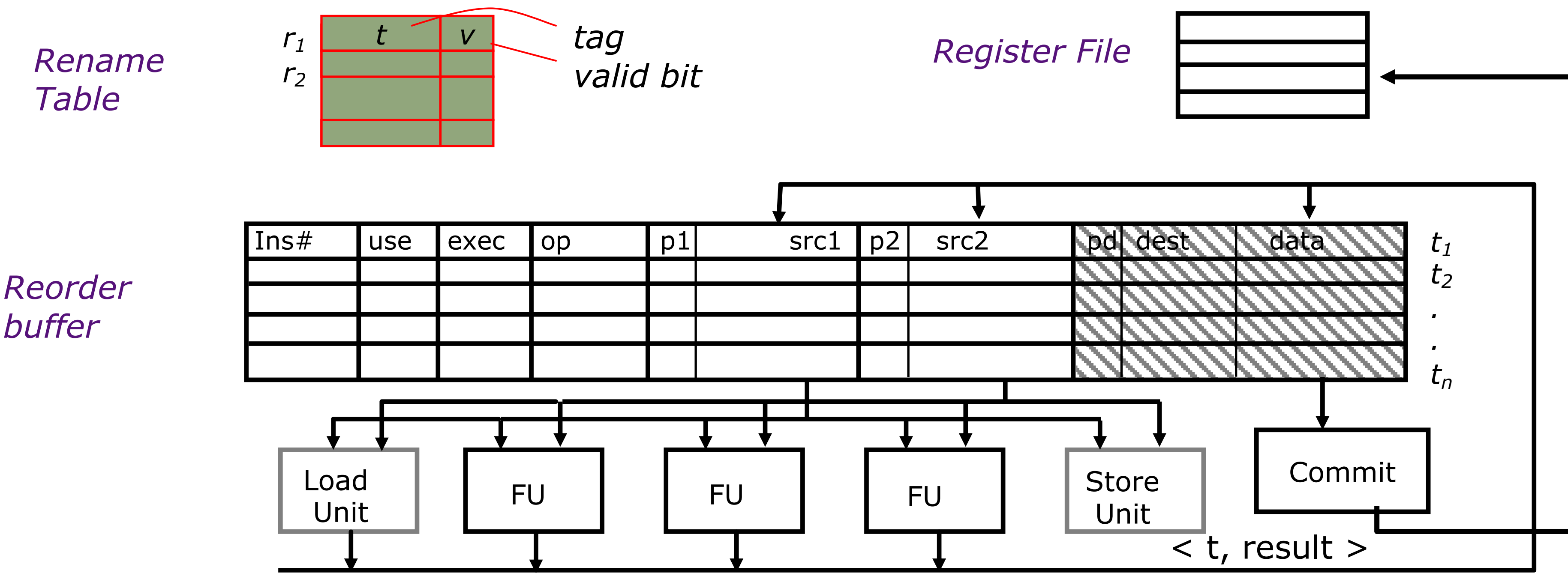


What is the update policy of rename table?
What events cause mis-speculation?

Greedy

Renaming Table

Micro-architectural speculative cache to speed up tag look up.

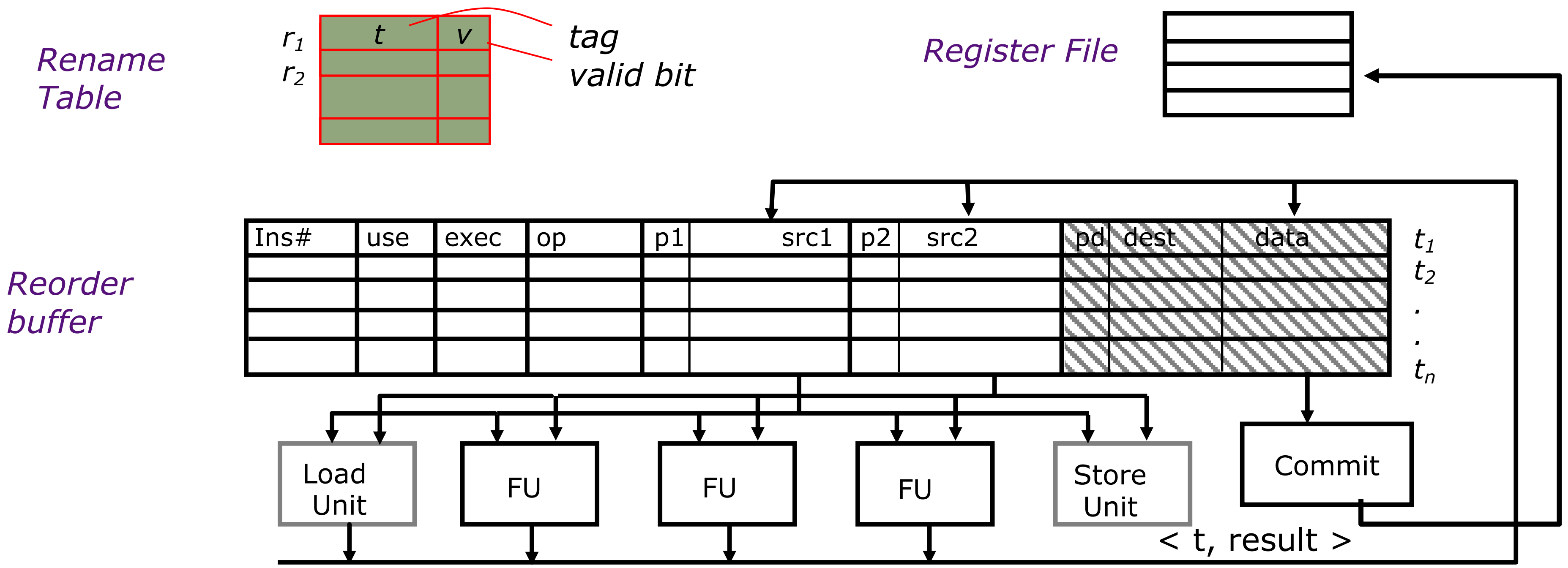


What is the update policy of rename table? *Greedy*

What events cause mis-speculation? *Exceptions & branch mispredicts*

Renaming Table

Micro-architectural speculative cache to speed up tag look up.



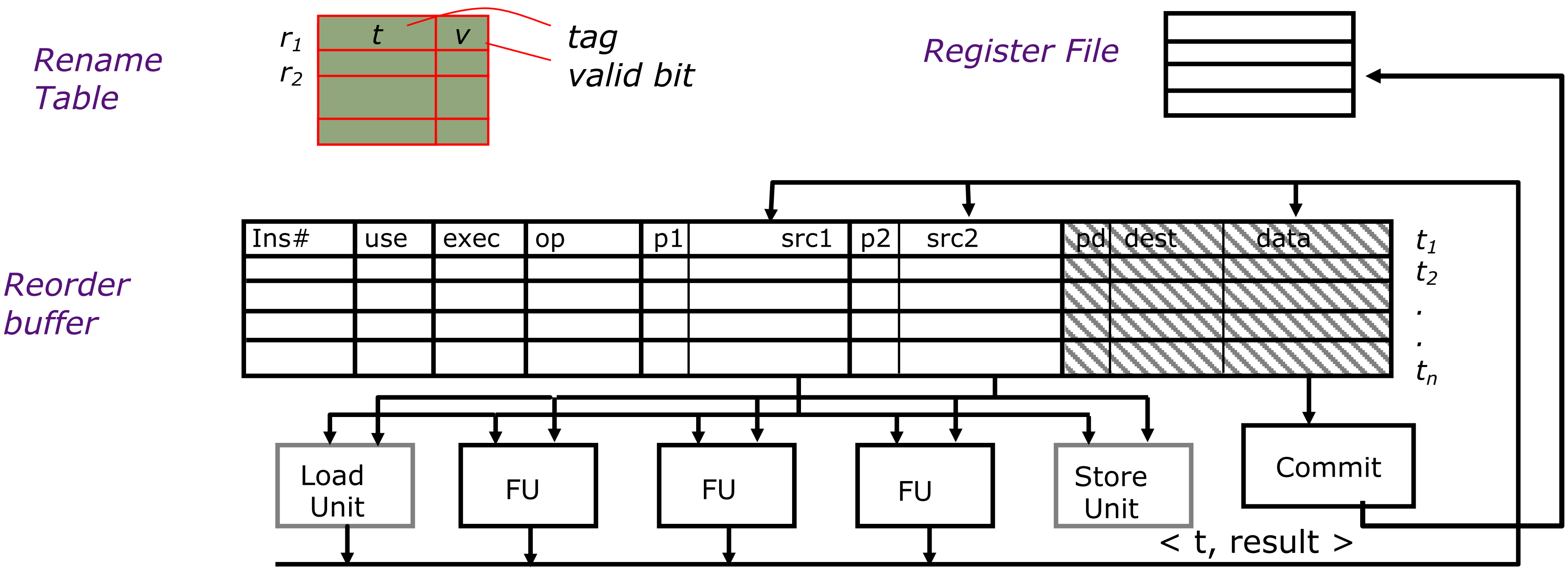
What is the update policy of rename table? *Greedy*

What events cause mis-speculation? *Exceptions & branch mispredicts*

How can we respond to mis-speculation?

Renaming Table

Micro-architectural speculative cache to speed up tag look up.



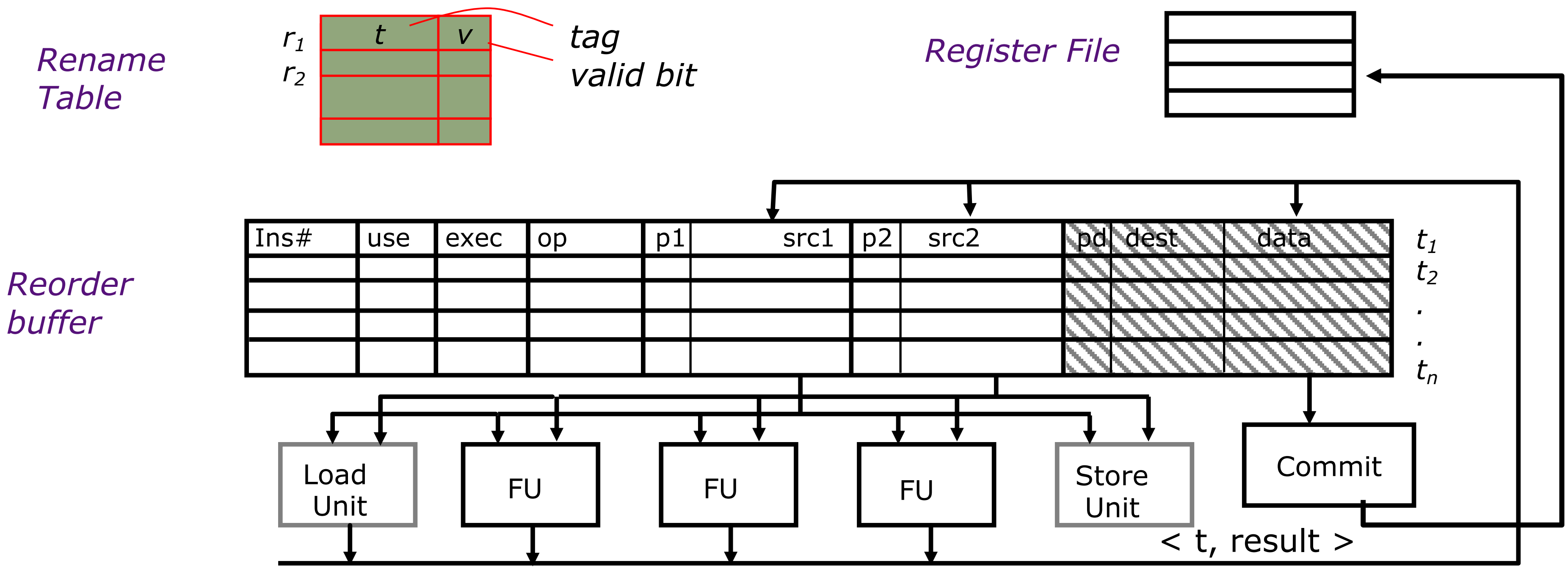
What is the update policy of rename table? *Greedy*

What events cause mis-speculation? *Exceptions & branch mispredicts*

How can we respond to mis-speculation? *Clear valid bits*

Renaming Table

Micro-architectural speculative cache to speed up tag look up.



What is the update policy of rename table? *Greedy*

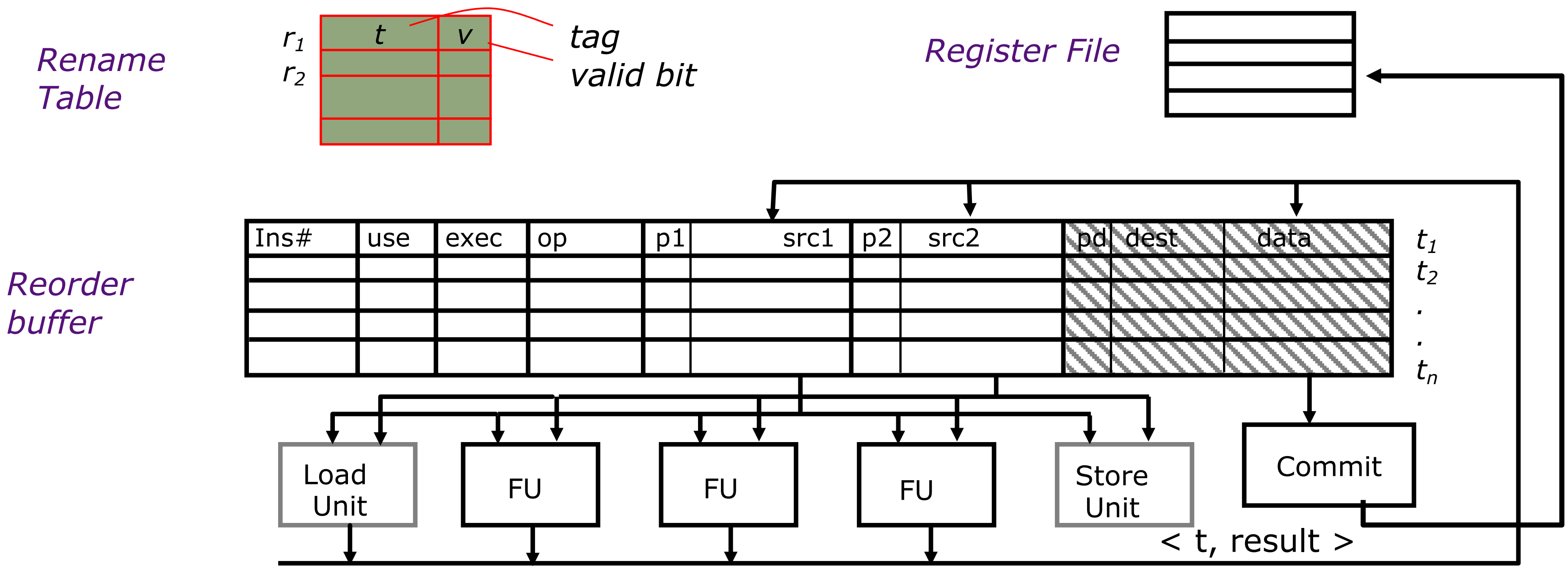
What events cause mis-speculation? *Exceptions & branch mispredicts*

How can we respond to mis-speculation? *Clear valid bits*

After being cleared, when can instructions be added to ROB?

Renaming Table

Micro-architectural speculative cache to speed up tag look up.



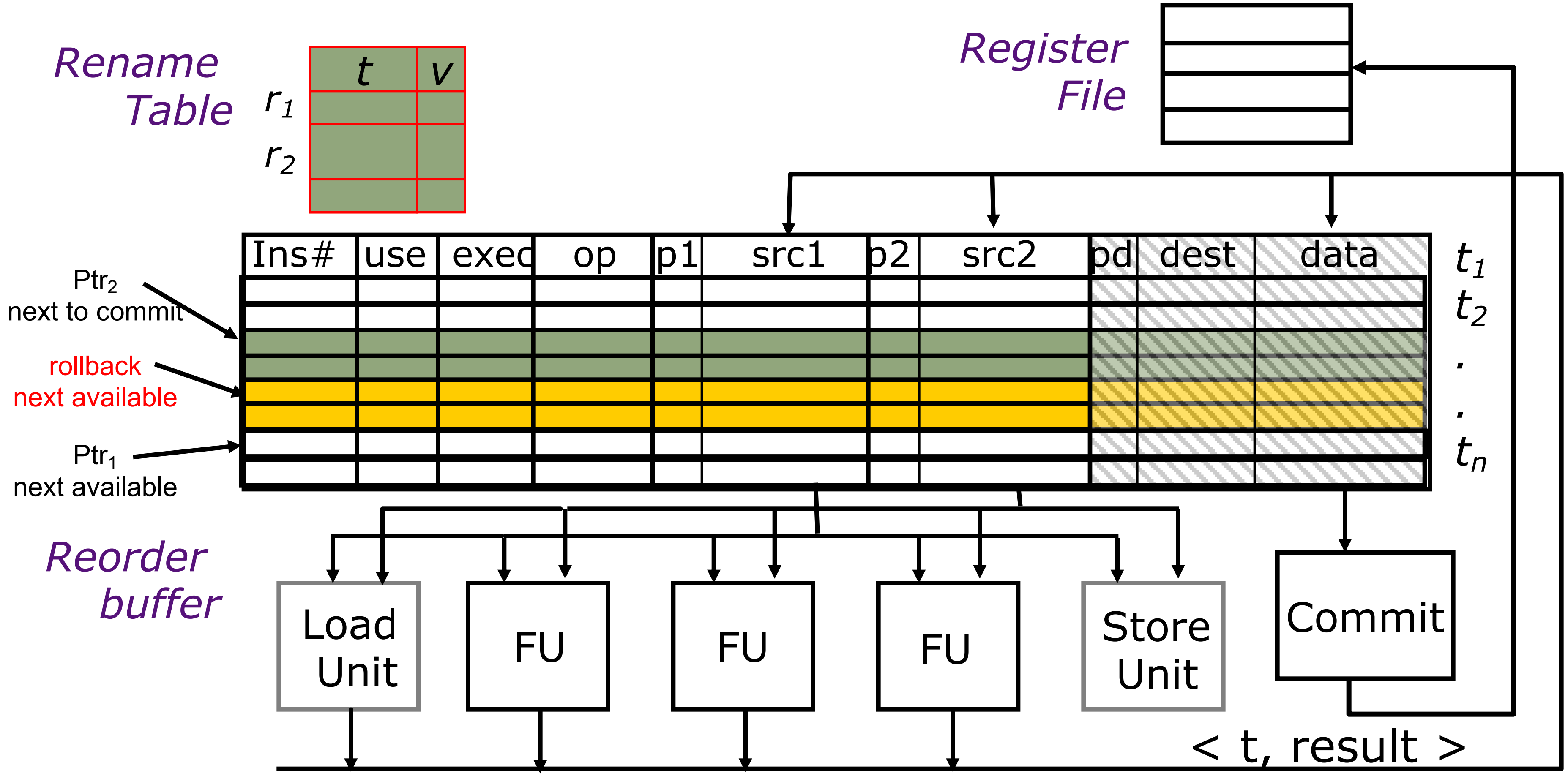
What is the update policy of rename table? *Greedy*

What events cause mis-speculation? *Exceptions & branch mispredicts*

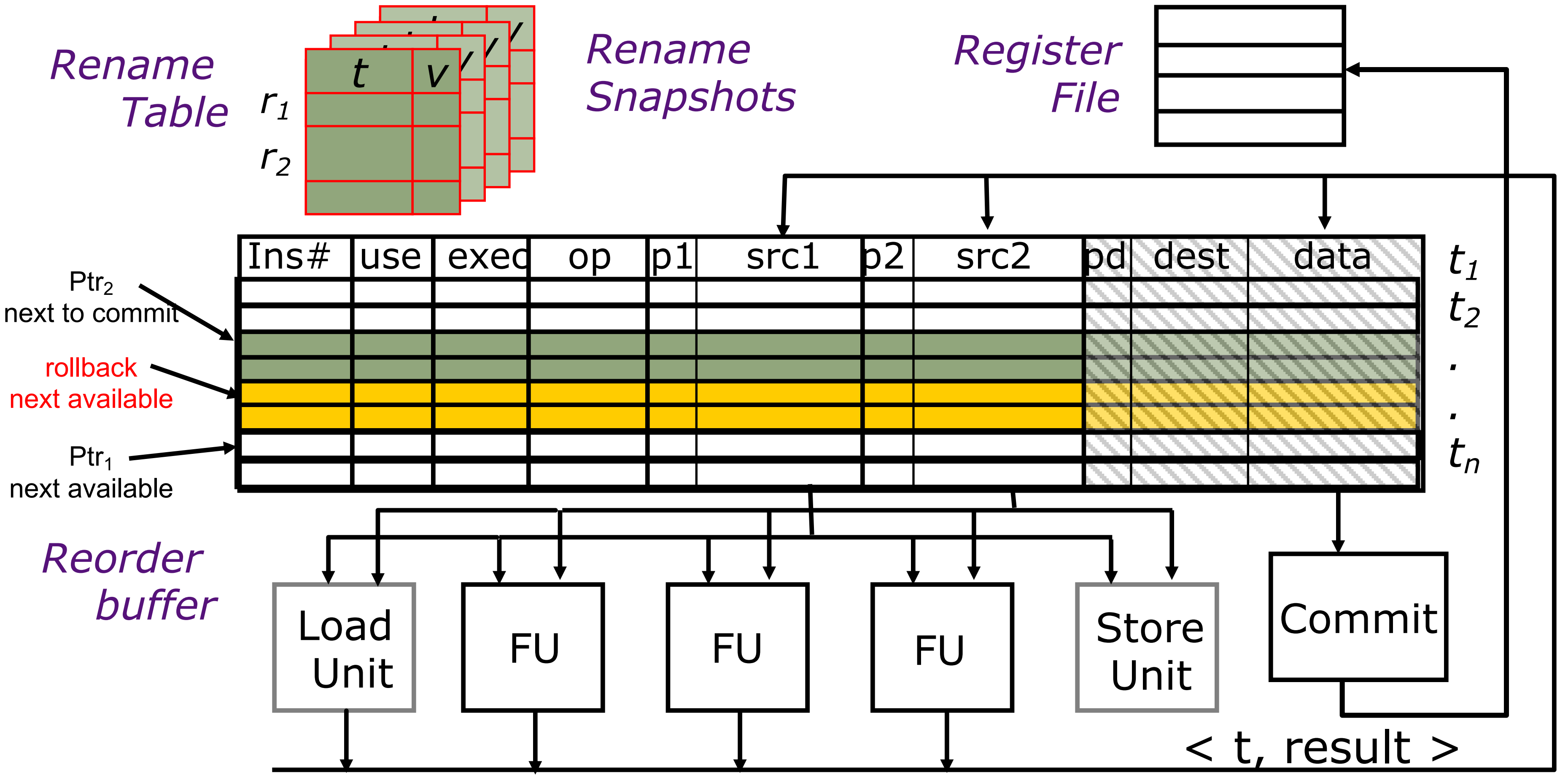
How can we respond to mis-speculation? *Clear valid bits*

After being cleared, when can instructions be added to ROB? *After drain*

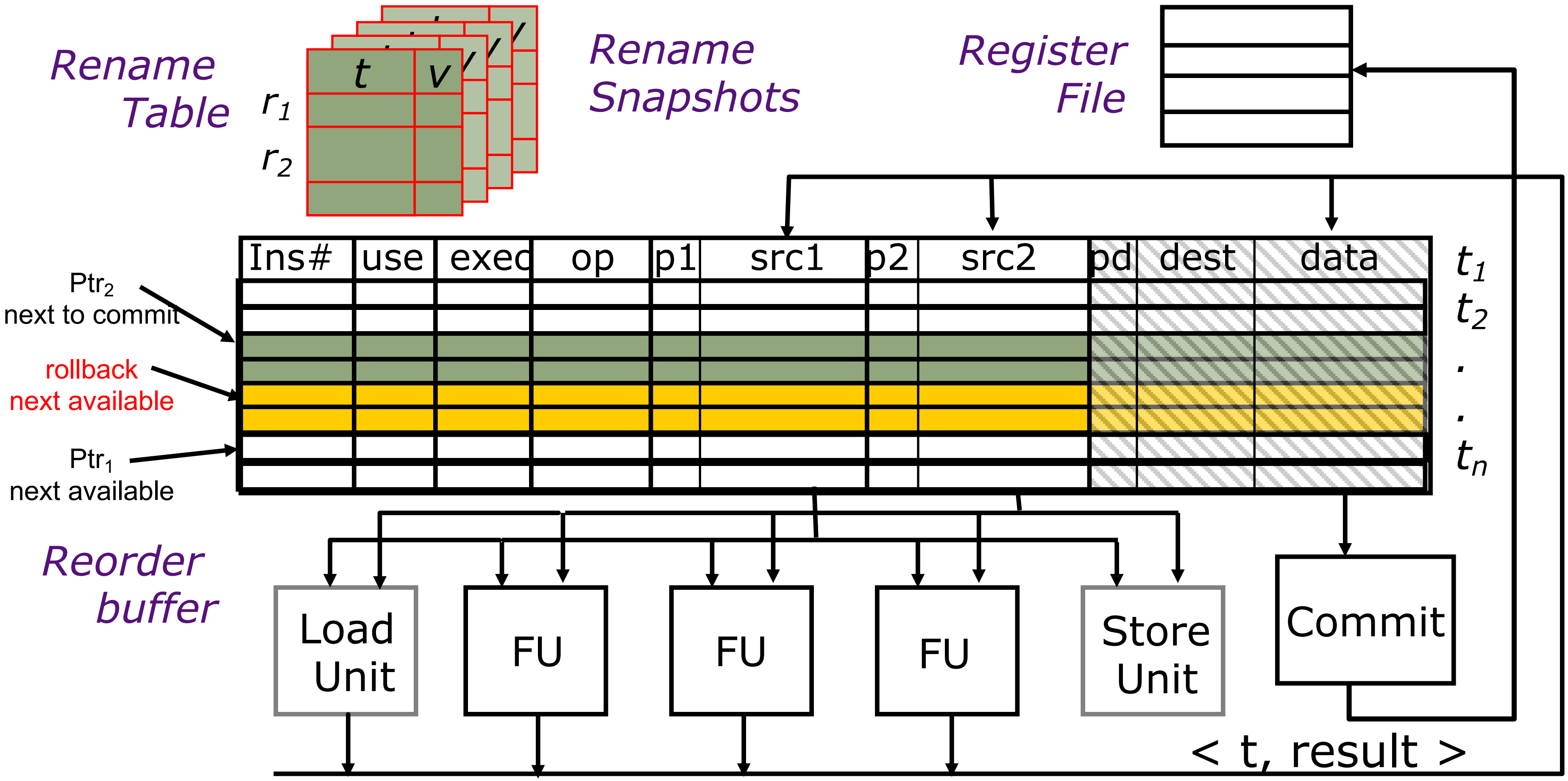
Recovering ROB/Renaming Table



Recovering ROB/Renaming Table



Recovering ROB/Renaming Table



Take snapshot of register rename table at each predicted branch, recover earlier snapshot if branch mispredicted

Map Table Recovery - Snapshots

Speculative value management of microarchitectural state

	Reg Map	V
R0	T20	X
R1	T08	
R2	T45	X
R3	T128	X
	⋮	
R30	T54	
R31	T88	X

Map Table Recovery - Snapshots

Speculative value management of microarchitectural state

	Reg Map	V
R0	T20	X
R1	T08	
R2	T45	X
R3	T128	X
	• • •	
R30	T54	
R31	T88	X

Snap Map	V
T20	X
T08	
T45	X
T128	X
• • •	
T54	
T88	X

Map Table Recovery - Snapshots

Speculative value management of microarchitectural state

	Reg Map	V
R0	T20	X
R1	T73	X
R2	T45	X
R3	T128	X
	⋮	
R30	T54	
R31	T88	X

Snap Map	V
T20	X
T08	
T45	X
T128	X
⋮	
T54	
T88	X

Map Table Recovery - Snapshots

Speculative value management of microarchitectural state

	Reg Map	V
R0	T20	X
R1	T73	X
R2	T45	X
R3	T128	
⋮	⋮	
R30	T54	
R31	T88	X

Snap Map	V
T20	X
T08	
T45	X
T128	X
⋮	
T54	
T88	X

Map Table Recovery - Snapshots

Speculative value management of microarchitectural state

	Reg Map	V	Snap Map	V	Snap Map	V
R0	T20	X	T20	X	T20	X
R1	T73	X	T73	X	T08	
R2	T45	X	T45	X	T45	X
R3	T128		T128		T128	X
	⋮		⋮		⋮	
R30	T54		T54		T54	
R31	T88	X	T88	X	T88	X

Map Table Recovery - Snapshots

Speculative value management of microarchitectural state

	Reg Map	V	Snap Map	V	Snap Map	V
R0	T20	X	T20	X	T20	X
R1	T73	X	T73	X	T08	
R2	T45	X	T45	X	T45	X
R3	T128		T128		T128	X
	⋮		⋮		⋮	
R30	T54		T54		T54	
R31	T88	X	T88	X	T88	X

What kind of value management is this?

Map Table Recovery - Snapshots

Speculative value management of microarchitectural state

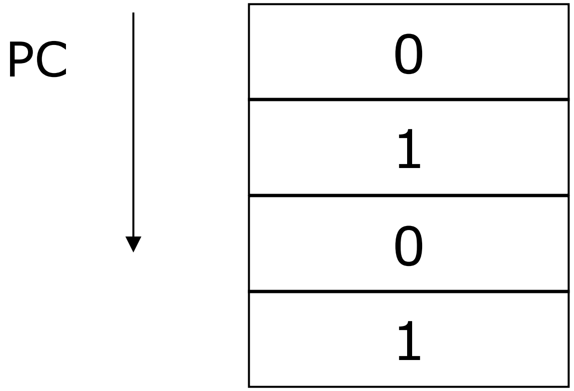
	Reg Map	V	Snap Map	V	Snap Map	V
R0	T20	X	T20	X	T20	X
R1	T73	X	T73	X	T08	
R2	T45	X	T45	X	T45	X
R3	T128		T128		T128	X
	⋮		⋮		⋮	
R30	T54		T54		T54	
R31	T88	X	T88	X	T88	X

What kind of value management is this? Greedy!!

Branch Predictor Recovery

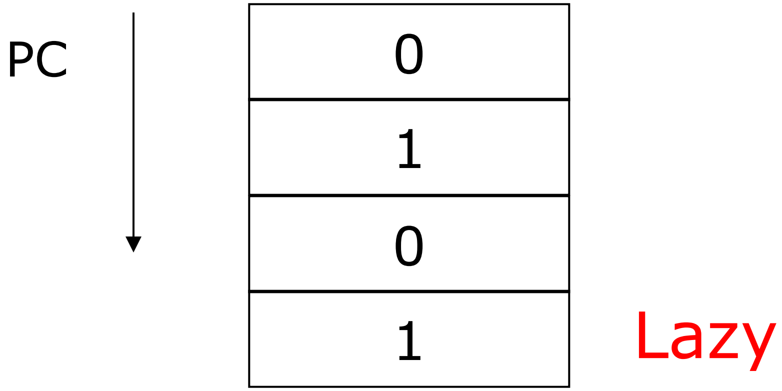
Branch Predictor Recovery

- 1-Bit Counter Recovery



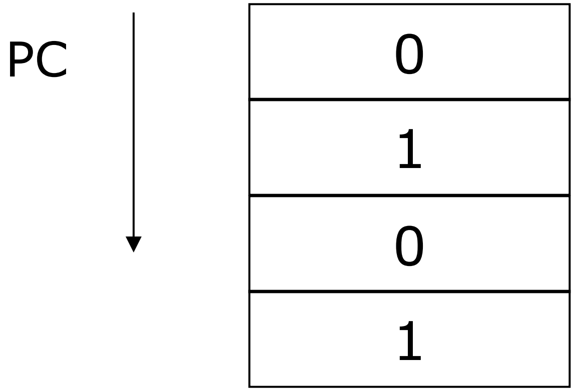
Branch Predictor Recovery

- 1-Bit Counter Recovery



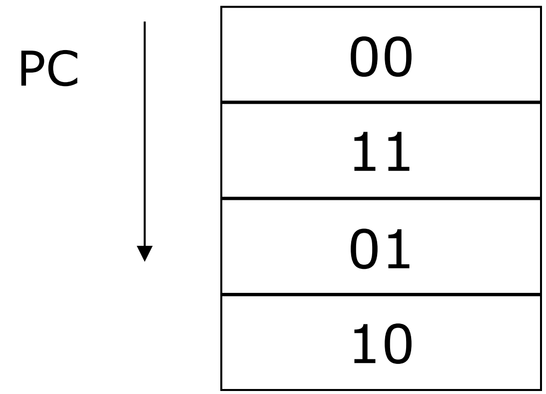
Branch Predictor Recovery

- 1-Bit Counter Recovery



Lazy

- 2-Bit Counter Recovery



Branch Predictor Recovery

- 1-Bit Counter Recovery

PC ↓

0
1
0
1

Lazy

- 2-Bit Counter Recovery

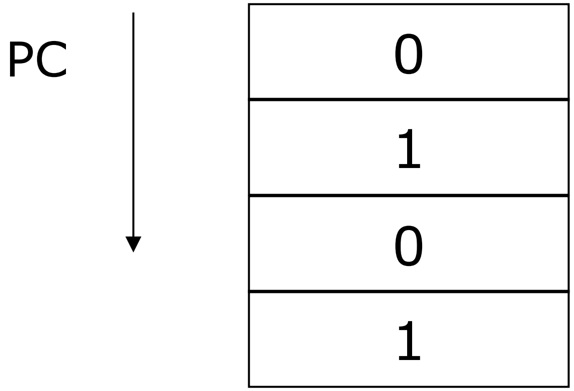
PC ↓

00
11
01
10

Lazy

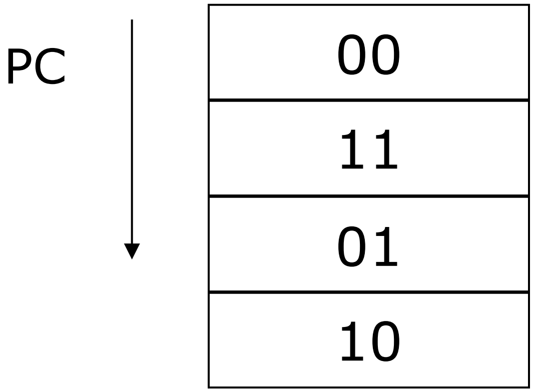
Branch Predictor Recovery

- 1-Bit Counter Recovery



Lazy

- 2-Bit Counter Recovery



Lazy

- Global History Recovery

10101010

Branch Predictor Recovery

- 1-Bit Counter Recovery

PC ↓

0
1
0
1

Lazy

- 2-Bit Counter Recovery

PC ↓

00
11
01
10

Lazy

- Global History Recovery

10101010

Greedy

Branch Predictor Recovery

- 1-Bit Counter Recovery

PC ↓

0
1
0
1

Lazy

- 2-Bit Counter Recovery

PC ↓

00
11
01
10

Lazy

- Global History Recovery

10101010

Greedy

- Local History Recovery

PC ↓

10101010
01010101

Branch Predictor Recovery

- 1-Bit Counter Recovery

PC ↓

0
1
0
1

Lazy

- 2-Bit Counter Recovery

PC ↓

00
11
01
10

Lazy

- Global History Recovery

10101010

Greedy

- Local History Recovery

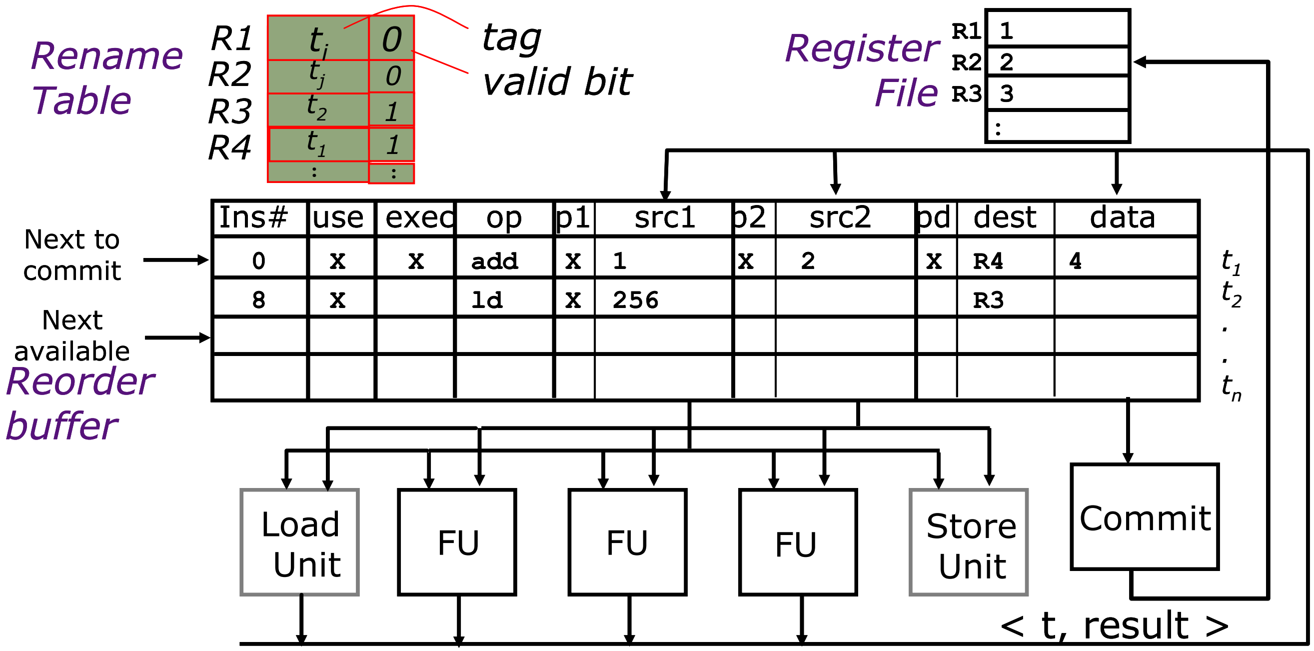
PC ↓

10101010
01010101

Greedy!!

O-o-O Execution with ROB

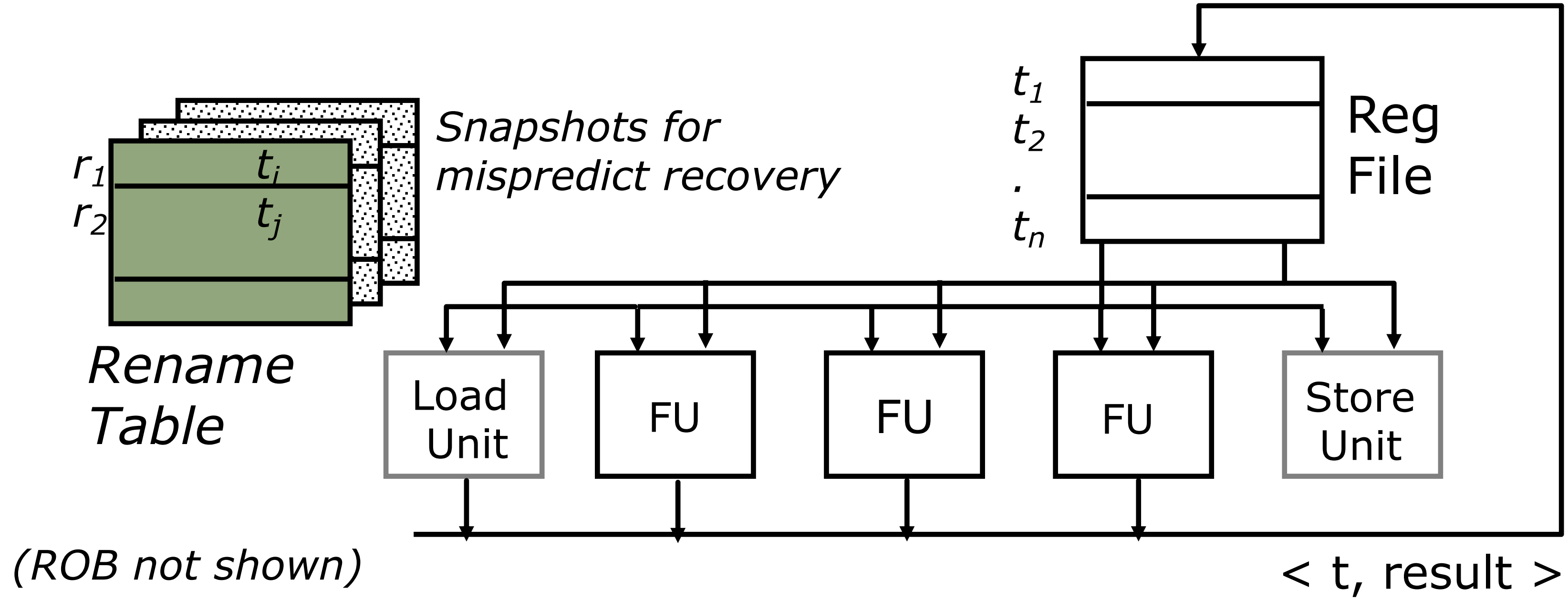
Data-in-ROB design



Basic Operation:

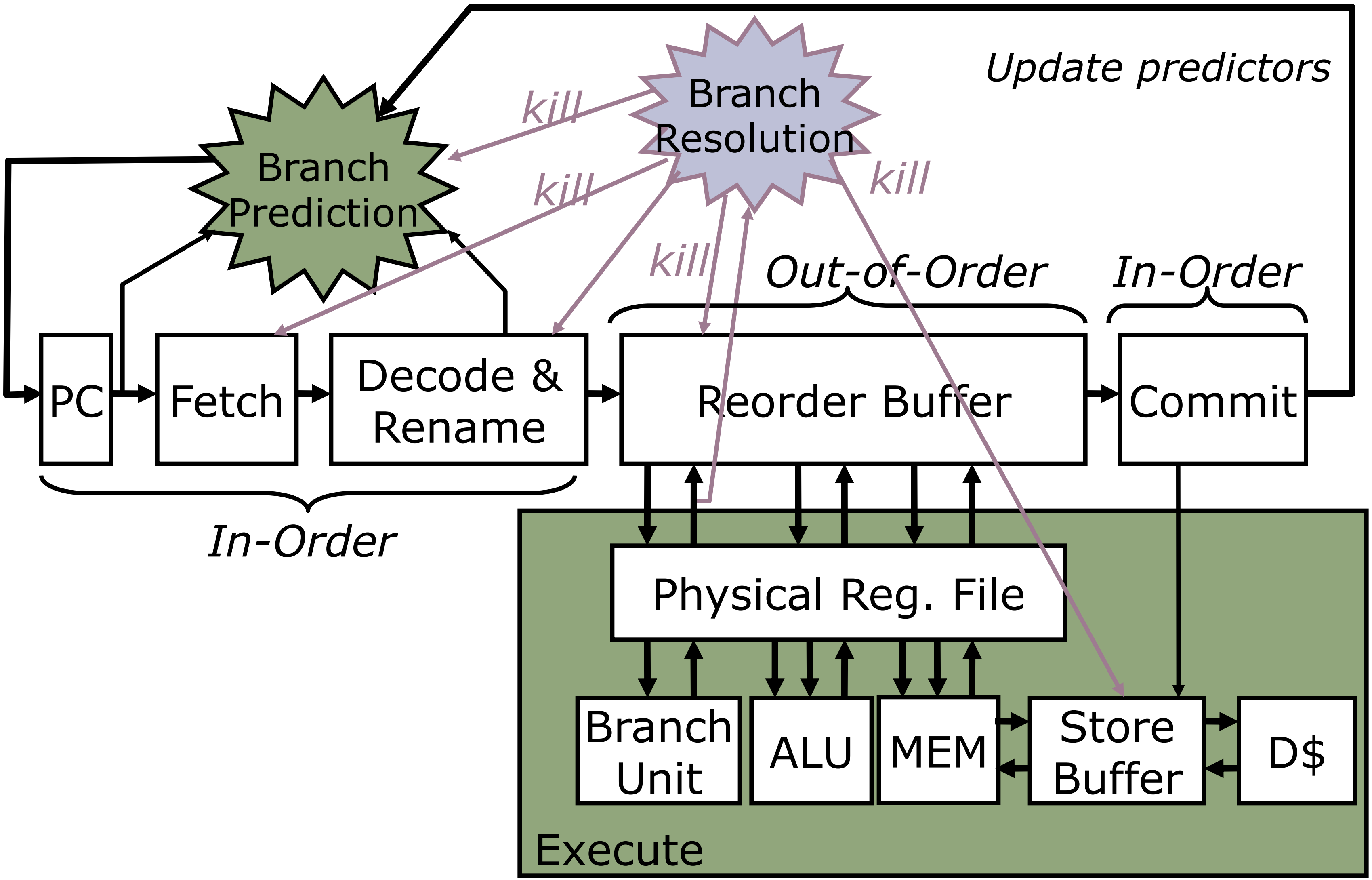
- Enter op and tag or data (if known) for each source
- Replace tag with data as it becomes available
- Issue instruction when all sources are available
- Save dest data when operation finishes
- Commit saved dest data when instruction commits

Unified Physical Register File
(MIPS R10K, Alpha 21264, Pentium 4)



- One regfile for both *committed* and *speculative* values (no data in ROB)
- During decode, instruction result allocated new physical register, source regs translated to physical regs through rename table
- Instruction reads data from regfile at start of execute (not in decode)
- Write-back updates reg. busy bits on instructions in ROB (assoc. search)
- Snapshots of rename table taken at every branch to recover mispredicts
- On exception, renaming undone in reverse order of issue *(MIPS R10000)*

Speculative & Out-of-Order Execution



Lifetime of Physical Registers

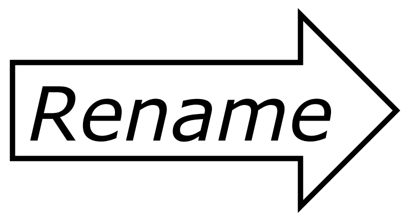
- Physical regfile holds committed and speculative values
- Physical registers decoupled from ROB entries *(no data in ROB)*

a) ld r1, (r3)
b) add r3, r1, #4
c) sub r1, r3, r9
d) add r3, r1, r7
e) ld r6, (r1)
f) add r8, r6, r3
g) st r8, (r1)
h) ld r3, (r11)

Lifetime of Physical Registers

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

a) ld r1, (r3)
b) add r3, r1, #4
c) sub r1, r3, r9
d) add r3, r1, r7
e) ld r6, (r1)
f) add r8, r6, r3
g) st r8, (r1)
h) ld r3, (r11)



Lifetime of Physical Registers

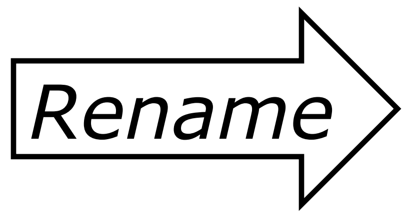
- Physical regfile holds committed and speculative values
- Physical registers decoupled from ROB entries *(no data in ROB)*

a)	ld r1, (r3)		ld P1, (P*x*)
b)	add r3, r1, #4		add P2, P1, #4
c)	sub r1, r3, r9		sub P3, P2, P*y*
d)	add r3, r1, r7	*Rename* →	add P4, P3, P*z*
e)	ld r6, (r1)		ld P5, (P3)
f)	add r8, r6, r3		add P6, P5, P4
g)	st r8, (r1)		st P6, (P3)
h)	ld r3, (r11)		ld P7, (P*w*)

Lifetime of Physical Registers

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

a) ld r1, (r3)
b) add r3, r1, #4
c) sub r1, r3, r9
d) add r3, r1, r7
e) ld r6, (r1)
f) add r8, r6, r3
g) st r8, (r1)
h) ld r3, (r11)



ld P1, (P*x*)
add P2, P1, #4
sub P3, P2, P*y*
add P4, P3, P*z*
ld P5, (P3)
add P6, P5, P4
st P6, (P3)
ld P7, (P*w*)

When can we reuse a physical register?

Lifetime of Physical Registers

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

		Rename →
a)	ld r1, (r3)	ld P1, (P*x*)
b)	add r3, r1, #4	add P2, P1, #4
c)	sub r1, r3, r9	sub P3, P2, P*y*
d)	add r3, r1, r7	add P4, P3, Pz
e)	ld r6, (r1)	ld P5, (P3)
f)	add r8, r6, r3	add P6, P5, P4
g)	st r8, (r1)	st P6, (P3)
h)	ld r3, (r11)	ld P7, (P*w*)

When can we reuse a physical register?

When next write to same architectural register commits

Physical Register Management

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

P0
P1
P3
P2
P4
⋮

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

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

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

P0
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd

Physical Register Management

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

P0
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd

Physical Register Management

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

P0
P1
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld					r1		

Physical Register Management

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

P0
P1
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld					r1		

Physical Register Management

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

P0
P1
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld		P7			r1		

Physical Register Management

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

P0
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld		P7			r1		

Physical Register Management

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

P0
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1		

Physical Register Management

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

P0
P1
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1		

Physical Register Management

Rename Table

R0	
R1	P8
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

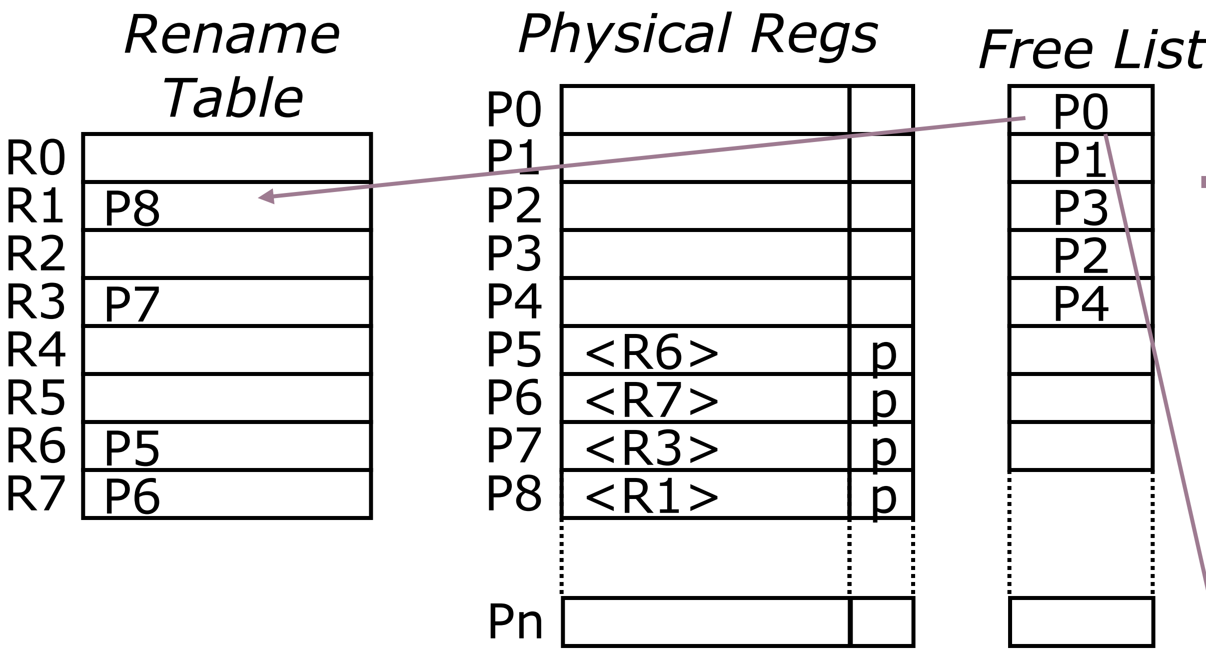
P0
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

P0
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

P0
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

~~P0~~
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

~~P0~~
P1
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

~~P0~~
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

~~P0~~
P1
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3		

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

~~P0~~
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3		

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	P7
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

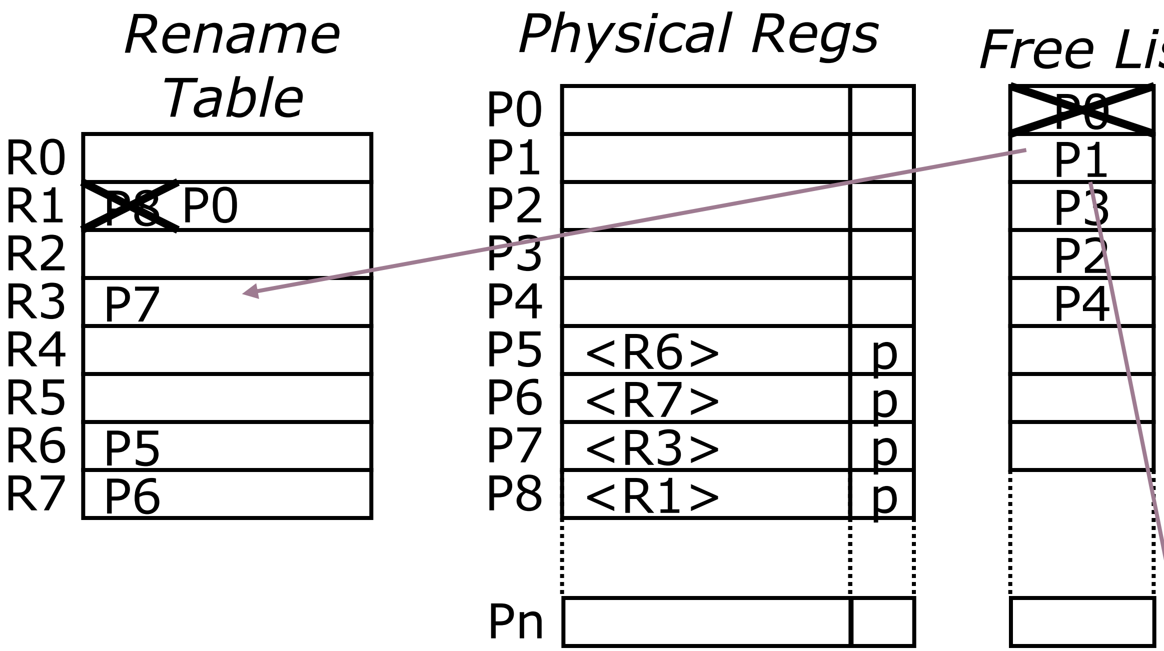
~~P0~~
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	

Physical Register Management



ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	

```
   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	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

~~P0~~
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

~~P0~~
P1
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

~~P0~~
~~P1~~
P3
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

~~P0~~
~~P1~~
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

~~P0~~
~~P1~~
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

~~P0~~
~~P1~~
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6		

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

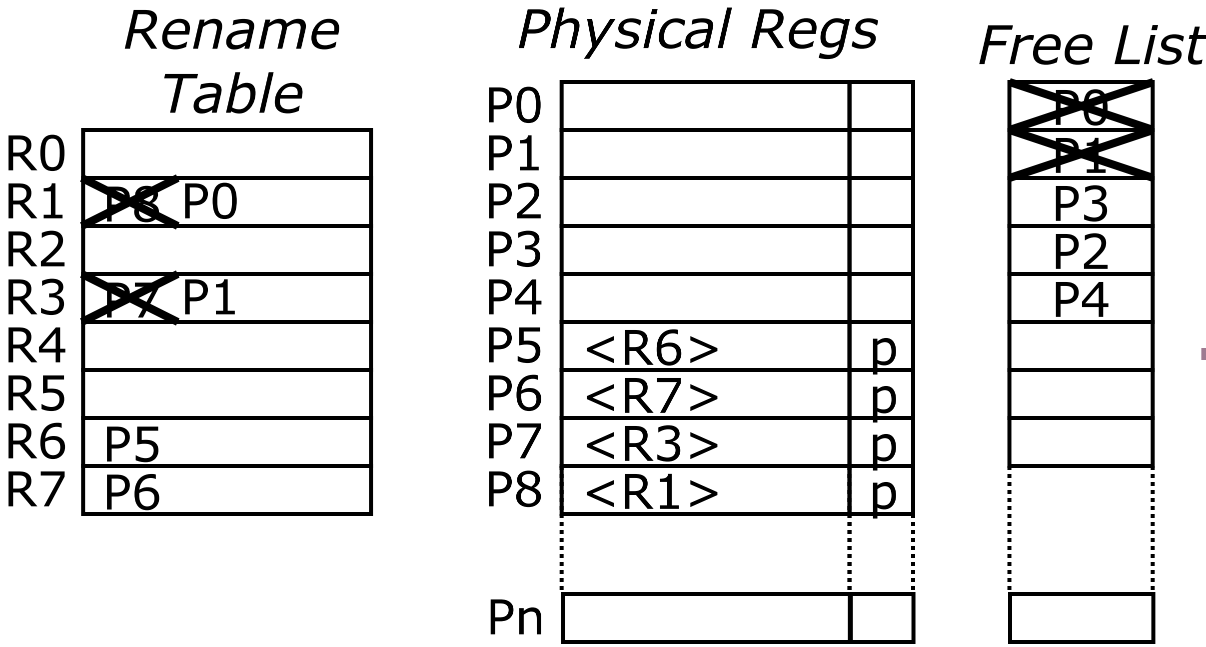
~~P0~~
~~P1~~
P3
P2
P4

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6		

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	P5
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

~~P0~~
~~P1~~
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	~~P5~~ P3
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

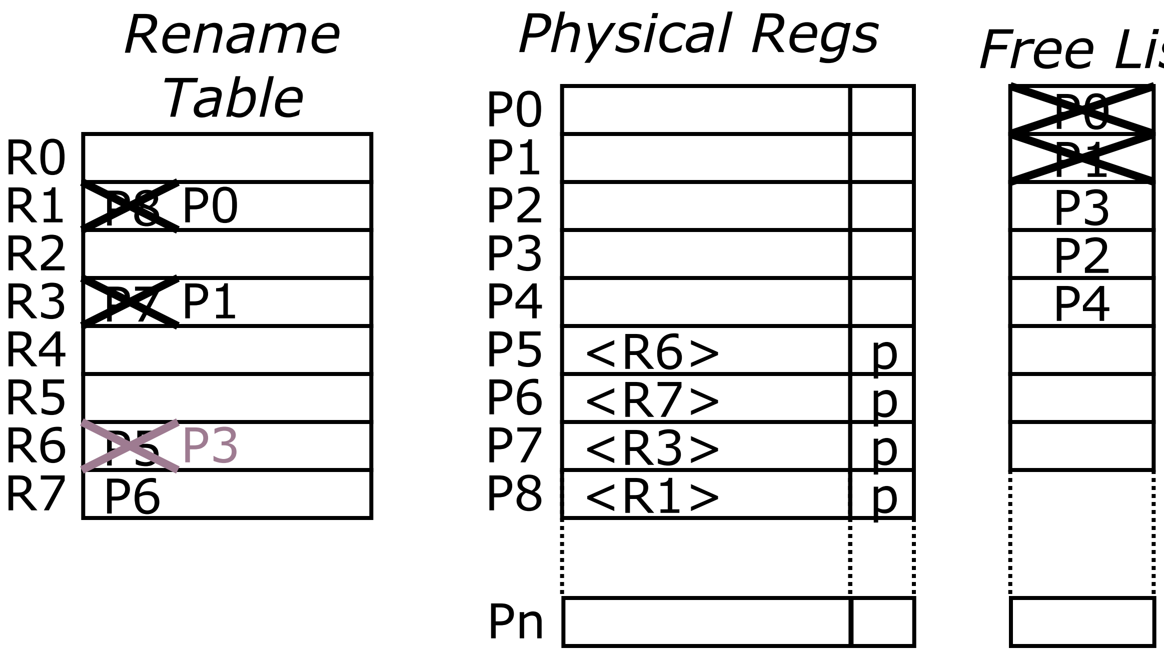
~~P0~~
~~P1~~
P3
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	

Physical Register Management

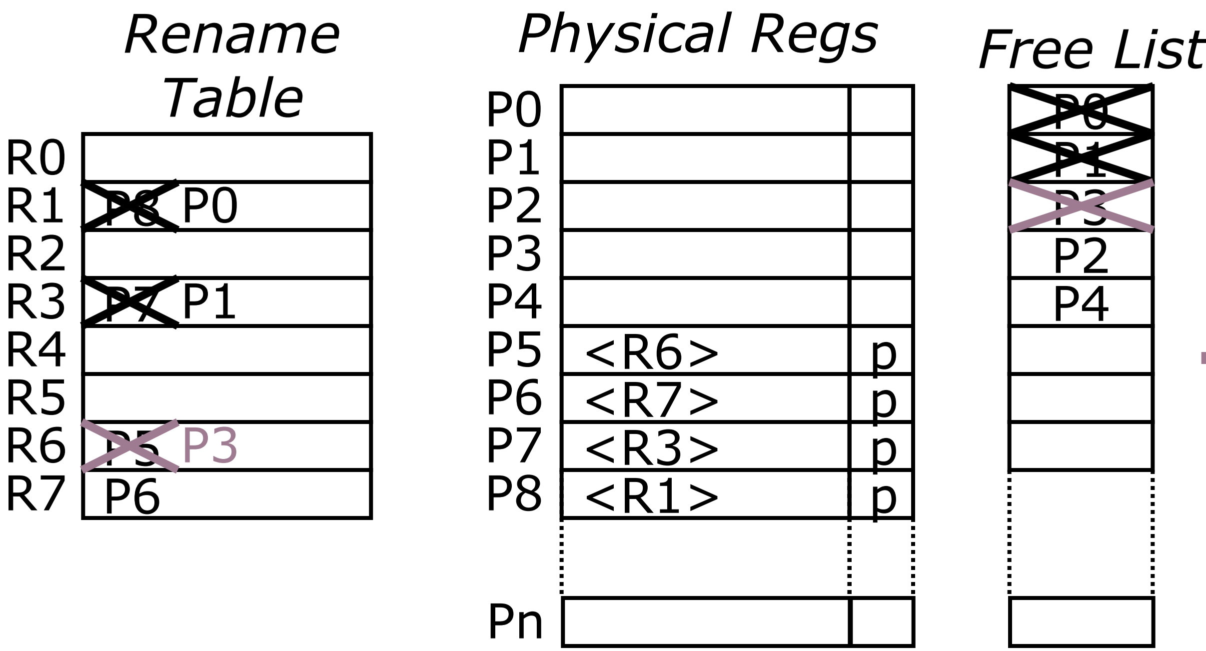


ld r1, 0(r3)
add r3, r1, #4
→ sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	~~P5~~ P3
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

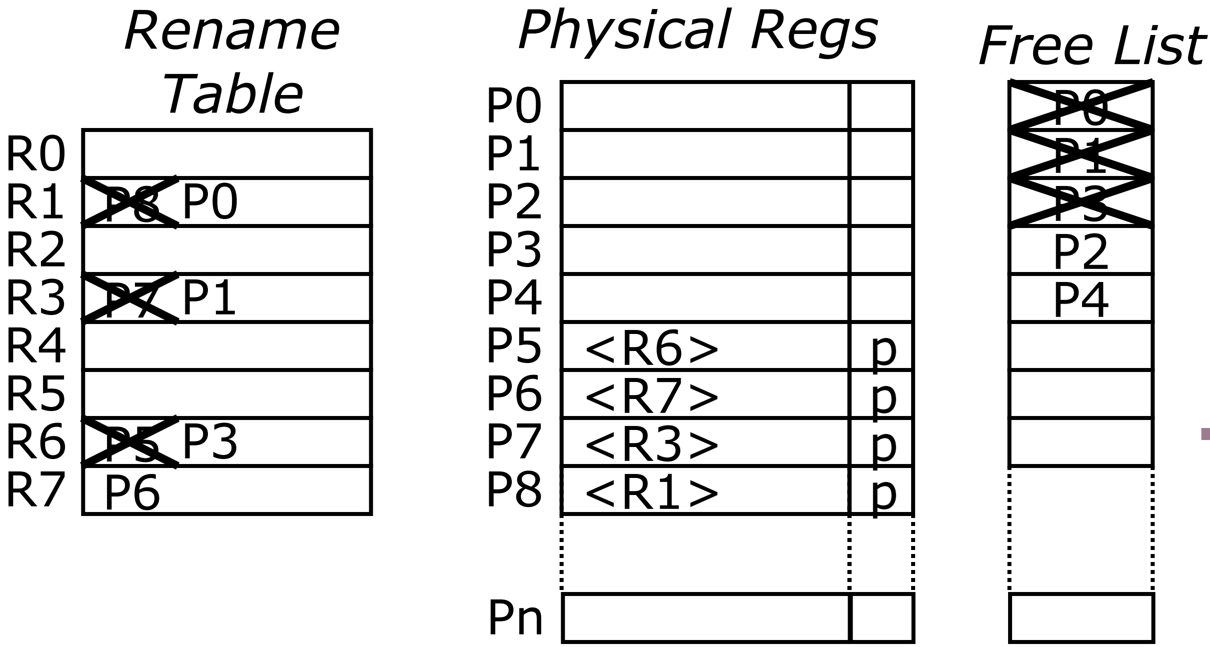
~~P0~~
~~P1~~
~~P3~~
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3

Physical Register Management

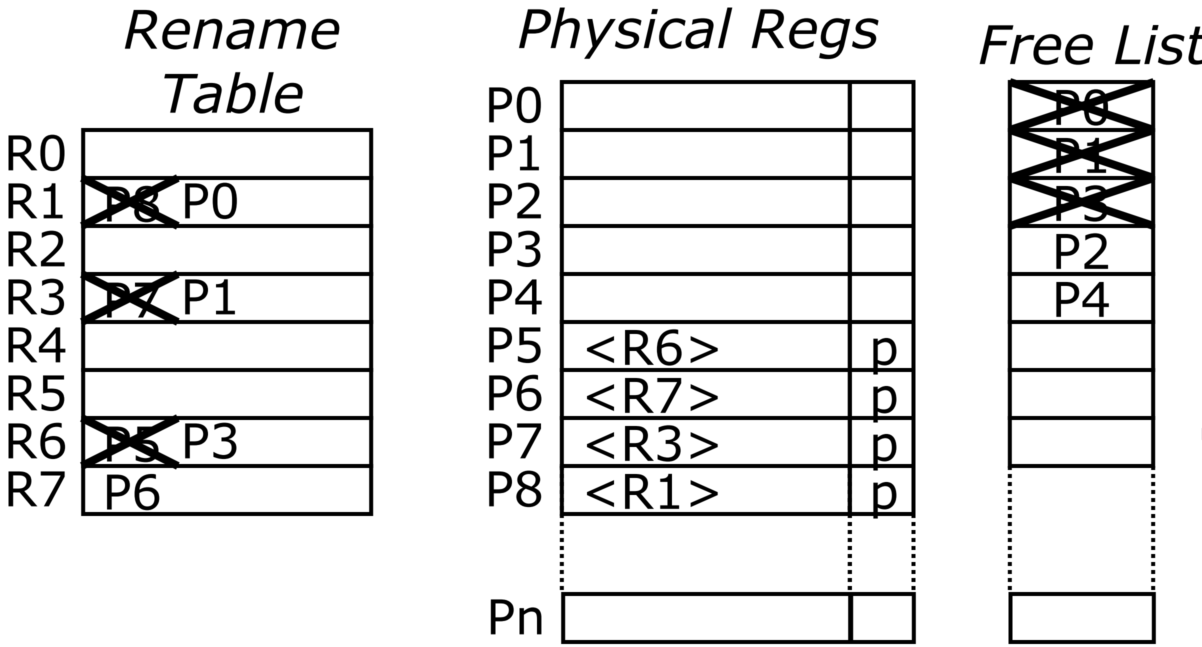


ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
→ add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3		

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	~~P5~~ P3
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
P2
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3		

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ P1
R4	
R5	
R6	~~P5~~ P3
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

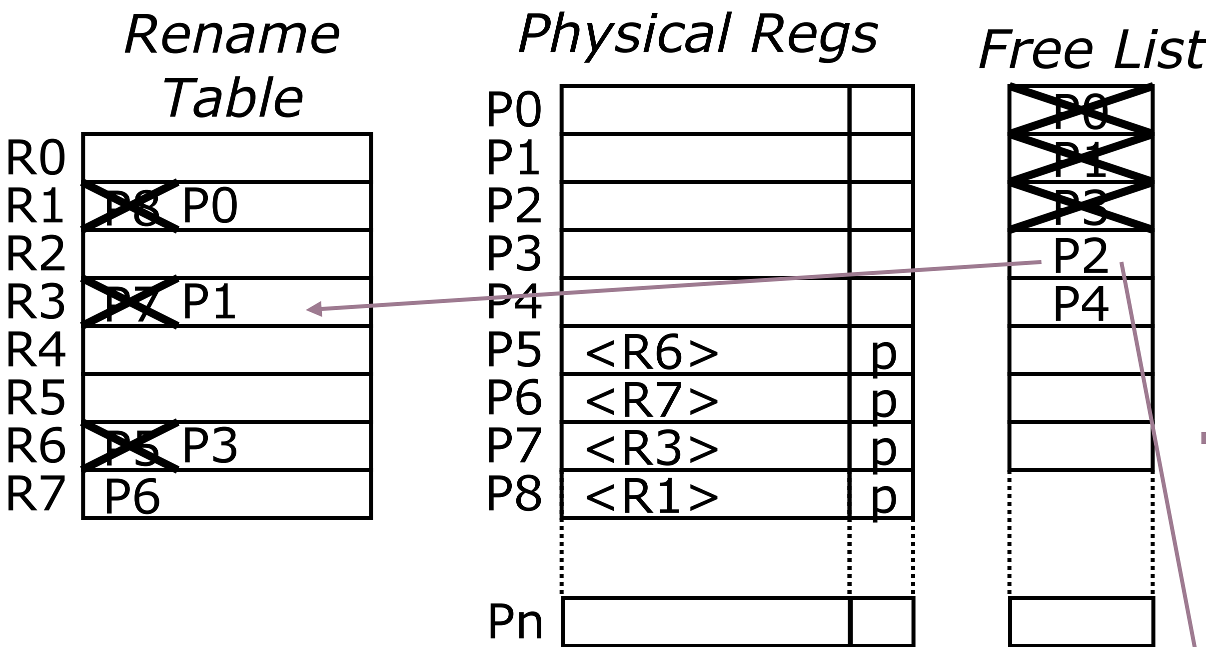
~~P0~~
~~P1~~
~~P3~~
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ P3
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ P3
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

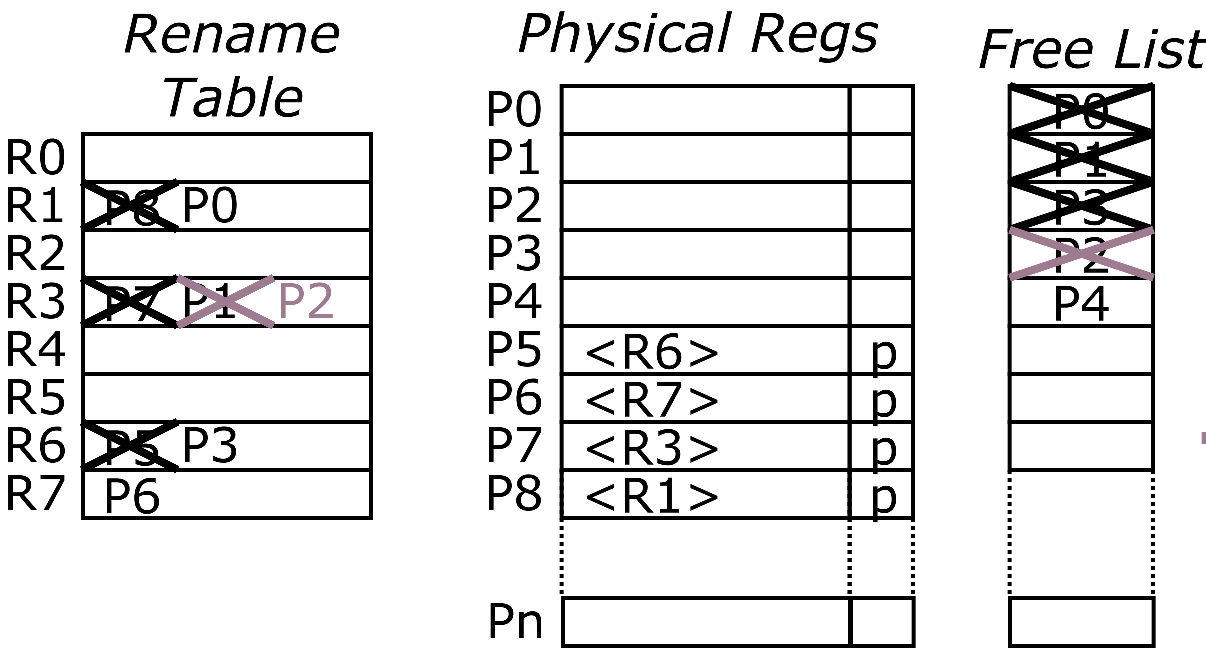
~~P0~~
~~P1~~
~~P3~~
P2
P4
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2

Physical Register Management

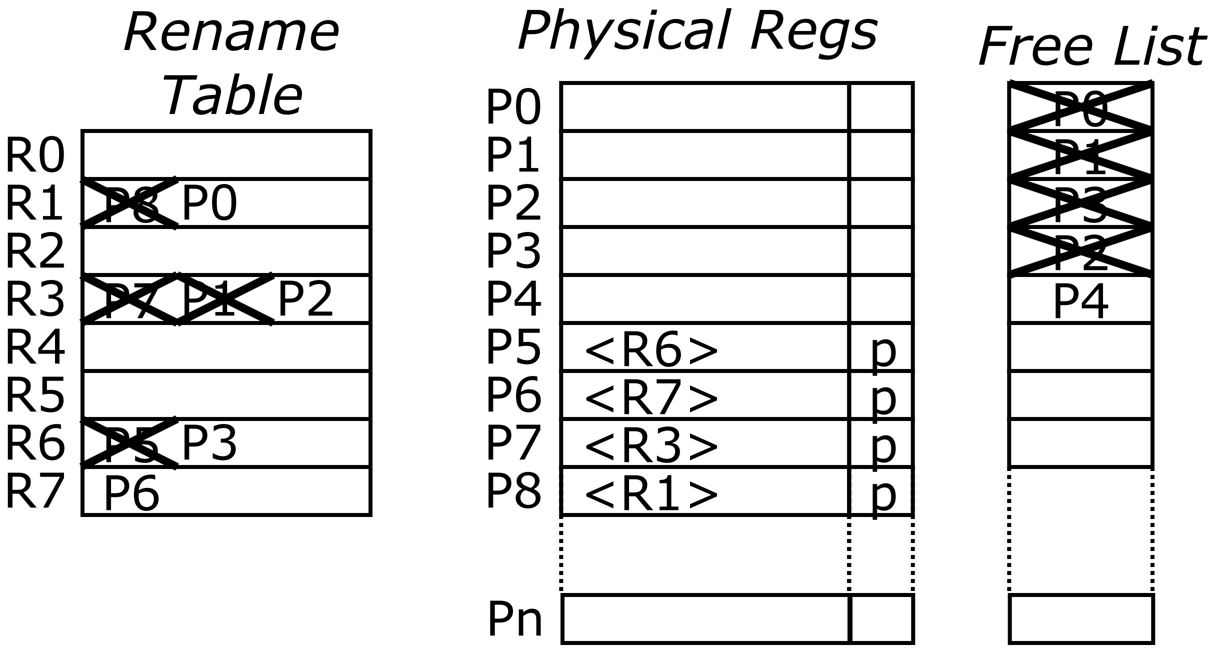


ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
→ add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2

Physical Register Management

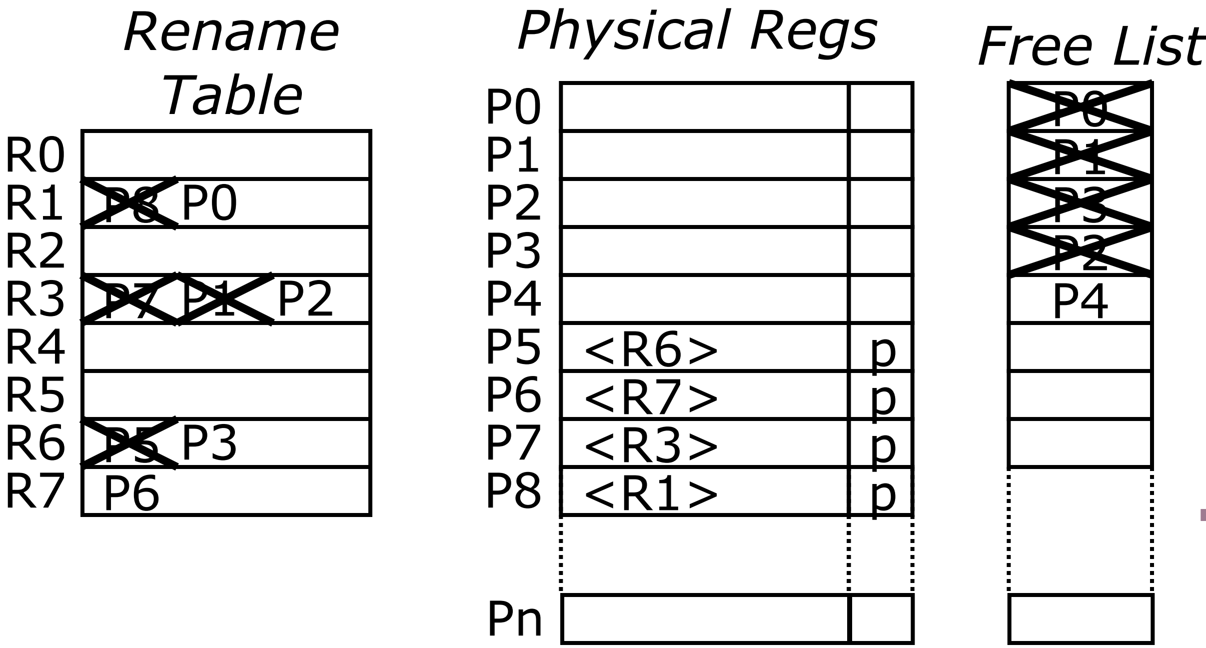


ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2

Physical Register Management

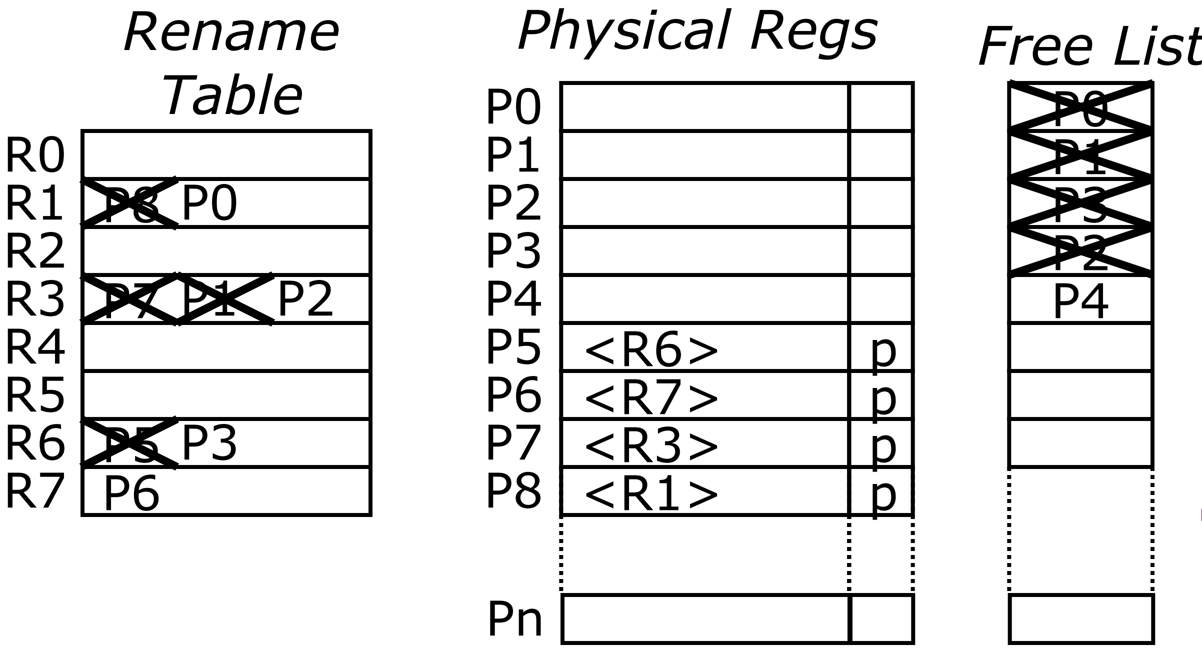


ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
→ ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2

Physical Register Management

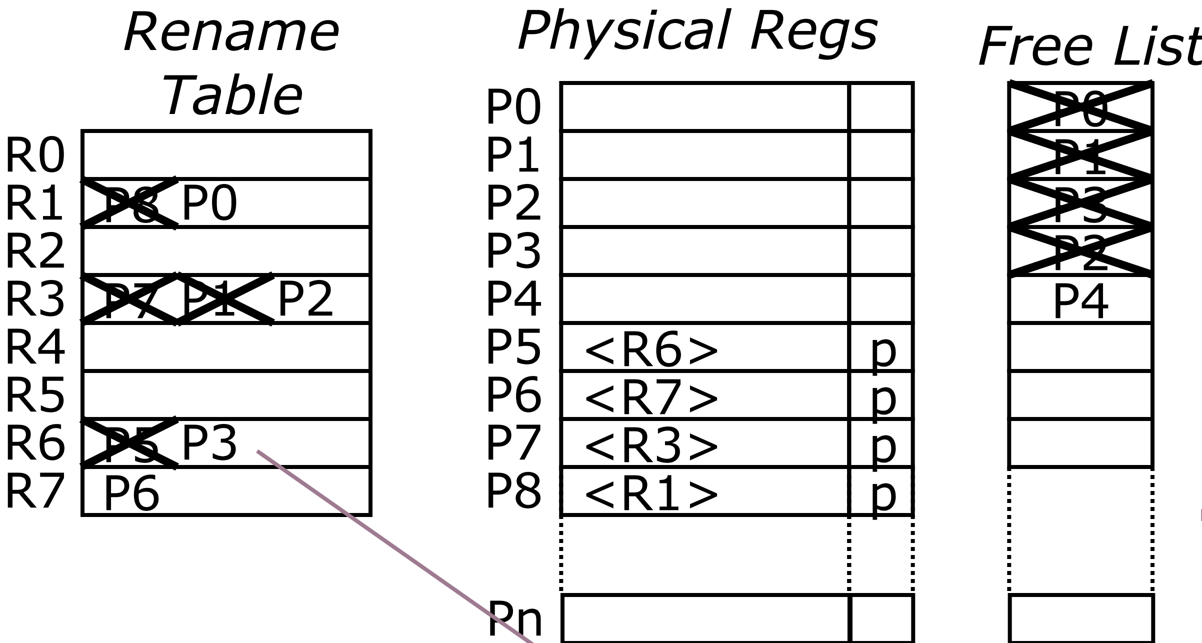


ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
→ ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6		

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6		

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ P3
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

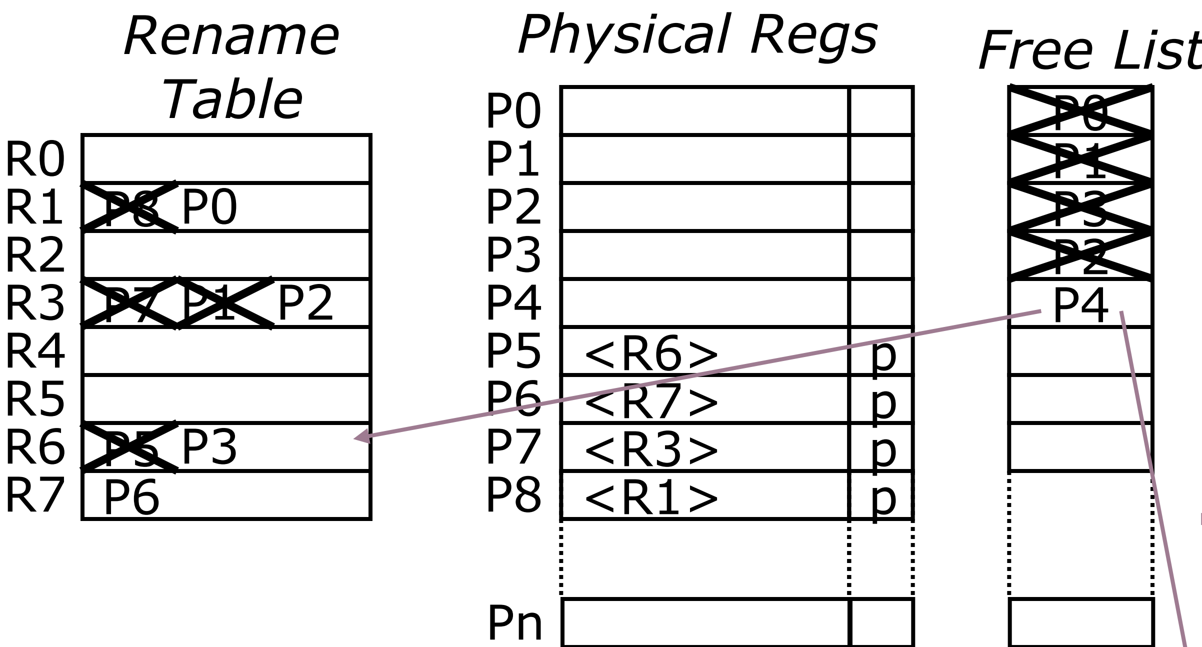
~~P0~~
~~P1~~
~~P3~~
~~P2~~
P4
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6	P3	

Physical Register Management

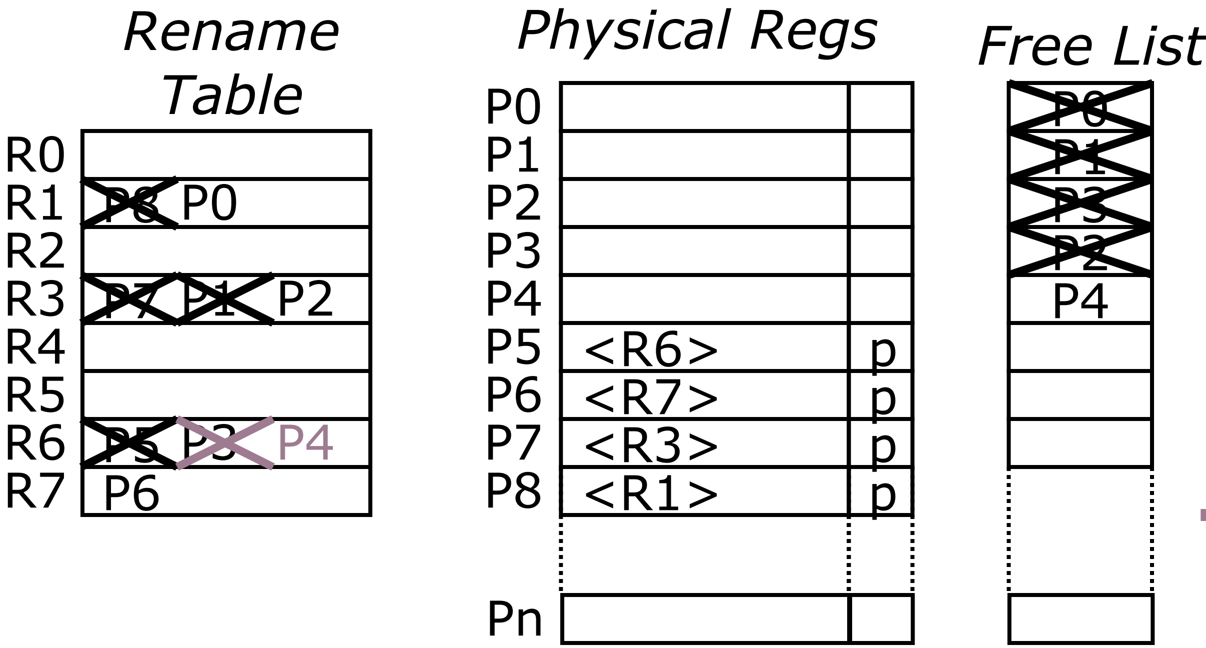


ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
➡ ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6	P3	

Physical Register Management

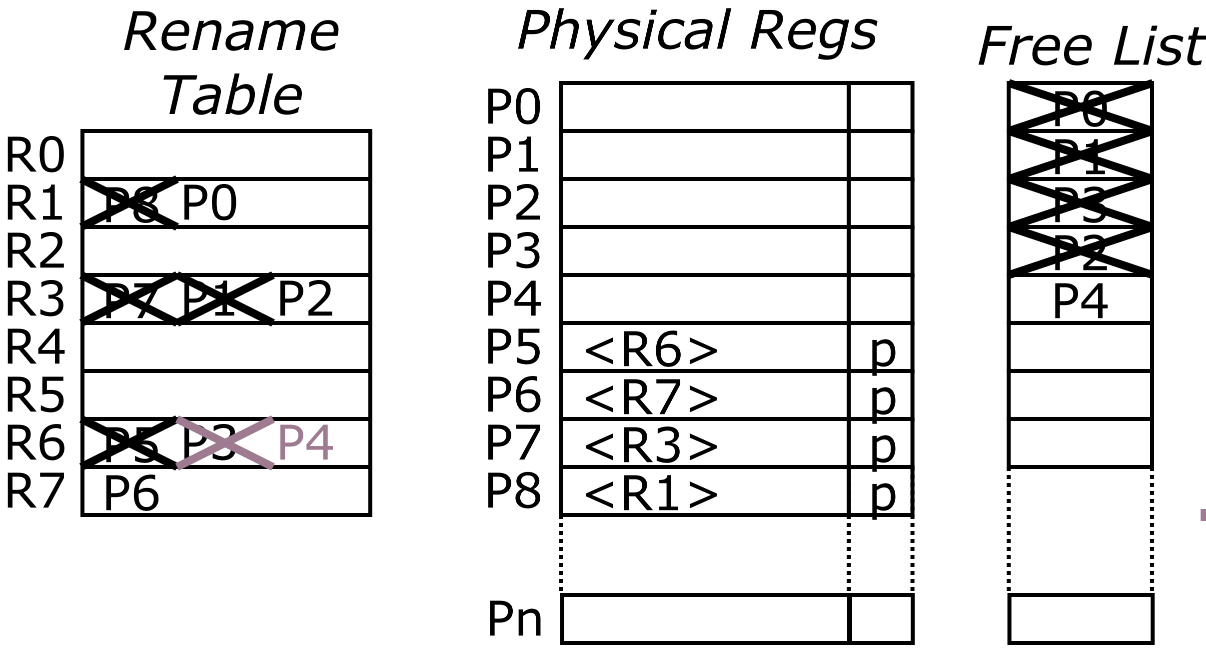


ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
→ ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6	P3	

Physical Register Management

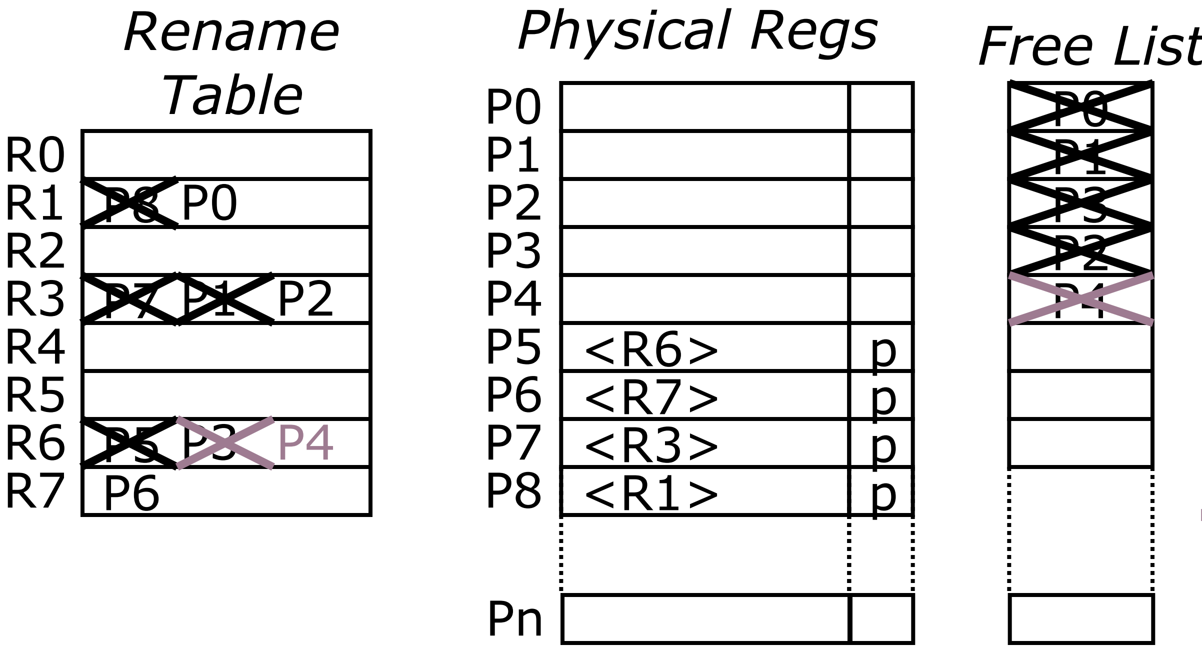


ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
→ ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6	P3	P4

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6	P3	P4

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6	P3	P4

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0		
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
⋮		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
⋮

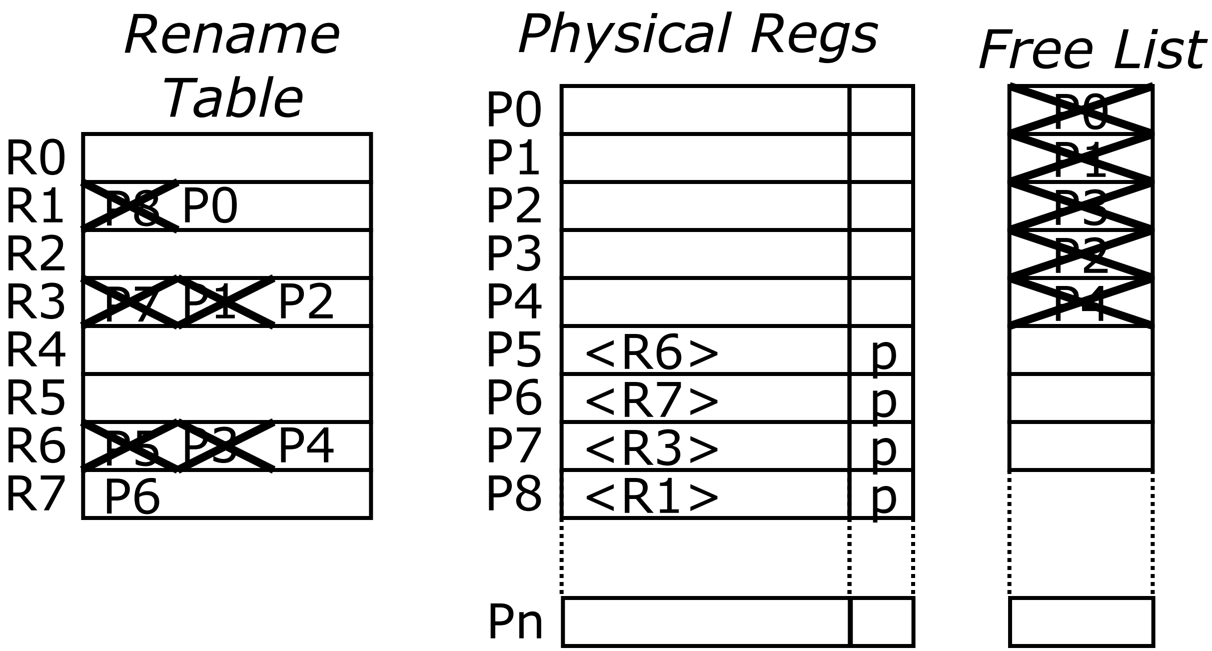
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6	P3	P4

← Execute & Commit

Physical Register Management



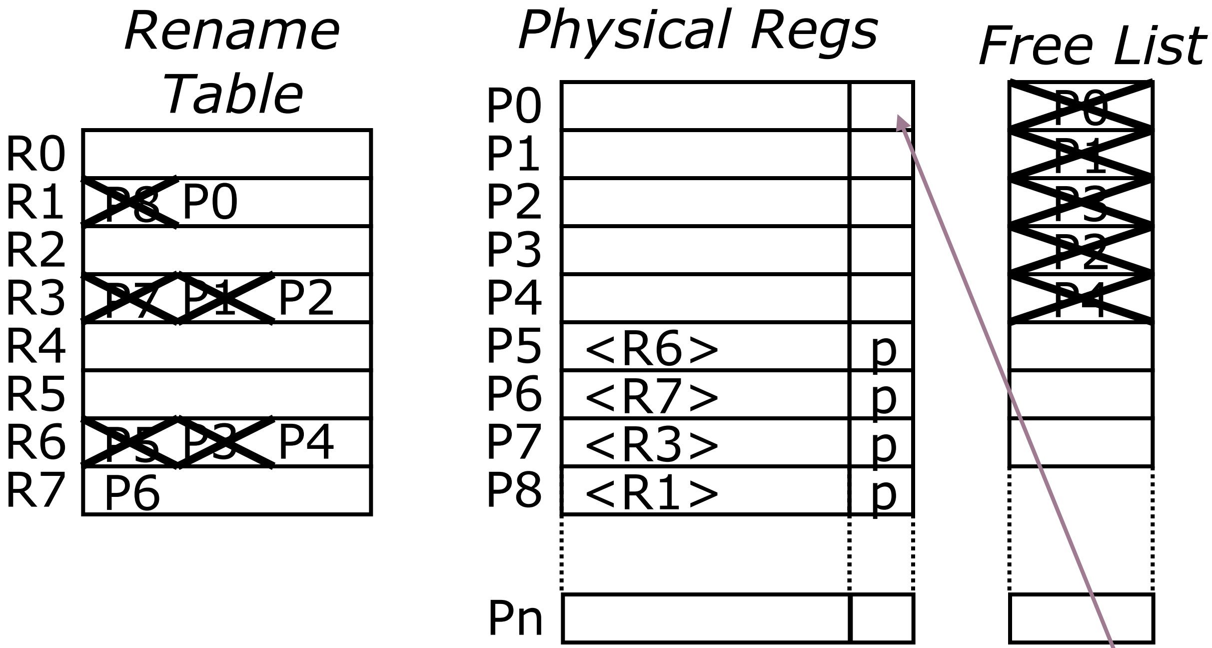
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6	P3	P4

Execute & Commit

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add		P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld		P0			r6	P3	P4

Execute & Commit

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0	<R1>	p
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	<R1>	p
...		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
...

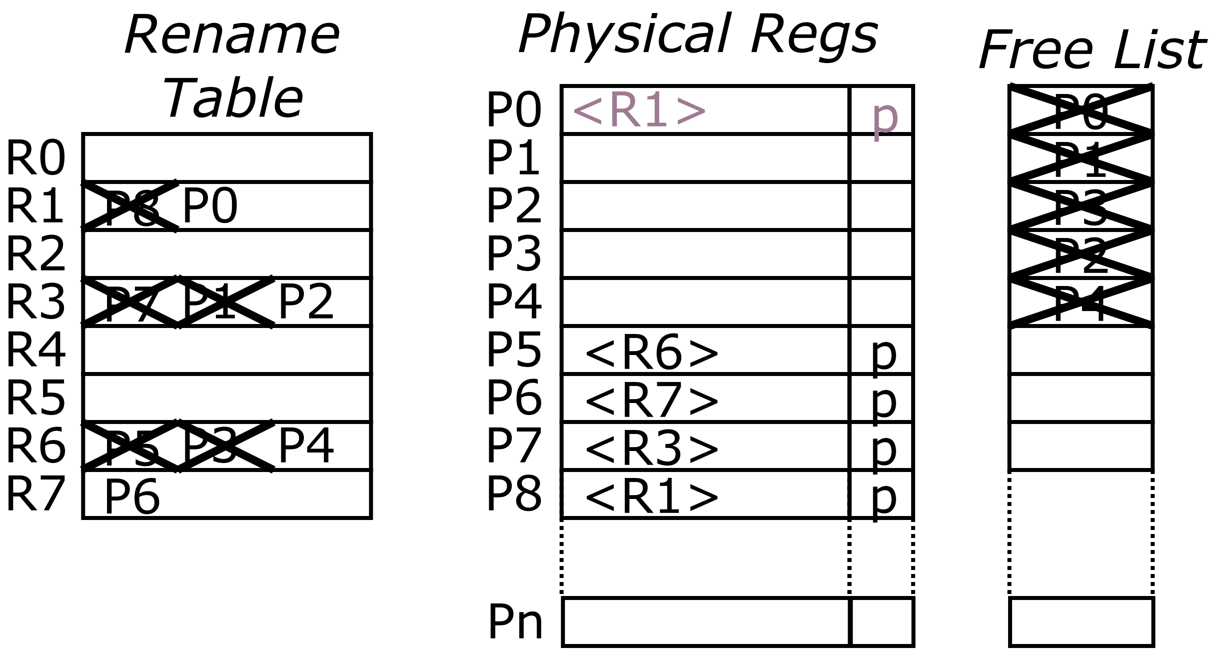
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

← Execute & Commit

Physical Register Management



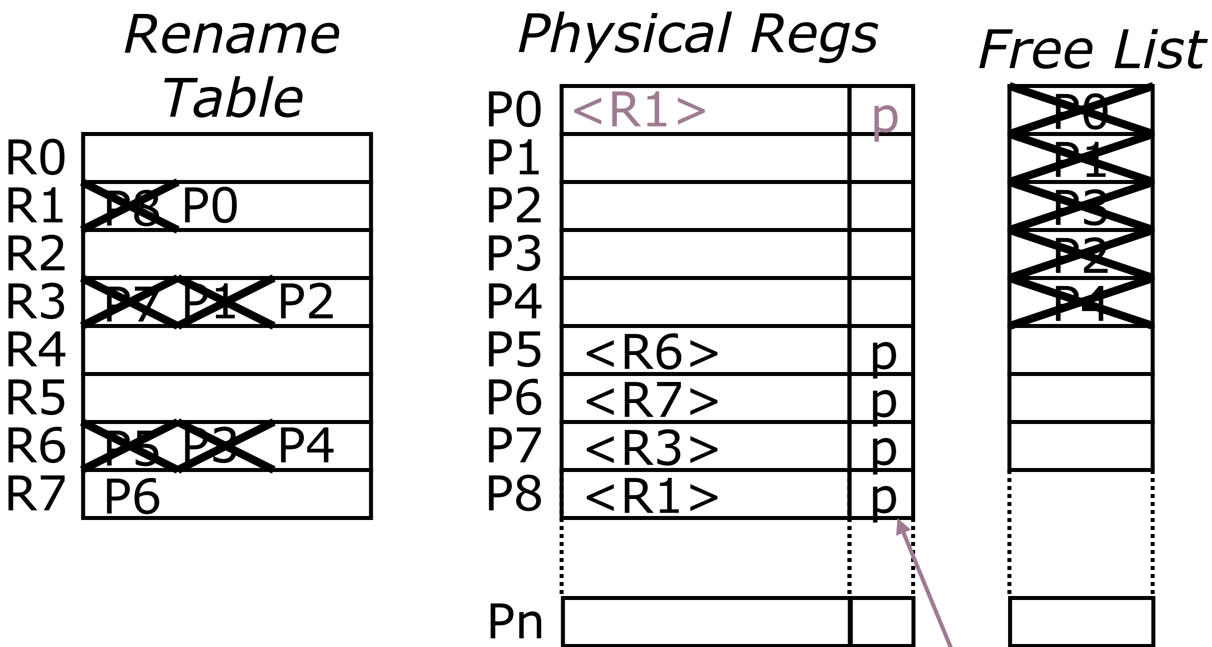
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

Execute & Commit

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

Execute & Commit

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0	<R1>	p
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	~~<R1>~~	~~p~~
⋮		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

← Execute & Commit

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0	<R1>	p
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8	~~<R1>~~	~~p~~
...		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
...

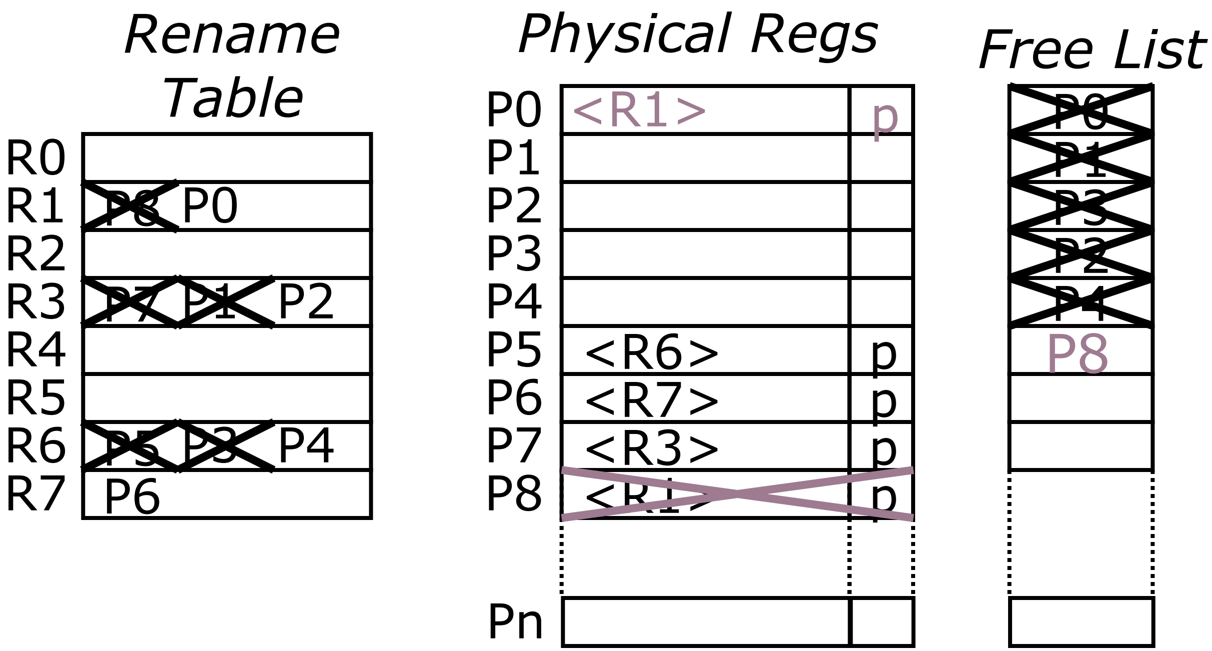
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

Execute & Commit

Physical Register Management



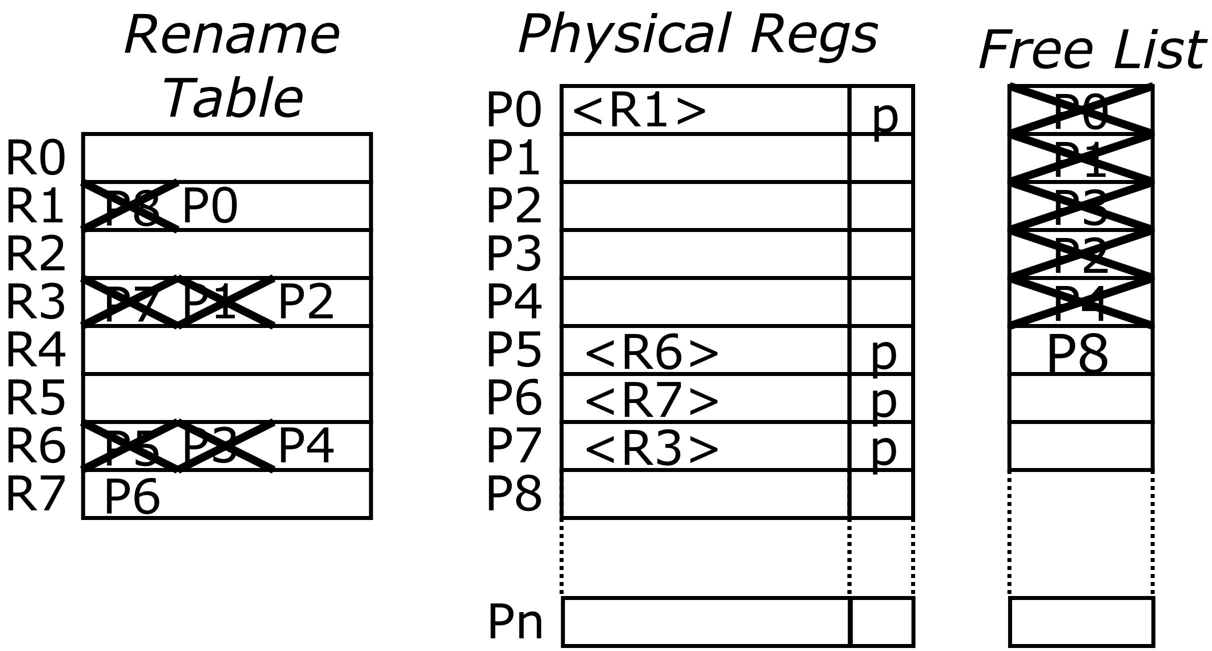
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

Execute & Commit

Physical Register Management



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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0	<R1>	p
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8		
...		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
P8
...

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

← Execute & Commit

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0	<R1>	p
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8		
⋮		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
P8
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

← Execute & Commit

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0	<R1>	p
P1		
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8		
⋮		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
P8
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add		P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

Execute & Commit

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0	<R1>	p
P1	<R3>	p
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8		
⋮		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
P8
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x		add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add	p	P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

← Execute & Commit

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0	<R1>	p
P1	<R3>	p
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8		
⋮		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
P8
⋮

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

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x	x	add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add	p	P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

← Execute & Commit

Physical Register Management

Rename Table

R0	
R1	~~P8~~ P0
R2	
R3	~~P7~~ ~~P1~~ P2
R4	
R5	
R6	~~P5~~ ~~P3~~ P4
R7	P6

Physical Regs

P0	<R1>	p
P1	<R3>	p
P2		
P3		
P4		
P5	<R6>	p
P6	<R7>	p
P7	<R3>	p
P8		
⋮		
Pn		

Free List

~~P0~~
~~P1~~
~~P3~~
~~P2~~
~~P4~~
P8
⋮

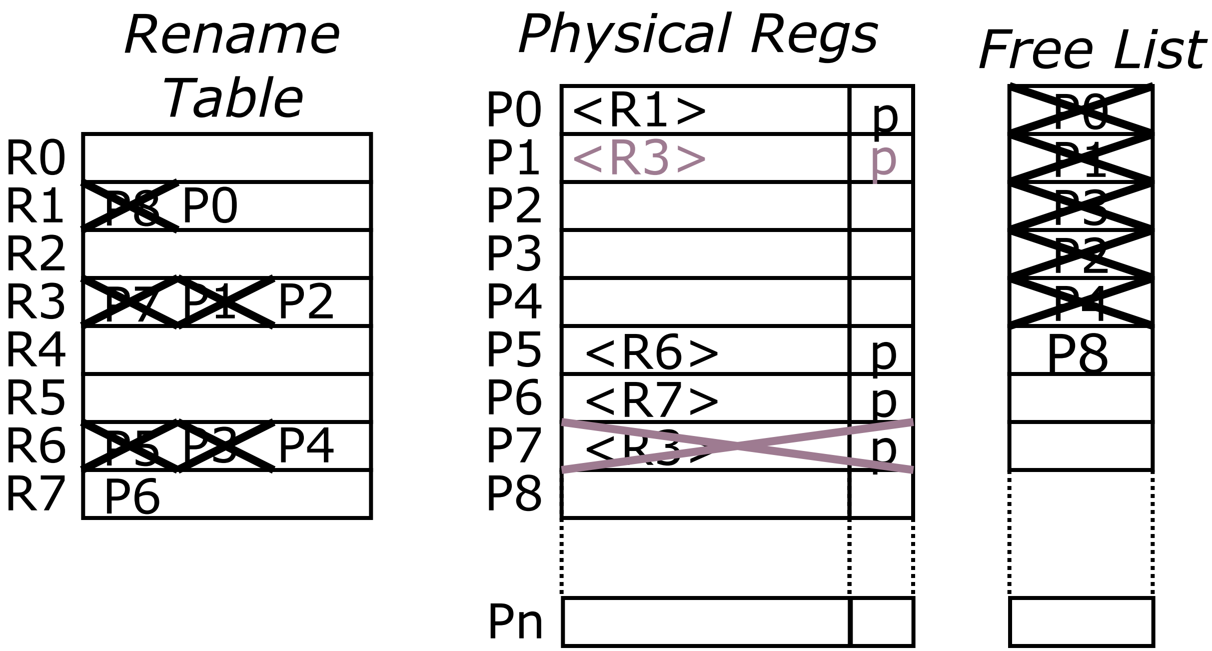
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x	x	add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add	p	P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

← Execute & Commit

Physical Register Management



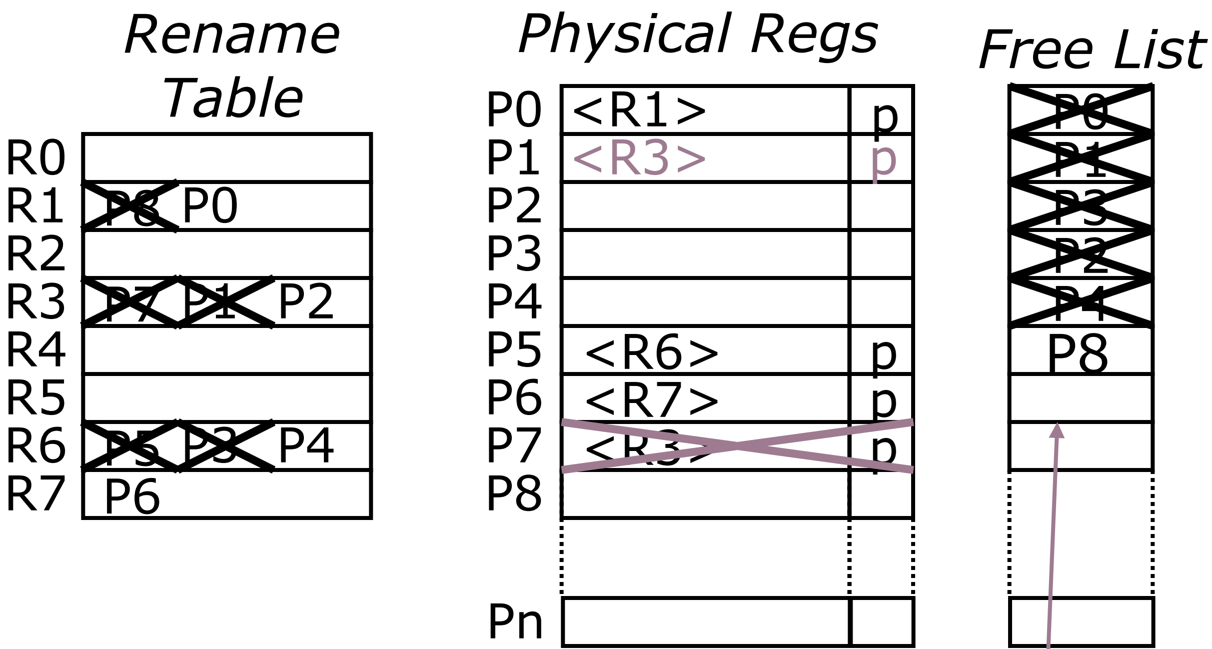
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x	x	add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add	p	P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

Execute & Commit

Physical Register Management



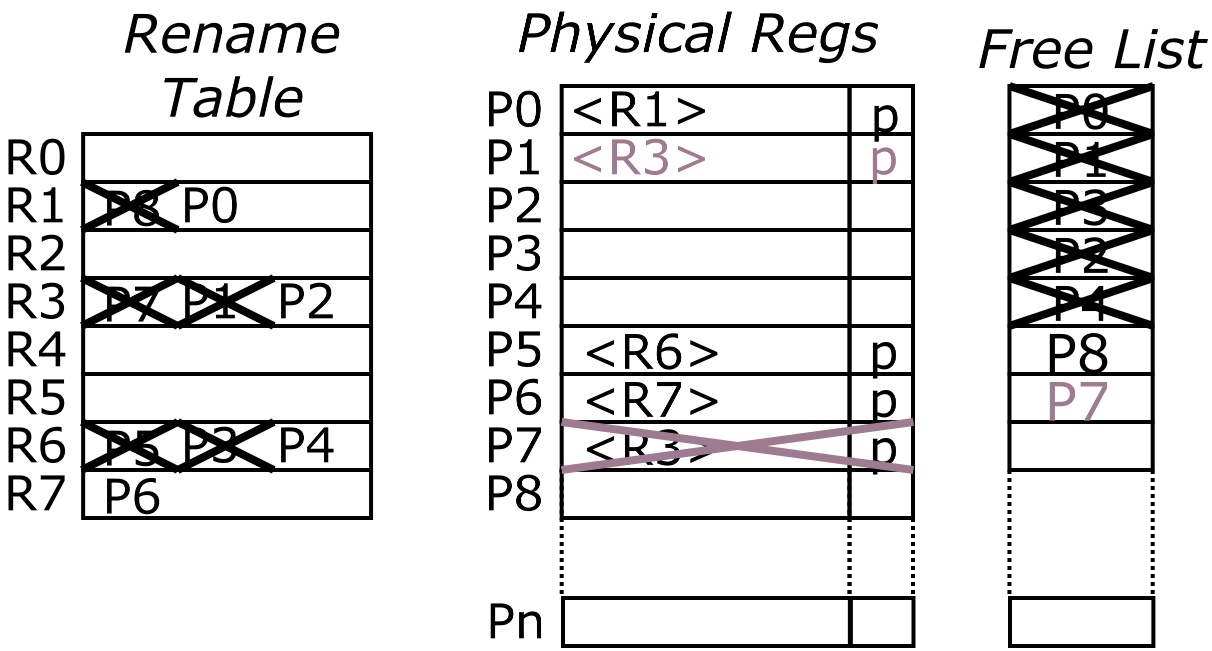
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x	x	add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add	p	P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

Execute & Commit

Physical Register Management



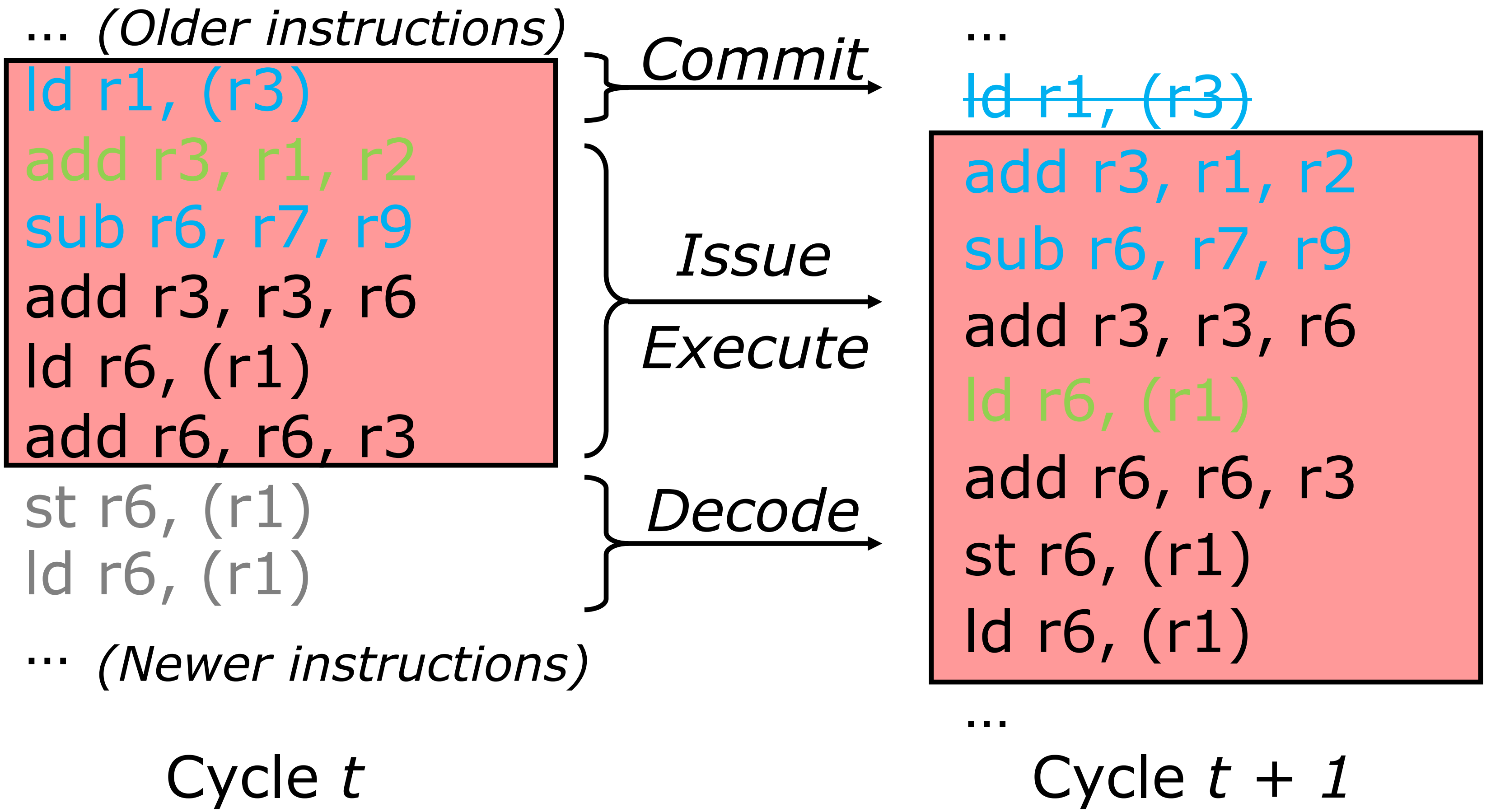
ld r1, 0(r3)
add r3, r1, #4
sub r6, r7, r6
add r3, r3, r6
ld r6, 0(r1)

ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			r1	P8	P0
x	x	add	p	P0			r3	P7	P1
x		sub	p	P6	p	P5	r6	P5	P3
x		add	p	P1		P3	r3	P1	P2
x		ld	p	P0			r6	P3	P4

← Execute & Commit

Reorder Buffer Holds Active Instruction Window



Key: predecode, decoded, issued, executed, ~~commited~~

Issue Timing

i1	Add R1,R1,#1	Issue_1	Execute_1		
i2	Sub R1,R1,#1			Issue_2	Execute_2

Issue Timing

i1	Add R1,R1,#1	$Issue_1$	$Execute_1$		
i2	Sub R1,R1,#1			$Issue_2$	$Execute_2$

How can we issue earlier?

Issue Timing

i1	Add R1,R1,#1	$Issue_1$	$Execute_1$		
i2	Sub R1,R1,#1			$Issue_2$	$Execute_2$

How can we issue earlier?

Using knowledge of execution latency (bypass)

Issue Timing

i1	Add R1,R1,#1	Issue_1	Execute_1		
i2	Sub R1,R1,#1			Issue_2	Execute_2

How can we issue earlier?

Using knowledge of execution latency (bypass)

i1	LD R1, (R3)	Issue_1	Execute_1		
i2	Sub R1,R1,#1		Issue_2	Execute_2	

Issue Timing

i1	Add R1,R1,#1	$Issue_1$	$Execute_1$		
i2	Sub R1,R1,#1			$Issue_2$	$Execute_2$

How can we issue earlier?

Using knowledge of execution latency (bypass)

i1	LD R1, (R3)	$Issue_1$	$Execute_1$		
i2	Sub R1,R1,#1		$Issue_2$	$Execute_2$	

What might make this schedule fail?

Issue Timing

i1	Add R1,R1,#1	$Issue_1$	$Execute_1$		
i2	Sub R1,R1,#1			$Issue_2$	$Execute_2$

How can we issue earlier?

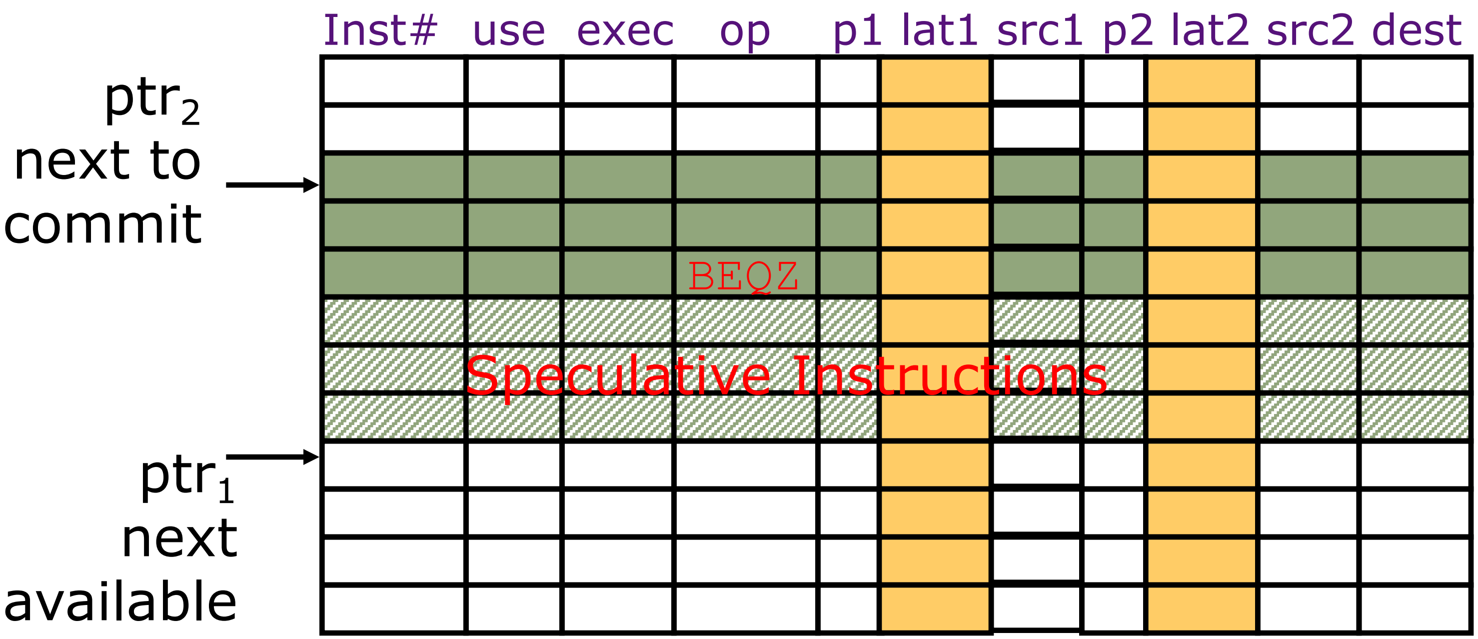
Using knowledge of execution latency (bypass)

i1	LD R1, (R3)	$Issue_1$	$Execute_1$		
i2	Sub R1,R1,#1		$Issue_2$	$Execute_2$	

What might make this schedule fail?

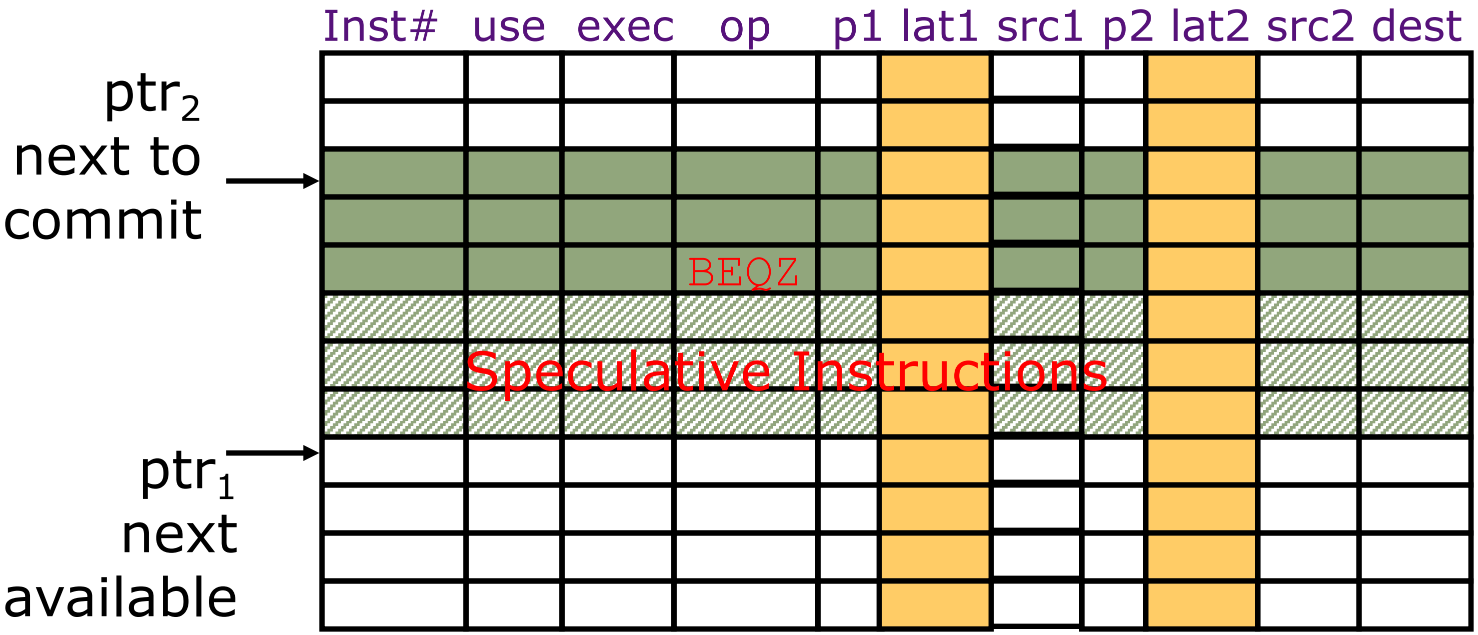
If execution latency wasn't as expected

Issue Queue with latency prediction



Issue Queue (Reorder buffer)

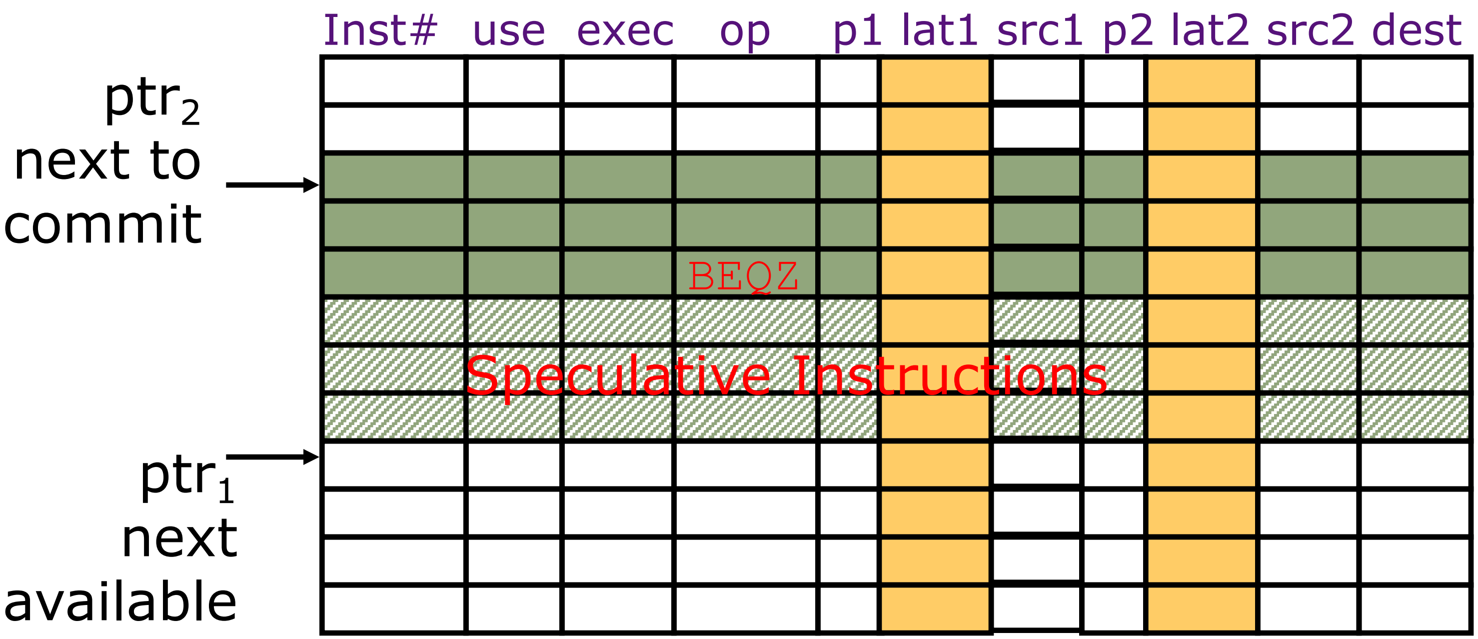
Issue Queue with latency prediction



Issue Queue (Reorder buffer)

- Fixed latency: latency included in queue entry ('bypassed')

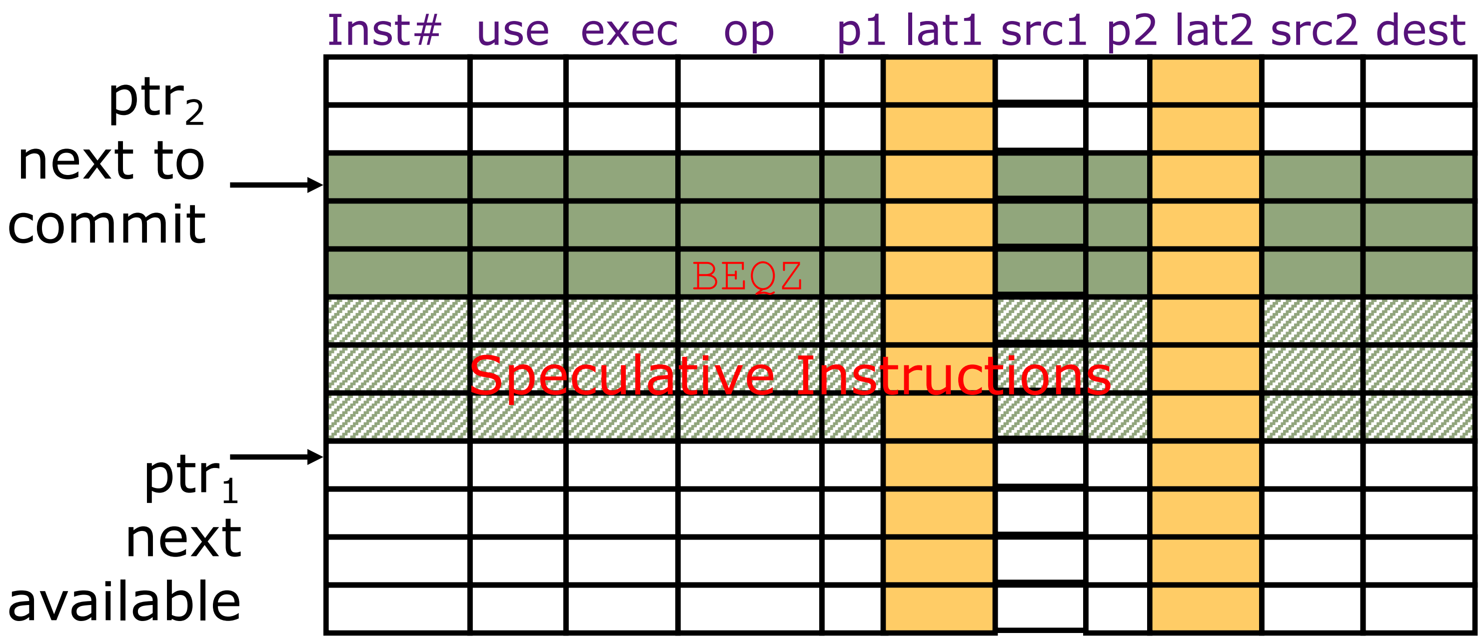
Issue Queue with latency prediction



Issue Queue (Reorder buffer)

- Fixed latency: latency included in queue entry ('bypassed')
- Predicted latency: latency included in queue entry (speculated)

Issue Queue with latency prediction

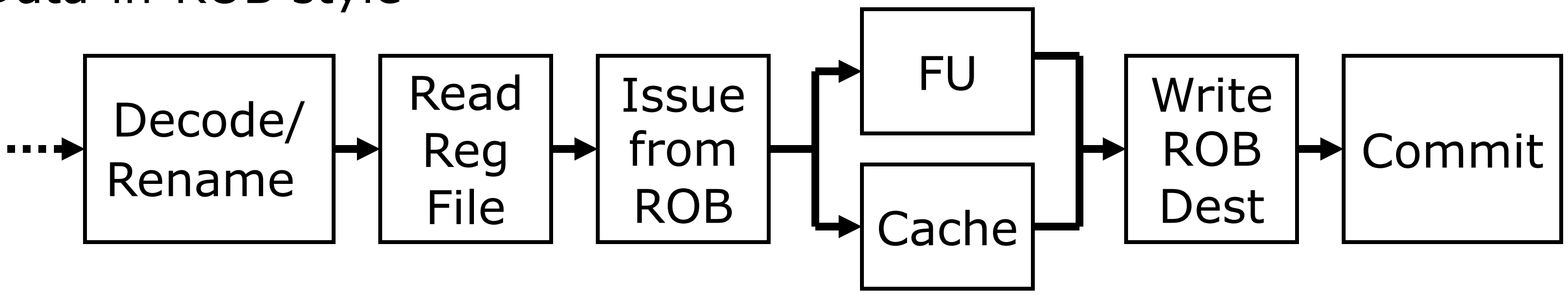


Issue Queue (Reorder buffer)

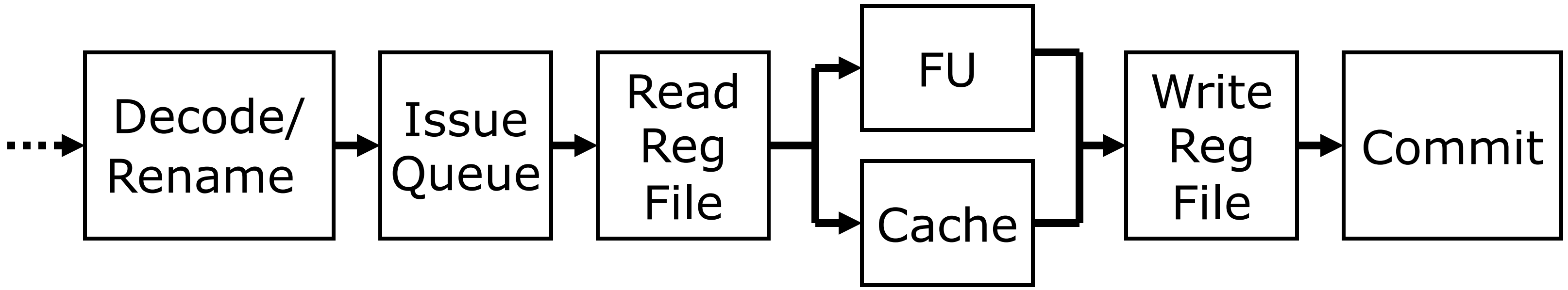
- Fixed latency: latency included in queue entry (‘bypassed’)
- Predicted latency: latency included in queue entry (speculated)
- Variable latency: wait for completion signal (stall)

Data-in-ROB vs. Unified RegFile

Data-in-ROB style

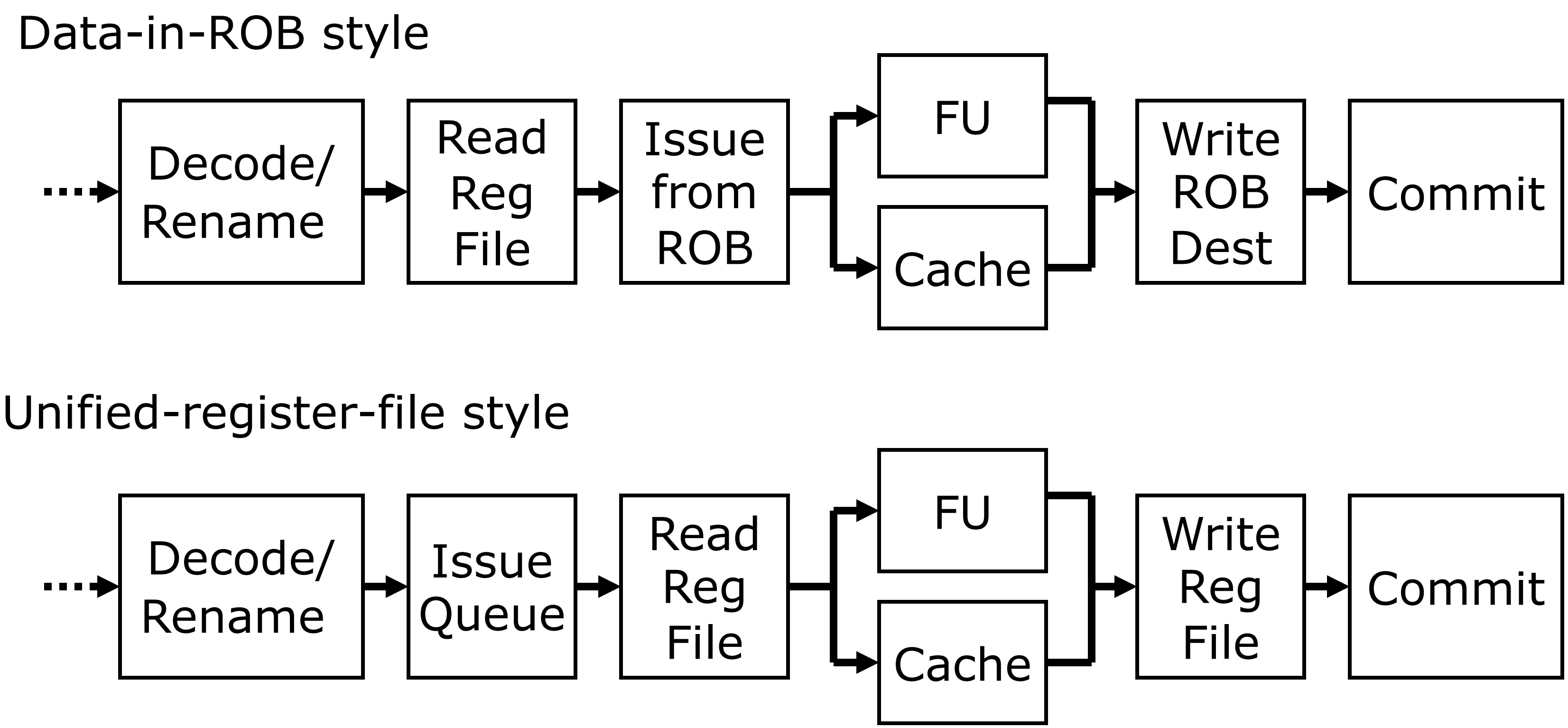


Unified-register-file style



How does issue speculation differ, e.g., on cache miss?

Data-in-ROB vs. Unified RegFile



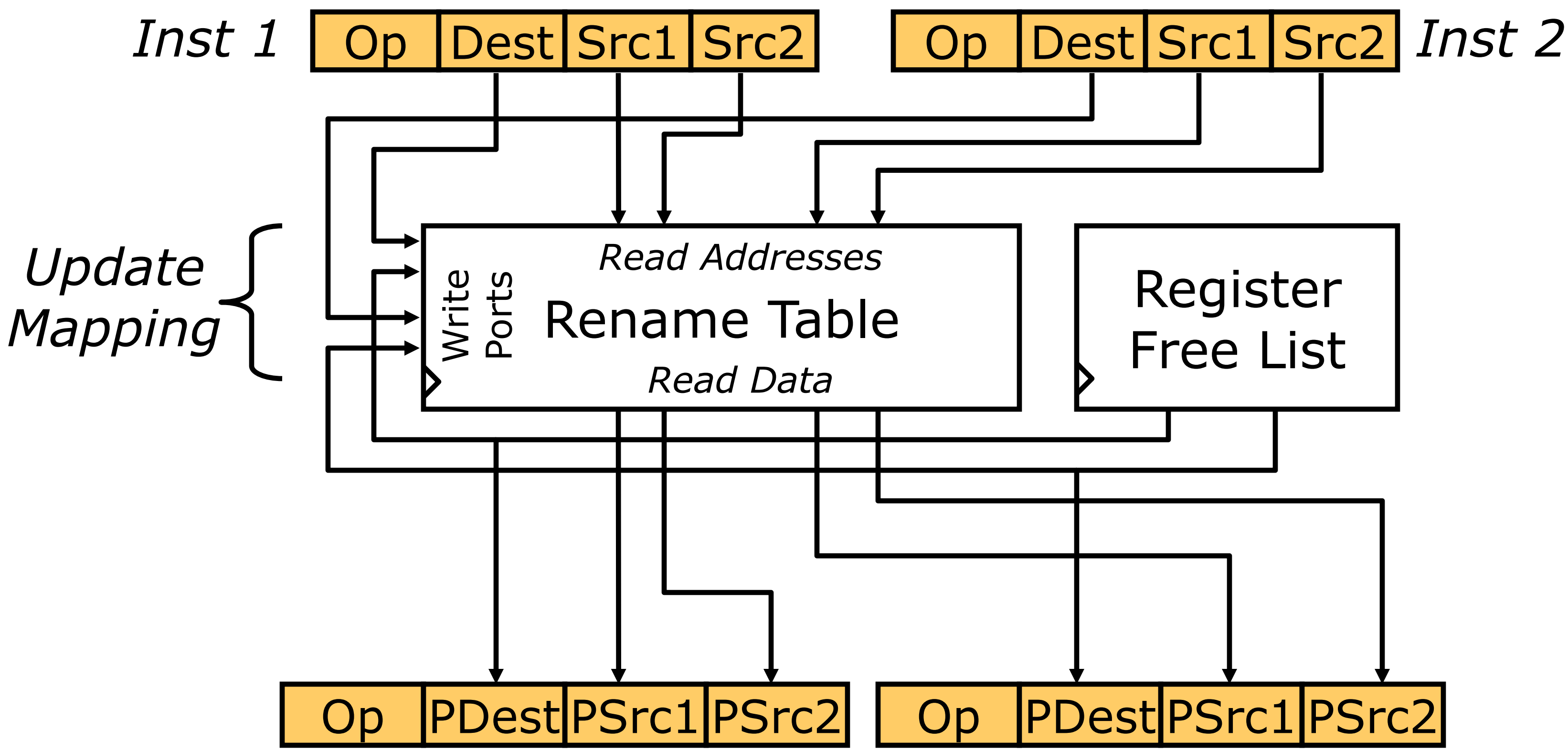
How does issue speculation differ, e.g., on cache miss?

Dependency loop shorter for data-in-ROB style

Thank you!

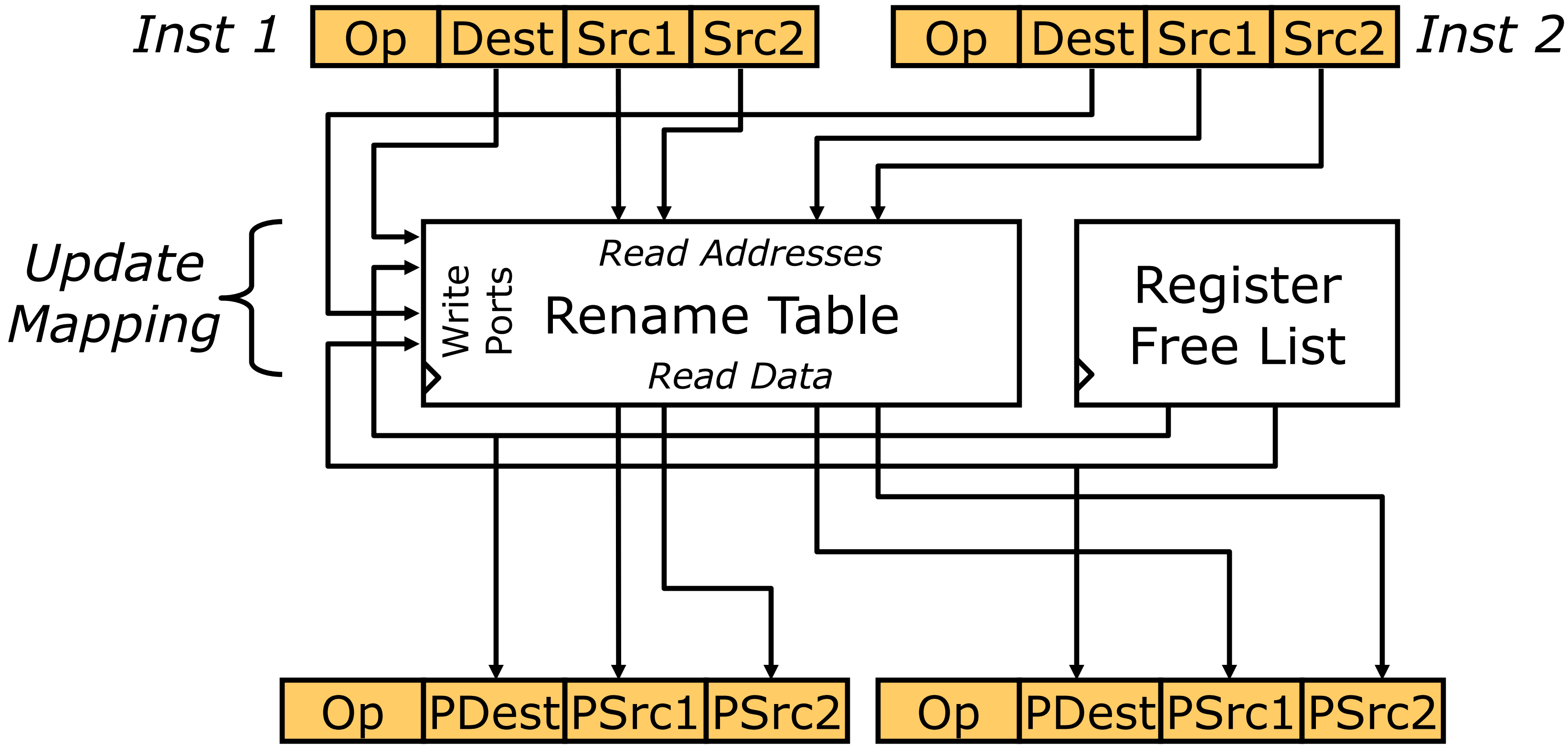
Superscalar Register Renaming

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



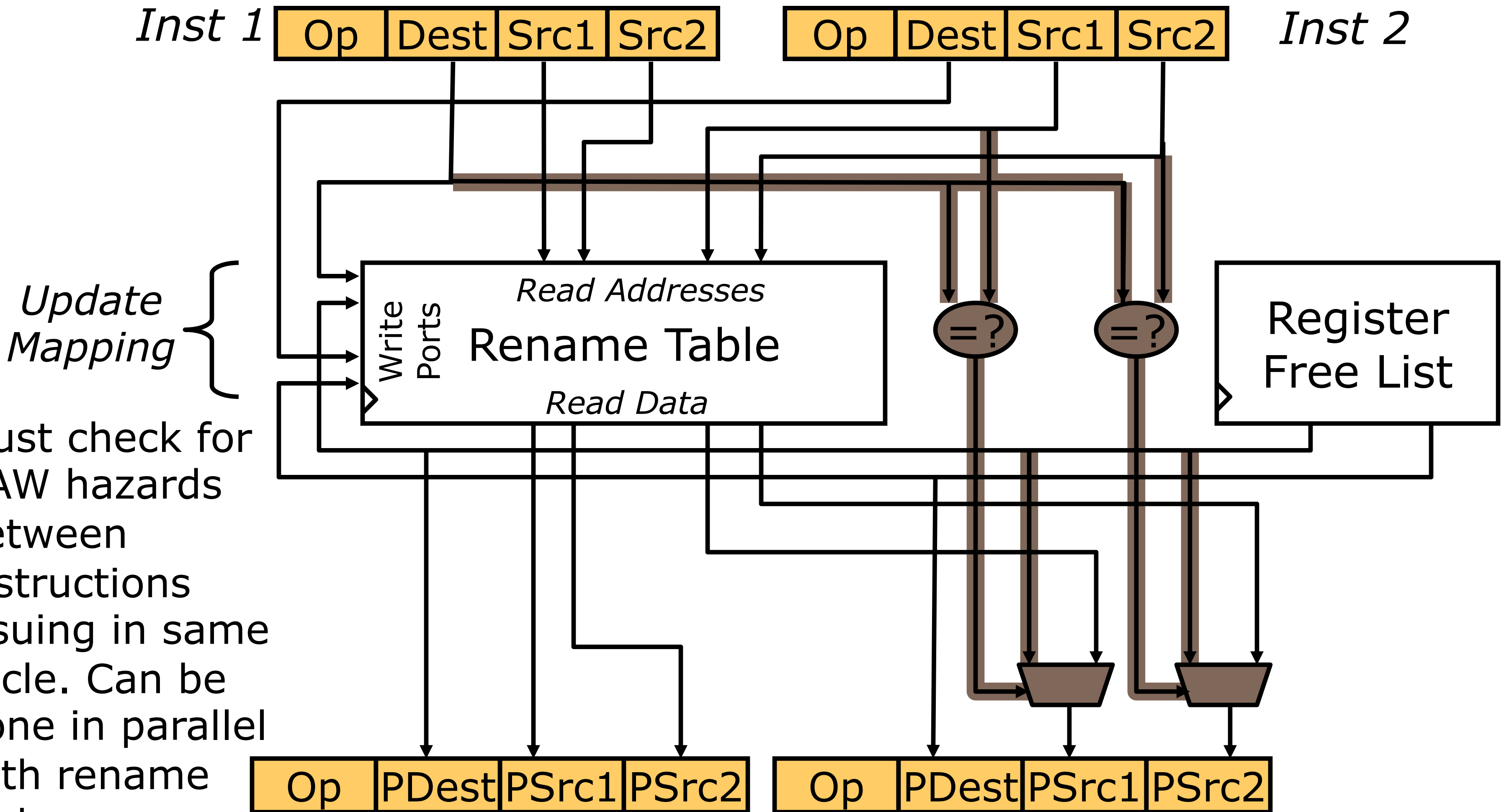
Superscalar Register Renaming

- 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)

Split Issue and Commit Queues

- How large should the ROB be?
  - Think Little's Law…

Split Issue and Commit Queues

- How large should the ROB be?
  - Think Little's Law…
- Can split ROB into issue and commit queues

Split Issue and Commit Queues

- How large should the ROB be?
  - Think Little's Law…
- Can split ROB into issue and commit queues

Issue Queue

use	op	p1	PR1	p2	PR2

Commit Queue

ex	Rd	LPRd	PRd

Split Issue and Commit Queues

- How large should the ROB be?
  - Think Little's Law…
- Can split ROB into issue and commit queues

Issue Queue

use	op	p1	PR1	p2	PR2

Commit Queue

ex	Rd	LPRd	PRd

- Commit queue: Allocate on decode, free on commit
- Issue queue: Allocate on decode, free on <u>dispatch</u>

Split Issue and Commit Queues

- How large should the ROB be?
  - Think Little's Law…
- Can split ROB into issue and commit queues

Issue Queue

use	op	p1	PR1	p2	PR2	tag

Commit Queue

ex	Rd	LPRd	PRd

- Commit queue: Allocate on decode, free on commit
- Issue queue: Allocate on decode, free on <u>dispatch</u>

Split Issue and Commit Queues

- How large should the ROB be?
  - Think Little's Law…
- Can split ROB into issue and commit queues

Issue Queue

use	op	p1	PR1	p2	PR2	tag

Commit Queue

ex	Rd	LPRd	PRd

- Commit queue: Allocate on decode, free on commit
- Issue queue: Allocate on decode, free on <u>dispatch</u>
- Pros: Smaller issue queue → simpler dispatch logic

Split Issue and Commit Queues

- How large should the ROB be?
  - Think Little's Law…
- Can split ROB into issue and commit queues

Issue Queue

use	op	p1	PR1	p2	PR2	tag

Commit Queue

ex	Rd	LPRd	PRd

- Commit queue: Allocate on decode, free on commit
- Issue queue: Allocate on decode, free on <u>dispatch</u>
- Pros: Smaller issue queue → simpler dispatch logic
- Cons: More complex mis-speculation recovery

Speculating Both Directions?

An alternative to branch prediction is to execute both directions of a branch *speculatively*

Speculating Both Directions?

An alternative to branch prediction is to execute both directions of a branch *speculatively*

- Resource requirement is proportional to the number of concurrent speculative executions

Speculating Both Directions?

An alternative to branch prediction is to execute both directions of a branch *speculatively*

- Resource requirement is proportional to the number of concurrent speculative executions

- Only half the resources engage in useful work when both directions of a branch are executed speculatively

Speculating Both Directions?

An alternative to branch prediction is to execute both directions of a branch *speculatively*

- Resource requirement is proportional to the number of concurrent speculative executions

- Only half the resources engage in useful work when both directions of a branch are executed speculatively

- Branch prediction takes less resources than speculative execution of both paths

Speculating Both Directions?

An alternative to branch prediction is to execute both directions of a branch *speculatively*

- Resource requirement is proportional to the number of concurrent speculative executions
- Only half the resources engage in useful work when both directions of a branch are executed speculatively
- Branch prediction takes less resources than speculative execution of both paths

With accurate branch prediction, it is more cost effective to dedicate all resources to the predicted direction