Instruction Level Parallelism

Explicit register renaming

Explicit Register Renaming

- Tomasulo provides ***Implicit Register Renaming***
  - User registers renamed to reservation station tags
- ***Now we introduce Explicit Register Renaming***:
  - Use ***physical*** register file that is larger than number of registers specified by the ISA
- Key insight: Allocate a new physical destination register for every instruction that writes
  - Very similar to a compiler transformation called Static Single Assignment (SSA) form — but in hardware!
  - Removes all chance of WAR or WAW hazards
  - Like Tomasulo, good for allowing full out-of-order completion
  - Like hardware-based dynamic compilation?

Advanced Computer Architectures

Explicit register renaming (MIPS R10000 Style)

Map Table

PRF

register file

RF

R0	32bit	P0
R1		P3
…		
R31		

P0	32bit
P1	
P2	
P3	
…	
P61	
P62	

floating point register file

FP RF

F0	64bit	P30
F2		
…		
F30		

N.B. In order to map a FP register to a physical one, we need to use two physical ones (see size)

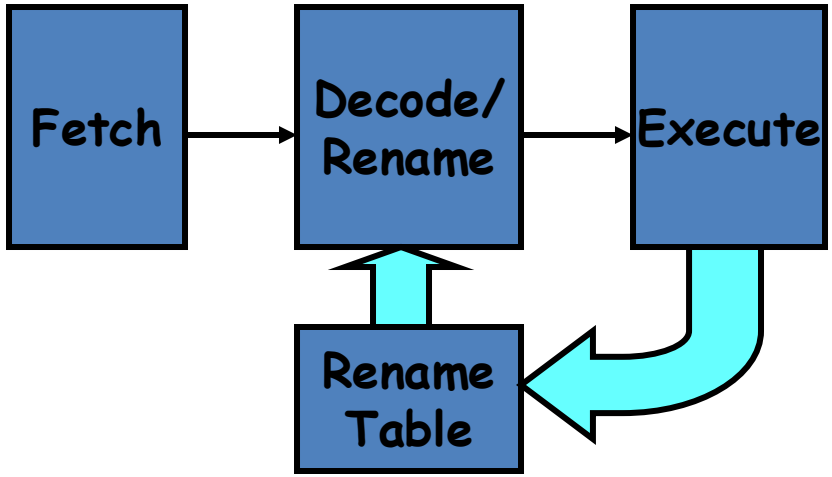
- **Physical Register File** larger than ISA Register File
- On issue, each instruction that writes a result is allocated new physical register from **Freelist**
- When a physical register P0 is "dead" (or not "live"), we free up

Freelist

P32	P34	P36	P38	…	P60	P62

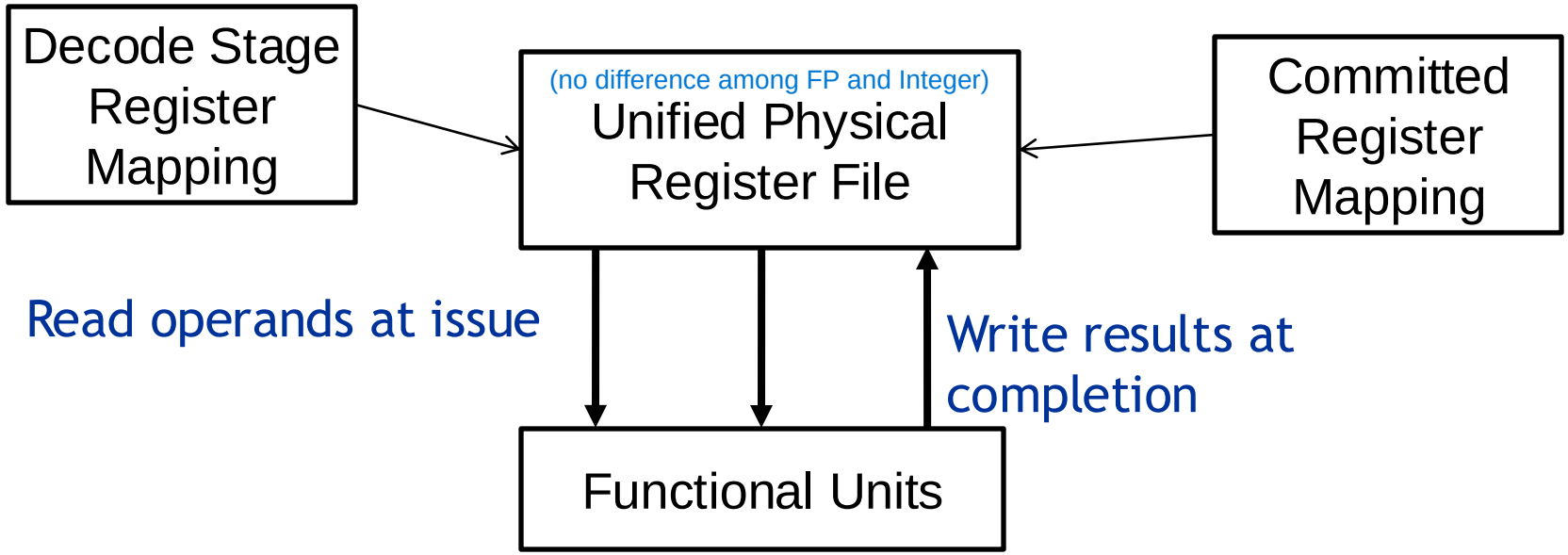
Explicit Register Renaming

- Mechanism? Keep a translation table:
  - ISA register ➔ physical register mapping
  - When register written, replace entry with new register from freelist
  - Physical register becomes free when not used by any active instructions



Unified Physical Register File

- 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 execution
- Commit only updates mapping of architectural register to physical register, no data movement



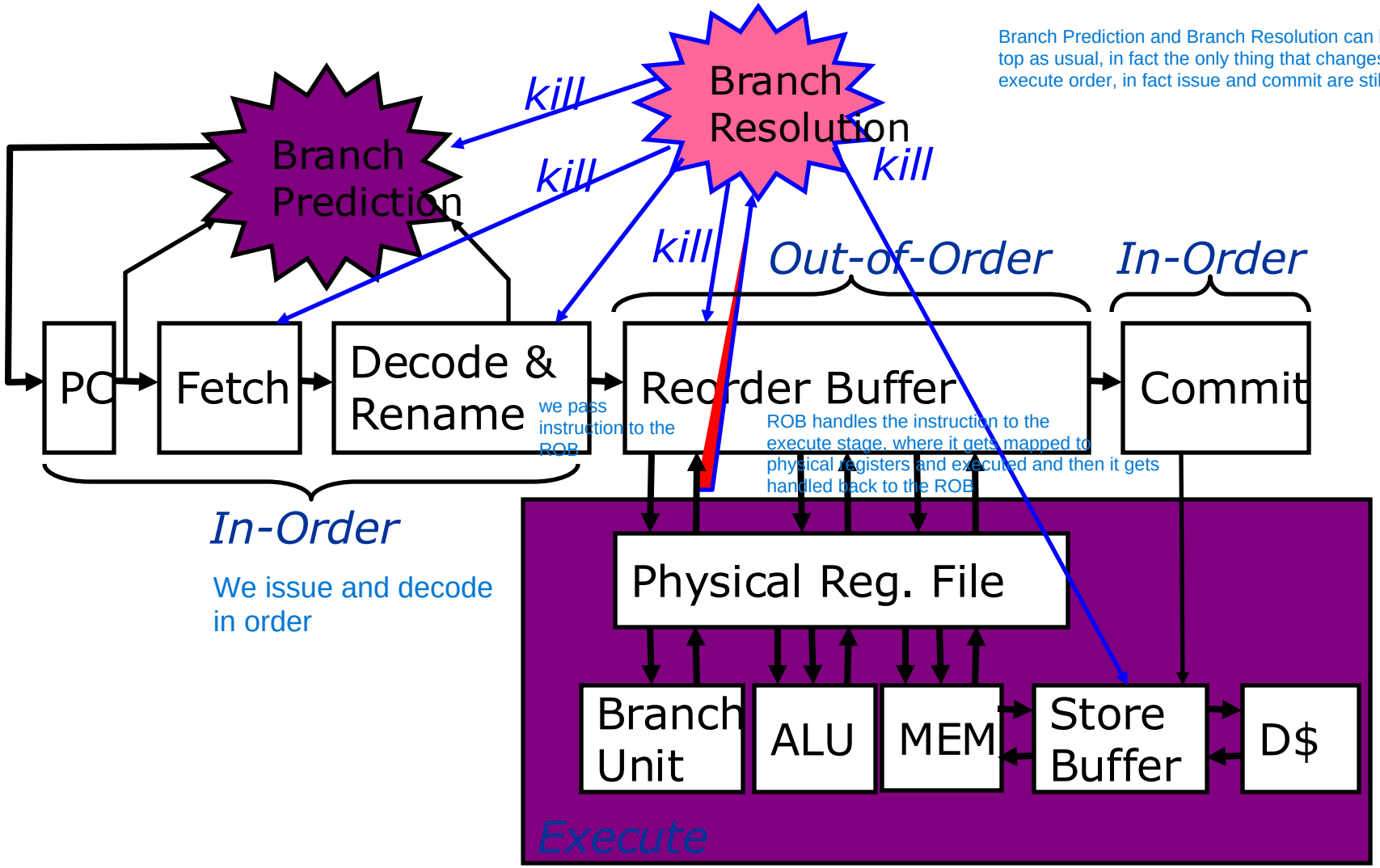
Instruction Commit

- Record that the mapping between an architectural register number and physical register number is no longer speculative
- Free up any physical registers being used to hold the “older” value of the architectural register
- Deallocating registers is more complicated:
  - Before freeing up a physical register we must know that it no longer corresponds to an architectural register and that no further uses of the physical register are outstanding
  - A physical register corresponds to an architectural register until the architectural register is **rewritten**
  - However, there may be uses of the physical register outstanding. The processor must check if any source operand corresponds to that register in the functional units queue. If it does not appear then it can be deallocated.
    - Alternatively the processor can simply wait until another instr. that writes the same architectural register commits. This may tie up a physical register slightly longer than necessary, but it is easy to implement

Hw register renaming

- **Renaming map**: simple data structure that supplies the physical register number of the register that currently corresponds to the requested architectural register
- **Instruction commit**: update permanently the renaming table to indicate that the physical register holding the destination value corresponds to the actual architectural register
- Use ROB to enforce in-order commit

Pipeline Design with Physical Regfile



Advantages of Explicit Renaming

- Decouples *renaming* from *scheduling:*
  - Pipeline can be exactly like "standard" DLX pipeline (perhaps with multiple operations issued per cycle)
  - Or, pipeline could be tomasulo-like or a scoreboard, etc.
  - Standard forwarding or bypassing could be used
- Allows data to be fetched from single register file
  - No need to bypass values from reorder buffer
  - This can be important for balancing pipeline
- Many processors use a variant of this technique:
  - R10000, Alpha 21264, HP PA8000

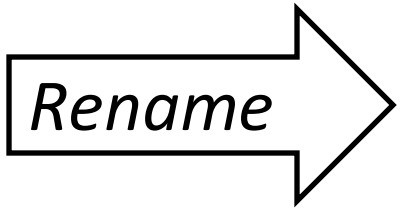
Explicit Renaming Support

- Rapid access to a table of translations
- A physical register file that has more registers than specified by the ISA
- Ability to figure out which physical registers are free.
  - No free registers ➔ stall on issue
- Thus, register renaming doesn't require reservation stations. However:
  - Many modern architectures use explicit register renaming + Tomasulo-like reservation stations to control execution.
- Two Questions:
  - How do we manage the "free list"?
  - How does Explicit Register Renaming mix with Precise Interrupts?

Lifetime of Physical Registers

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

```
ld x1, (x3)                ld P1, (Px)
addi x3, x1, #4            addi P2, P1, #4
sub x6, x7, x9             sub P3, Py, Pz
add x3, x3, x6             add P4, P2, P3
ld x6, (x1)                ld P5, (P1)
add x6, x6, x3             add P6, P5, P4
sd x6, (x1)                sd P6, (P1)
ld x6, (x11)               ld P7, (Pw)
```



When can we reuse a physical register?

*When **next** writer of same architectural register commits*

Advanced Computer Architectures

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)
```

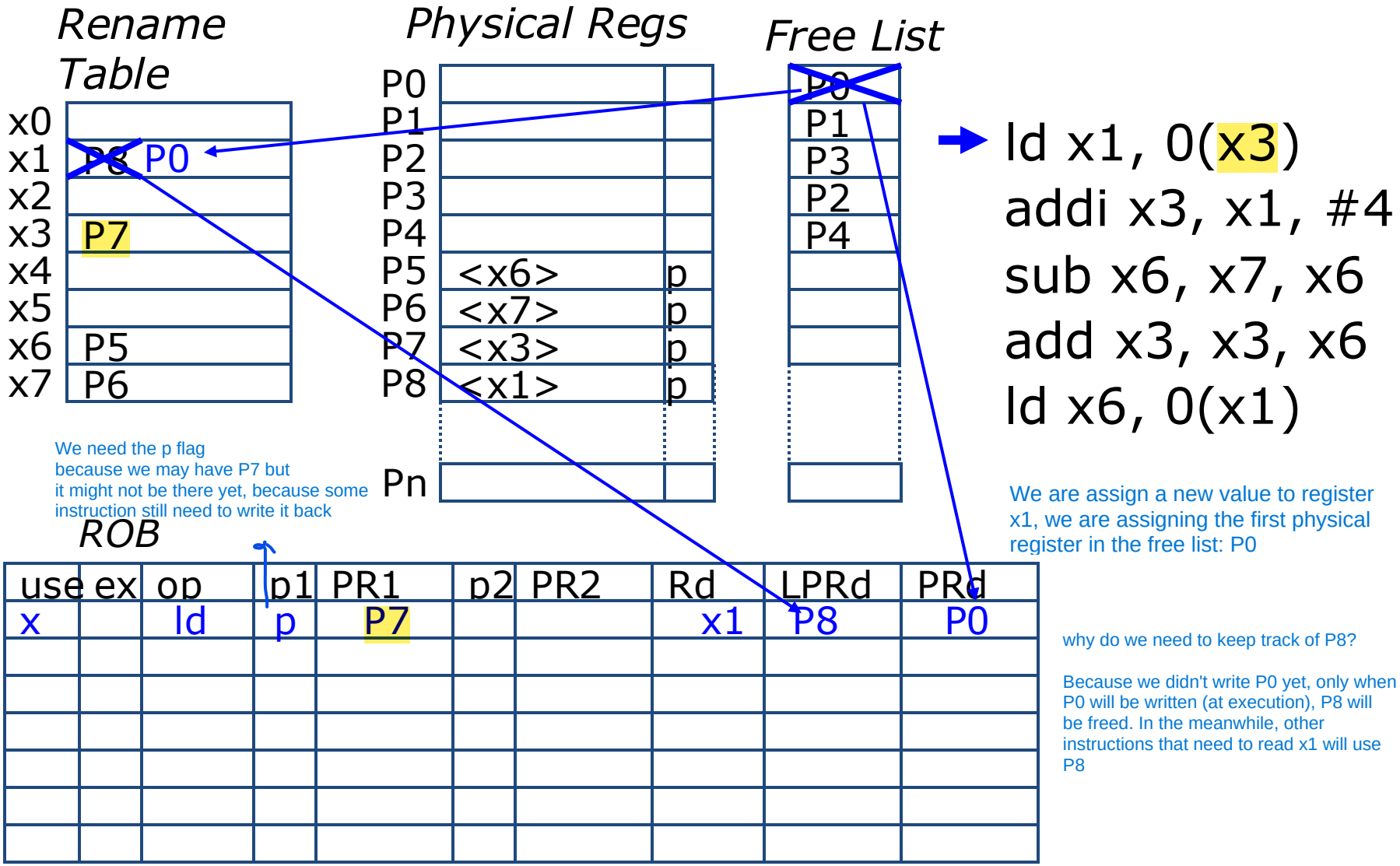
ROB

			source1		source2			old	new
use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd

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

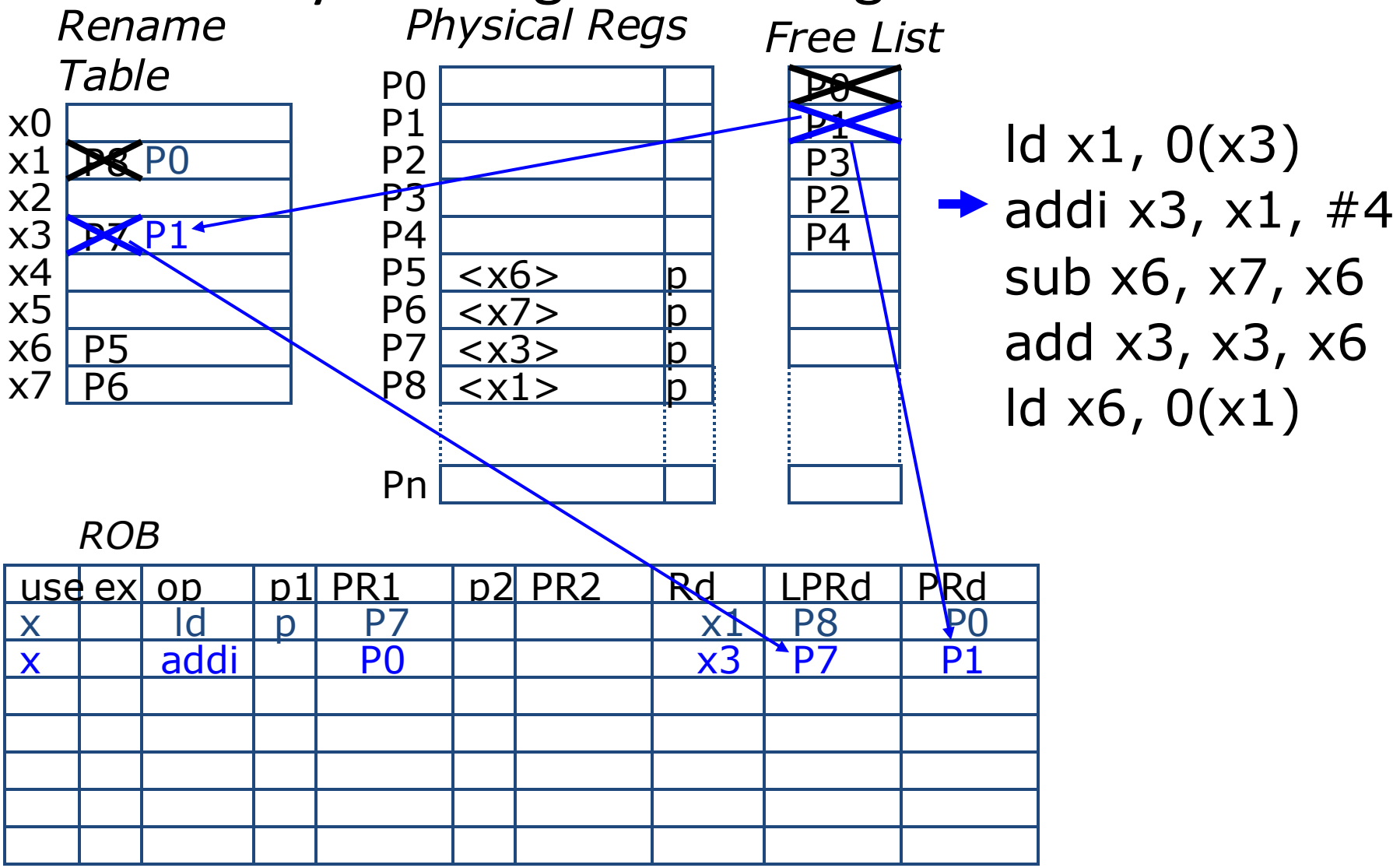
Advanced Computer Architectures

Physical Register Management



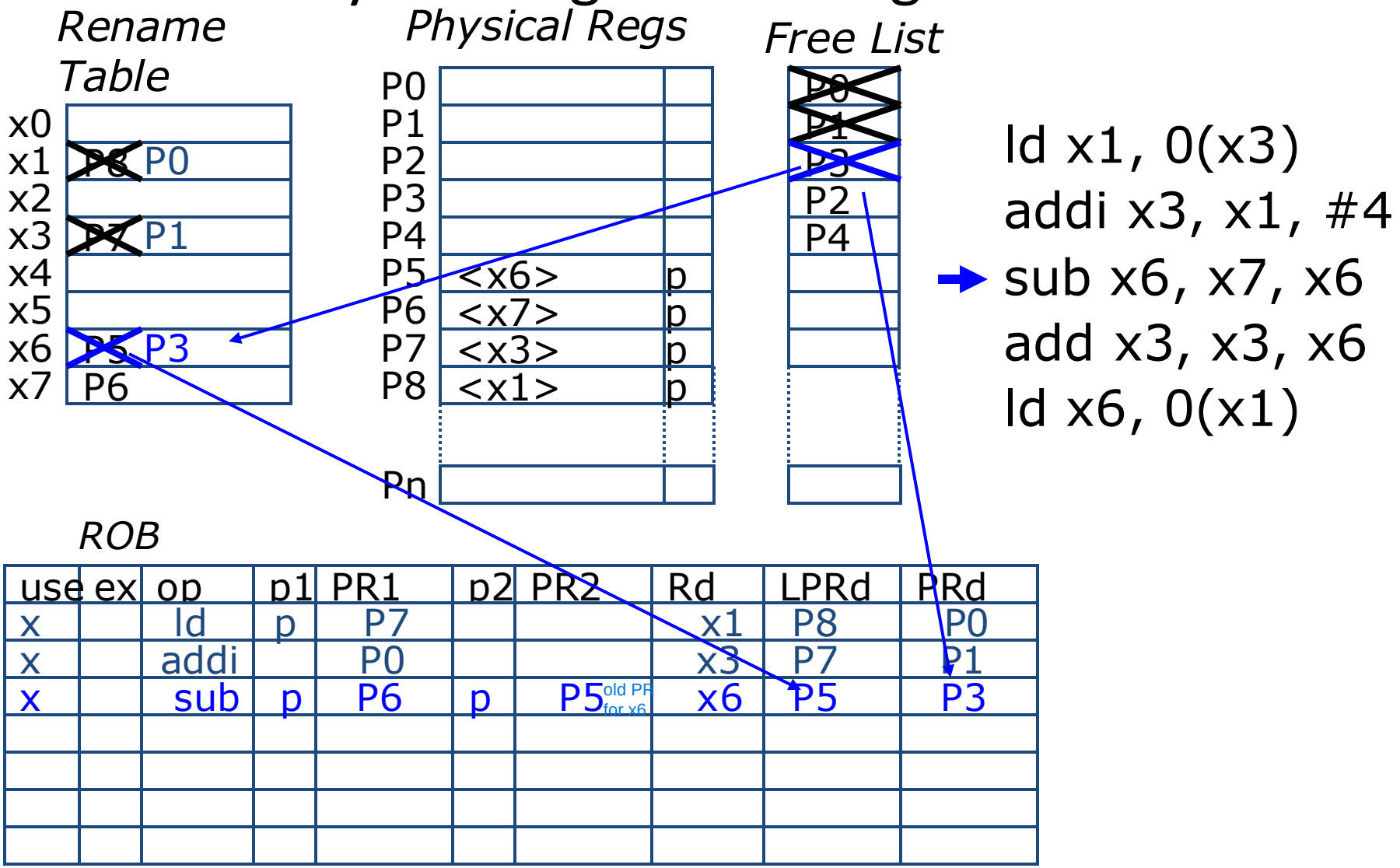
Advanced Computer Architectures

Physical Register Management



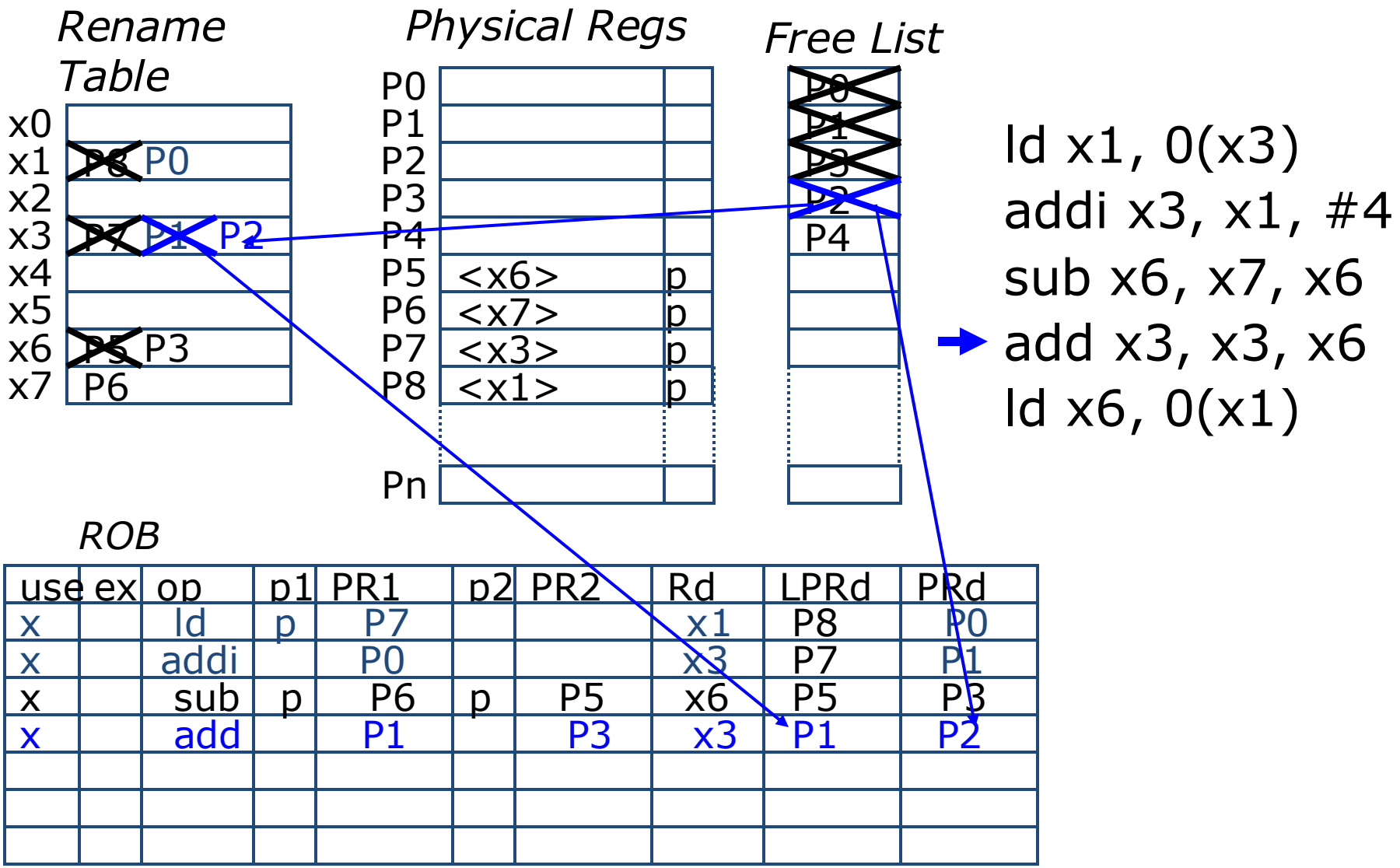
use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		ld	p	P7			x1	P8	P0
x		addi		P0			x3	P7	P1

Physical Register Management



Advanced Computer Architectures

Physical Register Management

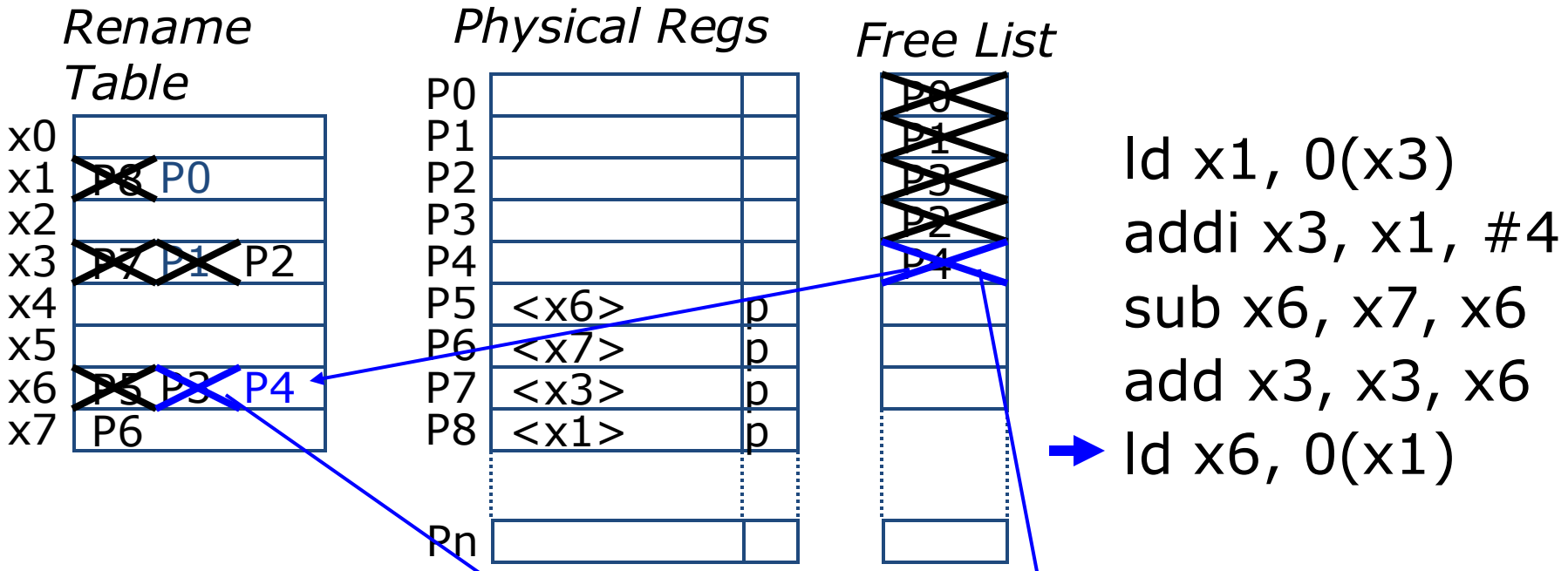


ROB

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

Advanced Computer Architectures

Physical Register Management

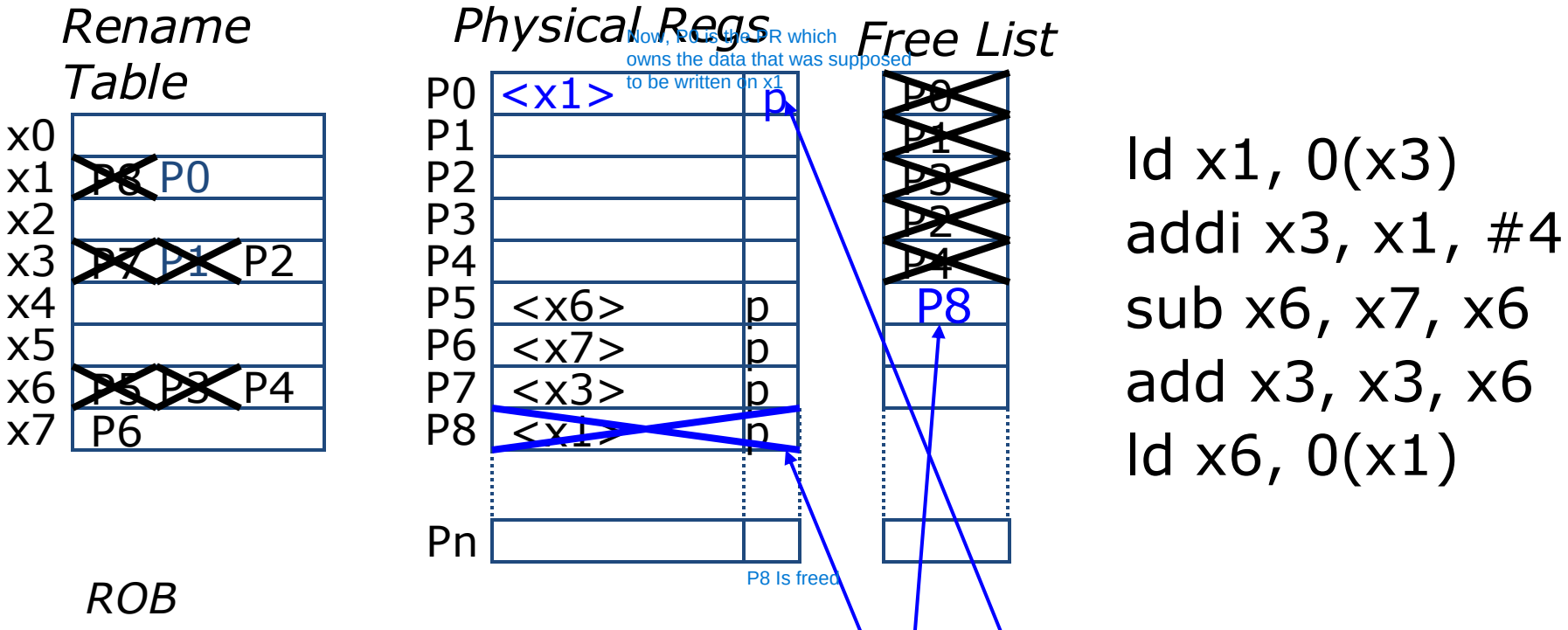


ROB

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
x		ld		P0			x6	P3	P4

Advanced Computer Architectures

Physical Register Management




```
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
x	x	ld	p	P7			x1	P8	**P0**
x		addi	p	P0			x3	P7	P1
x		sub	p	P6	p	P5	x6	P5	P3
x		add		P1		P3	x3	P1	P2
x		ld	p	P0			x6	P3	P4

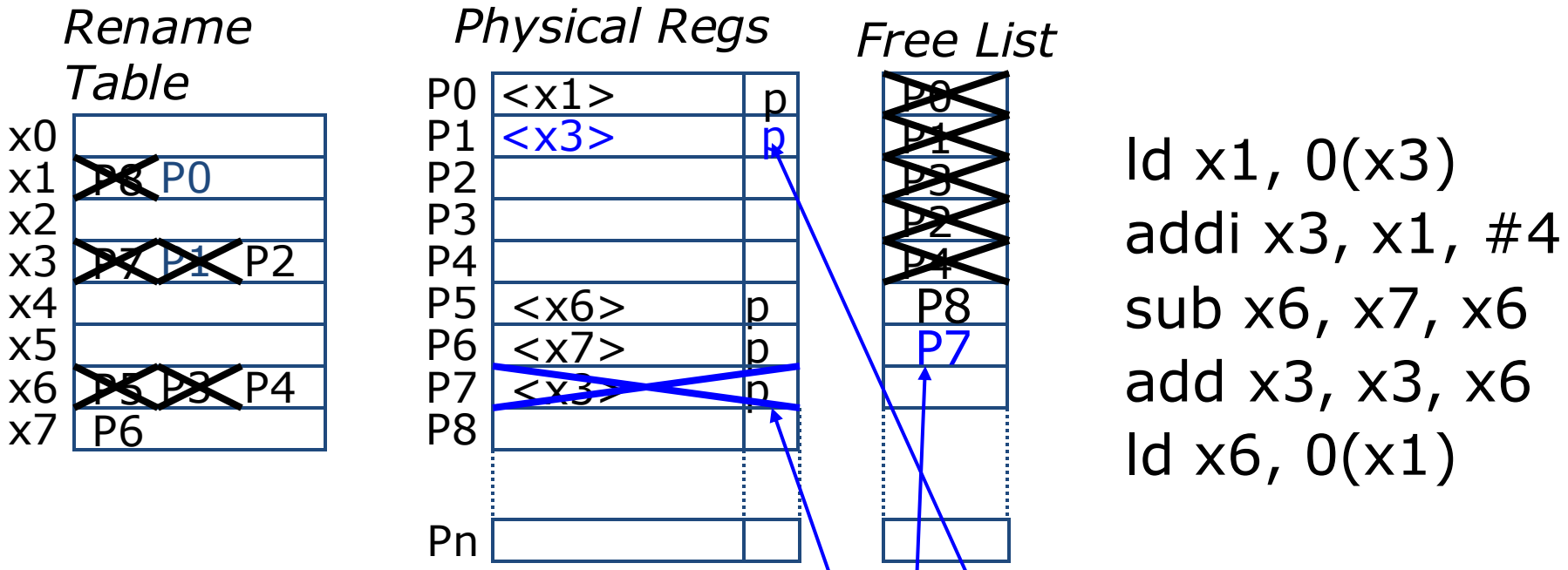
Execute & Commit

As soon as the first ld writes back, the new PR of x1, that is P0, now contains the valid value and so it gets marked as physical available in the table

That is, we are informing all the guys waiting for the data that the data is now available.

Advanced Computer Architectures

Physical Register Management

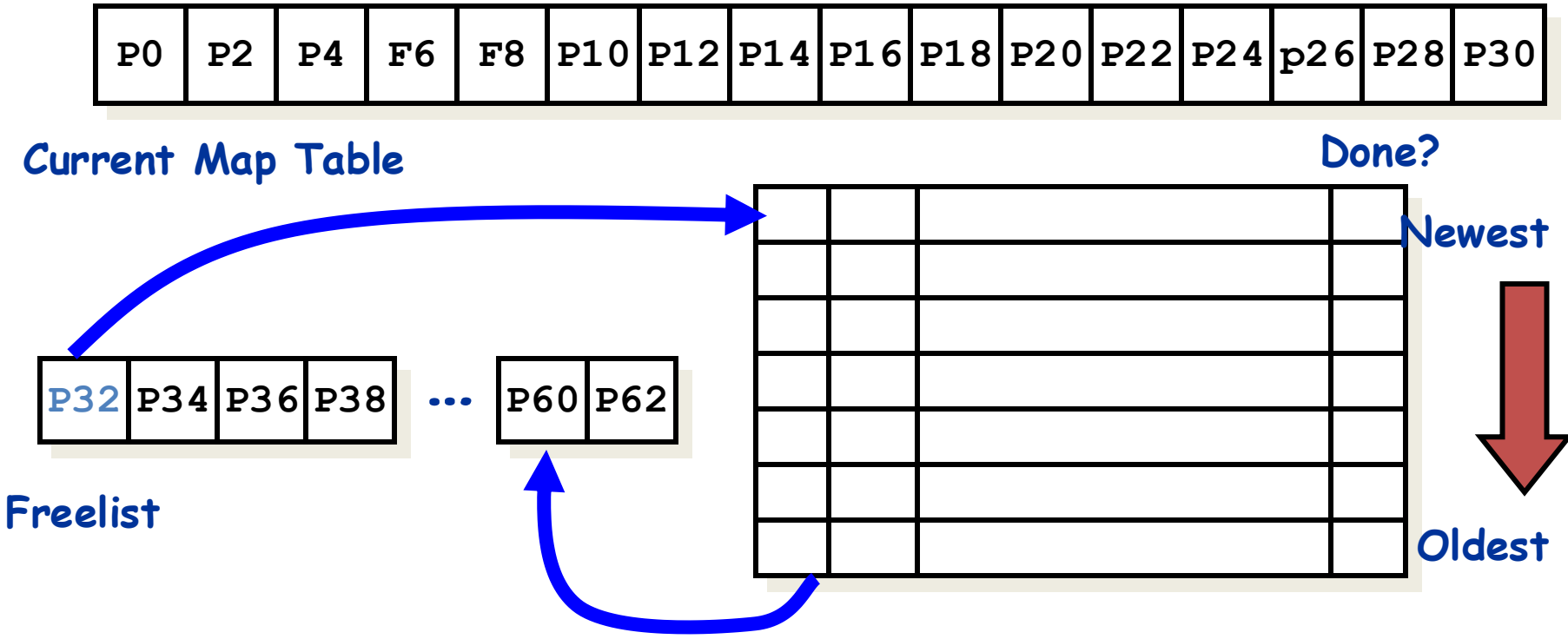


ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	ld	p	P7			x1	P8	P0
x	x	addi	p	P0			x3	P7	P1
x		sub	p	P6	p	P5	x6	P5	P3
x		add	p	P1		P3	x3	P1	P2
x		ld	p	P0			x6	P3	P4

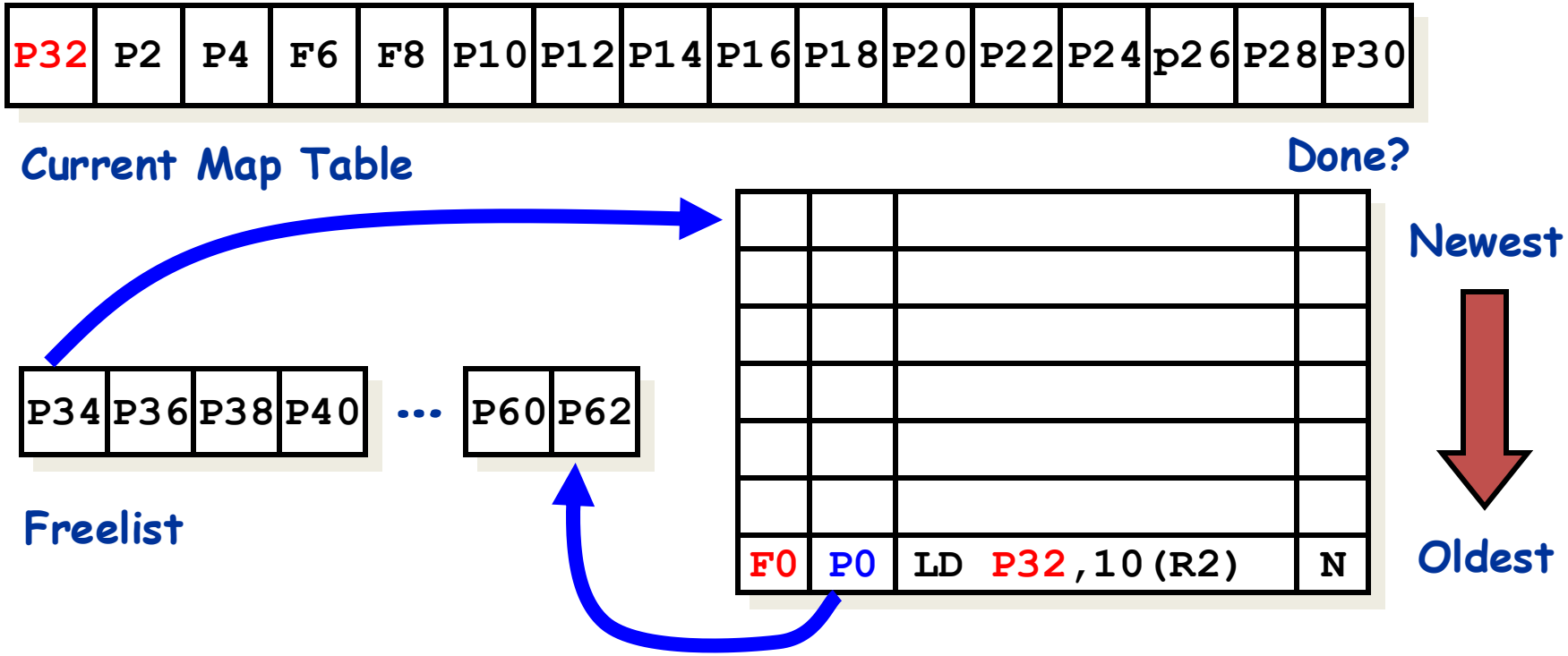
Execute & Commit

Explicit register renaming (MIPS R10000 Style)



- Physical register file larger than ISA register file
- On issue, each instruction that modifies a register is allocated new physical register from freelist

Explicit register renaming: (MIPS R10000 Style)

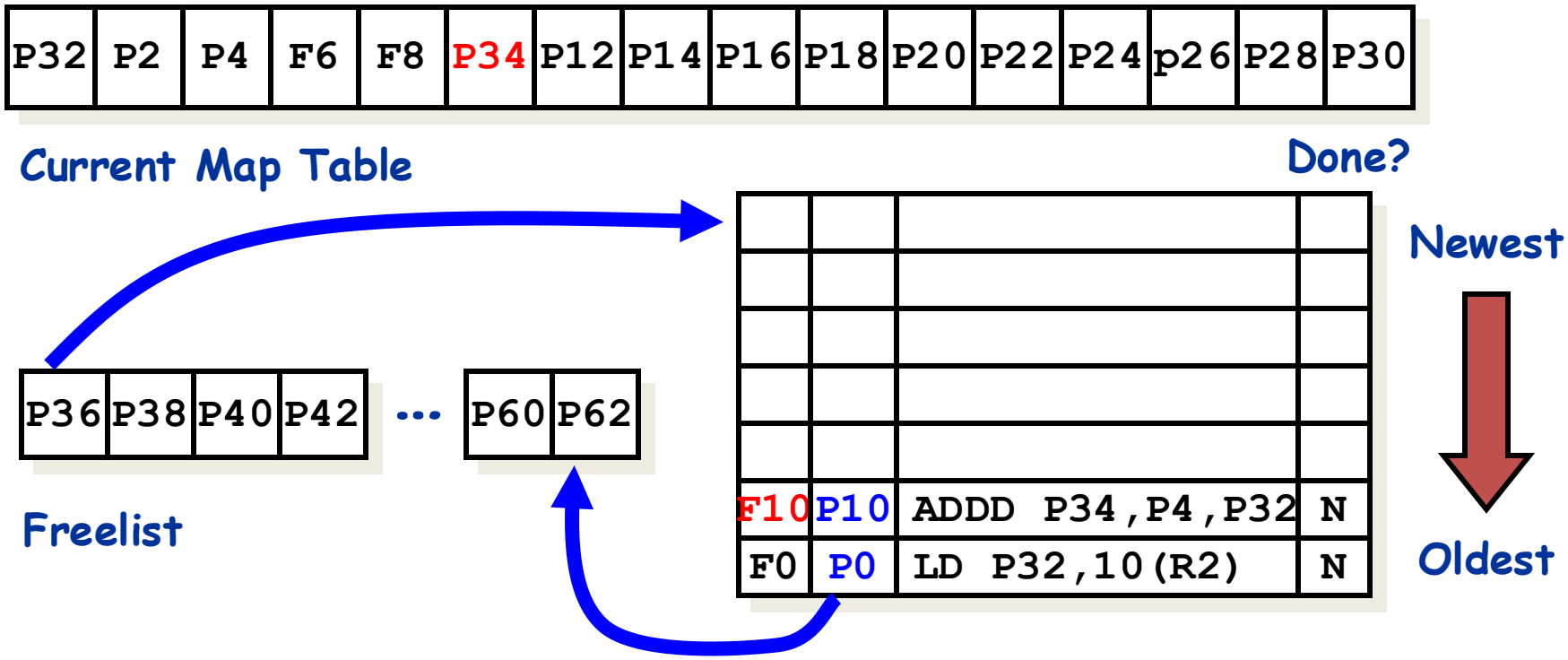


Note that physical register P0 is "dead" (or not "live") past the point of this load.

When we go to commit the load, we free up

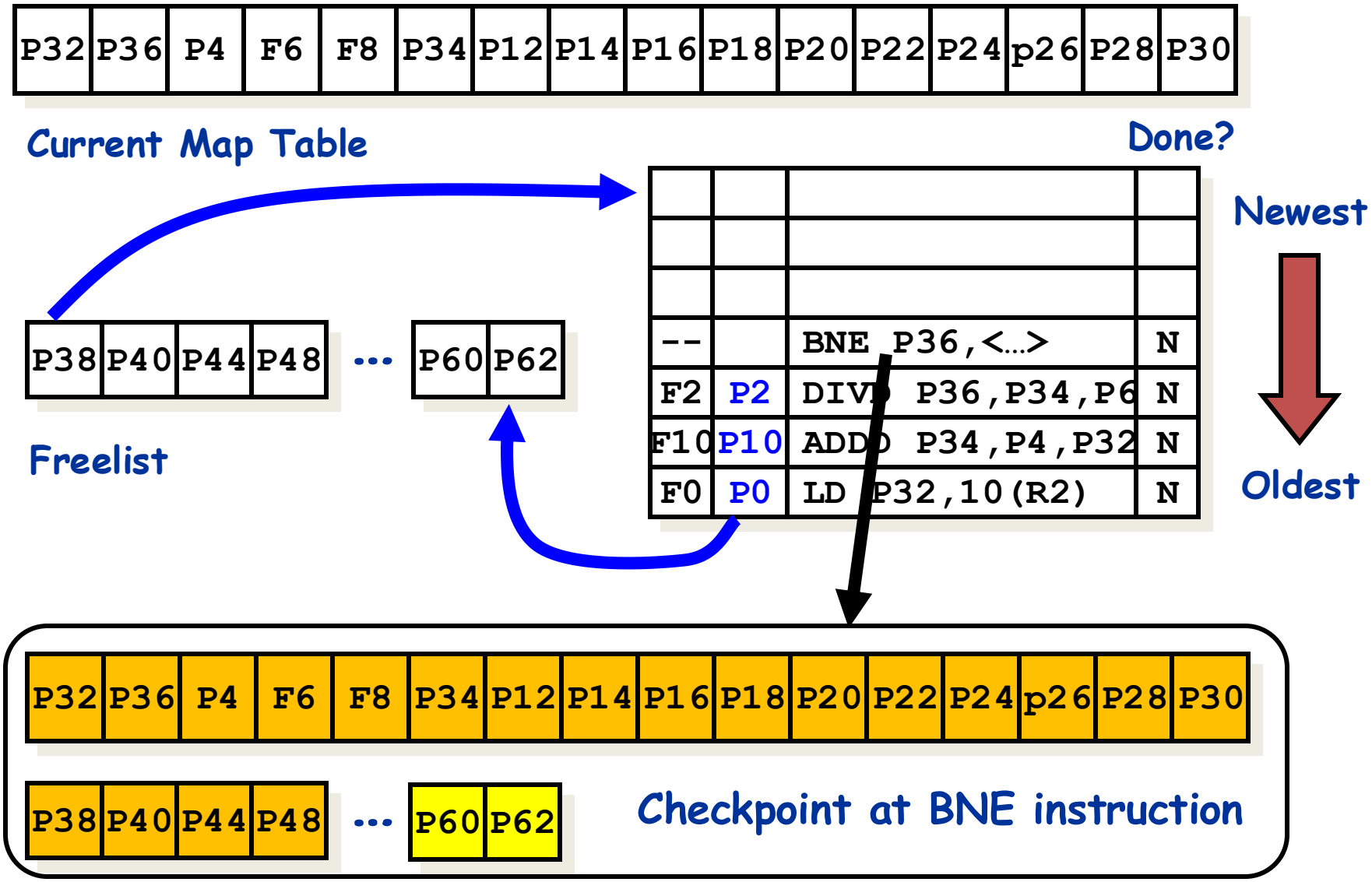
Advanced Computer Architectures

Explicit register renaming: (MIPS R10000 Style)



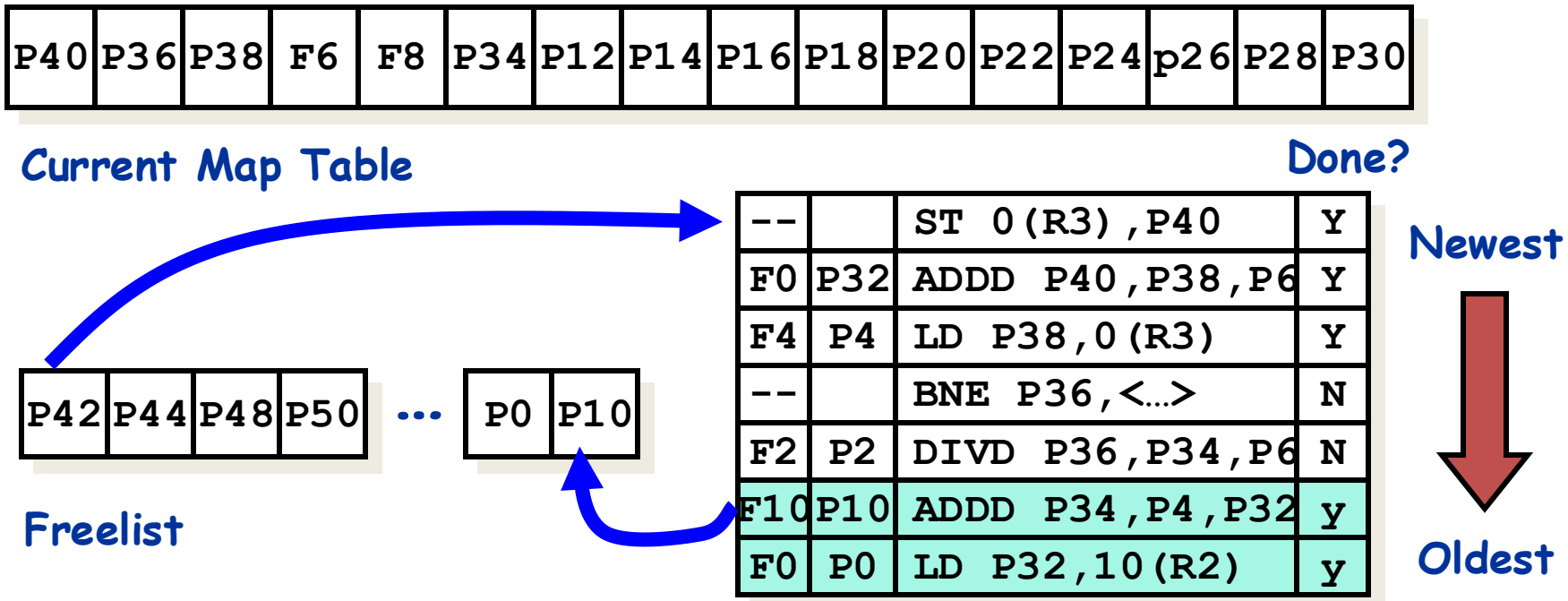
Advanced Computer Architectures

Explicit register renaming: (MIPS R10000 Style)

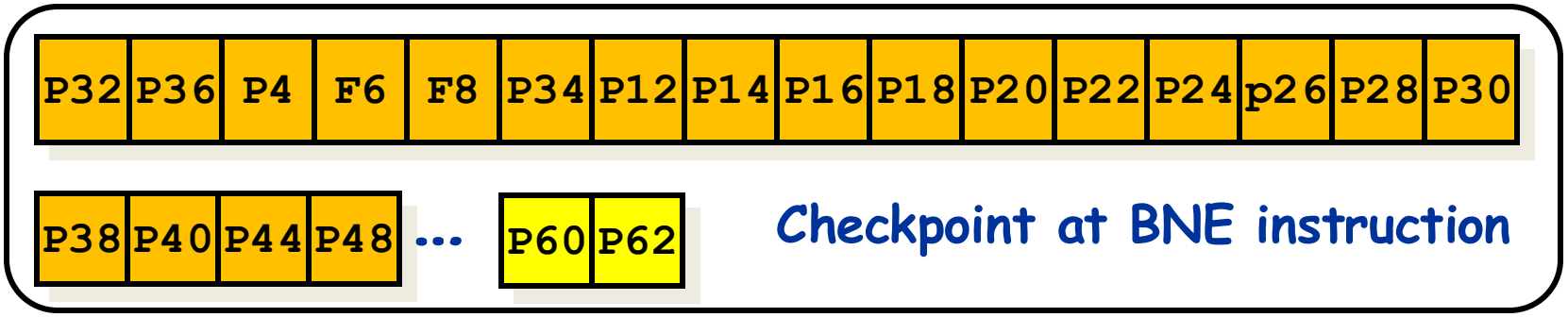


Advanced Computer Architectures

Explicit register renaming: (MIPS R10000 Style)

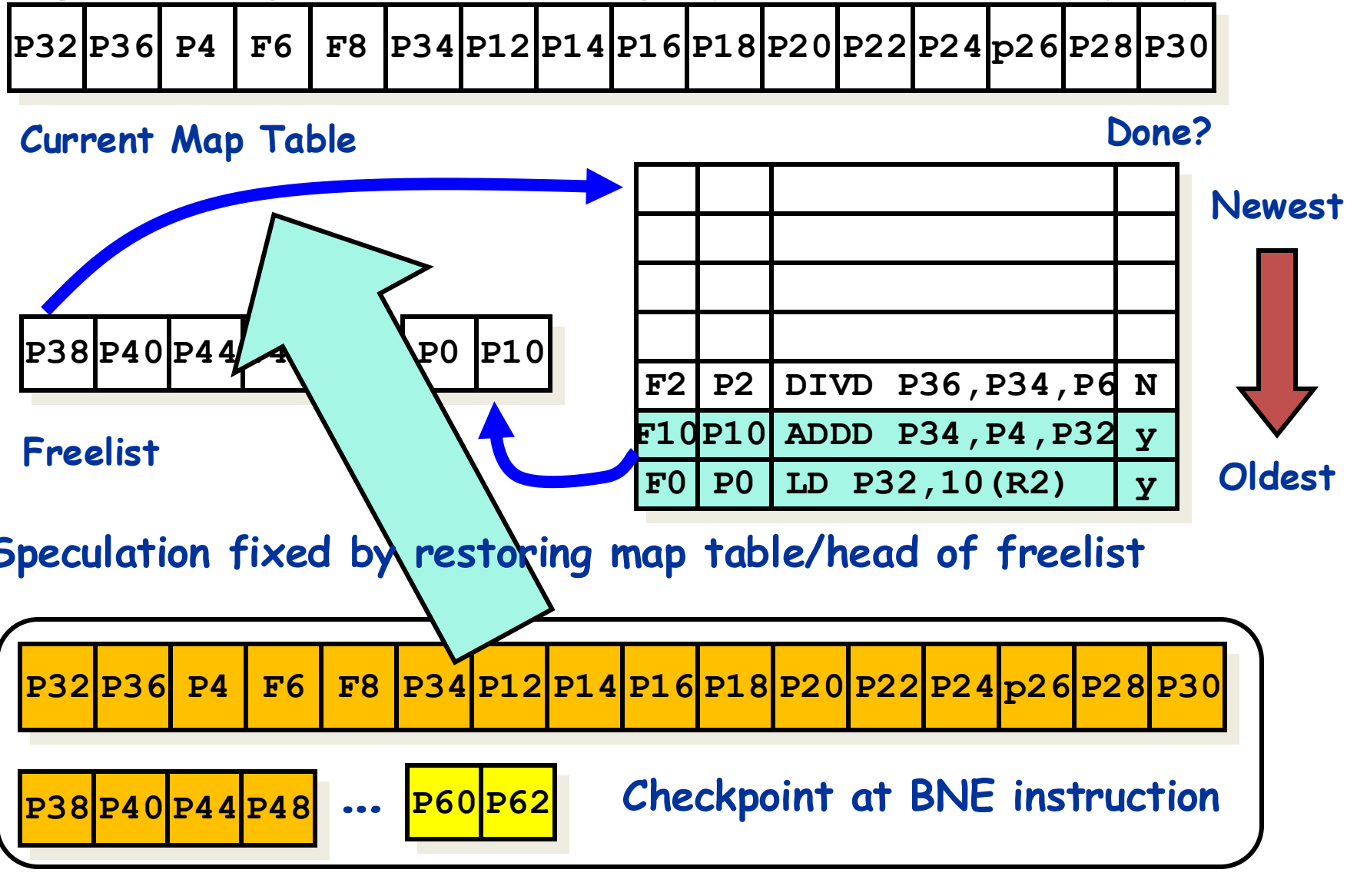


What happens if erroneous speculation?



Advanced Computer Architectures

Explicit register renaming: (MIPS R10000 Style)



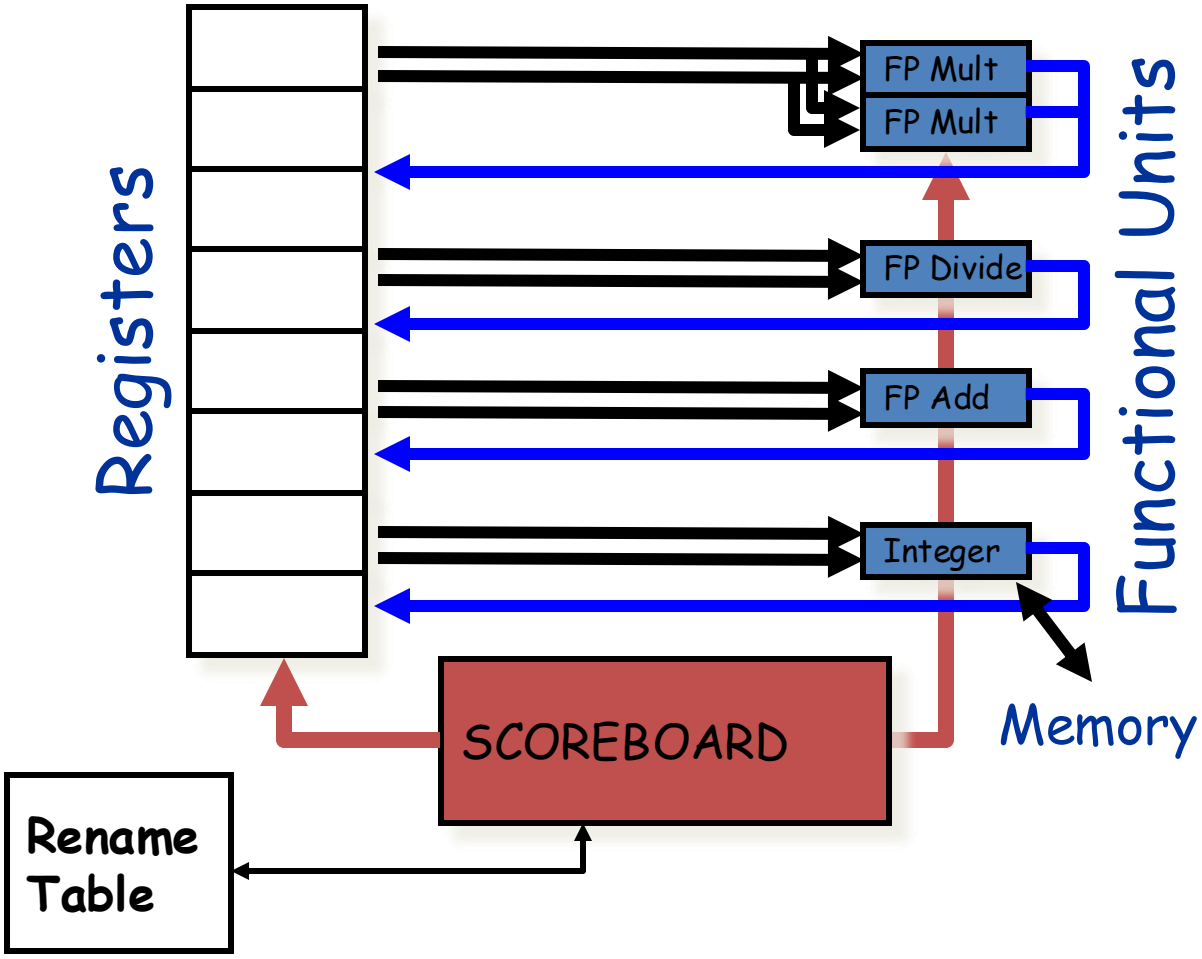
Explicit Register Renaming

- Tomasulo provides *Implicit Register Renaming*
  - User registers renamed to reservation station tags
- Explicit Register Renaming:
  - Use *physical* register file that is larger than number of registers specified by ISA
- Keep a translation table:
  - ISA register => physical register mapping
  - When register is written, replace table entry with new register from freelist.
  - Physical register becomes free when not being used by any instructions in progress.
- Pipeline can be exactly like "standard" DLX pipeline
  - IF, ID, EX, etc….
- Advantages:
  - Removes all WAR and WAW hazards
  - Like Tomasulo, good for allowing full out-of-order completion
  - Allows data to be fetched from a single register file
  - Makes speculative execution/precise interrupts easier:
    - All that needs to be "undone" for precise break point is to undo the table mappings

26

Advanced Computer Architectures

Question: Can we use explicit register renaming with scoreboard?



Stages of Scoreboard Control With Explicit Renaming

N.B. A difference wrt the base version of scoreboard (without register renaming) is that we are not checking anymore for WAW, this because it is no more a problem

- Issue—decode instructions & check for structural hazards & allocate new physical register for result
  - Instructions issued in program order (for hazard checking)
  - Don't issue if no free physical registers
  - Don't issue if structural hazard
- Read operands—wait until no hazards, read operands
  - All real dependencies (RAW hazards) resolved in this stage, since we wait for instructions to write back data.
- Execution—operate on operands
  - The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard
- Write result —finish execution

In base version of scoreboard we had to check for WAR at this stage, again now it is not a problem

- Note: No checks for WAR or WAW hazards!

Scoreboard Example

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock	*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
FU	P0	P2	P4	P6	P8	P10	P12		P30

Initialized Rename Table – registers from P32 in the free list

Renamed Scoreboard 1

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU	P0	P2	P4	P32	P8	P10	P12		P30

Each instruction allocates free register

black means there is a "P" in the renaming table (that means physical available.
Red means no "p"

Advanced Computer Architectures

Renamed Scoreboard 2

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2		
LD	F2	45+	R3	2			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
2	*FU*	P0	P34	P4	P32	P8	P10	P12		P30

Renamed Scoreboard 3

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3	2	3		
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	No								
	Divide	No								

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
3	*FU*	P36	P34	P4	P32	P8	P10	P12		P30

Advanced Computer Architectures

Renamed Scoreboard 4

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

As always, the read of a written data can happens only in the clock cycle next to the one of the WB

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	Yes	Sub	P38	P32	P34		Int2	Yes	No
	Divide	No								

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
4	*FU*	P36	P34	P4	P32	P38	P10	P12		P30

Renamed Scoreboard 5

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
5	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 6

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6		
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
10	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
6	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 7

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6		
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
9	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
7	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 8

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
8	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
8	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 9

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
7	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 10

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10			

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
6	Mult1	Yes	Multd	~~P36~~	P34	~~P4~~			Yes	Yes
	Add	Yes	Addd	P42	P38	~~P34~~			Yes	Yes
	Divide	Yes	Divd	~~P40~~	P36	P32	Mult1		No	Yes

WAR Hazard gone!

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
10	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Notice that P32 not listed in Rename Table
Still live. Must not be reallocated by accident

Renamed Scoreboard 11

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
5	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
11	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

40

Renamed Scoreboard 12

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
4	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	*...*	*F30*
12	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 13

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
3	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 14

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
2	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
14	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

43

Renamed Scoreboard 15

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
1	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
15	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

44

Renamed Scoreboard 16

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
0	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
16	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 17

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
17	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

46

Renamed Scoreboard 18

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5	18		
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
40	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
18	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Register renaming vs. ROB

- Instruction commit simpler than with ROB;
- Deallocating registers more complex;
- Dynamic mapping of architectural to physical registers complicates design and debugging;
- Used in PowerPC603/604, Pentium II-III-4, MIPS 10000/12000, Alpha 21264; Sandy-Bridge
  - 20 to 80 registers are added.

48

Summary

- Explicit Renaming: more physical registers than needed by ISA.
  - Rename table: tracks current association between architectural registers and physical registers
  - Uses a translation table to perform compiler-like transformation on the fly
- With Explicit Renaming:
  - All registers concentrated in single register file
  - Can utilize bypass network that looks more like 5-stage pipeline
  - Introduces a register-allocation problem
    - Need to handle branch misprediction and precise exceptions differently, but ultimately makes things simpler
- For precise exceptions and branch prediction:
  - Clearly need something like reorder buffer

49

We want to go beyond CPI=1, this can only be done with multiple issue

Multiple issue

- Necessary to have the issue logic to handle two or more instructions at once, including possible dependences between instructions
- Most fundamental bottleneck in dynamically scheduled superscalar processors
  - Need logic to handle issuing every possible combination of dependent instructions in the same clock cycle
  - Since the number of possibilities increases with the square of the number of instructions that can be issued in one clock cycle, difficult to implement issue logic for more than 4 instructions

Basic strategy for multiple issue

- Assign a reservation station and a reorder buffer entry for every instruction in the next issue bundle
  - if not available only a subset of instructions is considered in sequential order
- Analyze all dependences among the instructions
- If an instruction in the bundle depends on an earlier instruction of the same bundle, use the assigned reorder buffer number to update the reservation table for the dependent instruction.

All this is done in parallel in a single clock cycle!

- In addition we need to be able to commit multiple instructions in a single clock cycle
- Intel I7 uses a similar technique

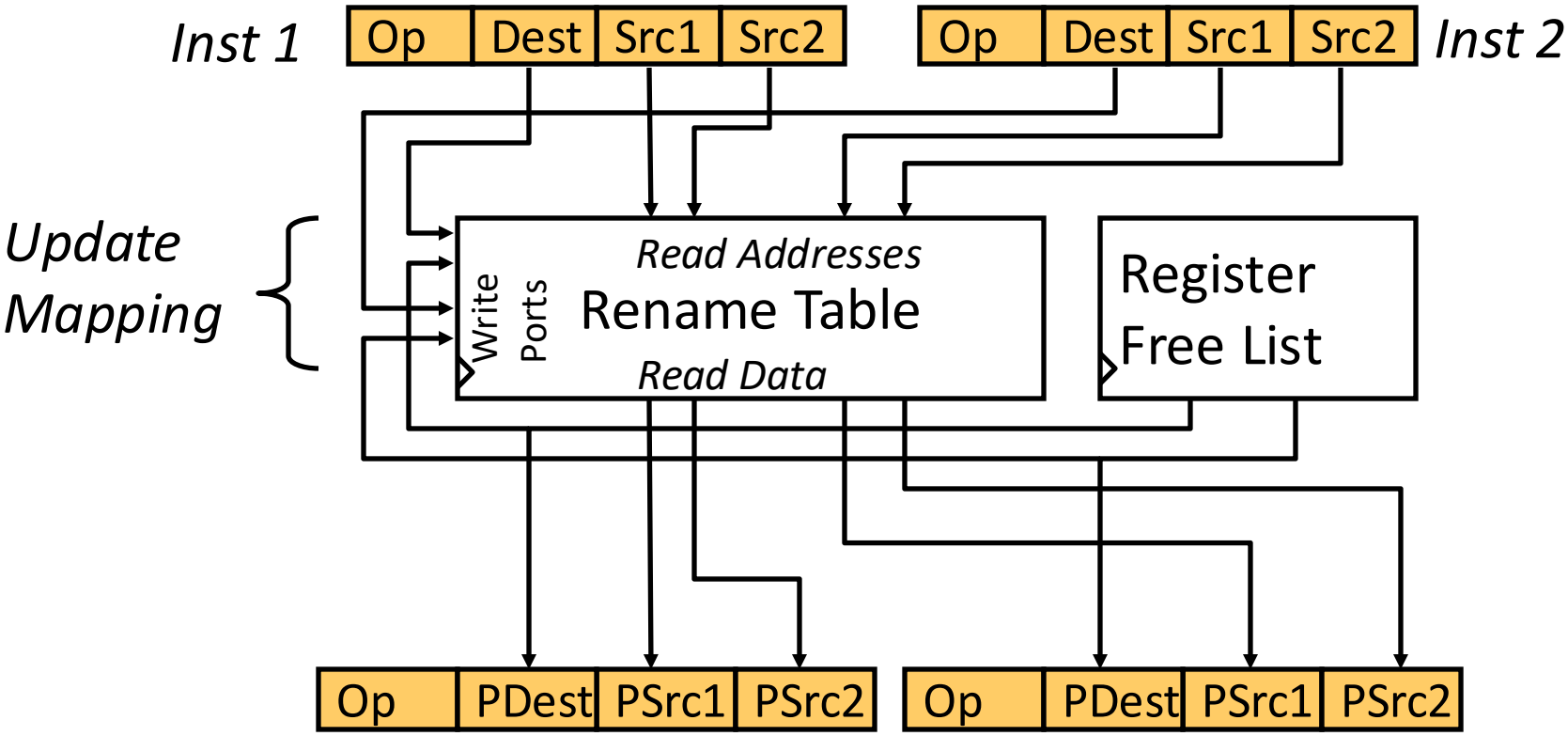
Multiple instructions issue with register renaming

- The issue logic pre-reserves enough physical registers for the entire issue bundle
- The issue logic determines what dependences exist in the bundle.
  - If a dependence does not exist within the bundle, the register renaming structure is used to determine the physical register that holds, or will hold, the result on which instruction depends. The result is from an earlier issue bundle, and the register renaming table will have the correct register number
  - If an instruction depends on an instruction that is earlier in the bundle, then the pre-reserved physical register in which the result will be placed is used to update the information for the issuing instruction

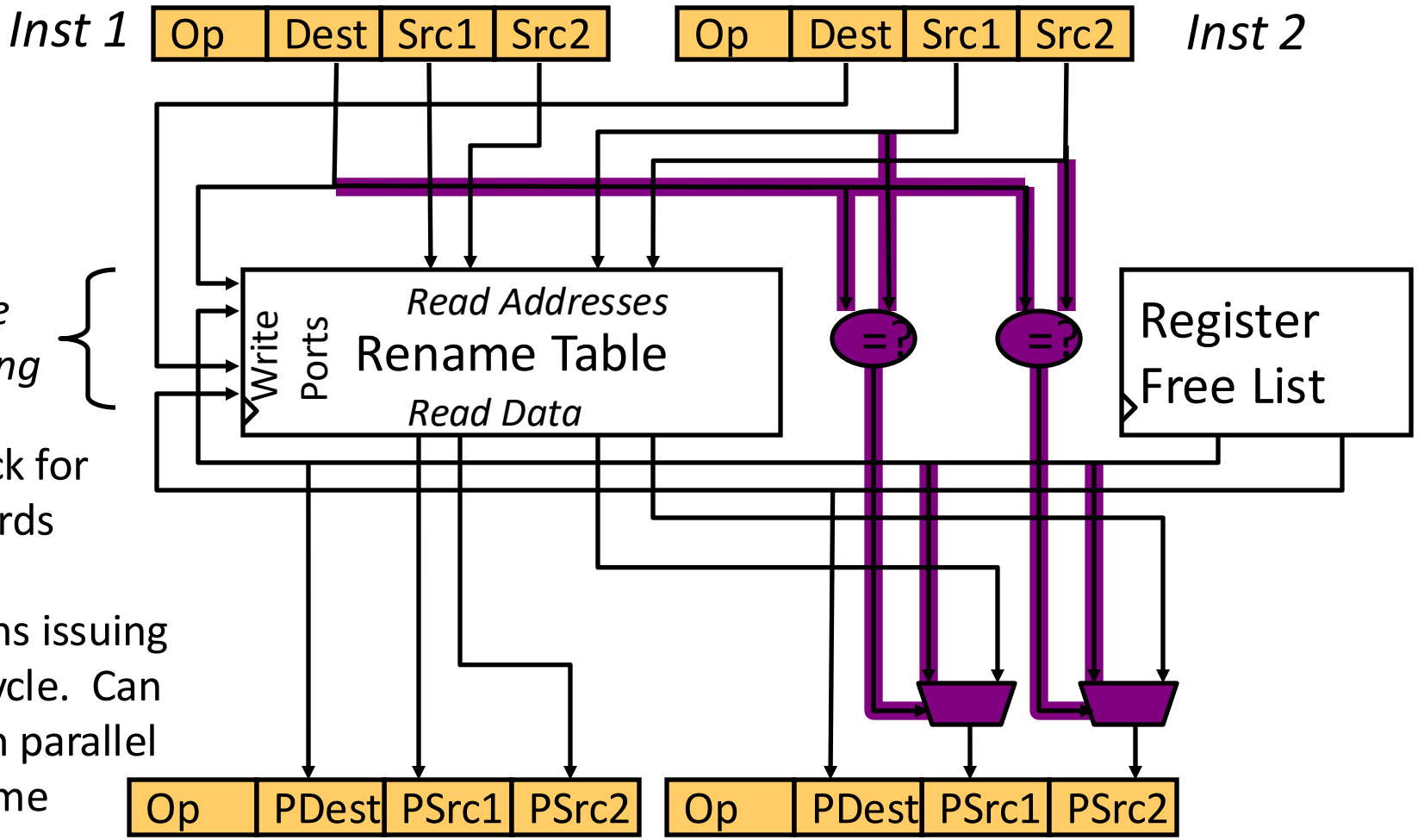
All this is done in parallel in a single clock cycle!

Superscalar Register Renaming: two-issue

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



Advanced Computer Architectures



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

Speculation and energy efficiency

- Speculation raises power consumption but lowers execution time by more than it increases the average power consumption

  ➔ Total energy consumed may be less depending on the number of instructions incorrectly executed

  – Experimental results show that in scientific code misspeculation is on average small while it is significant (30% on average) in integer code

Summary

- Modern computer architects predict everything:
  - Branches/Data Dependencies/Data!
- Fairly simple hardware structures can do a good job of predicting branches:
  - Branch Target Buffer (BTB) identifies branches and branch offsets
  - Branch History Table (BHT) does prediction
- More Sophisticated prediction: Correlation
  - Different branches depend on one another!

Summary

- Explicit Renaming: more physical registers than ISA
  - Separates *renaming* from *scheduling*
    - Opens up lots of options for resolving RAW hazards
  - Rename table: tracks current association between architectural registers and physical registers
  - Potentially complicated rename table management
- Parallelism hard to get from real hardware