Instruction Level Parallelism

Register renaming

ILP limits

Superscalar processors

Outline

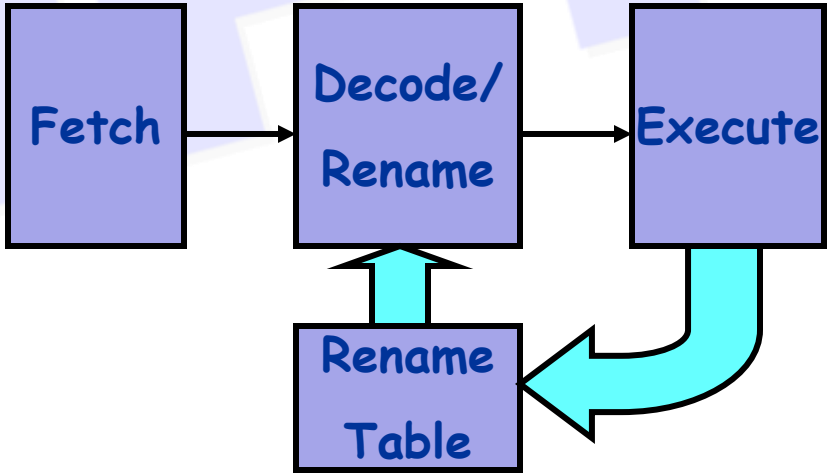
- Explicit Register renaming
- Alias analysis
- ILP limits
- Superscalar processors

Explicit Register Renaming

- Make use of a *physical* register file that is larger than number of registers specified by 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?

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



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

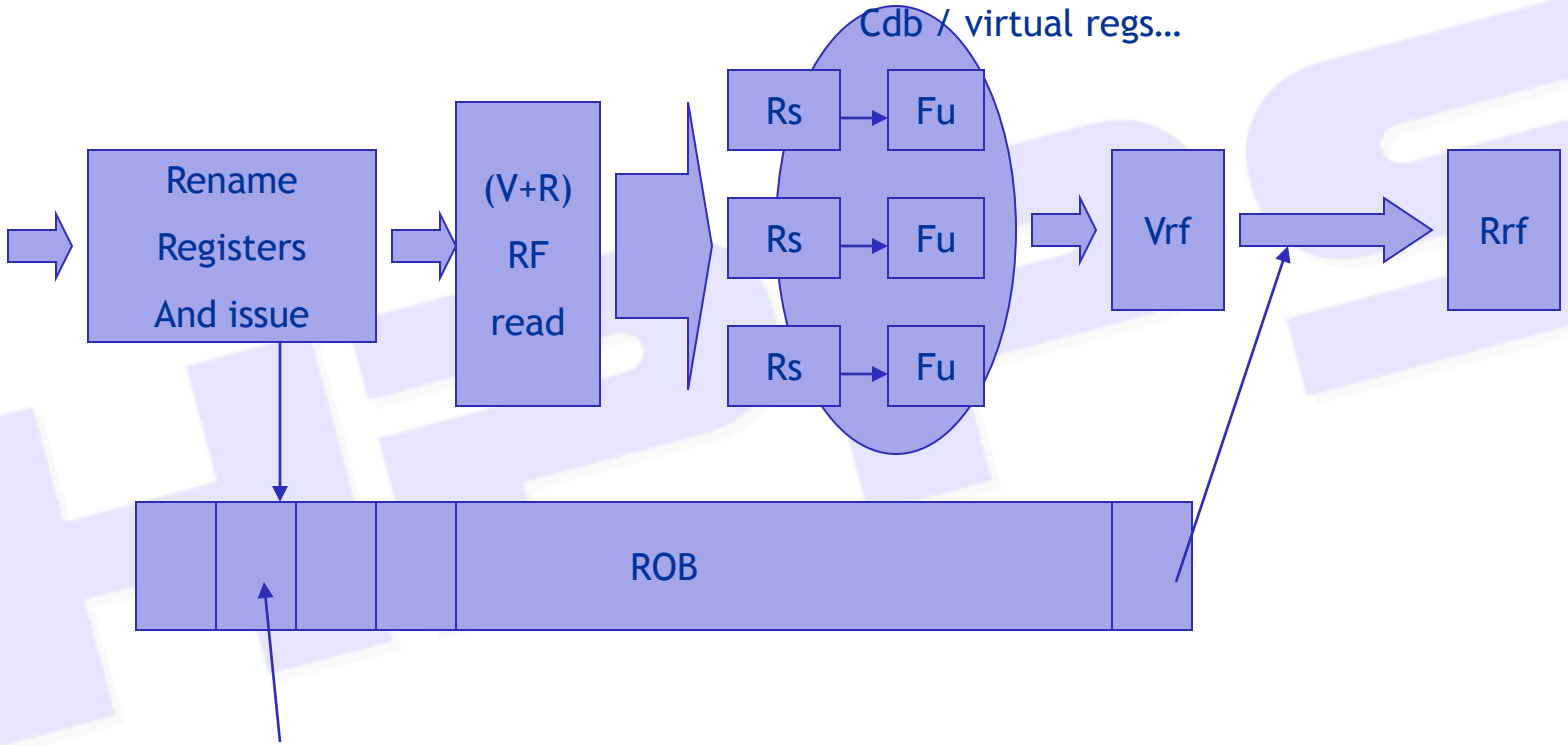
Interrupts and register renaming

- Another way to get precise interrupt points:
  - All that needs to be “undone” for precise break point is to undo the table mappings
  - Provides an interesting mix between reorder buffer and future file
    - Results are written immediately back to register file
    - Registers *names* are “freed” in program order (by ROB)

HW Register Renaming

- It is a Reorder buffer that does not keep the results but only enforces in-order commit.
- Register File is extended with extra registers to hold speculative values.
- When issuing an instruction, rename all the speculative operands to the speculative registers. On commit copy the speculative register into the real one.
- Operands are read from the RF (real or speculative) or via the CDB.

HW Register Renaming

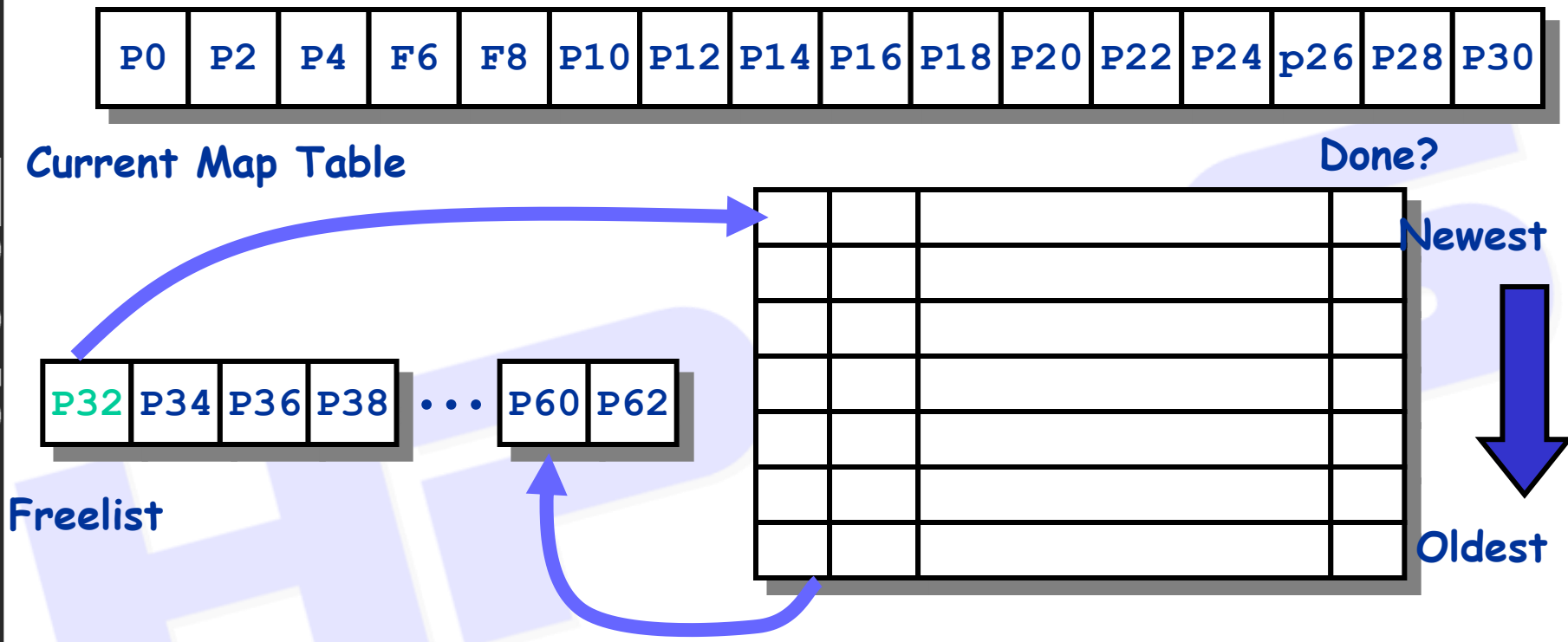


Maps virtual to physical registers

Explicit Renaming Support

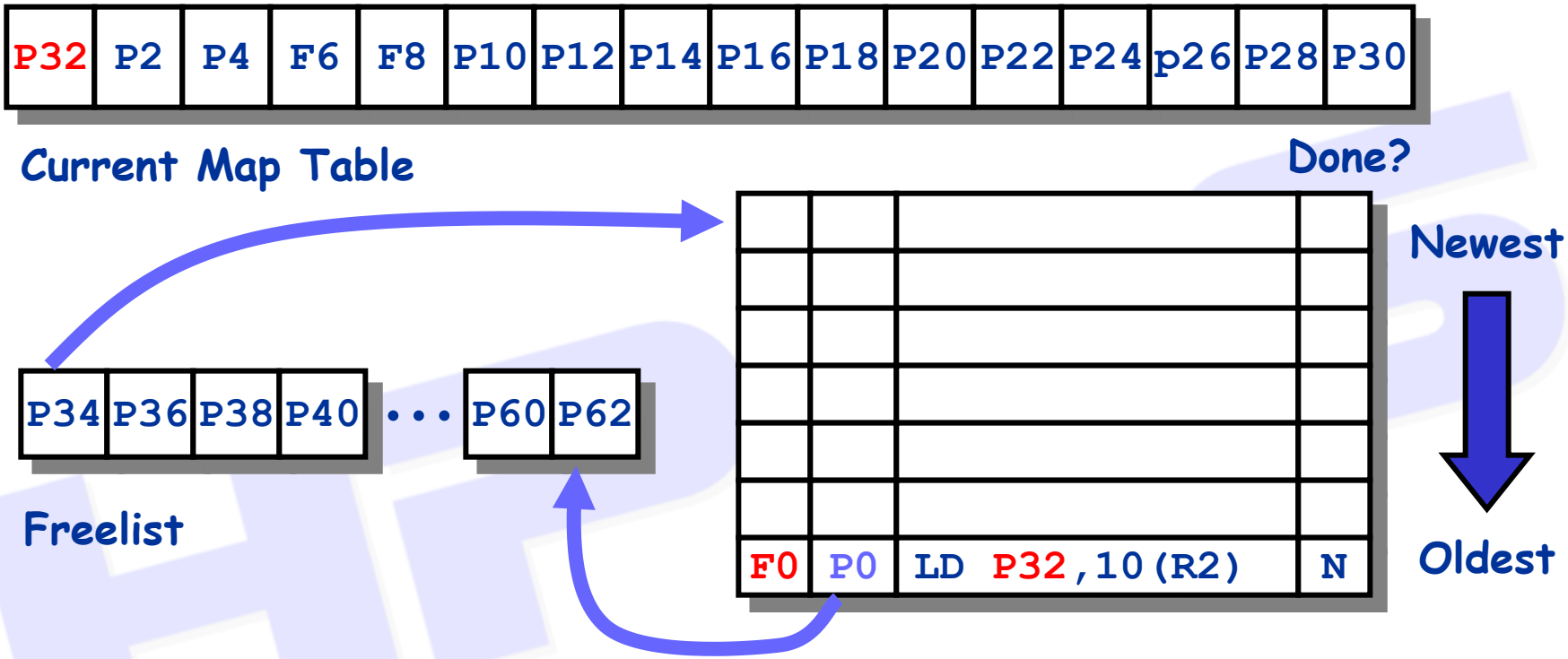
- Rapid access to a table of translations
- A physical register file that has more registers than specified by the ISA
- Ability to figure out which physical registers are free.
  - No free registers ⇒ stall on issue
- Thus, register renaming doesn't require reservation stations. However:
  - Many modern 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?

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

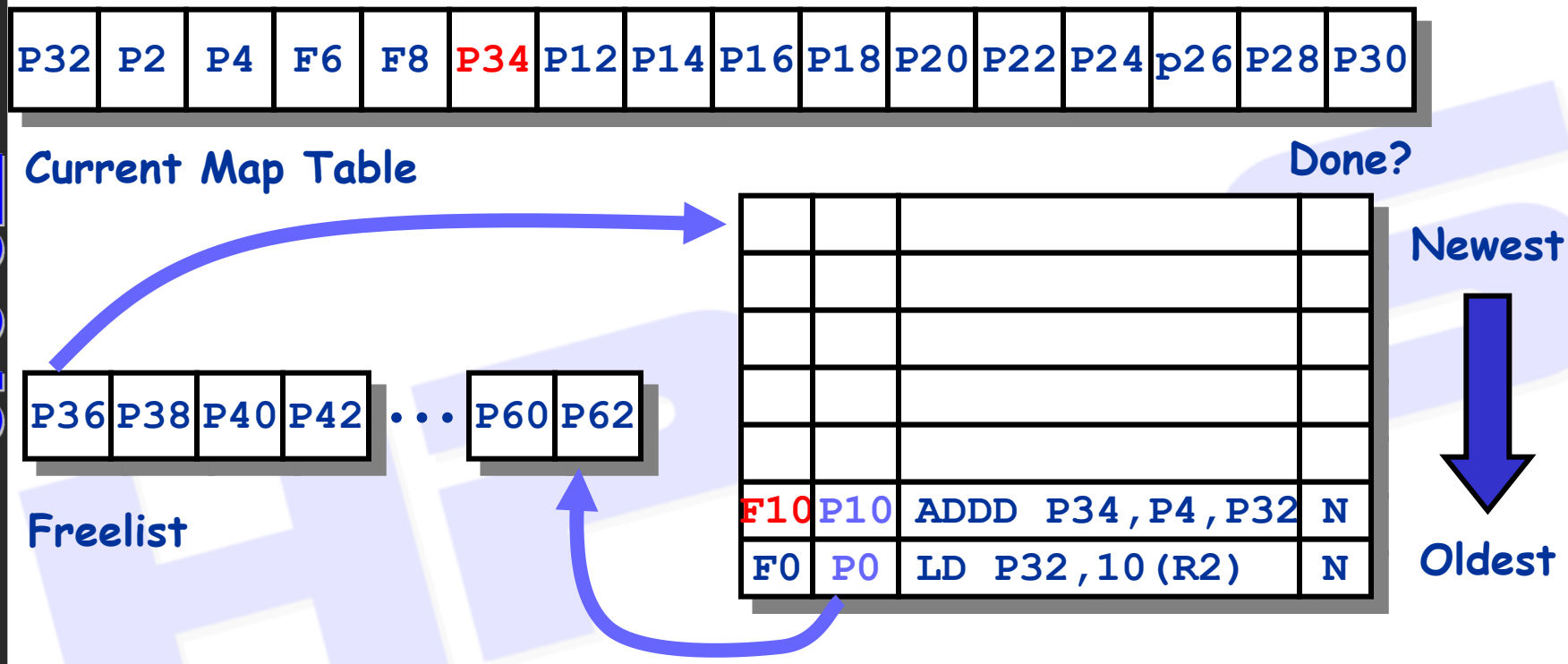
Explicit register renaming: (MIPS R10000 Style)



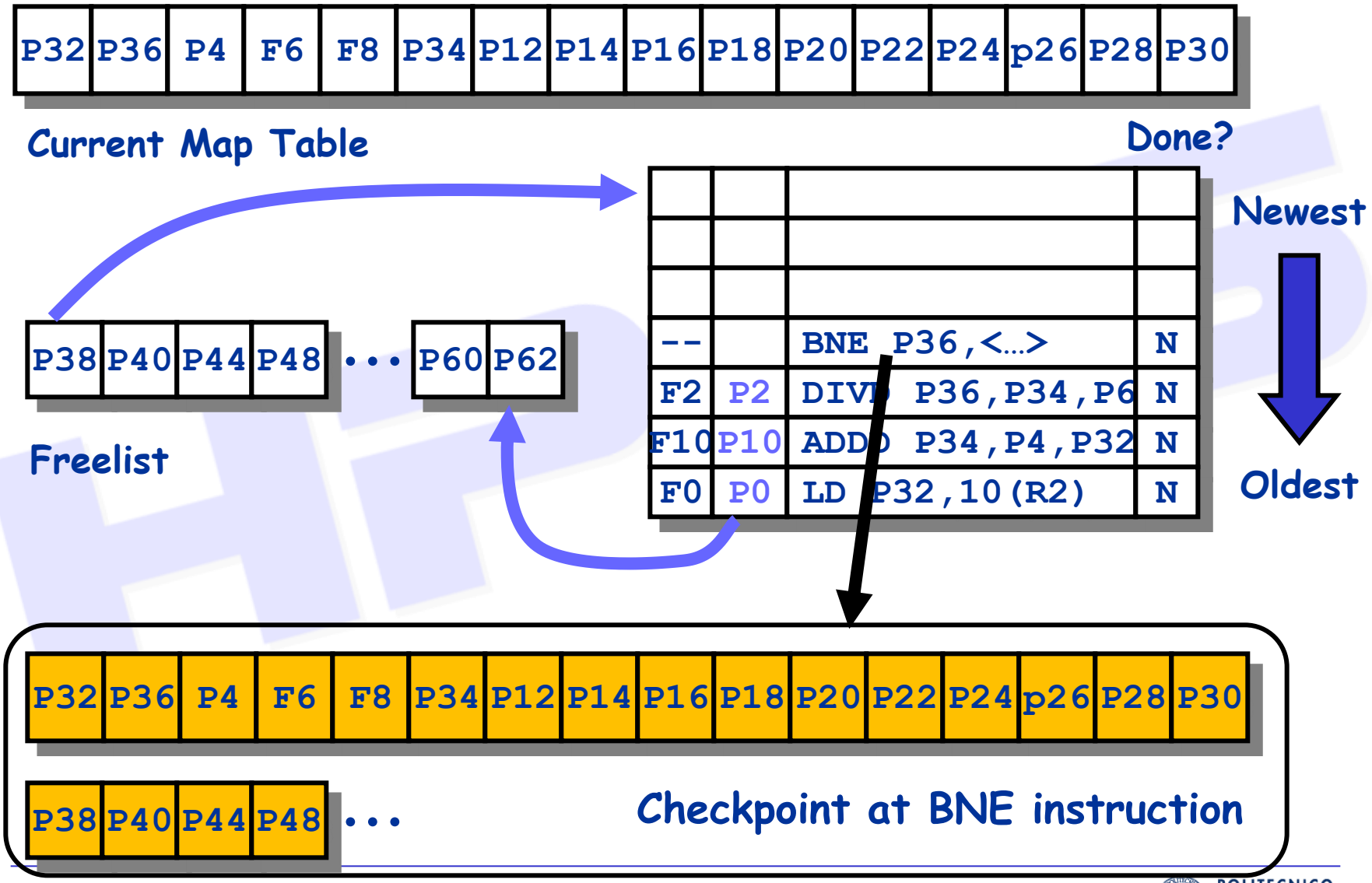
Note that physical register P0 is “dead” (or not “live”) past the point of this load.

When we go to commit the load, we free up

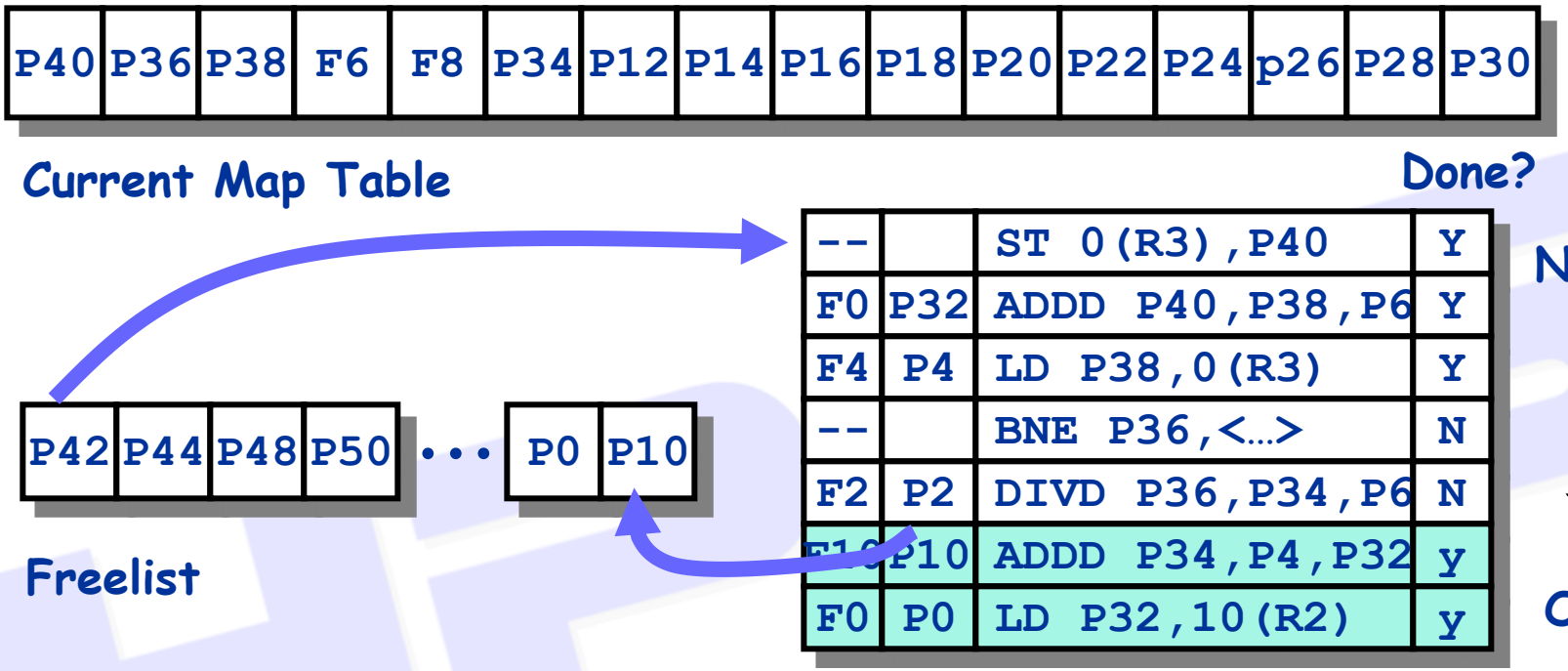
Explicit register renaming: (MIPS R10000 Style)

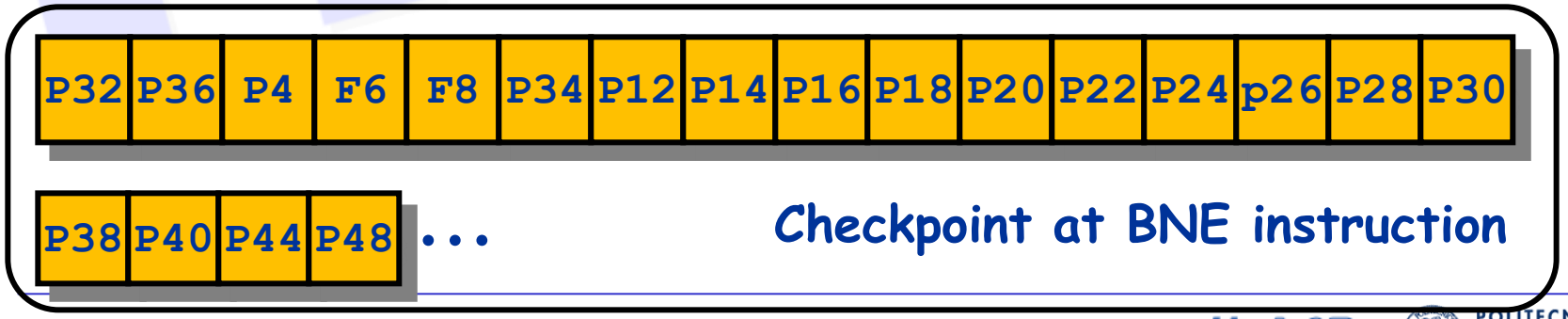


Explicit register renaming: (MIPS R10000 Style)

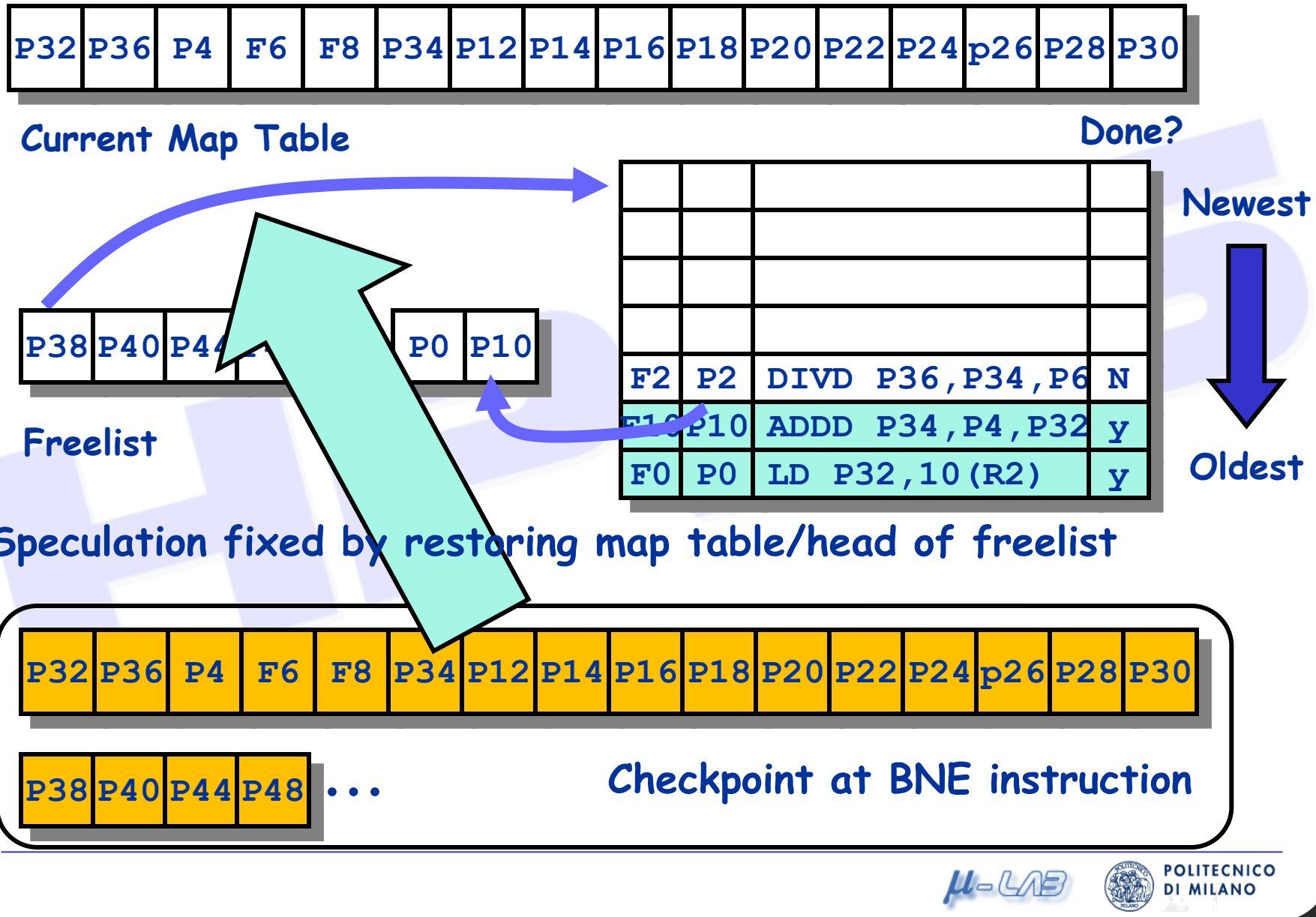


Explicit register renaming: (MIPS R10000 Style)





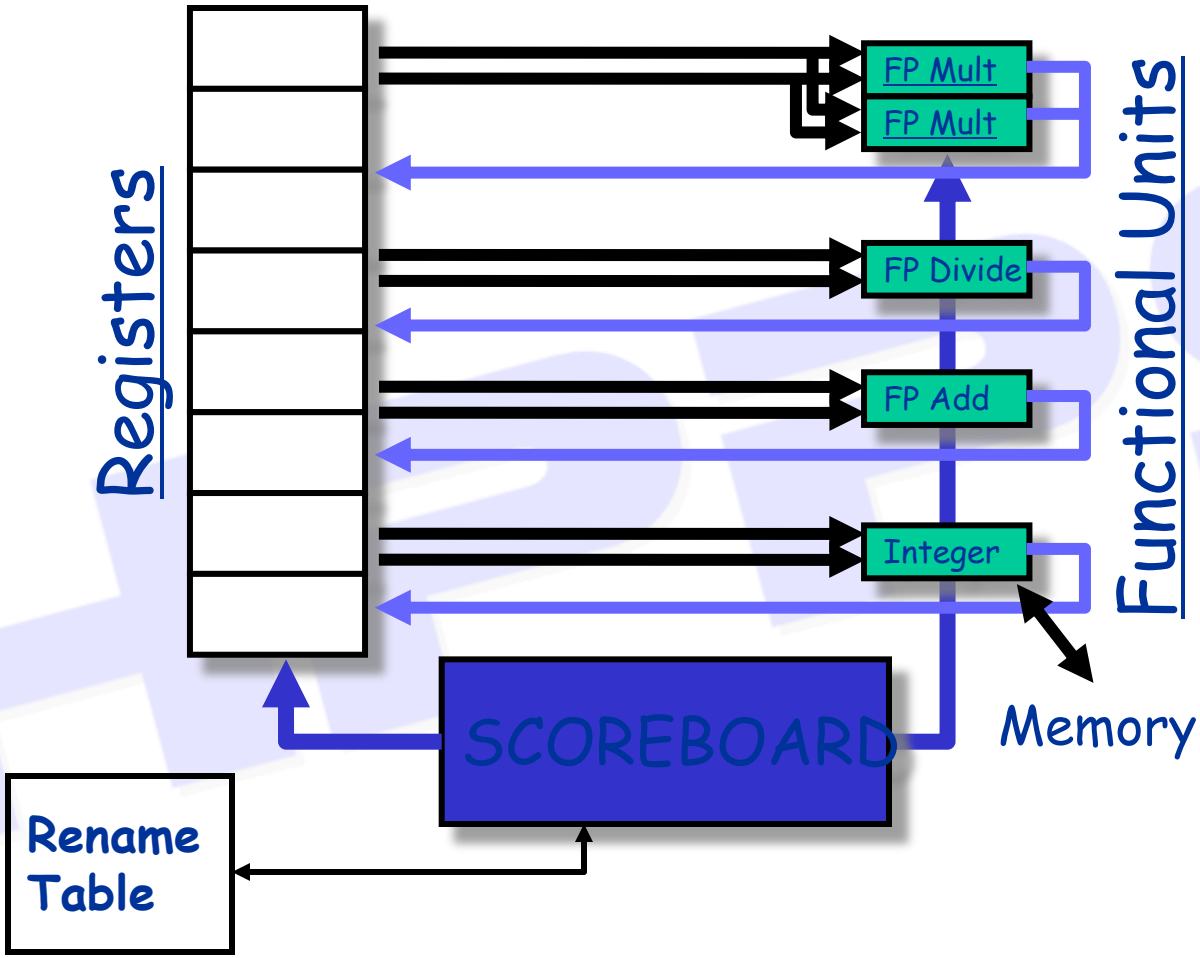
Explicit register renaming: (MIPS R10000 Style)



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

Question: Can we use explicit register renaming with scoreboard?



Stages of Scoreboard Control With Explicit Renaming

- 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
- 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

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

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

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				

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

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

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

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

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

Explicit Renaming Support Includes:

- 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 $\Rightarrow$ 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.

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/future file (next time)

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;
  - 20 to 80 registers are added.

Speculating through multiple branches

- Speculating from multiple branches simultaneously: a benefit in the case of:
  - Very high branch frequency, or
  - Significant clustering of branches, or
  - Long delays in functional units.
- Complicates speculation recovery, otherwise is straightforward;
- More complex: predicting and speculating *more than one branch per cycle*.

Effects of realistic branch and jump prediction

- Perfect branch prediction obviously impossible: when not highly accurate, mispredicted branches become a barrier to finding parallelism;
- Branch prediction mechanisms a major point of optimization in leading-edge CPUs.

Effects of finite registers

- Reducing the number of registers available for renaming has great impact of extraction of available parallelism; increasingly relevant with increasing intrinsic level of parallelism in a benchmark!

Imperfect Alias Analysis

- Perfect analysis at compile time impossible (run-time compiled memory references, pointer. Accessed variables etc.);
- Run-time alias analysis: a priori (if no constraints are placed on number of simultaneous memory references is allowed) requires unlimited number of comparisons.

Imperfect Alias Analysis

- Consider three models of memory alias analysis, in addition to perfect analysis:
  1. *Global/stack perfect*: assumes perfect prediction for all global+stack references, conflict on all heap references (based on improvements in compiler technology);
  2. *Inspection*: accesses are examined to see if they can be determined not to interfere at compile time. Also, accesses based on registers that point to different allocation areas (e.g., global area and stack area) are assumed never to alias;
  3. *None*: all memory references are assumed to conflict.

Imperfect Alias Analysis

- Model 1 gives results quite similar to perfect alias analysis, model 2 is not much better than model 3.
- In practice, dynamically scheduled CPUs rely on dynamic memory disambiguation.
- Three factors limiting their efficiency:
  1. To achieve perfect dynamic disambiguation for a load ⇨ necessary to know memory addresses of all previous stores that have not yet committed.
     Memory address speculation: dependency is *assumed* not to exist or else predicts through hw mechanism, load is stalled if dependency is predicted.
     To check on prediction correctness: CPU examines destination address of each completing store preceding in program order the given load; if dependency that should have been enforced occurs, CPU uses speculative restart mechanism to redo load and following instructions (supported with suitable instruction set extension);
  2. Only a small number of memory references can be disambiguated per clock cycle;
  3. Number of load/store buffers determines how much earlier or later in the instruction stream a load or a store can be moved.

Relationship between precise interrupts and speculation

- Speculation is a form of guessing
  - Branch prediction, data prediction
  - If we speculate and are wrong, need to back up and restart execution to point at which we predicted incorrectly
  - This is exactly same as precise exceptions!
- Branch prediction is a very important!
  - Need to “take our best shot” at predicting branch direction.
  - If we issue multiple instructions per cycle, lose lots of potential instructions otherwise:
    - Consider 4 instructions per cycle
    - If take single cycle to decide on branch, waste from 4 - 7 instruction slots!
- Technique for both precise interrupts/exceptions and speculation: *in-order completion or commit*
  - This is why reorder buffers in all new processors

Beyond CPI = 1

- Initial goal to achieve CPI = 1
- Can we improve beyond this?
- Two approaches
- Superscalar:
  - varying no. instructions/cycle (1 to 8),
  - scheduled by compiler or by HW (Tomasulo)
  - e.g. IBM PowerPC, Sun UltraSparc, DEC Alpha, HP 8000
  - The successful approach (to date) for general purpose computing
- Anticipated success lead to use of Instructions Per Clock cycle (IPC) vs. CPI

Limits to ILP

- Conflicting studies of amount
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication
- How much ILP is available using existing mechanisms with increasing HW budgets?
- Do we need to invent new HW/SW mechanisms to keep on processor performance curve?

Limits to ILP

Assumptions for ideal/perfect machine to start:

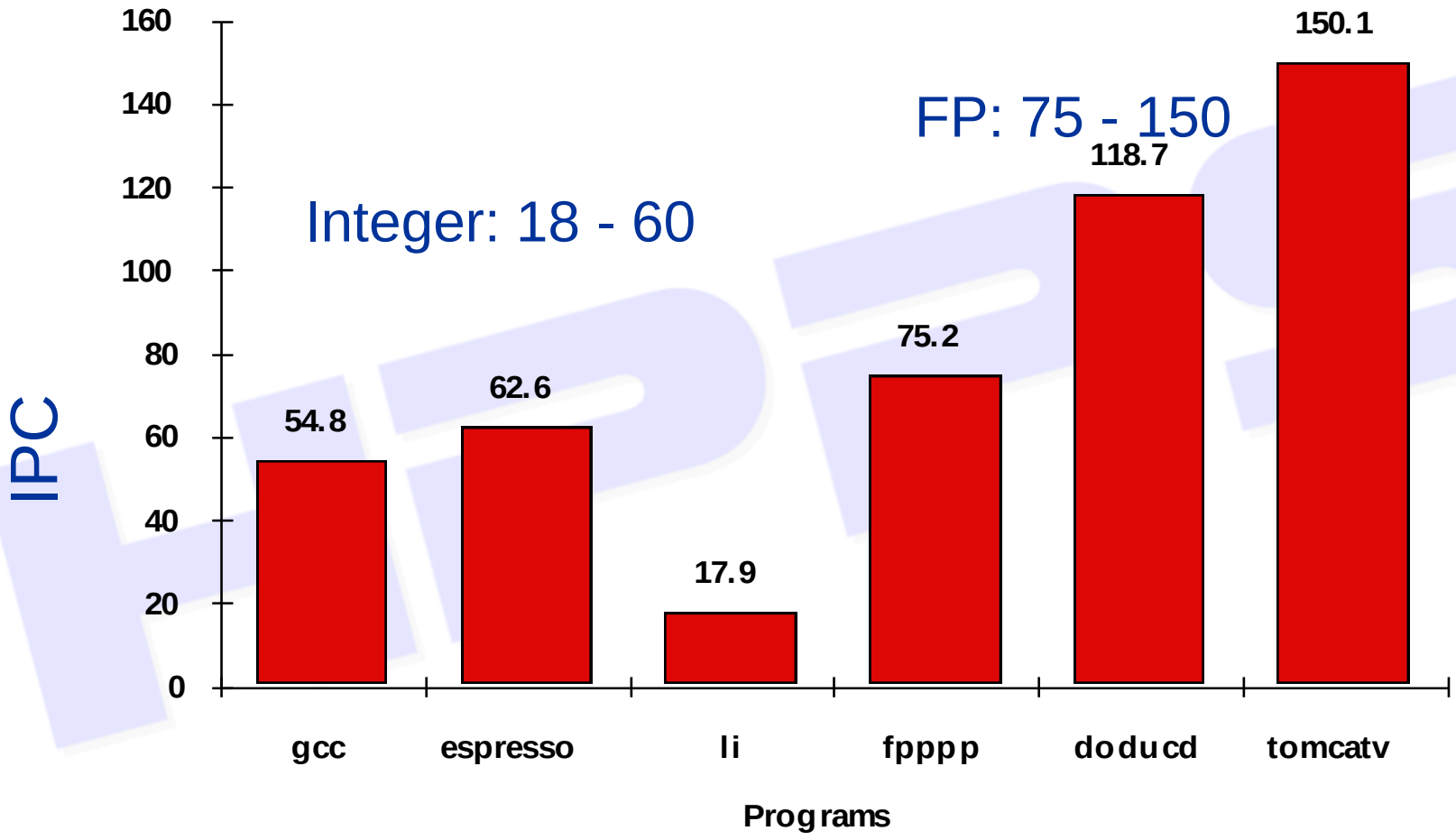
1. *Register renaming*–infinite virtual registers and all WAW & WAR hazards are avoided
2. *Branch prediction*–perfect; no mispredictions
3. *Jump prediction*–all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis*–addresses are known & a store can be moved before a load provided addresses not equal

1 cycle latency for all instructions; unlimited number of instructions issued per clock cycle

Initial assumptions

- CPU can issue at once unlimited number of instructions, looking arbitrarily far ahead in computation;
- No restrictions on types of instructions that can be executed in one cycle (including loads and stores);
- All functional unit latencies = 1; any sequence of depending instructions can issue on successive cycles;
- Instructions in execution: "***in flight***".
- Perfect caches = all loads, stores execute in one cycle ⇨ only fundamental limits to ILP are taken into account.
- Obviously, results obtained are VERY optimistic! (no such CPU can be realized…);
- Benchmark programs used: six from SPEC92 (three FP-intensive ones, three integer ones).

Upper Limit to ILP: Ideal Machine



Limits on window size

- Dynamic analysis is necessary to approach perfect branch prediction (*impossible at compile time*!);
- A perfect dynamic-scheduled CPU should:
  1. Look arbitrarily far ahead to find set of instructions to issue, predict all branches perfectly;
  2. Rename all registers uses (⇨ no WAW, WAR hazards);
  3. Determine whether there are data dependencies among instructions in the issue packet; rename if necessary;
  4. Determine if memory dependencies exist among issuing instructions, handle them;
  5. Provide enough replicated functional units to allow all ready instructions to issue.

Limits on instruction windows

- Size affects the number of comparisons necessary to determine RAW dependences
- Example: # comparisons to evaluate data dependences among n register-to-register instructions in the issue phase (with an infinite # of regs) =

$$2\sum_{i=1}^{n-1} i = 2\frac{(n-1)n}{2} = n^2 - n$$

  - Window size = 2000 ↳ almost 4 Million comparisons!
  - Issue window of 50 instructions requires 2450 comparisons!
  - Today window size between 32 and 126 instructions

- Today's CPUs: constraints deriving from the limited number of registers + search for dependent instructions + in-order issue

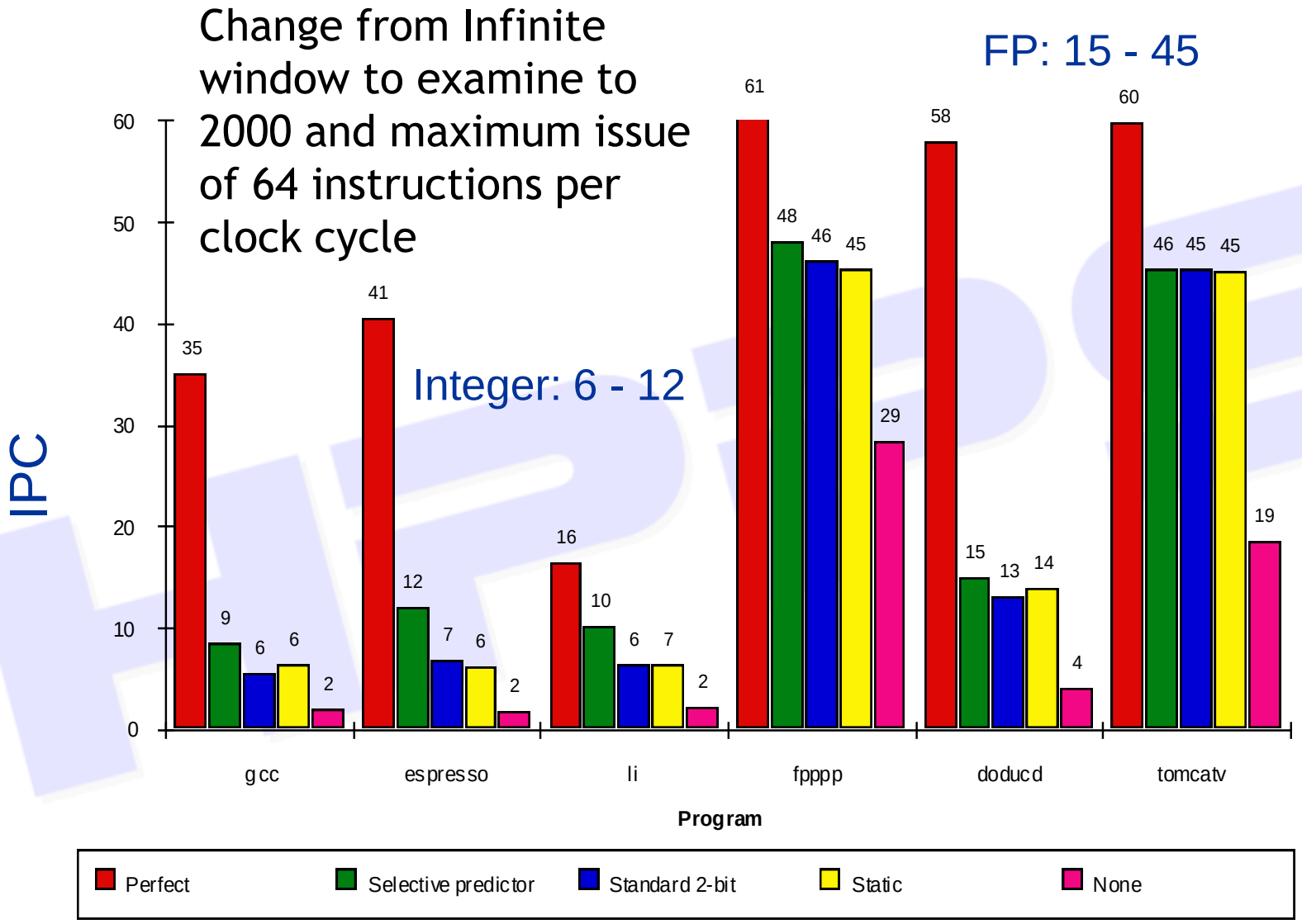
Limits on window size, maximum issue count

All instructions in the ***window*** must be kept in the processor ⇨

number of comparisons required at each cycle =
maximum completion rate x
window size x
number of operands per instruction ⇨
total window size limited by storage + comparisons + limited issue rate

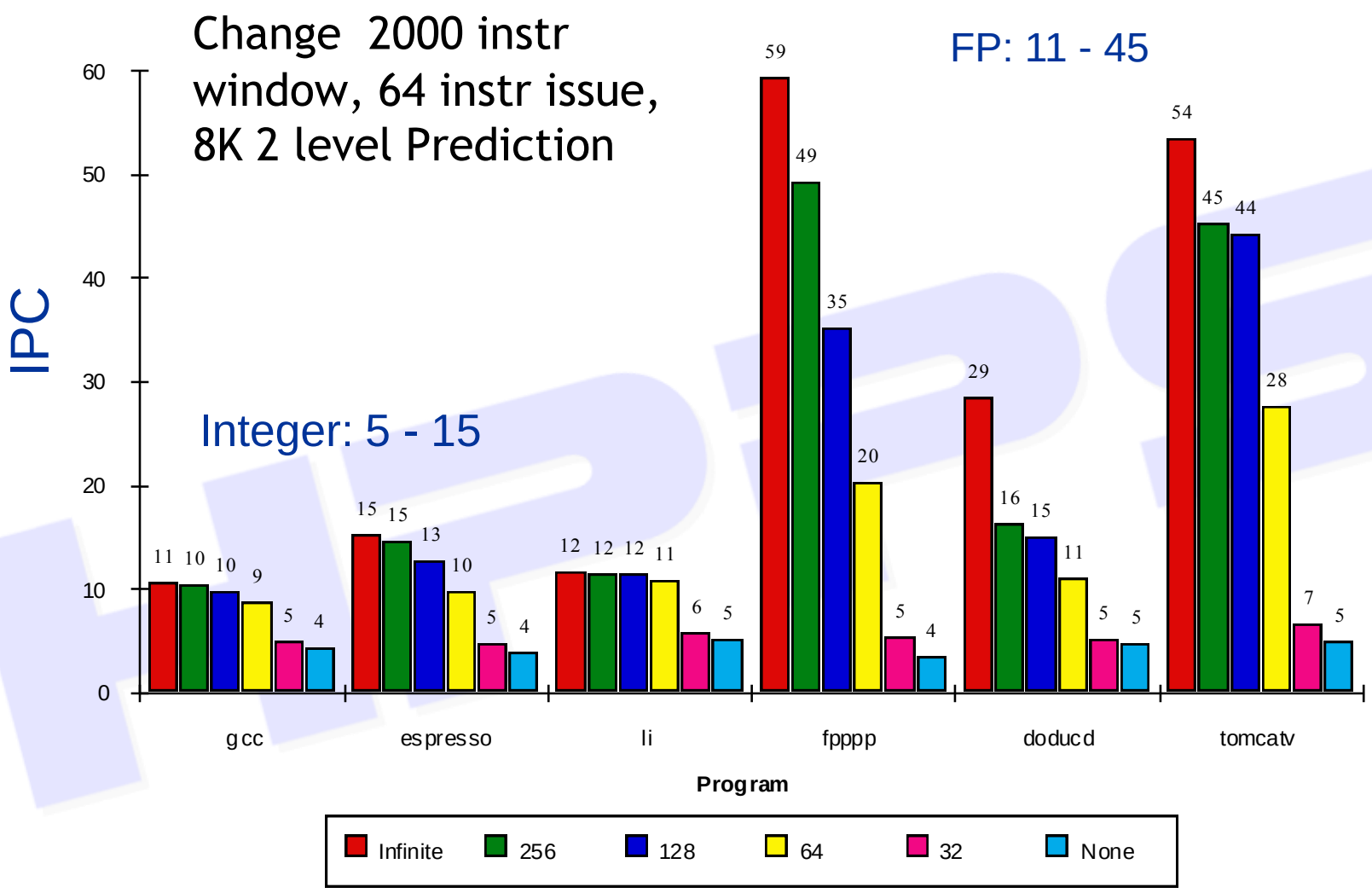
(today: window size 32-200 ⇨ up to over 2400 comparisons!)

More Realistic HW: Branch Impact



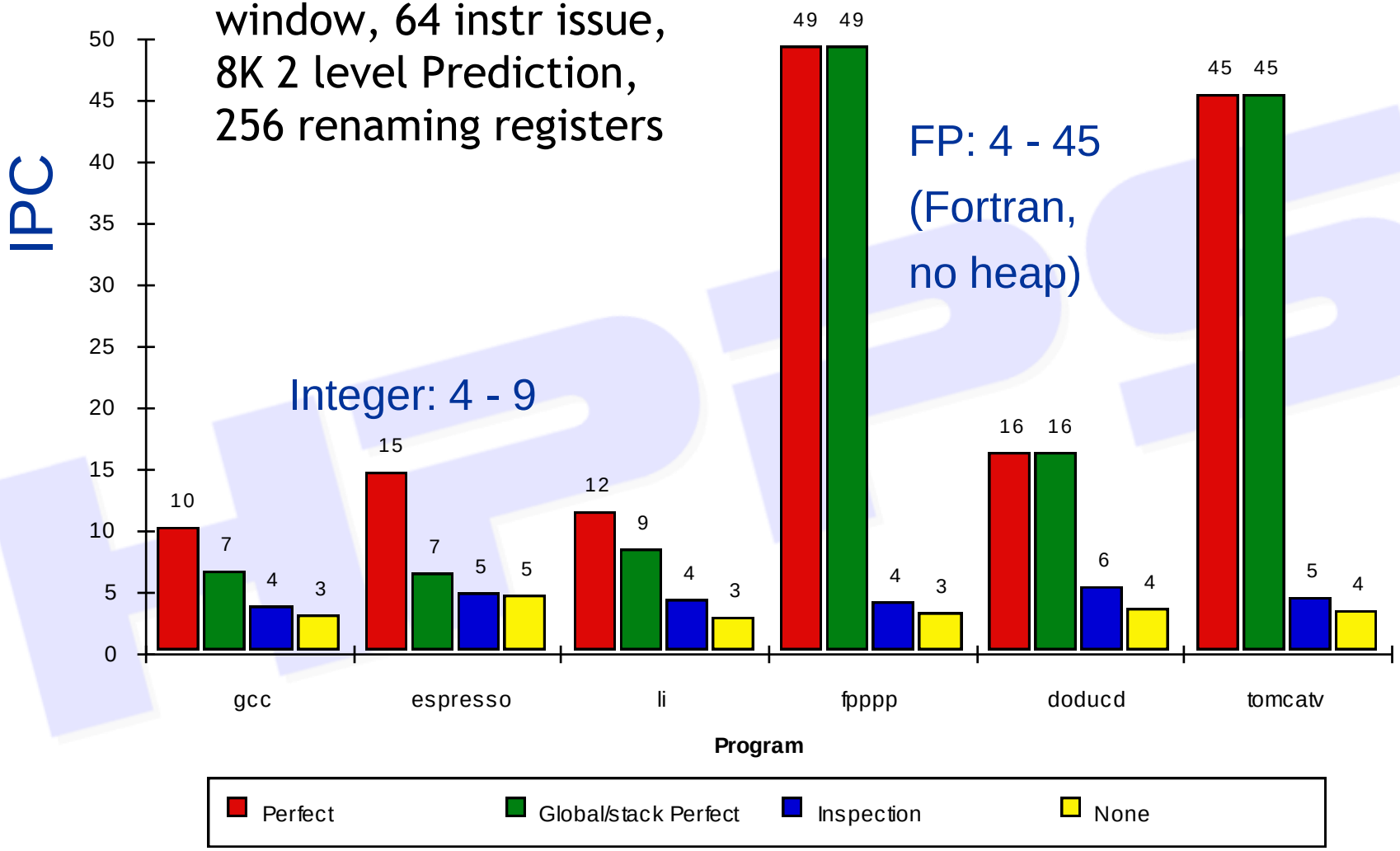
Perfect Pick Cor. or BHT BHT (512) Profile No prediction

More Realistic HW: Register Impact



More Realistic HW: Alias Impact

Change 2000 instr window, 64 instr issue, 8K 2 level Prediction, 256 renaming registers



Integer: 4 - 9

FP: 4 - 45 (Fortran, no heap)

Other limits of today's CPUs

- N. of functional units
  - For instance: not more than 2 memory references per cycle
- N. of busses
- N. of ports for the register file

All these limitations define that the maximum number of instructions that can be issued, executed or committed in the same clock cycle is much smaller than the window size

Issue capabilities

Processor	Year Shipped in Systems	Initial Clock rate (MHz)	Issue Structure	Scheduling	Issue Capabilities: Max.	Load Store	Integer ALU	FP	Branch	SPEC (Measure of estimate)
DEC A1-Pha 21064	1992	150	Dynamic	Static	2	1	1	1	1	100 int 150 FP
Intel Pentium	1994	66	Dynamic	Static	2	2	2	1	1	65 int 65FP
DEC Alpha 21164	1995	300	Static	Static	4	2	2	2	1	330 inc 500 FP
Intel P6	1995	150	Dynamic	Dynamic	3	1	2	1	1	>200 int
PowerPC 620	1995	133	Dynamic	Dynamic	4	1		1	1	225 int 300 FP
MIPS R10000	1996	200	Dynamic	Dynamic	4	1	2	2	1	300 int 600 FP

Superscalar Processors

- Issues multiple instructions at each clock cycle.
- If instructions are dependent, only consecutive ready instructions are issued (in-order issue).
- This decision is made at run-time by the processor.

=> Variability in the issue rate.

Superscalar Processors

Can be:

- Statically scheduled:
  Do not allow (**issue**) instructions behind stalls to proceed or
- Dynamically scheduled and speculative (allow instructions behind RAW hazards to proceed).

- Loop unrolling is one of the techniques to optimize code for superscalar execution

Example: P6

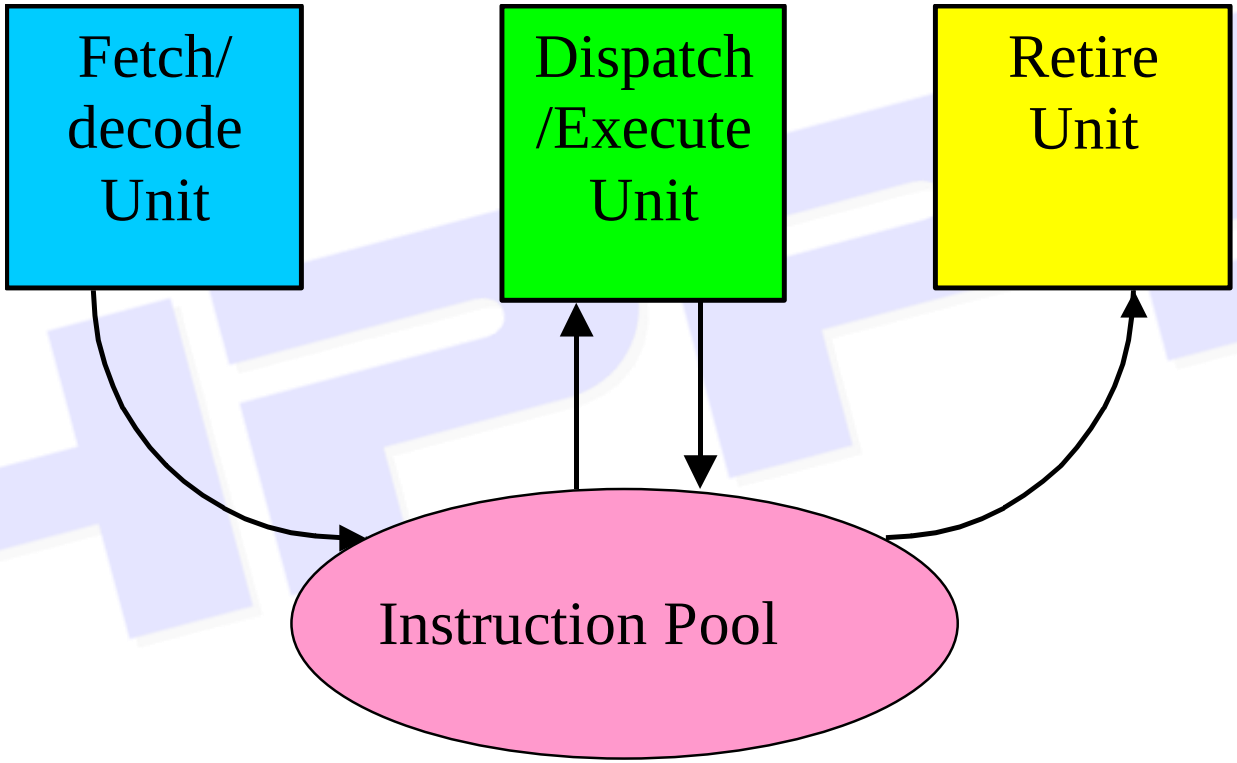
Processor	First issue	Clock frequency	L1 cache	L2 cache
Pentium Pro	1995	100-200 MHz	8KB I + 8KB D	256-1025 KB
Pentium II	1998	233-450	16KB + 16KB	256-512
Pentium II Xeon	1999	400-450	16KB + 16KB	512-2 MB
Celeron	1999	500-900	16KB + 16KB	128
Pentium III	1999	450-1100	16KB + 16KB	256-512
Pentium III Xeon	2000	700-900	16KB + 16KB	1-2 MB

Example: P6

- P6 microarchitecture: CPU with dynamic scheduling, translates every instruction IA-32 in a sequence of micro-operations (uops) executed by the pipeline;
- Uops: typical RISC instructions;
- In every clock cycle read, decode and translate up to three IA-32 instructions into micro-ops

Example: P6

- 3 way superscalar
- Basic Idea, three engines



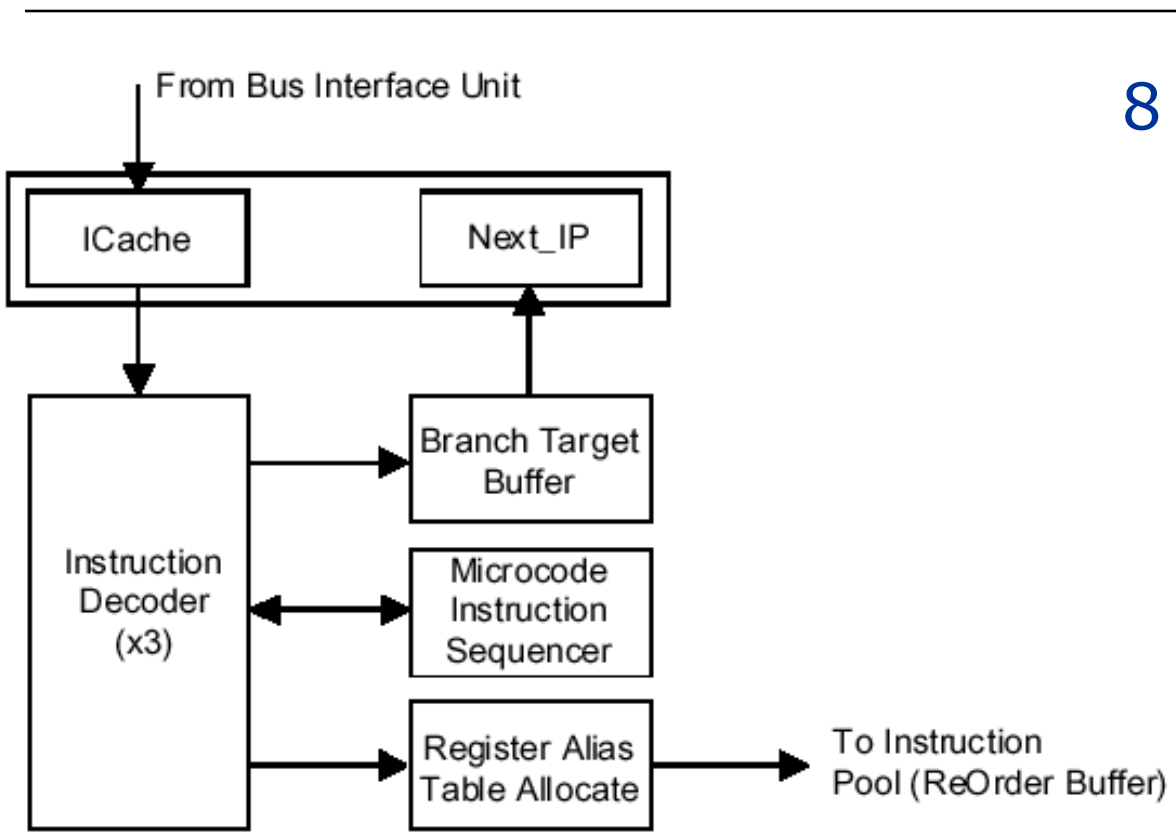
P6 Pipeline

- **Fetch/Decode Unit:** decodes instructions and puts them in the instruction pool in-order.
  - converts the instructions in micro-ops that represent instruction code executed by the pipeline
    - The micro-ops are typical RISC instructions
    - In each clock cyle: fetch and decode up to 3 instructions
- **Dispatch/Execute Unit:** out-of-order issue from the instruction pool in a reservation station and out-of-order execution of micro-ops.
- **Retire Unit**
  Reorders the instructions and commits speculative results to the architectural state.

P6 pipeline stages

- 14 pipeline stages
  - 8 stages for the in-order issue, decode and dispatch
    - The next instruction is selected during the IF stage using a 2-levels branch predictor with 512 elements
    - Decode and issue include register renaming with 40 virtual registers
    - Dipatch to one of the 20 reservation stations and to one of the 40 positions of the Reorder Buffer
  - 3 stages for the out-of-order exection to one of the functional units (5 types: FX, FP, branches, memory addressing, memory access)
    - Execution pipeline: 1 clock cycle (simple ALU ops) up to 32 (FP division)
  - 3 stages for commit

P6 Instruction Decode



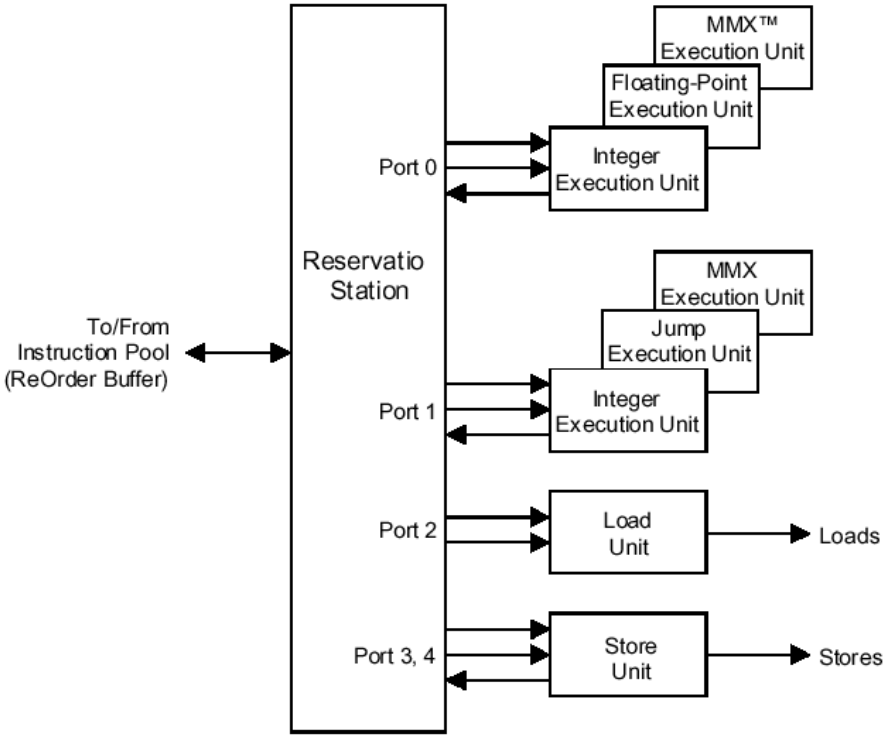
8 pipeline stages

- The decoder fetches 16 bytes at each clock cycle from the cache
- 3 parallel decoders convert most of the instructions into one or more *triadic* micro-ops. Some instruction need microcode (several micro-ops) to be executed. Throughput=6 microops per clock cycle.
- Register Alias Table unit converts logical reg. ref. into virtual reg. ref. (40 registers).
- In-order Issue to reservation stations and reorder buffer

P6 Instruction Dispatch/Execute

20 entries RS

3 pipeline stages



- Out of order execution through the *reservation station* unit
- This happens when:
  - All the operands are ready
  - The resource needed is ready.
- Maximum throughput: 5 micro-ops/cycle.

If micro-ops are branches, their execution is compared with the predicted address (in the Fetch phase). If mispredicted the JEU changes the status of all the micro-ops behind the branch and removes them from the instruction pool.

P6 Instruction Retire

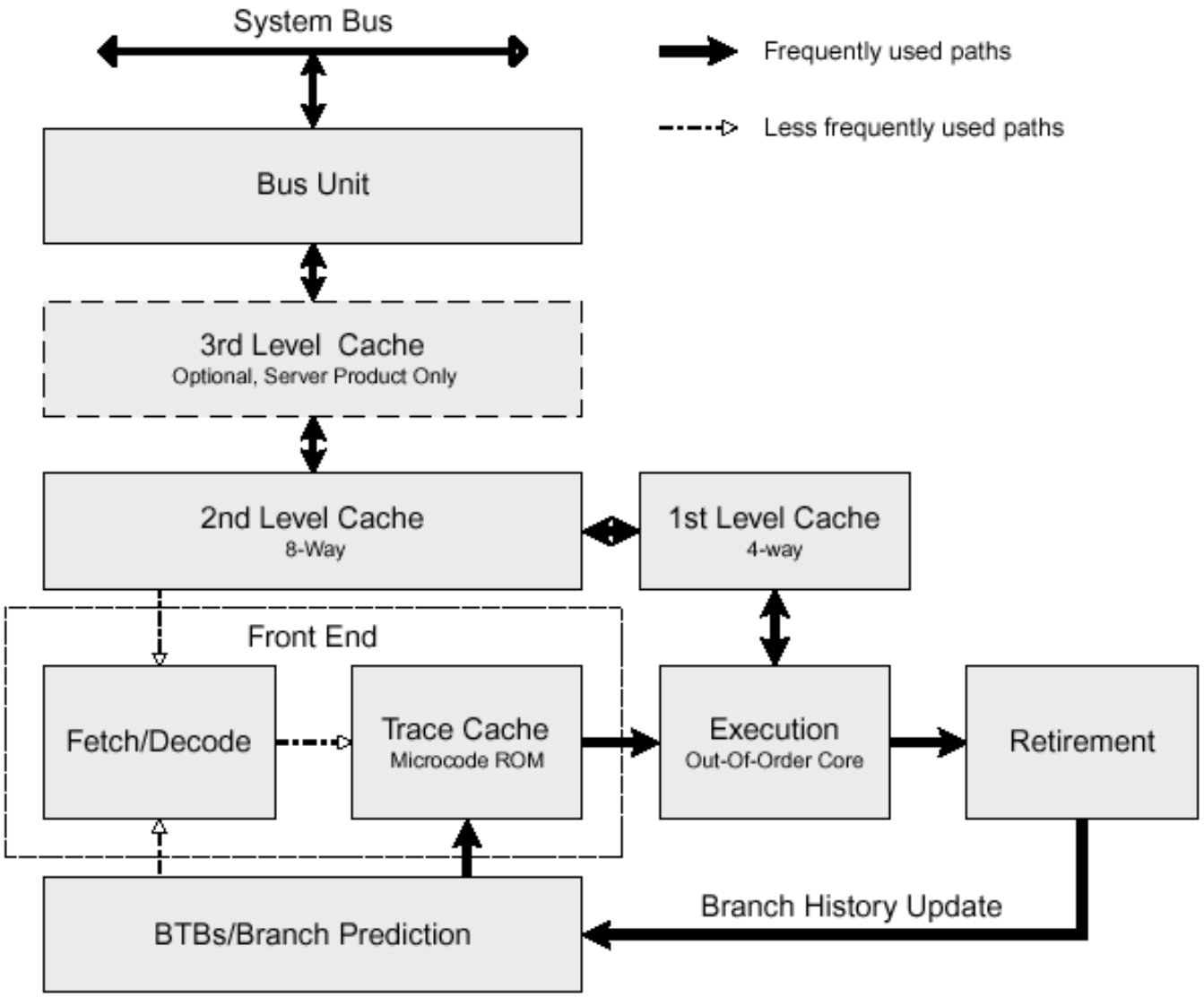
- The retire unit looks for micro-ops that have been executed and can be removed from the pool.
- The original architectural target of the micro-ops is written.
- This is done in-order by committing an instruction only if:
  - Previous instructions have been committed
  - The instruction has been executed.
- Up to 3 micro-ops can be retired at each clock cycle.

P6

- On average 20% of conditional branches are wrongly predicted and use a static prediction scheme (forward branches not taken, backward prediction taken)
- Due to speculation on average 1.2 more microops are issued than those that are committed

- IPC = 1.15 for SpecInt benchmarks
- IPC = 2.0 for SpecFP benchmarks

Pentium IV


System Bus
Frequently used paths
Less frequently used paths
Bus Unit
3rd Level Cache
Optional, Server Product Only
2nd Level Cache
8-Way
1st Level Cache
4-way
Front End
Fetch/Decode
Trace Cache
Microcode ROM
Execution
Out-Of-Order Core
Retirement
Branch History Update
BTBs/Branch Prediction

Pentium IV

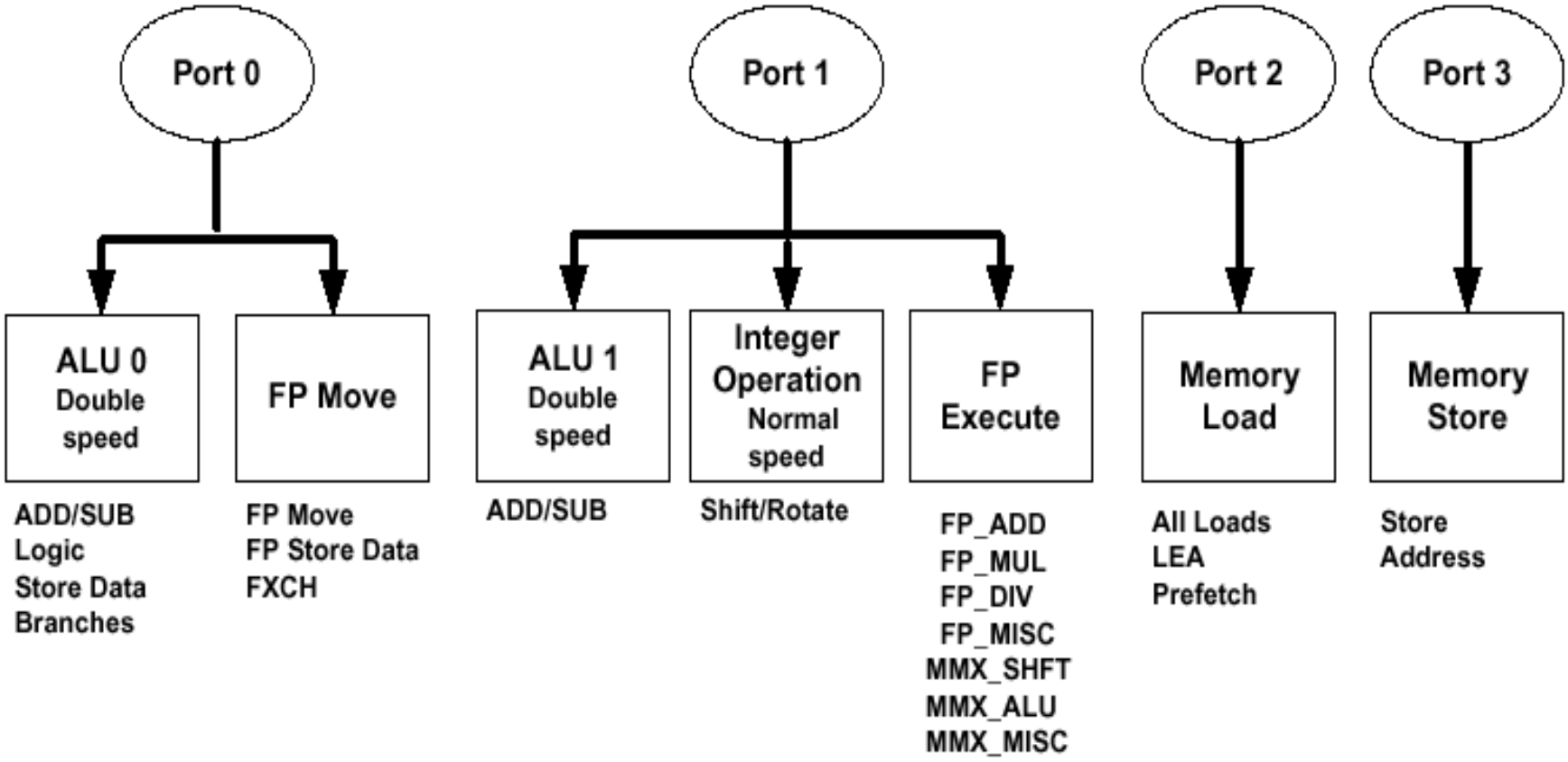
- NetBurst micro-architecture
  - 20 pipeline stages (hyper-pipeline)
  - 1.4 GHz to 2GHz and more
- 3 prefetching mechanisms
  - Hardware instruction prefetcher (based on BTB).
  - Software controlled data cache prefetching.
  - L3->L2 data and instruction hardware prefetcher
- Execution Trace Cache
  - TC stores decoded IA-32 instructions or micro-ops.
  - Removes decoding costs
  - 12K micro-ops, 3 micro-ops per cycle fetch bandwidth
  - It stores traces built across predicted branches.
  - However some instructions need micro-code from ROM

Pentium IV

- Branch penalty delay can be much more than 10 cycles
- Uses BTB
- In case of a miss in the BTB, static prediction is used (backward=T, forward=NT)
- Use of software **branch hints** during the trace construction that override static prediction.

Pentium IV

- Execution Units and Issue Ports



Pentium IV

- 1 load and 1 store issue for each cycle.
- Loads can be reordered w.r.t. other loads and stores
- Loads can be executed speculatively
- Up to 4 outstanding load misses.
- Load/store forwarding