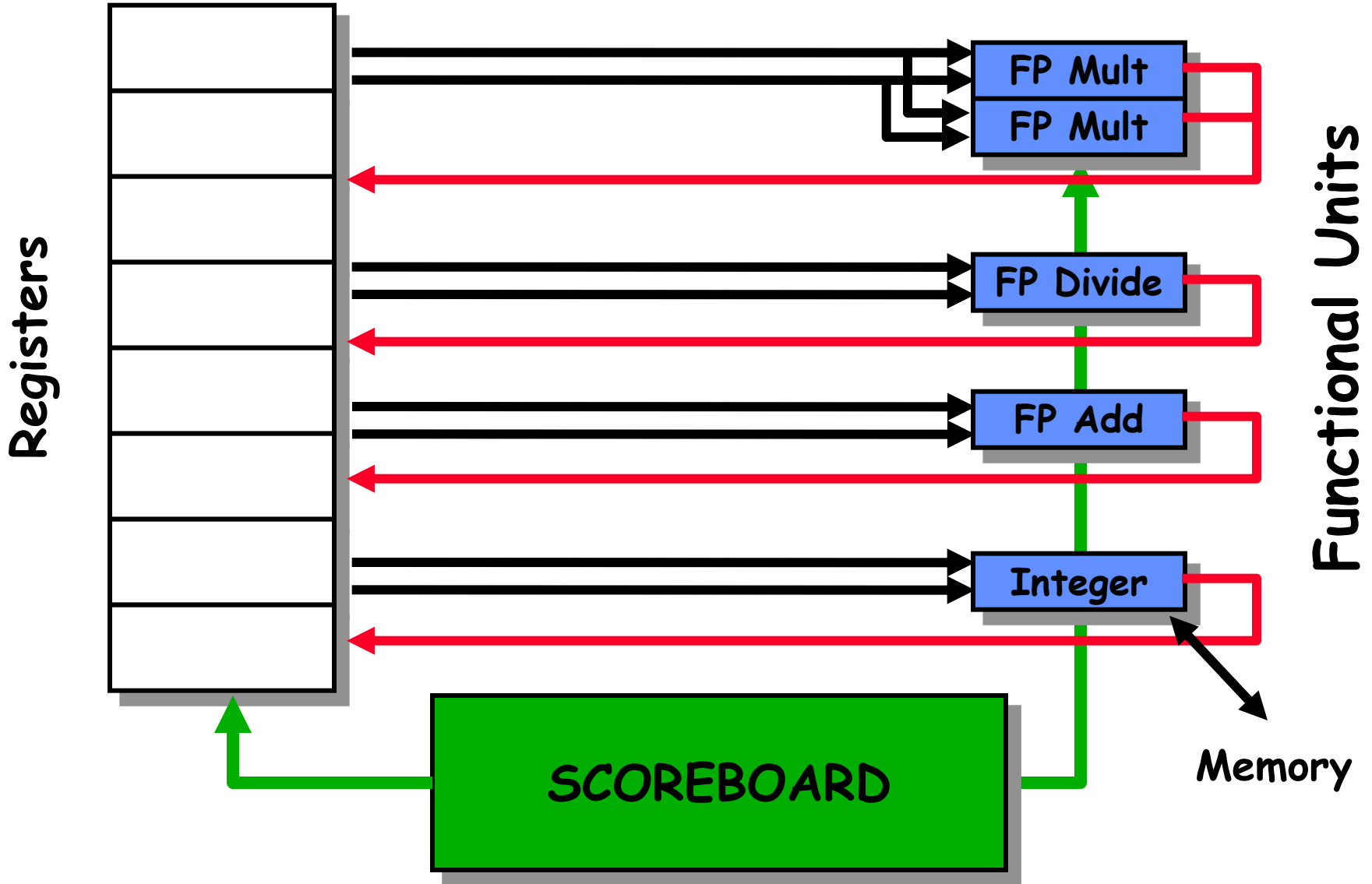


HY425
Αρχιτεκτονική Υπολογιστών

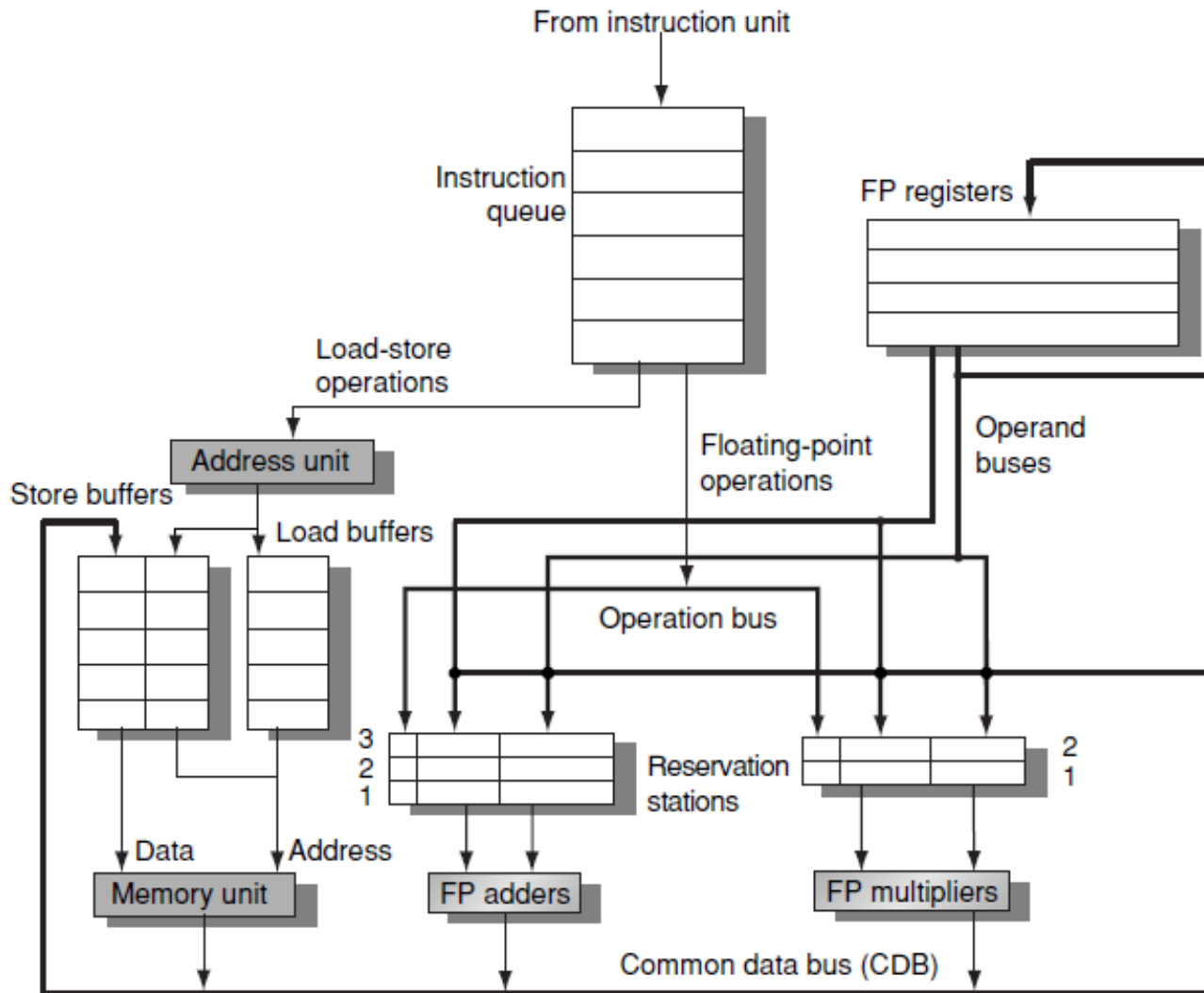
Precise Exception
Speculation
Reorder Buffer

Ιάκωβος Μαυροειδής

Αρχιτεκτονική Scoreboard (CDC 6600)



Tomasulo Organization



Tomasulo v. Scoreboard (IBM 360/91 v. CDC 6600)

Pipelined Functional Units

(6 load, 3 store, 3 +, 2 x/÷)

window size: \leq 14 instructions

No issue on structural hazard

WAR: renaming avoids

WAW: renaming avoids

Broadcast results from FU

Control: reservation stations

Multiple Functional Units

(1 load/store, 1 +, 2 x, 1 ÷)

\leq 5 instructions

same

stall completion

stall issue

Write/read registers

central scoreboard

Exception Behavior with ROB

Προσοχή να διατηρούνται

1. Data flow
2. Exception Behavior

Έχουμε μελετήσει
Θα μελετήσουμε

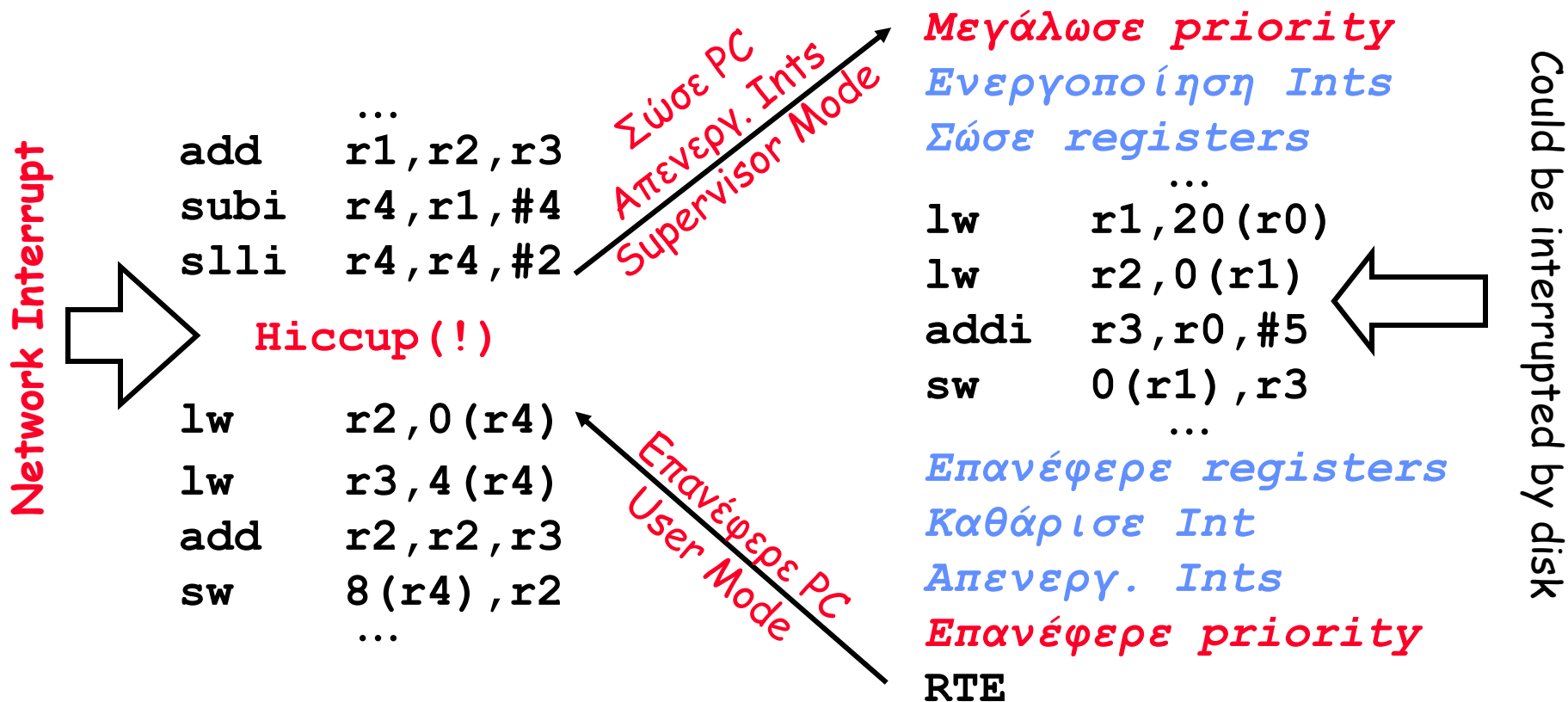
Δυναμικές δρομολόγηση
εντολών (hardware)

- Scoreboard (ελάττωση RAW stalls)
- Register Renaming
 - α) Tomasulo
(ελάττωση WAR και WAW stalls)
 - β) Reorder Buffer
- Branch prediction
(ελάττωση Control stalls)

Στατικές (software/compiler)

- Loop Unrolling
- Software Pipelining
- Trace Scheduling

Device Interrupt



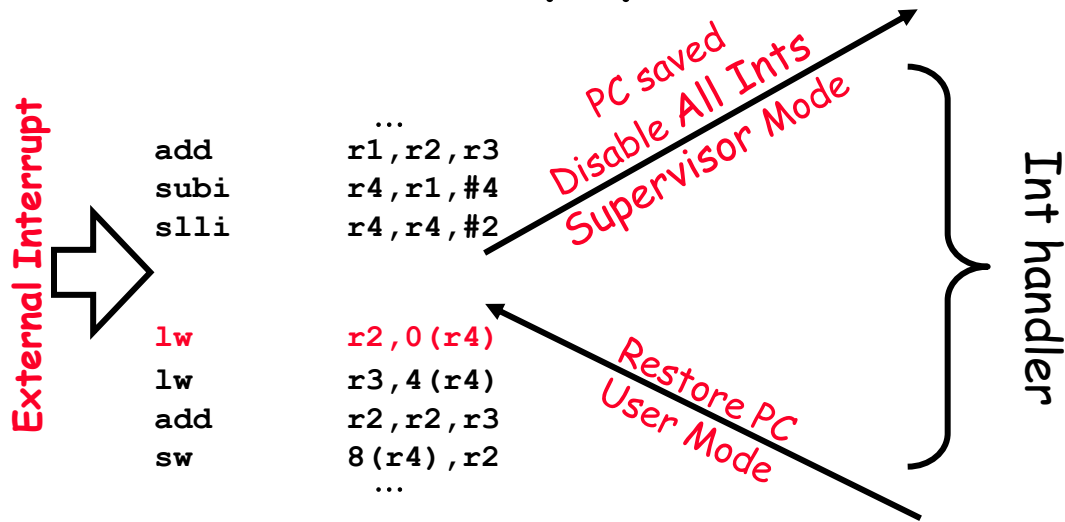
Note that priority must be raised to avoid recursive interrupts!

Types of Exception

- I/O device request
- Invoking an operating system service from a user program
- Tracing instruction execution
- Breakpoint (programmer-requested interrupt)
- Integer arithmetic overflow
- FP arithmetic anomaly
- Page fault (not in main memory)
- Misaligned memory accesses (if alignment is required)
- Memory protection violation
- Using an undefined or unimplemented instruction
- Hardware malfunctions
- Power failure

Precise Interrupts/Exceptions

- Ένα interrupt ή exception ονομάζεται *precise* εάν υπάρχει μία εντολή (ή interrupt point) για το οποίο:
 - Όλες οι προηγούμενες εντολές έχουν πλήρως εκτελεστεί.
 - Καμία εντολή (μαζί με την interrupting instruction) δεν έχει αλλάξει την κατάσταση της μηχανής.
- Αυτό σημαίνει ότι μπορείς να επανακινήσεις την εκτέλεση από το interrupt point και "να πάρεις τα σωστά αποτελέσματα"
 - Στο παράδειγμά μας: Interrupt point είναι η **lw** εντολή



Imprecise Interrupt/Exception

- An exception is *imprecise* if the processor state when an exception is raised does not look **exactly** as if the instructions were executed sequentially in strict program order
- Occurrence in two possibilities:
 - The pipeline may have *already completed instructions that are later in program order*
 - The pipeline may have *not yet completed some instructions that are earlier in program order*

Precise interrupt point απαιτεί πολλάπλα PCs όταν υπάρχουν delayed branches

```
addi  r4,r3,#4
sub   r1,r2,r3
PC:  bne  r1,there
PC+4: and  r2,r3,r5
<other insts>
```

← Interrupt point described as $\langle PC, PC+4 \rangle$

```
addi  r4,r3,#4
sub   r1,r2,r3
PC:  bne  r1,there
PC+4: and  r2,r3,r5
<other insts>
```

Interrupt point described as:
← $\langle PC+4, there \rangle$ (branch was taken)
or
 $\langle PC+4, PC+8 \rangle$ (branch was not taken)

Γιατί χρειαζόμαστε τα precise interrupts?

- Αρκετά interrupts/exceptions χρειάζονται να είναι restartable
 - I.e. TLB faults. Πρέπει να διορθώσει translation, και μετά restart load/store
 - IEEE gradual underflow, illegal operation, etc:

e.g. Για παράδειγμα : $f(x) = \frac{\sin(x)}{x}$
Για $x \rightarrow 0$

$$f(0) = \frac{0}{0} \Rightarrow NaN + illegal_operation$$

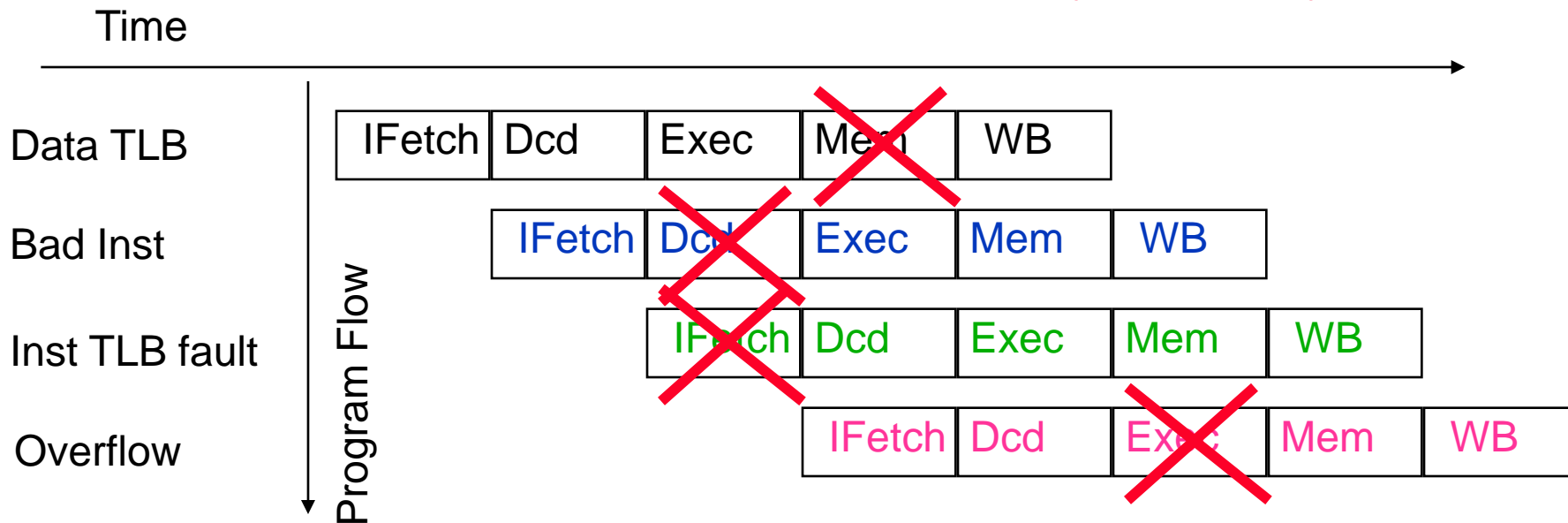
Want to take exception, replace NaN with 1, then restart.

- Restartability δεν απαιτεί preciseness. Ωστόσο, με preciseness είναι **πολύ πιο εύκολη** η επανεκίνηση.
- Απλοποιεί το λειτουργικό σύστημα πολύ
 - **Less state** needs to be saved away if unloading process.
 - Quick to restart (making for fast interrupts)

Precise Exceptions στην απλή 5-stage pipeline:

- Exceptions μπορούν να συμβούν σε διαφορετικά stages της pipeline (I.e. **out of order**):
 - Arithmetic exceptions occur in execution stage
 - TLB faults can occur in instruction fetch or memory stage
- How we guarantee precise exceptions? Η λύση είναι να μαρκάρεις την εντολή ως "δημιουργεί exception ή όχι" και περίμενε μέχρι το τέλος της MEM stage για να σηκώσεις το exception
 - Interrupts become marked NOPs (like bubbles) that are placed into pipeline instead of an instruction.
 - Assume that interrupt condition persists in case NOP flushed
 - Clever instruction fetch might start fetching instructions from interrupt vector, but this is complicated by need for supervisor mode switch, saving of one or more PCs, etc

Another look at the exception problem



- Χρήση της pipeline!
 - Κάθε εντολή έχει ένα *exception status*.
 - Καταγραφή *PCs* για κάθε εντολή στην pipeline.
 - Έλεξε *exception* όταν η εντολή φτάσει το *WB stage*
- Όταν η εντολή φτάσει το *WB stage* και έχει *exception*:
 - Σώσε *PC* \Rightarrow *EPC*, *Interrupt vector addr* \Rightarrow *PC*
 - Μετάτρεψε όλες τις επόμενες εντολές που έχουν γίνει *fetch* σε *NOPs*!
- Δουλεύει επειδή γίνεται *in-order-completion/WB*

Tomasulo Example Cycle 57

Instruction status:

Instruction	<i>j</i>	<i>k</i>	Issue	Exec Comp	Write Result	Busy	Address	
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3	15	16	Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5	56	57		
ADDD	F6	F8	F2	6	10	11		

Reservation Stations:

Time	Name	Busy	Op	<i>S1</i> <i>Vj</i>	<i>S2</i> <i>Vk</i>	<i>RS</i> <i>Qj</i>	<i>RS</i> <i>Qk</i>
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(A2)		(M-M+N	(M-M)	Result		

- In-order issue, out-of-order execution and completion.

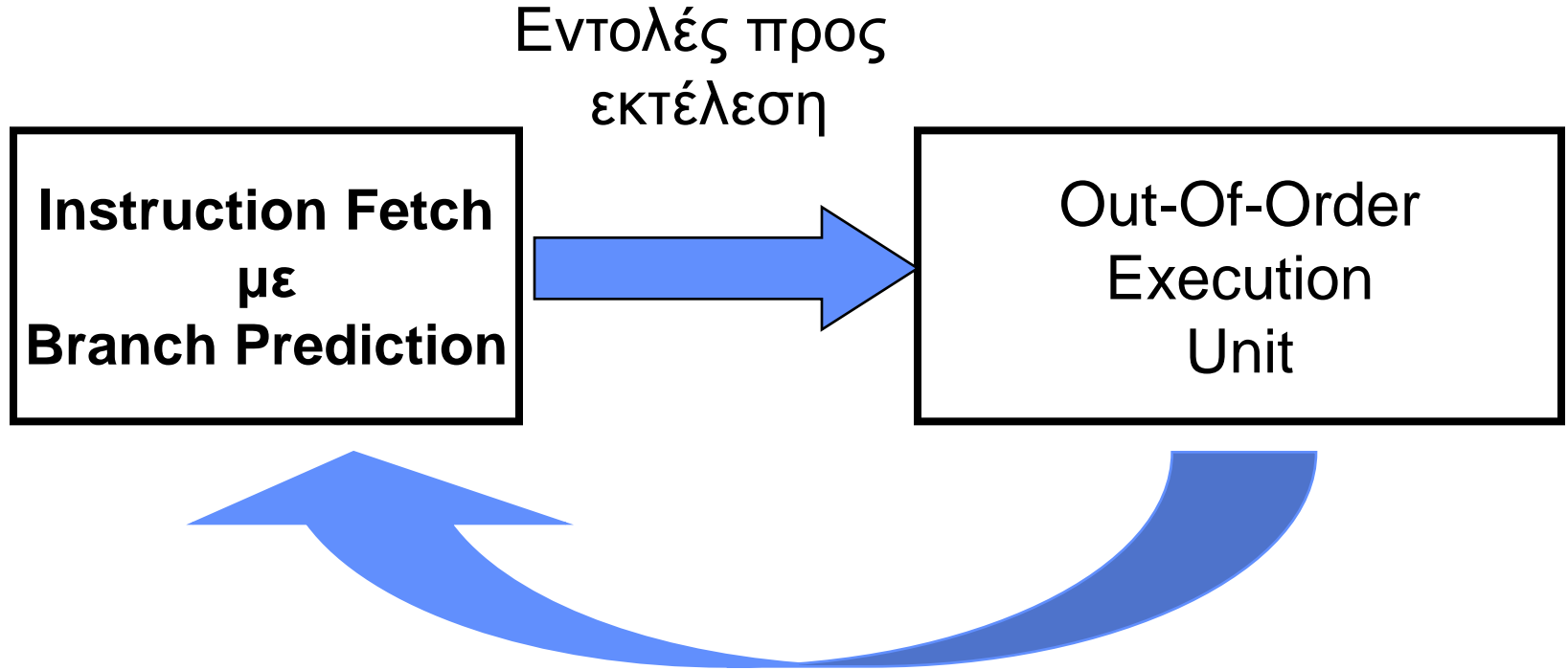
Scoreboard (out-of-order compl)

Instruction status:

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>Read Oper</i>	<i>Exec Comp</i>	<i>Write Result</i>	
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	5	6	7	8
MULTD	F0	F2	F4	6	9	19	20
SUBD	F8	F6	F2	7	9	11	12
DIVD	F10	F0	F6	8	21	61	62
ADDD	F6	F8	F2	13	14	16	22

<i>Issue</i>	<i>Exec Comp</i>	<i>Write Result</i>
1	3	4
2	4	5
3	15	16
4	7	8
5	56	57
6	10	11

Πρόβλημα: "Fetch" unit



Επιστροφή στο σωστό path
όταν βγει το αποτέλεσμα του branch

- Εντολές στο λανθασμένο predicted path έχουν ήδη εκτελεστεί.
- *Instruction fetch decoupled from execution*

Branch πρέπει να εκτελεστεί γρήγορα για loop overlap!

- Στο loop-unrolling παράδειγμα, στηριχθήκαμε ότι τα branches εκτελούνται από μια "γρήγορη" integer unit για να πετύχουμε overlap!

```
Loop:      LD          F0      0      R1
           MULTD     F4      F0     F2
           SD          F4      0      R1
           SUBI      R1      R1     #8
           BNEZ      R1      Loop
```

- Τι συμβαίνει αν το branch εξαρτάται από το αποτέλεσμα του multd??
 - Χάνουμε τελείως όλα τα πλεονεκτήματα!
 - Πρέπει να μπορούμε να μαντεύουμε "predict" το αποτέλεσμα του branch.
 - Αν μαντεύαμε ότι το branch είναι συνέχεια taken, θα είμασταν σωστοί τις περισσότερες φορές.

Prediction: Branches, Dependencies, Data

- Η πρόβλεψη είναι απαραίτητη για καλή απόδοση.
- Μελετήσαμε πώς προβλέπονται branches στο προηγούμενο μαθήματα. Μοντέρνες αρχιτεκτονικές τώρα προβλέπουν τα πάντα : *data dependencies, actual data, and results of groups of instructions:*
- Γιατί δουλεύει η πρόβλεψη?
 - Underlying algorithm has regularities.
 - Data that is being operated on has regularities.
 - Instruction sequence has redundancies that are artifacts of way that humans/compiler think about problems.

Πρόβλημα: out-of-order completion

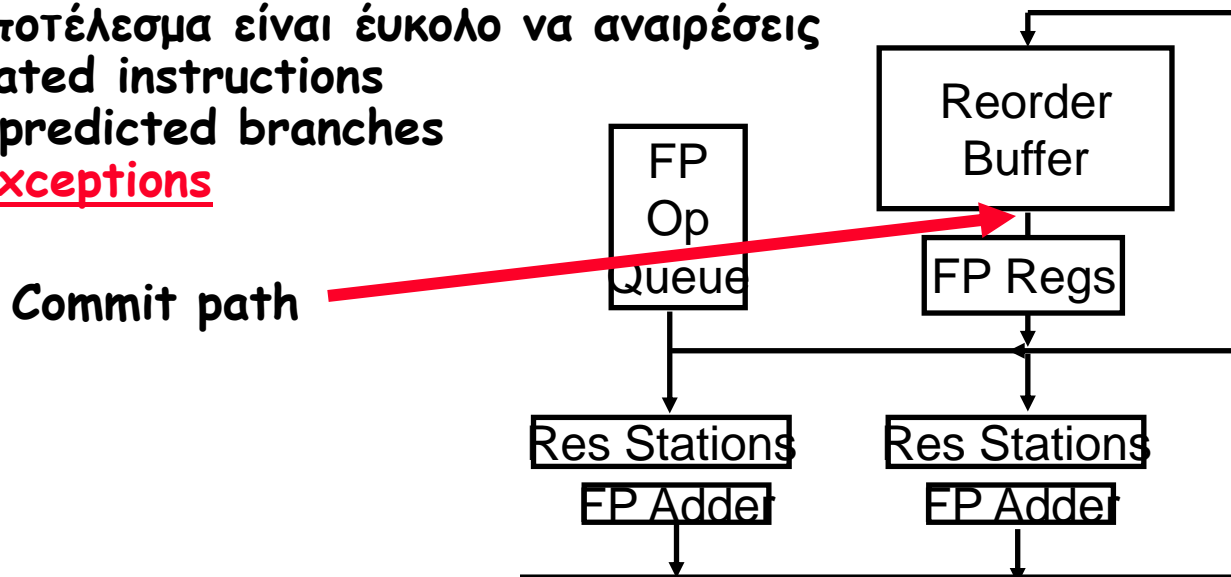
- Scoreboard και Tomasulo έχουν:
 - *In-order issue, out-of-order execution, out-of-order completion*
- Τρόπος να συγχρονίσεις το completion στάδιο των εντολών με την σειρά στο πρόγραμμα (i.e. with issue-order)
 - Easiest way is with *in-order completion (i.e. reorder buffer)*
 - Other Techniques (Smith paper): Future File, History Buffer

Σχέση μεταξύ Precise Interrupts και Speculation:

- Στο Issue στάδιο εντολών είναι σαν να προβλέπεις ότι οι προηγούμενες εντολές δεν έχουν exception.
 - 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!
- Τεχνική για precise interrupts/exceptions και speculation: *in-order completion or commit*
 - Γι'αυτό συναντάμε reorder buffers σε όλους τους καινούριους επεξεργαστές

Υποστήριξη precise interrupts από HW

- Ιδέα του Reorder Buffer (ROB):
 - Κράτα εντολές σε μία FIFO, ακριβώς με την σειρά που γίνονται issue.
 - » Each ROB entry contains PC, dest reg, result, exception status
 - Όταν η εντολή τελειώσει την εκτέλεση, τοποθέτησε τα αποτελέσματα στον ROB.
 - » Supplies operands to other instruction between execution complete & commit \Rightarrow more registers like RS
 - » Tag results with ROB buffer number instead of reservation station
 - Η εντολή αλλάζει την κατάσταση της μηχανής στο **commit στάδιο** όχι στο WB \Rightarrow in order commit \Rightarrow values at head of ROB placed in registers
 - Σαν αποτέλεσμα είναι εύκολο να αναιρέσεις speculated instructions σε mispredicted branches ή σε exceptions



Τέσσερα Στάδια του Tomasulo Αλγόριθμου με ROB

1. **Issue**—Πάρε εντολή από FP Op Queue

Αν υπάρχουν ελεύθερα reservation station και reorder buffer slot, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")

2. **Execution**—Εκτέλεσε εντολή στο Ex Unit(EX)

Όταν και οι τιμές και των 2 source regs είναι έτοιμες εκτέλεσε εντολή; αν όχι, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")

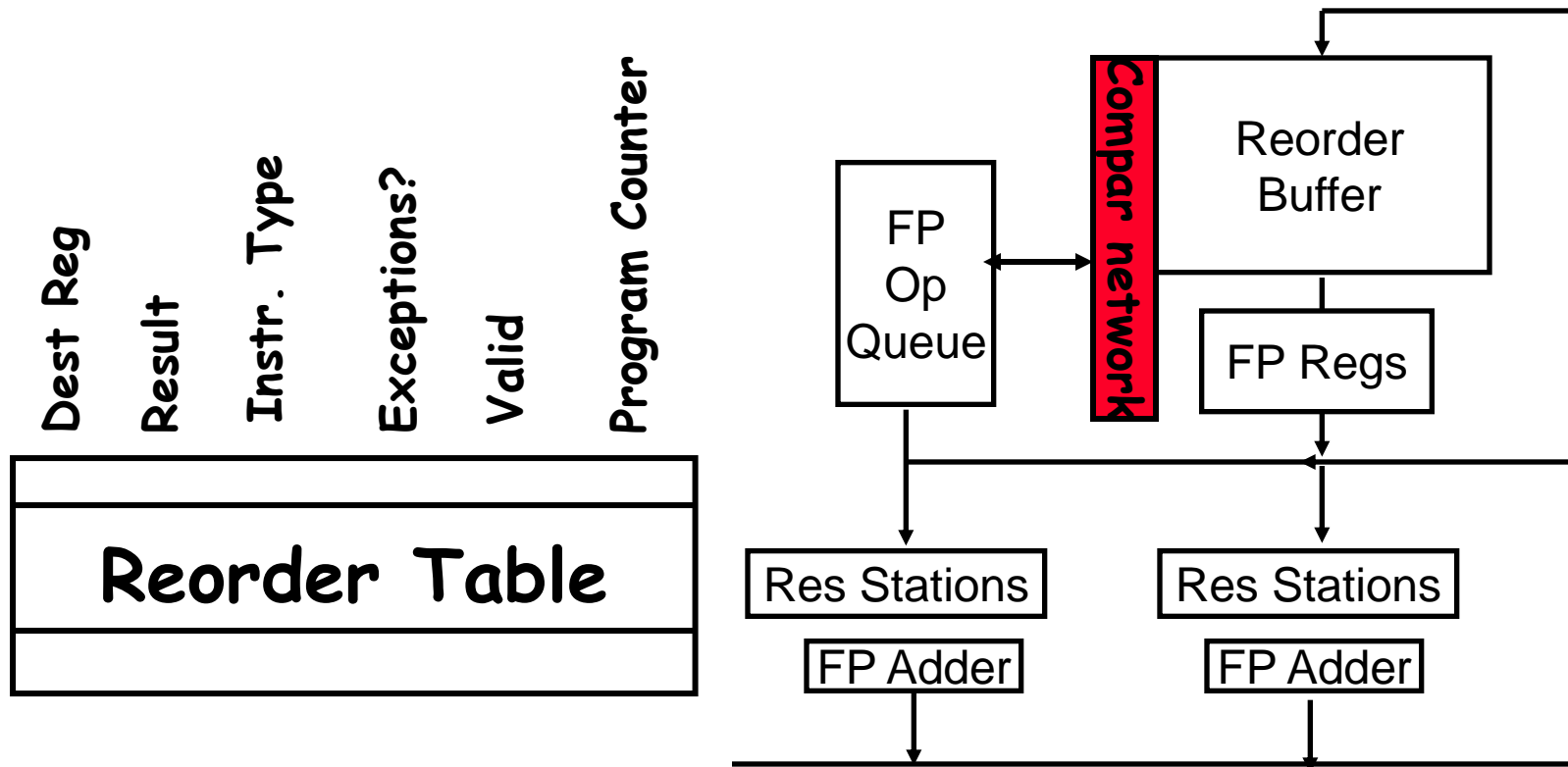
3. **Write result**—Τέλος εκτέλεσης (WB)

Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

4. **Commit**—Άλλαξε τιμή του dest register με το αποτέλεσμα από το reorder buffer

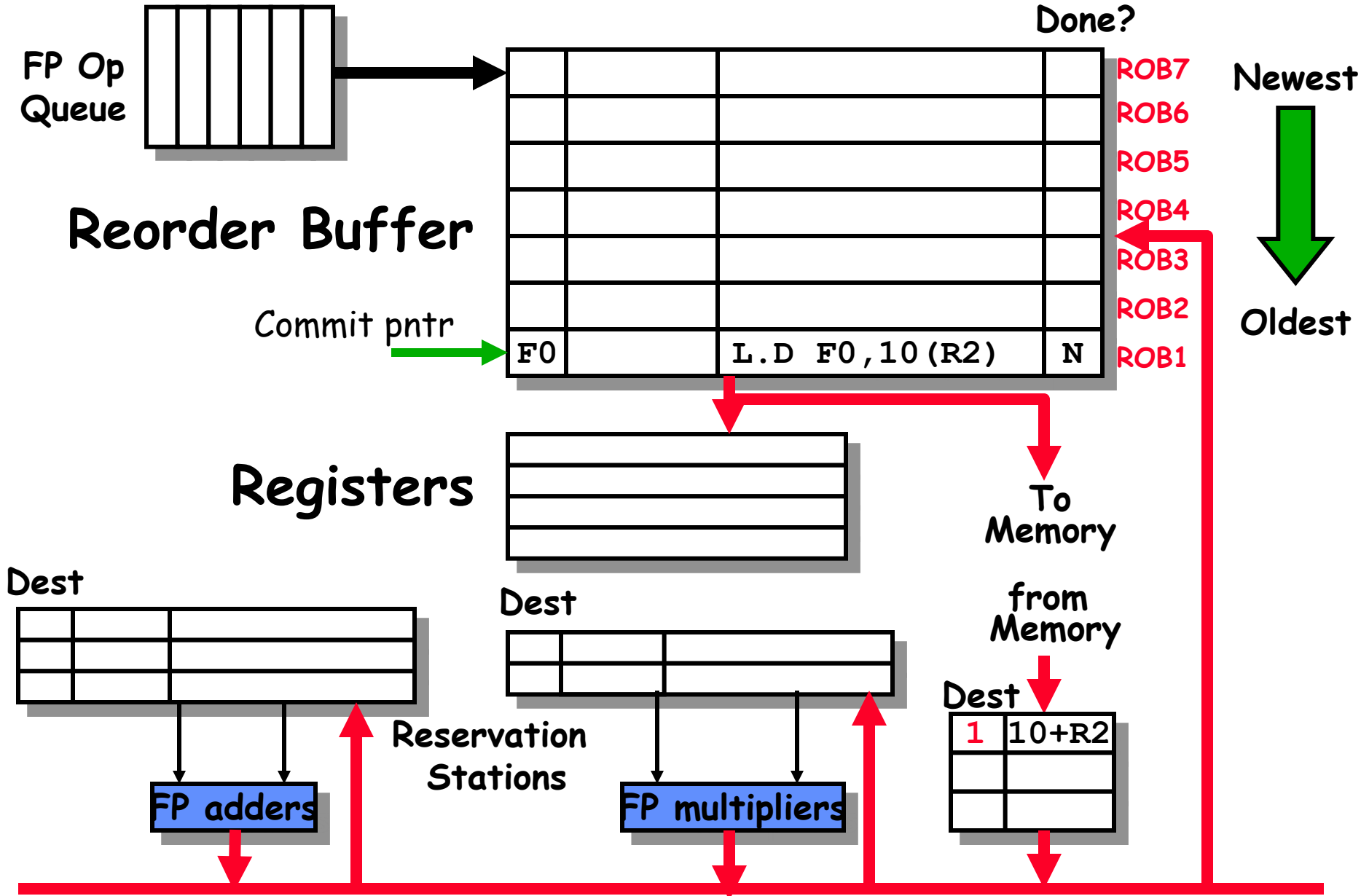
When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

HW με reorder buffer (ROB)?

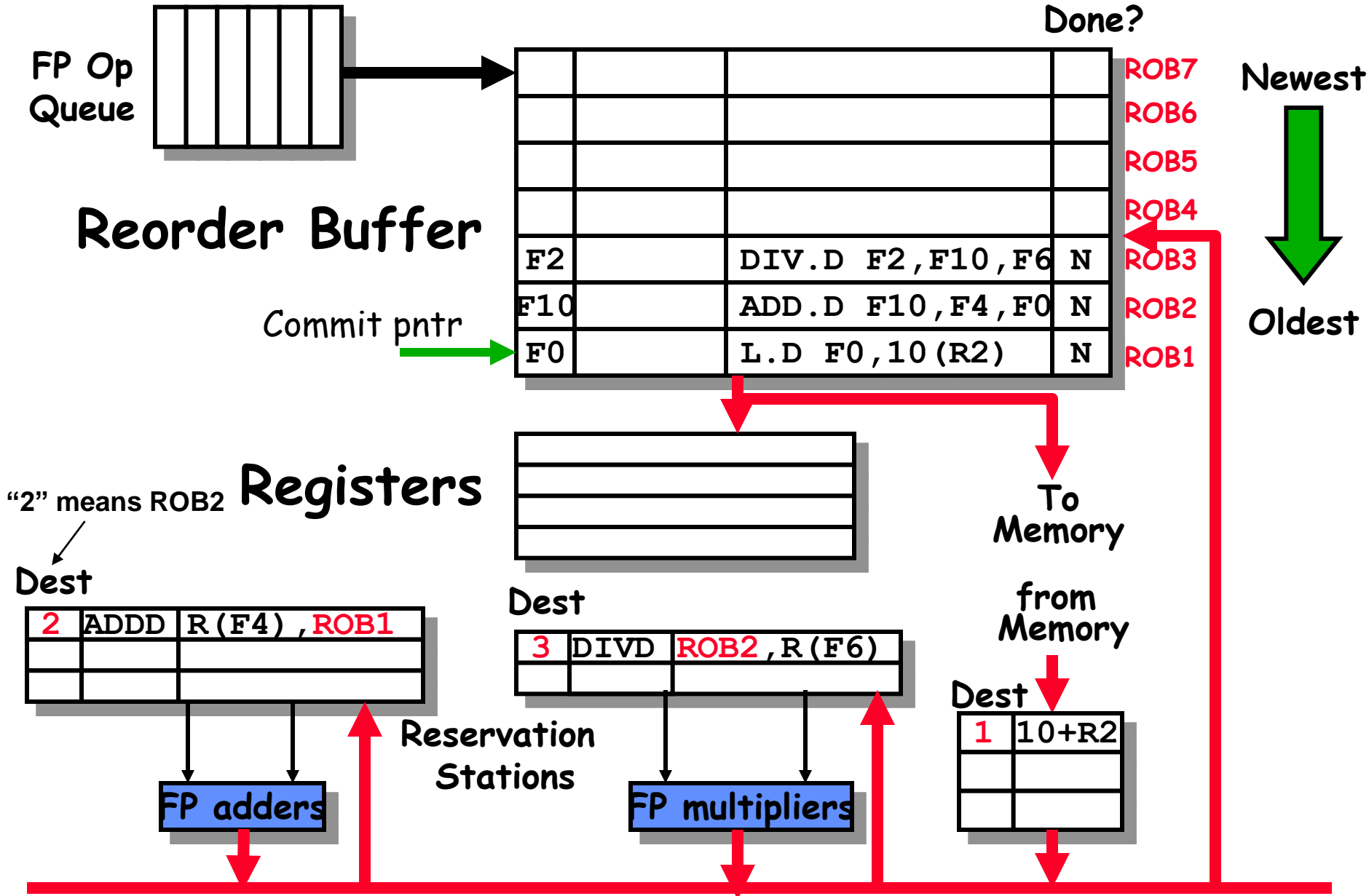


- Πώς βρίσκουμε την τελευταία έκδοση του register?
- Πολύπορτο ROB σαν να είναι register file

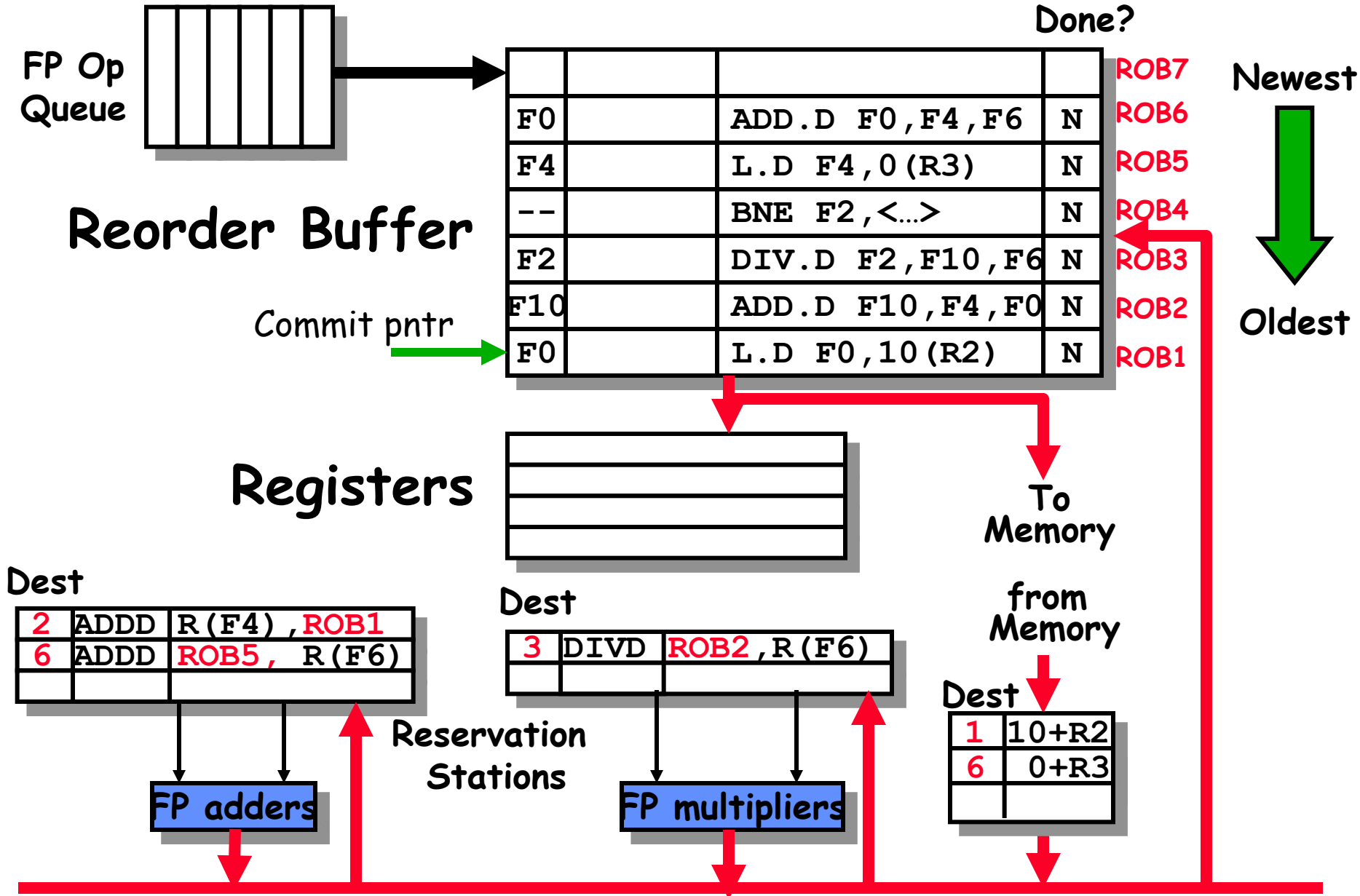
Tomasulo With Reorder buffer:



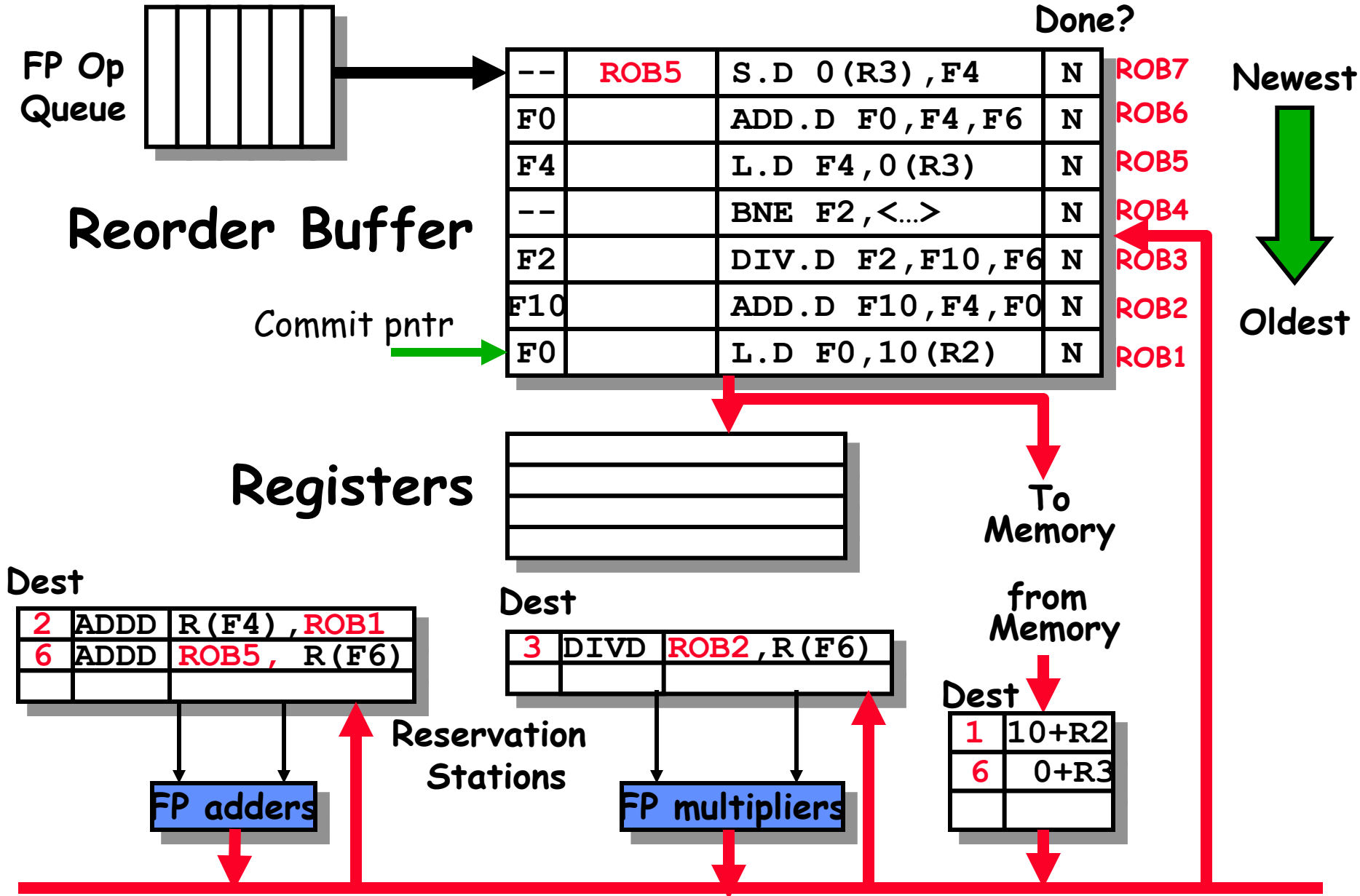
Tomasulo With Reorder buffer:



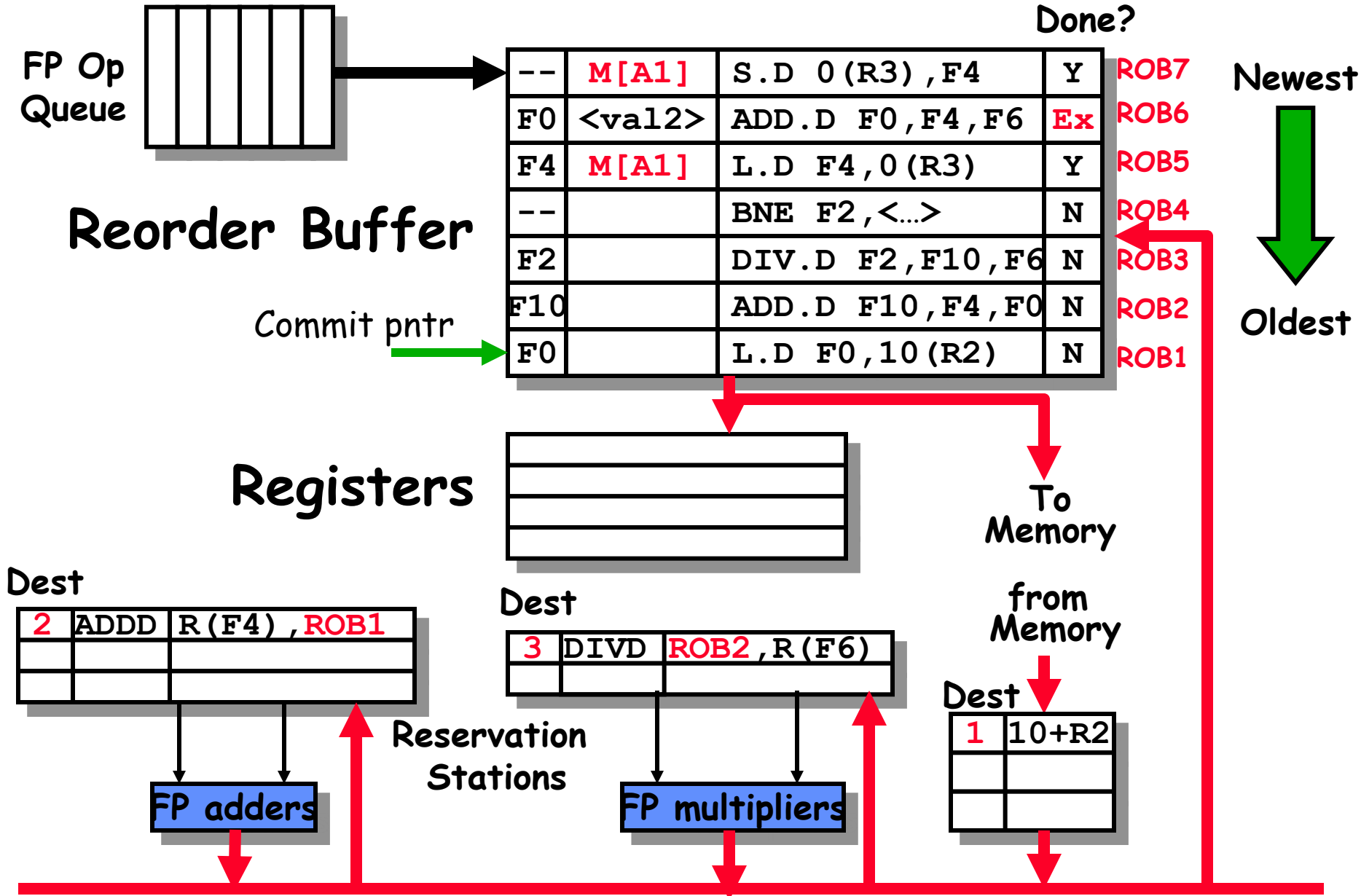
Tomasulo With Reorder buffer:



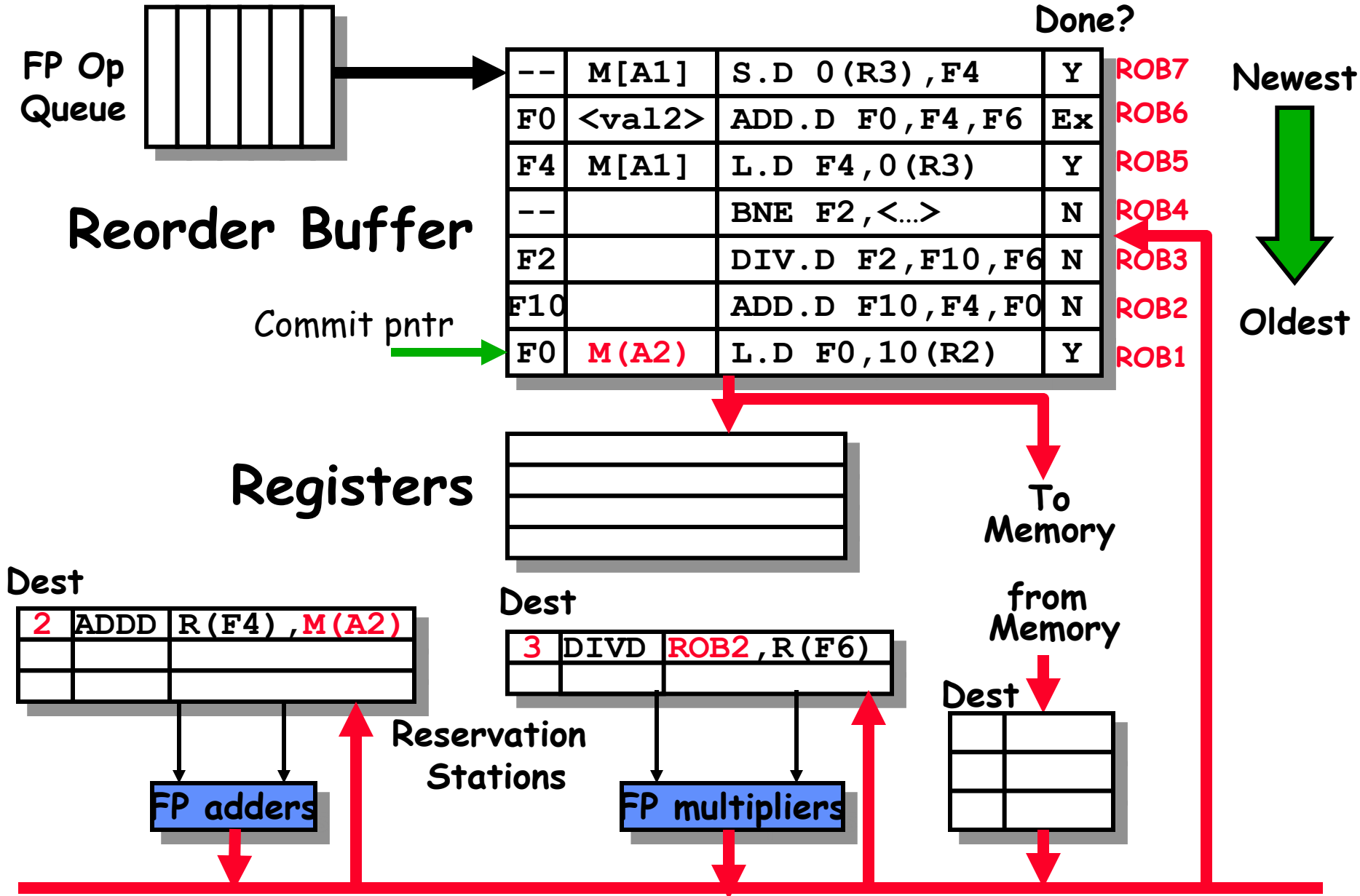
Tomasulo With Reorder buffer:



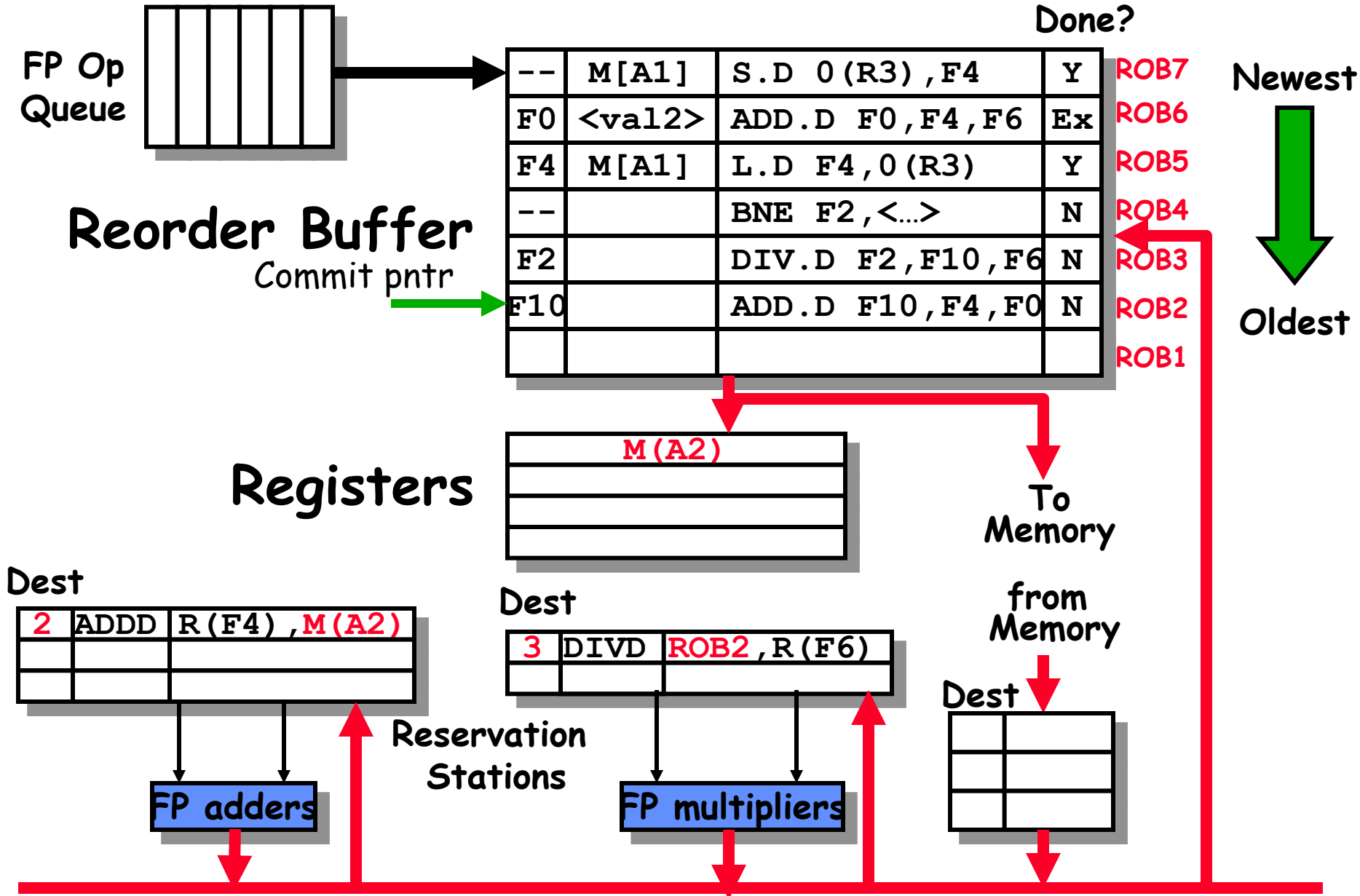
Tomasulo With Reorder buffer:



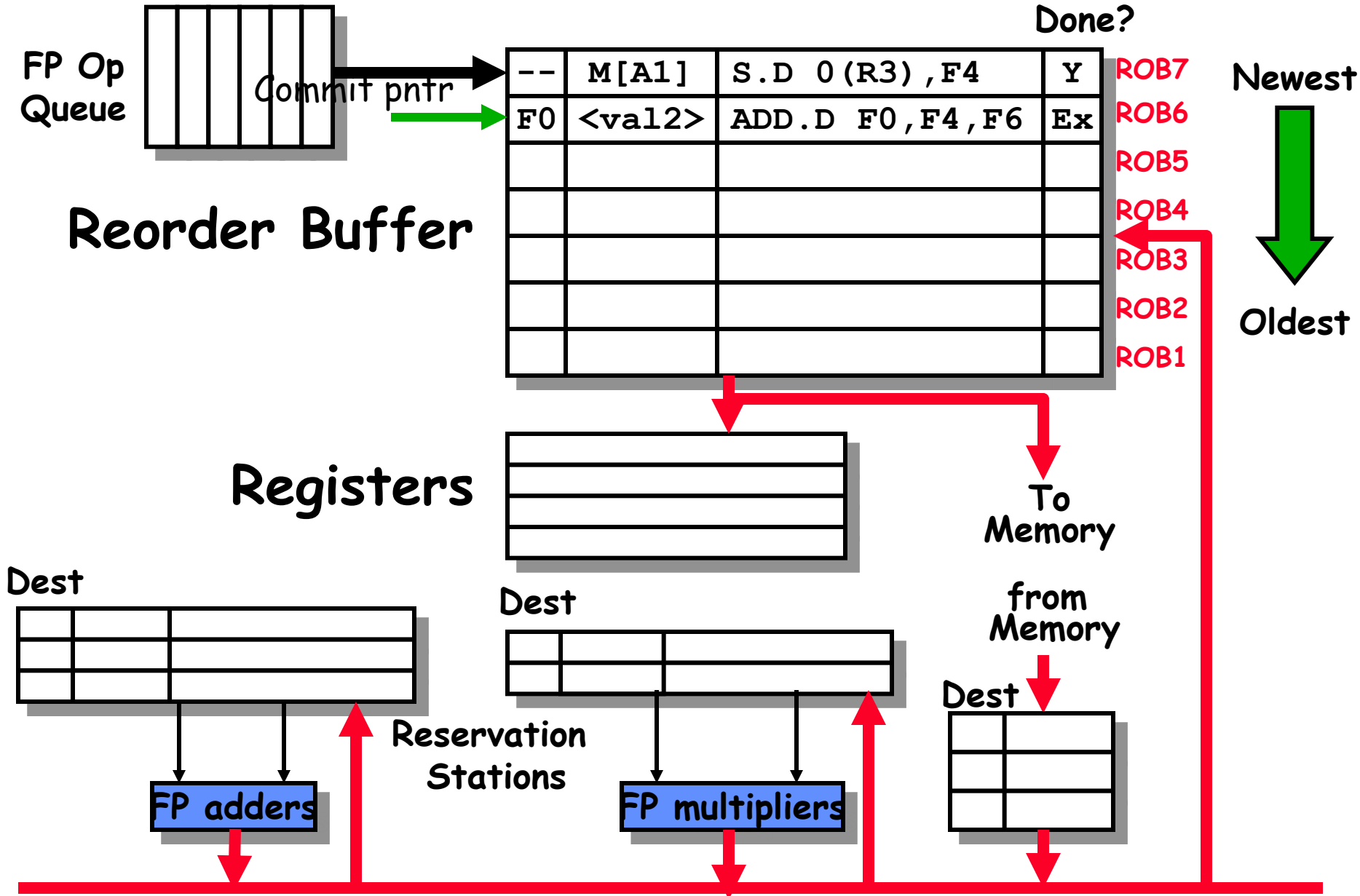
Tomasulo With Reorder buffer:



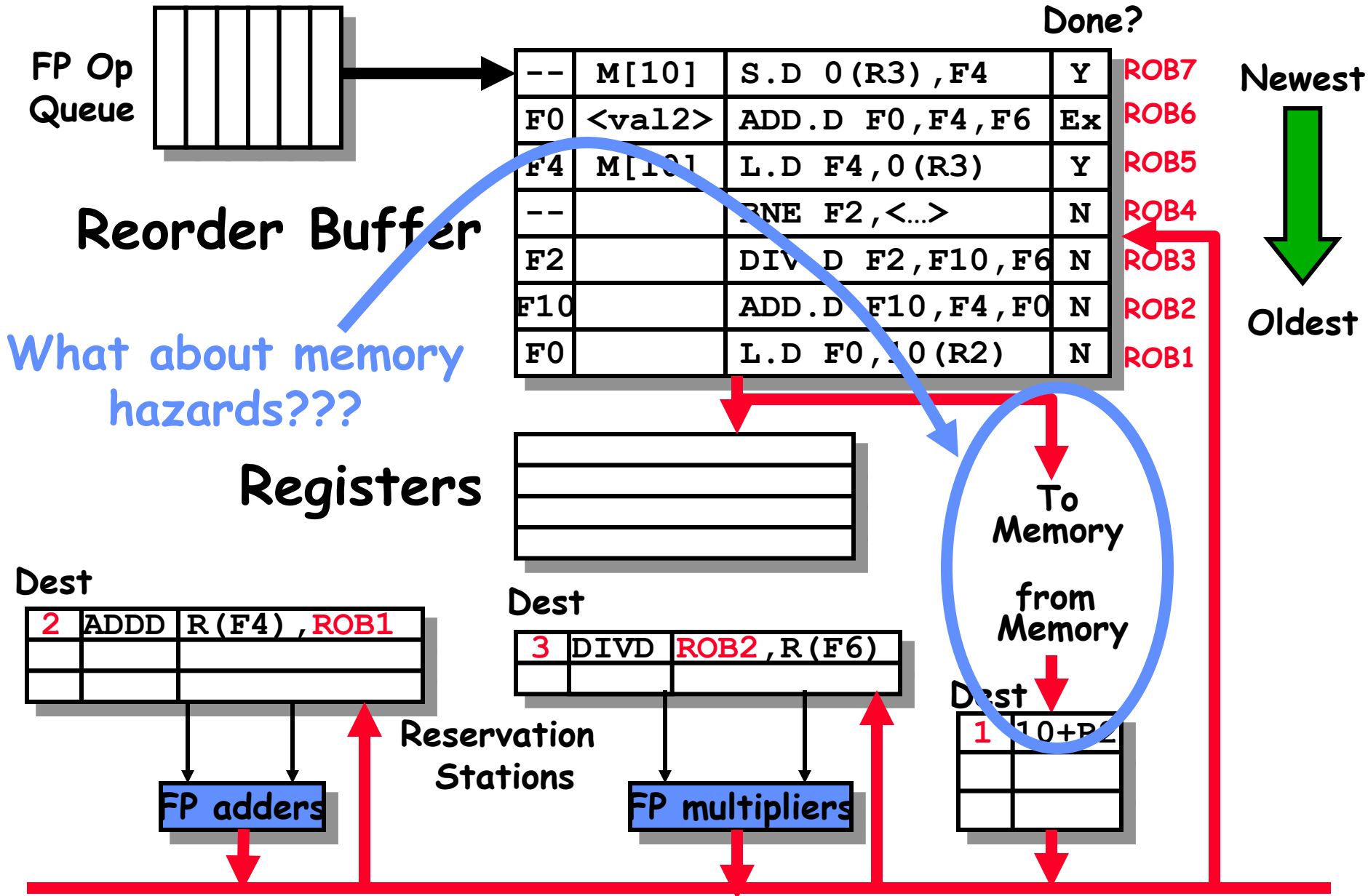
Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:



Memory Disambiguation: WAW/WAR Hazards στη μνήμη

- Like Tomasulo's algorithm, we must avoid hazards through memory:
 - WAW and WAR hazards through memory are eliminated with speculation because the **actual updating of memory occurs in order**, when a store is at the head of the ROB, and hence, no earlier loads or stores can still be pending.

Memory Disambiguation: Λύση στα RAW Hazards στη μνήμη

- Ερώτηση: Given a load that follows a store in program order, are the two related?
 - (υπάρχει ένα RAW hazard ανάμεσα στο store και στο load)?

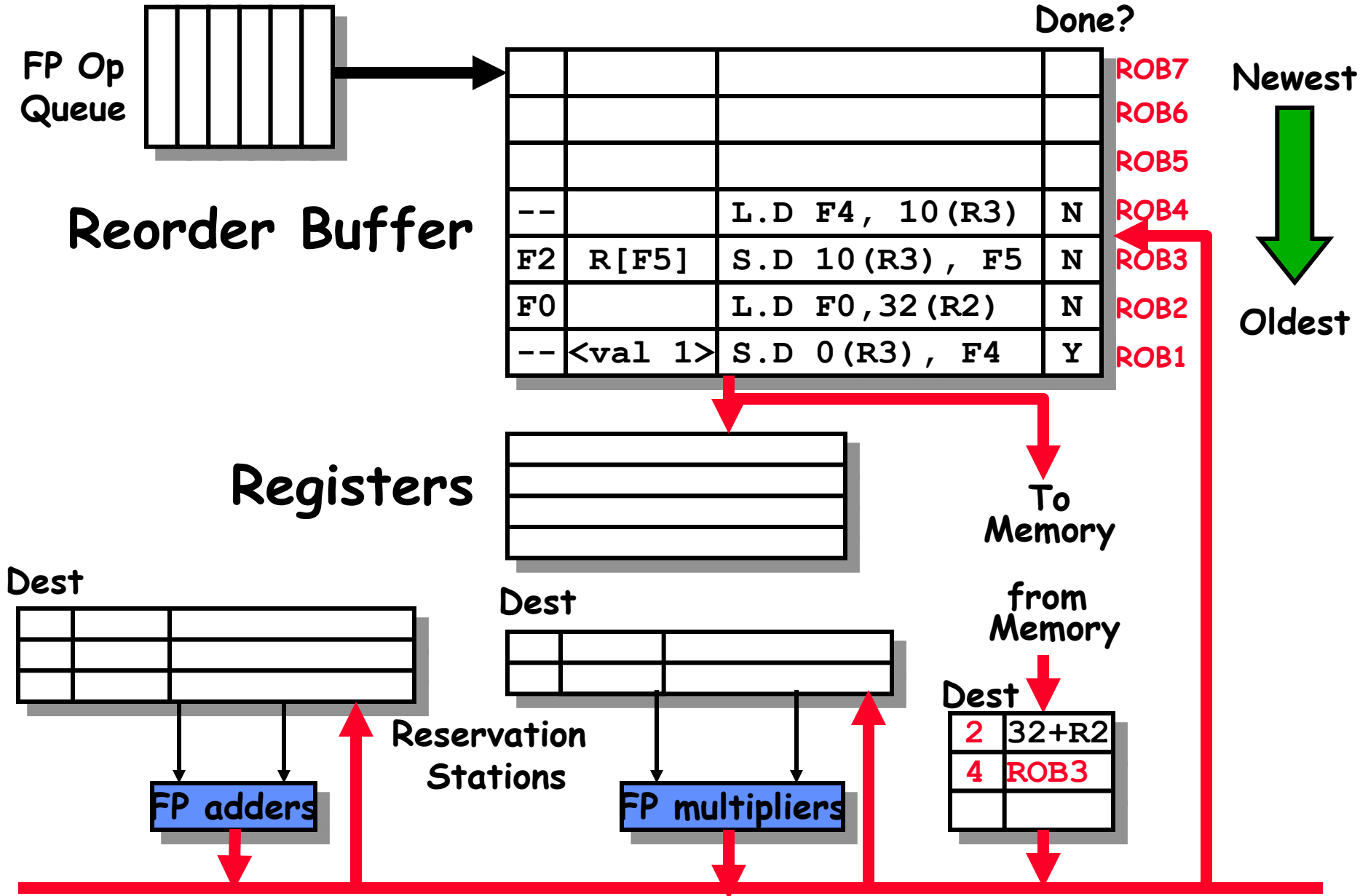
Eg: st 0 (R2) , R5
 ld R6 , 0 (R3)

- Μπορούμε να προχωρήσουμε και να αρχίσουμε το load ?
 - Store address could be delayed for a long time by some calculation that leads to R2 (divide?).
 - We might want to issue/begin execution of both operations in same cycle.
 - Λύση1: Answer is that we are not allowed to start load until we know that address $0(R2) \neq 0(R3)$
 - Λύση2: We might guess at whether or not they are dependent (called "**dependence speculation**") and use reorder buffer to fixup if we are wrong.

Hardware υποστήριξη για Memory Disambiguation

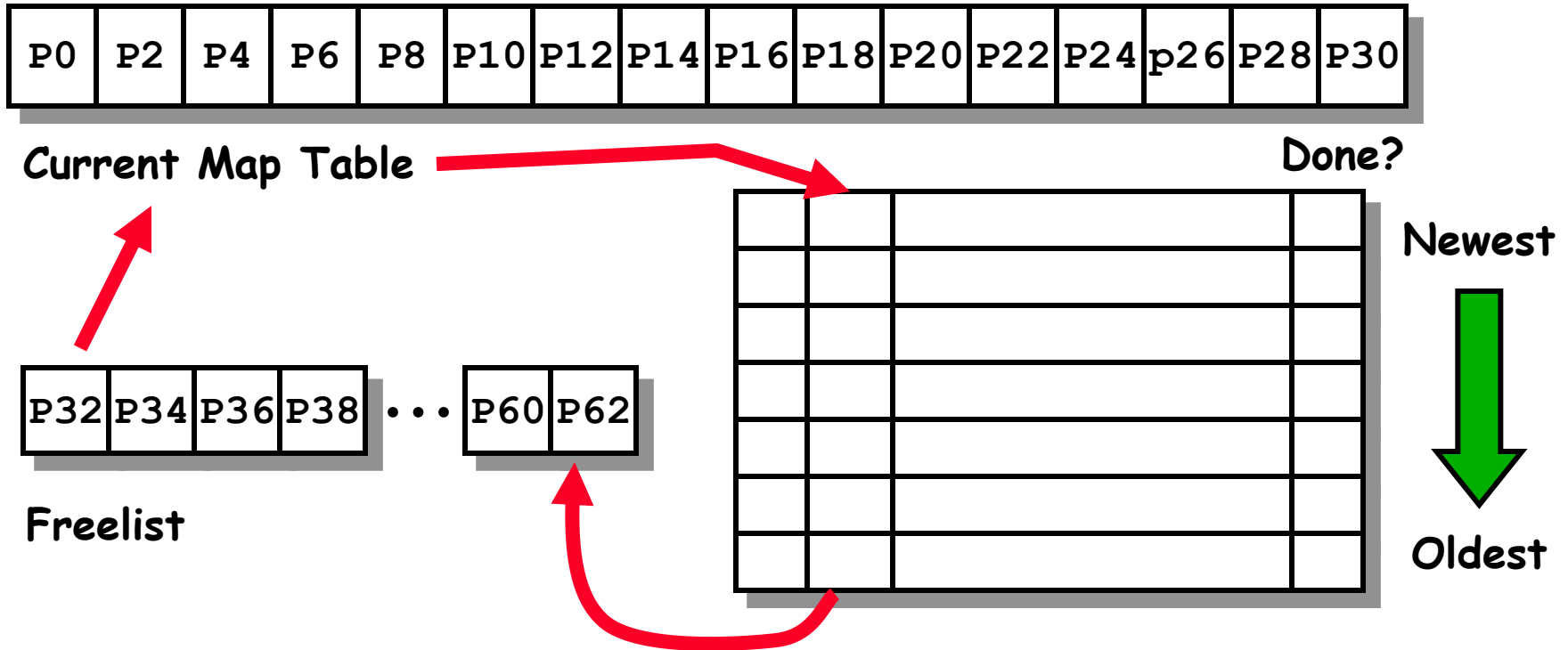
- **Store buffer** που κρατάει όλα τα εκκρεμή stores στη memory, σε program order.
 - Keep track of address (when becomes available) and value (when becomes available)
 - FIFO ordering: will retire stores from this buffer in program order
- Όταν κάνεις issue ένα load, κατέγραψε το head του buffer (γνώριζε ποια stores προηγούνται από εσένα).
- Όταν έχεις την διεύθυνση του load, έλεγξε τον buffer:
 - If *any* store prior to load is waiting for its address, stall load.
 - If load address matches earlier store address (associative lookup), then we have a *memory-induced RAW hazard*:
 - » store value available \Rightarrow return value
 - » store value not available \Rightarrow return ROB number of source
 - Otherwise, send out request to memory
- Τα stores περνάνε το commit στάδιο in order, άρα δεν υπάρχουν WAW hazards στη μνήμη.

Memory Disambiguation:



Explicit register renaming:

Hardware equivalent of static,
single-assignment (SSA) compiler form



- Physical register file μεγαλύτερα από ISA register file (π.χ. 32 phys regs και 16 ISA regs)
- Στο issue, κάθε εντολή που αλλάζει έναν register παίρνει έναν καινούριο physical register από την freelist
- Used on: R10000, Alpha 21264, HP PA8000

Explicit register renaming:

Hardware equivalent of static, single-assignment (SSA) compiler form

P32	P2	P4	P6	P8	P10	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30
-----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Current Map Table

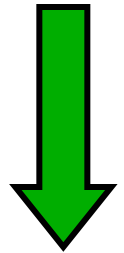
Done?

P34	P36	P38	P40	...	P60	P62
-----	-----	-----	-----	-----	-----	-----

Freelist

F0	P0	LD P32, 10 (R2)		N

Newest



Oldest

- 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:

Hardware equivalent of static, single-assignment (SSA) compiler form

P32	P2	P4	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30
-----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Current Map Table

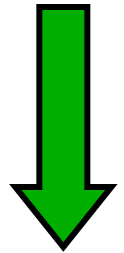
Done?

P36	P38	P40	P42	...	P60	P62
-----	-----	-----	-----	-----	-----	-----

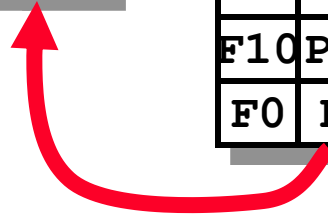
Freelist

F10	P10	ADDD P34, P4, P32	N
F0	P0	LD P32, 10(R2)	N

Newest



Oldest



Explicit register renaming:

Hardware equivalent of static, single-assignment (SSA) compiler form

P32	P36	P4	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30
-----	-----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Current Map Table

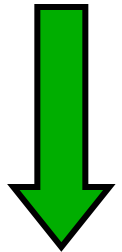
Done?

Freelist

P38	P40	P44	P48	...	P60	P62
-----	-----	-----	-----	-----	-----	-----

--				
--				
--				
--		BNE	P36, <...>	N
F2	P2	DIV	P36, P34, P6	N
F10	P10	ADD	P34, P4, P32	N
F0	P0	LD	P32, 10 (R2)	N

Newest



Oldest

P32	P36	P4	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30
-----	-----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

P38	P40	P44	P48	...	P60	P62
-----	-----	-----	-----	-----	-----	-----

Checkpoint at BNE instruction

Explicit register renaming:

Hardware equivalent of static, single-assignment (SSA) compiler form

P40	P36	P38	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30
-----	-----	-----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Current Map Table

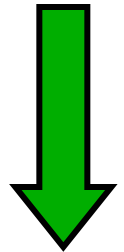
Done?

P42	P44	P48	P50	...	P0	P10
-----	-----	-----	-----	-----	----	-----

Freelist

--		ST 0 (R3) , P40	Y
F0	P32	ADDD P40 , P38 , P6	Y
F4	P4	LD P38 , 0 (R3)	Y
--		BNE P36 , <...>	N
F2	P2	DIVD P36 , P34 , P6	N
F10	P10	ADDD P34 , P4 , P32	y
F0	P0	LD P32 , 10 (R2)	y

Newest



Oldest

P32	P36	P4	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30
-----	-----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

P38	P40	P44	P48	...	P60	P62
-----	-----	-----	-----	-----	-----	-----

Checkpoint at BNE instruction

Explicit register renaming:

Hardware equivalent of static, single-assignment (SSA) compiler form

P32	P36	P4	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30
-----	-----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Current Map Table

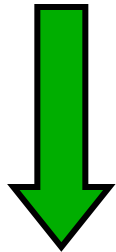
Done?

P38	P40	P44	P48	P52	P56	P60	P62
-----	-----	-----	-----	-----	-----	-----	-----

Freelist

F2	P2	DIVD P36, P34, P6	N	
F10	P10	ADDD P34, P4, P32	y	
F0	P0	LD P32, 10(R2)	y	

Newest



Oldest

Speculation error fixed by restoring map table and freelist

P32	P36	P4	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30
-----	-----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

P38	P40	P44	P48	...	P60	P62
-----	-----	-----	-----	-----	-----	-----

Checkpoint at BNE instruction