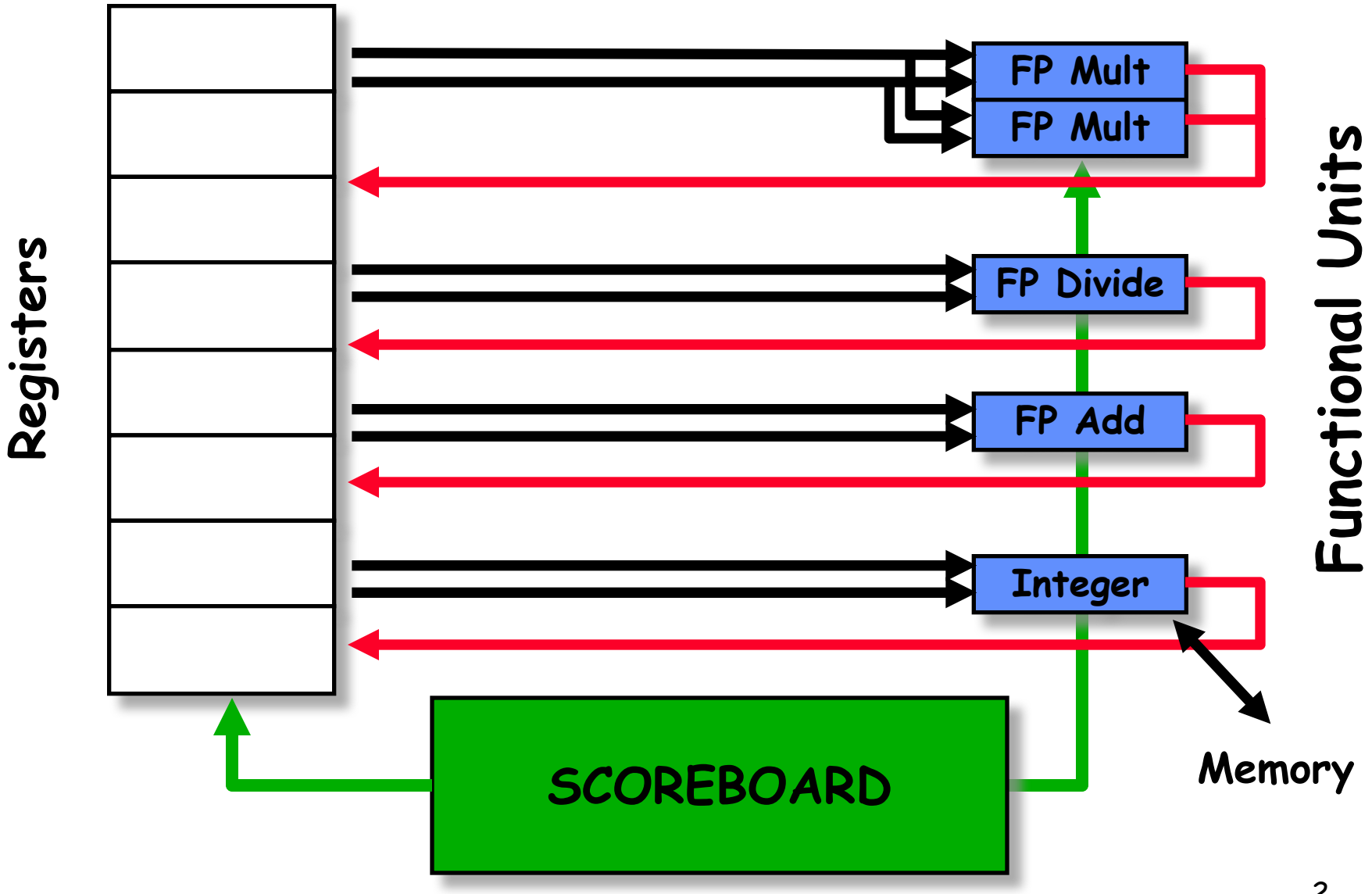
HY425
Αρχιτεκτονική Υπολογιστών

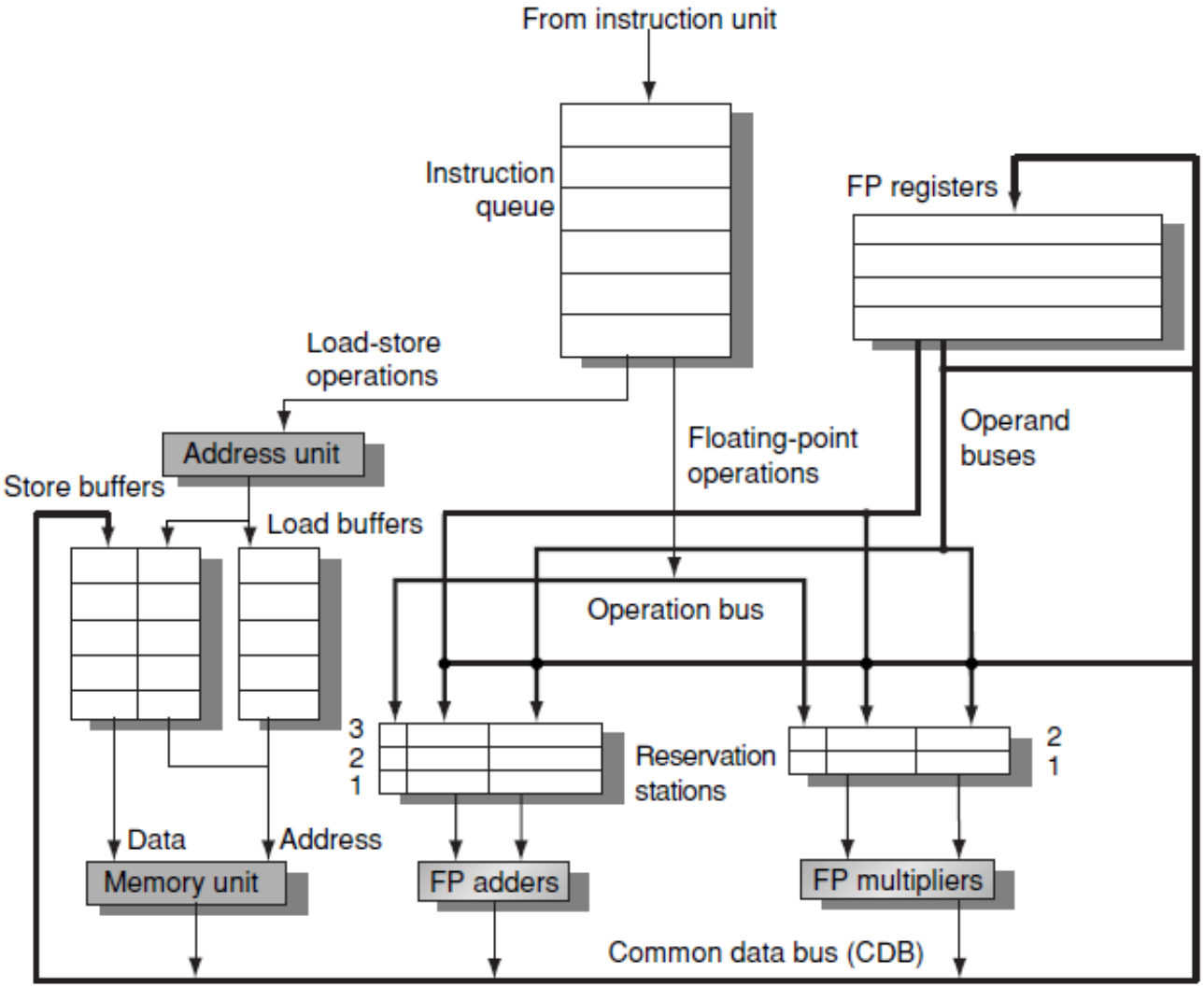
Precise Exceptions
Speculation
Reorder Buffer

Βασίλης Παπαευσταθίου
Ιάκωβος Μαυροειδής

Αρχιτεκτονική Scoreboard (CDC 6600)



Tomasulo Organization



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

Pipelined Functional Units	Multiple Functional Units
(6 load, 3 store, 3 +, 2 x/÷)	(1 load/store, 1 + , 2 x, 1 ÷)
window size: ≤ 14 instructions	≤ 5 instructions
No issue on structural hazard	same
WAR: renaming avoids	stall completion
WAW: renaming avoids	stall issue
Broadcast results from FU	Write/read registers
Control: reservation stations	central scoreboard

Exception Behavior with ROB

$CPI = CPI_{ideal} + Stalls_{structural} + Stalls_{RAW} + Stalls_{WAR} + Stalls_{WAW} + Stalls_{control}$

Προσοχή να διατηρουνται

1. **Data flow**
2. **Exception Behavior**

Έχουμε μελετήσει
Θα μελετήσουμε σημερα
Θα μελετήσουμε σε επόμενα μαθήματα

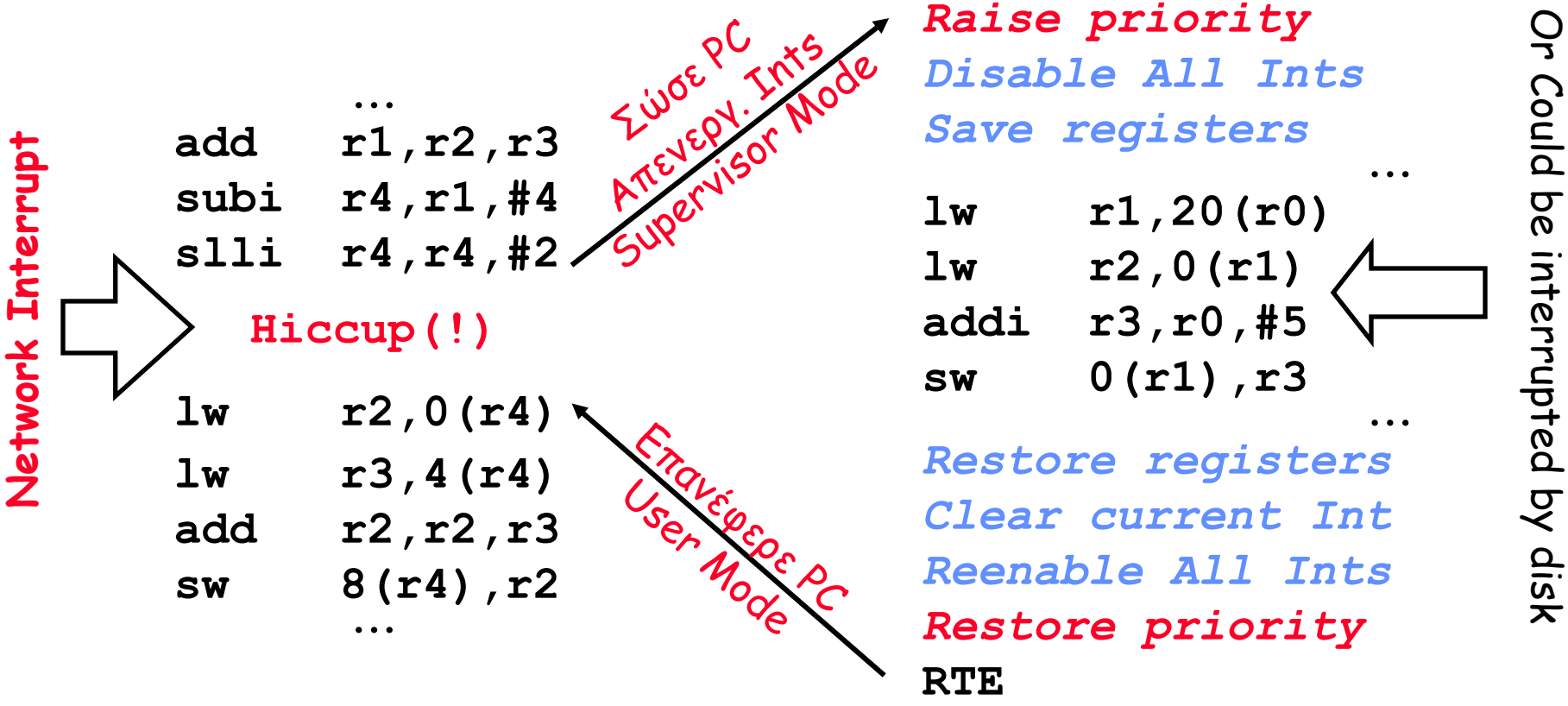
Δυναμικές δρομολόγηση εντολών (hardware)

- **Scoreboard (ελάττωση RAW stalls)**
- **Register Renaming**
  - **α)Tomasulo**
    - **(ελάττωση WAR και WAW stalls)**
  - **β)Reorder Buffer**
- **Branch prediction**
  - **(ελάττωση Control stalls)**
- **Multiple Issue (CPI < 1)**
- **Multithreading (CPI < 1)**

Στατικές (shoftware/compiler)

- **Loop Unrolling**
- **Software Pipelining**
- **Trace Scheduling**

Device Interrupt



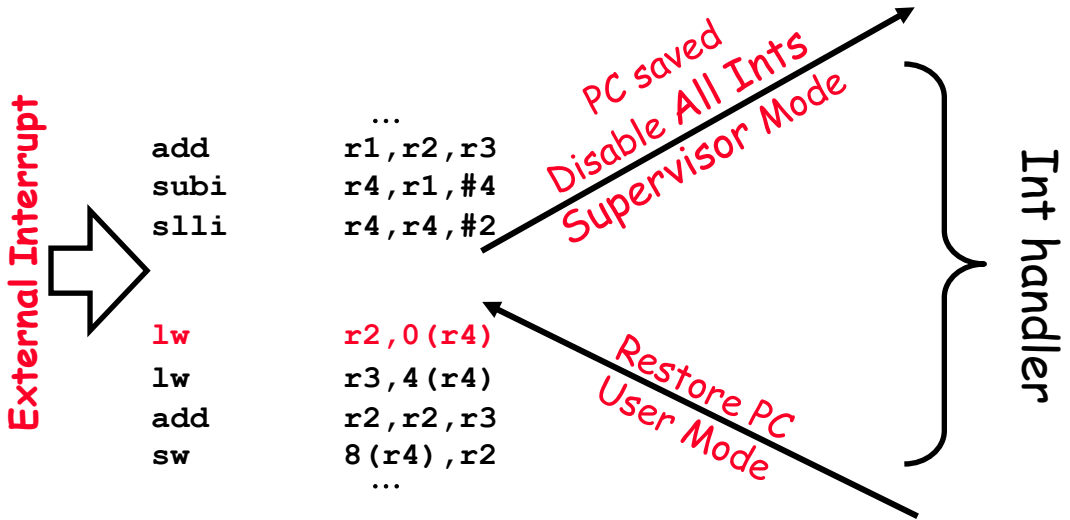
Note that priority must be raised to avoid recursive interrupts!

Types of Interrupts/Exceptions

- I/O device request
- Invoking an operating system service from a user program
- 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 μετά (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 <PC,PC+4>

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

Interrupt point described as:

⇦ <PC+4,there> (branch was taken)

or

<PC+4,PC+8> (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 $x \to 0$

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

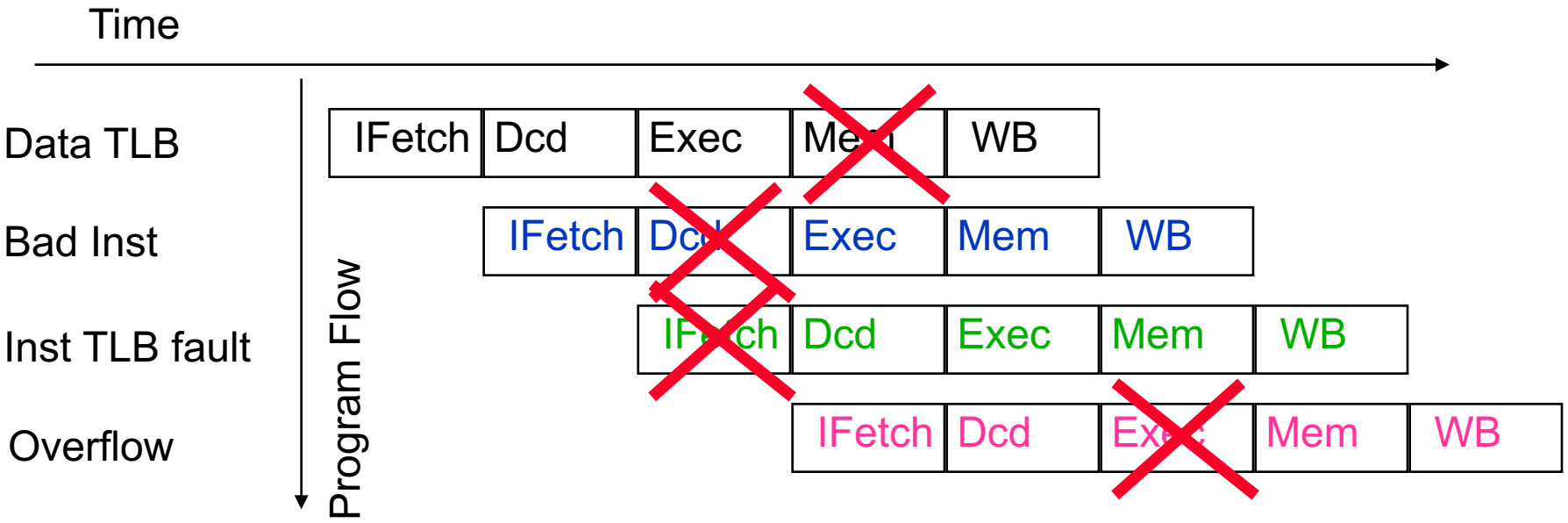
    Want to take exception, replace *NaN* with 1, then restart.

- Restartability δεν απαιτεί preciseness. Ωστόσο, με preciseness είναι *πολύ πιο εύκολη* η επανεκίνηση.
- Απλοποίει 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 ⇒ EPC, Interrupt vector addr ⇒ PC**
  - **Μετάτρεψε όλες τις επόμενες εντολές που έχουν γίνει fetched σε NOPs!**
- **Δουλεύει επειδή γίνεται in-order-completion/WB**

Tomasulo Example Cycle 57

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	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+M	(M-M)	Result			

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

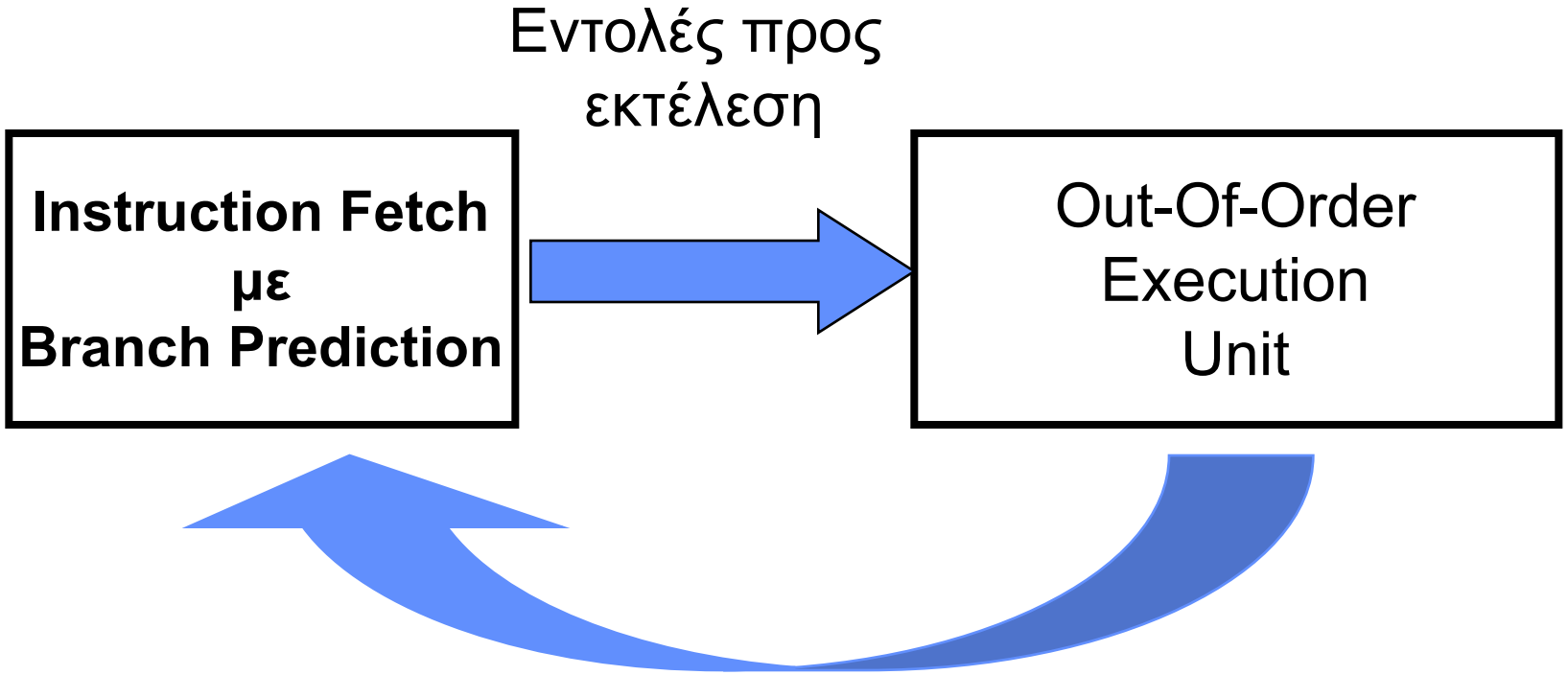
Scoreboard (out-of-order compl)

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
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

Issue	*Exec Comp*	*Write Result*
1	3	4
2	4	5
3	15	16
4	7	8
5	56	57
6	10	11

Πρόβλημα: "Fetch" unit



- **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/compilers 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

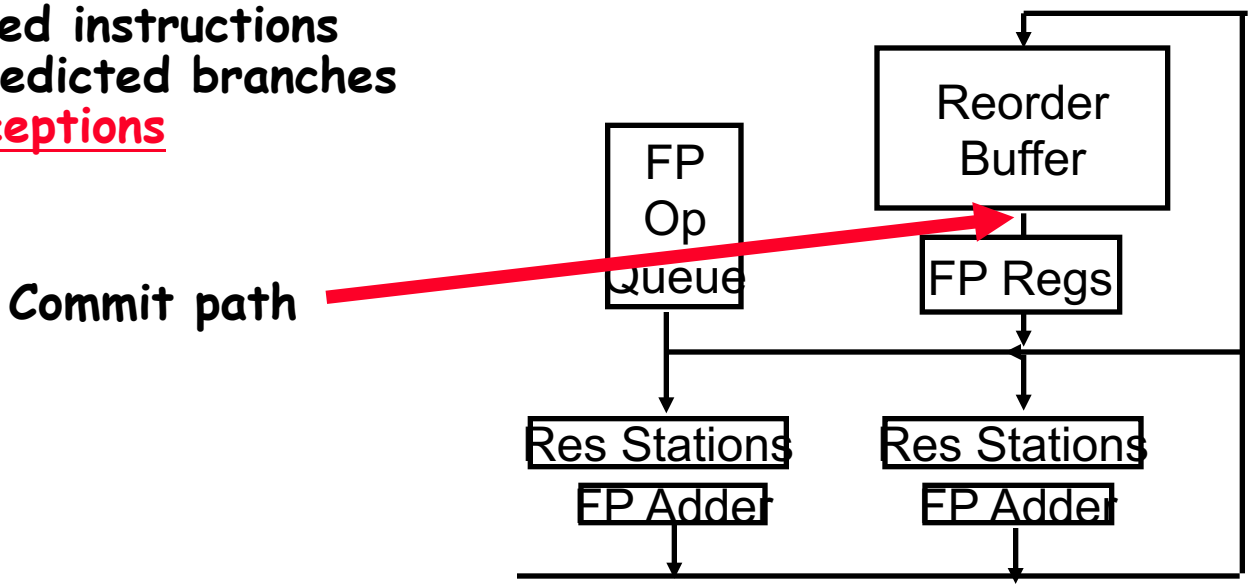
Σχέση μεταξύ Precise Interrupts και Speculation:

- 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!

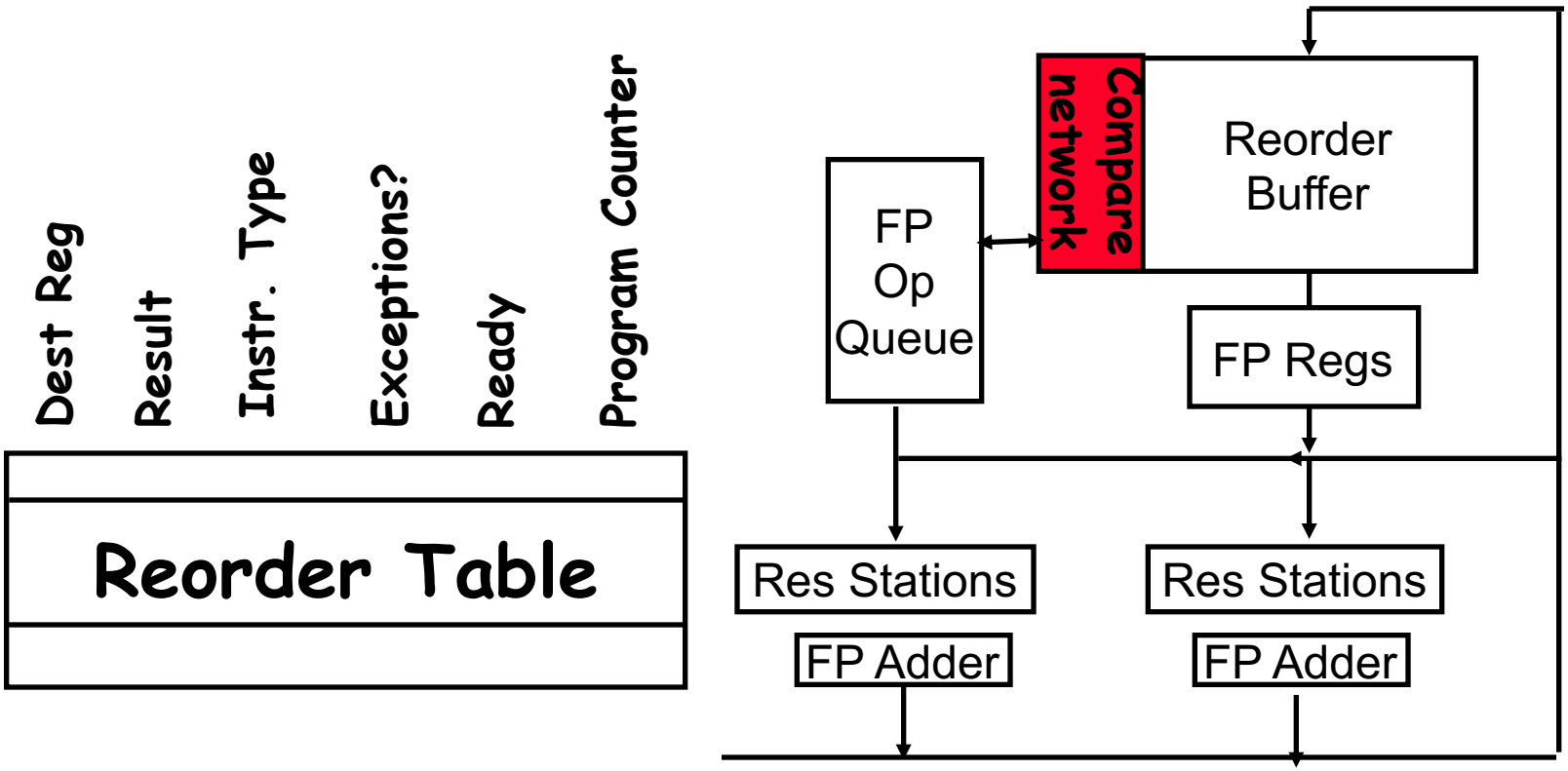
- Τεχνική για precise interrupts/exceptions και speculation: *in-order completion or commit*
  - Γι'αυτό συναντάμε reorder buffers σε όλους τους καινούριους επεξεργαστές

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/mem, result, exception status
  - Όταν η εντολή τελείωσει την εκτέλεση, τοποθέτησε τα αποτελέσματα στον ROB.
    - » Supplies operands to other instruction between execution complete & commit ⇒ more registers like RS
    - » Tag results with ROB buffer number instead of reservation station
  - Η εντολή αλλάζει την κατάσταση της μηχανης στο commit στάδιο όχι στο WB ⇒ in order commit ⇒ values at head of ROB placed in registers
  - Σαν αποτέλεσμα είναι έυκολο να αναιρέσεις speculated instructions σε mispredicted branches ή σε exceptions



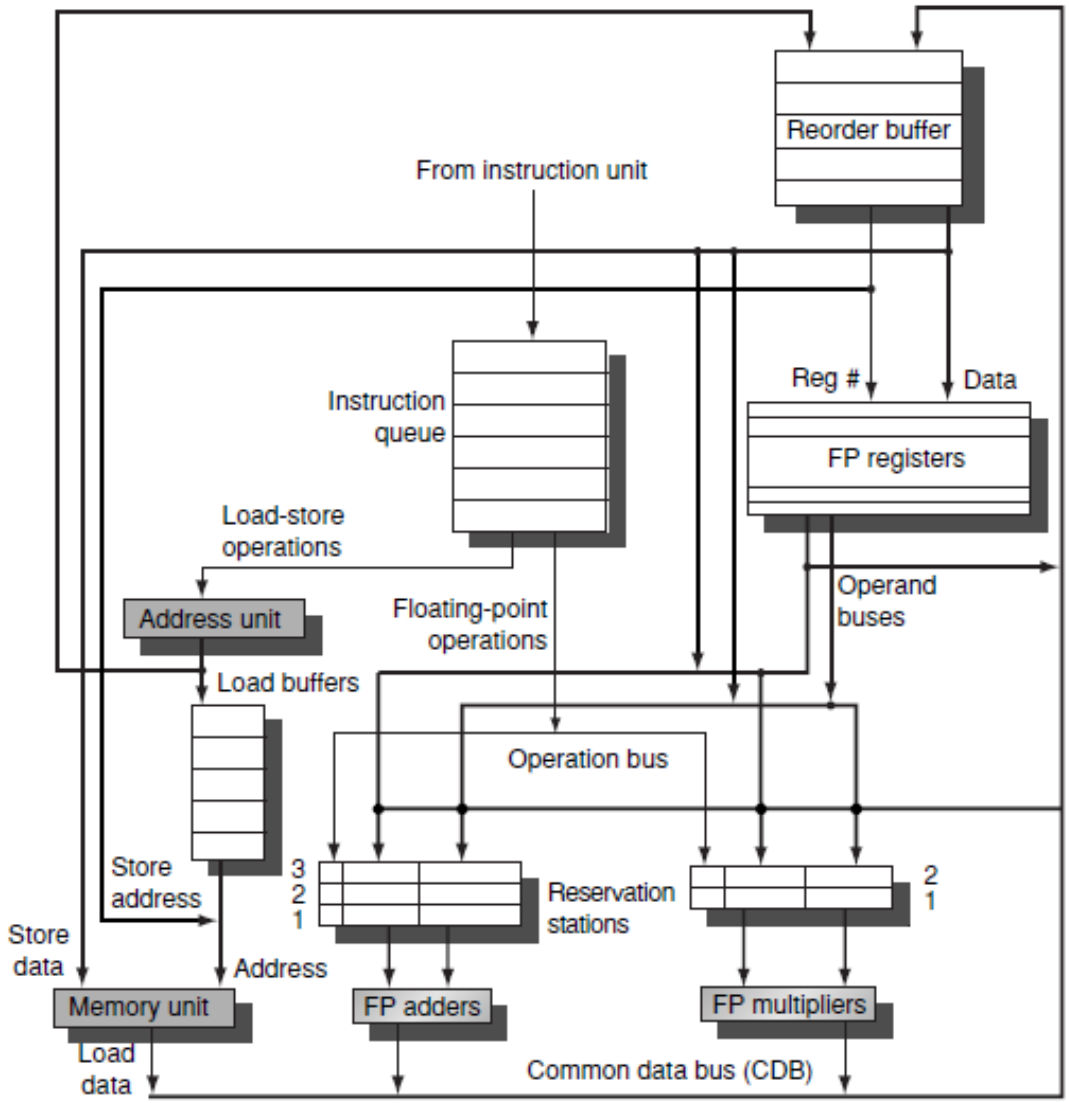
HW με reorder buffer (ROB)?



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

Integrate store buffer into ROB since in order commit. Stores use Result field for ROB tag until data ready on CDB.
Can we also integrate the reservation stations ?

Tomasulo with Reorder Buffer: Basic Block Diagram



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; άν όχι, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called "issue")

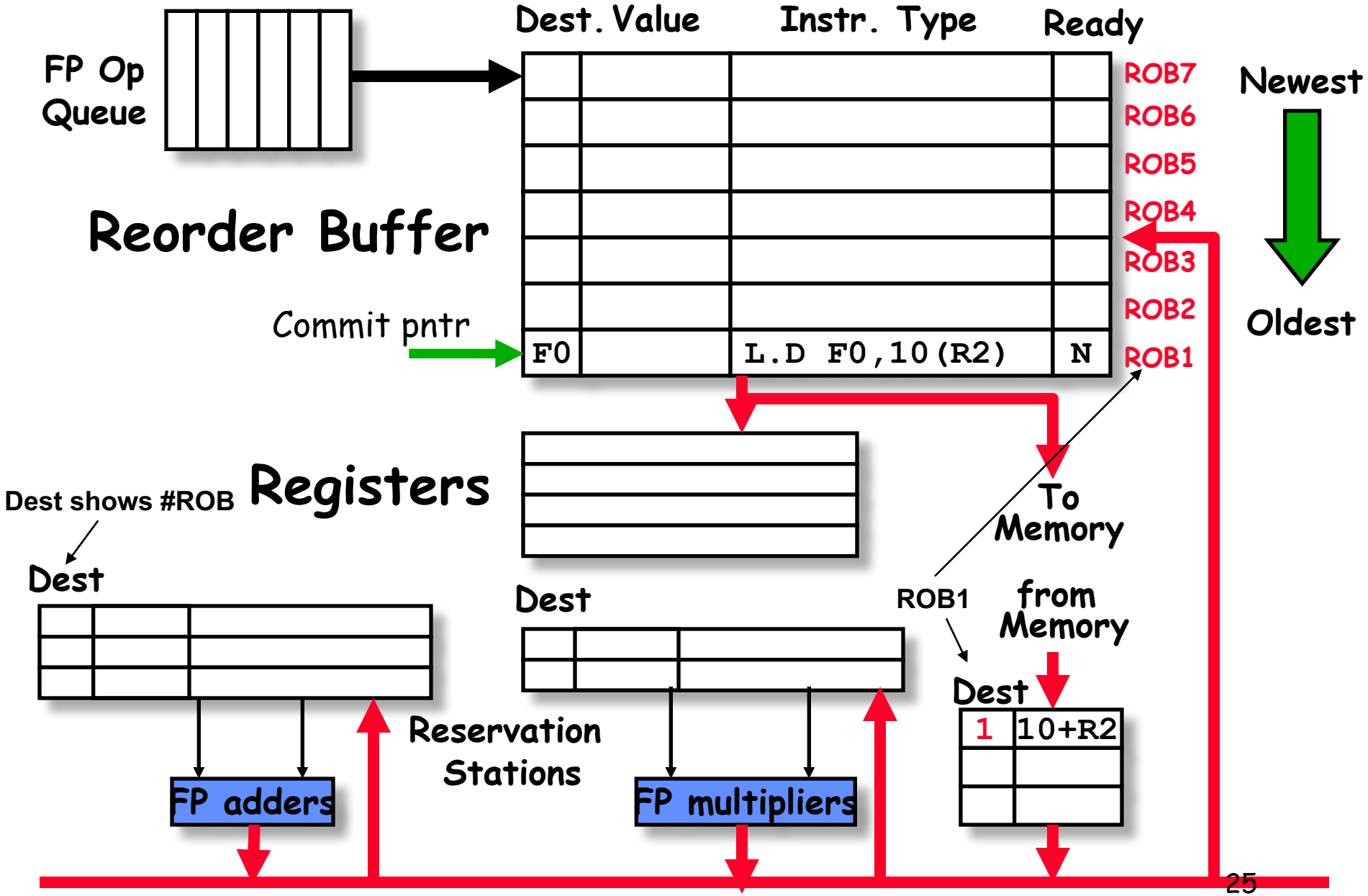
3. **Write result**—Τέλος εκτέλεσης (WB)

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

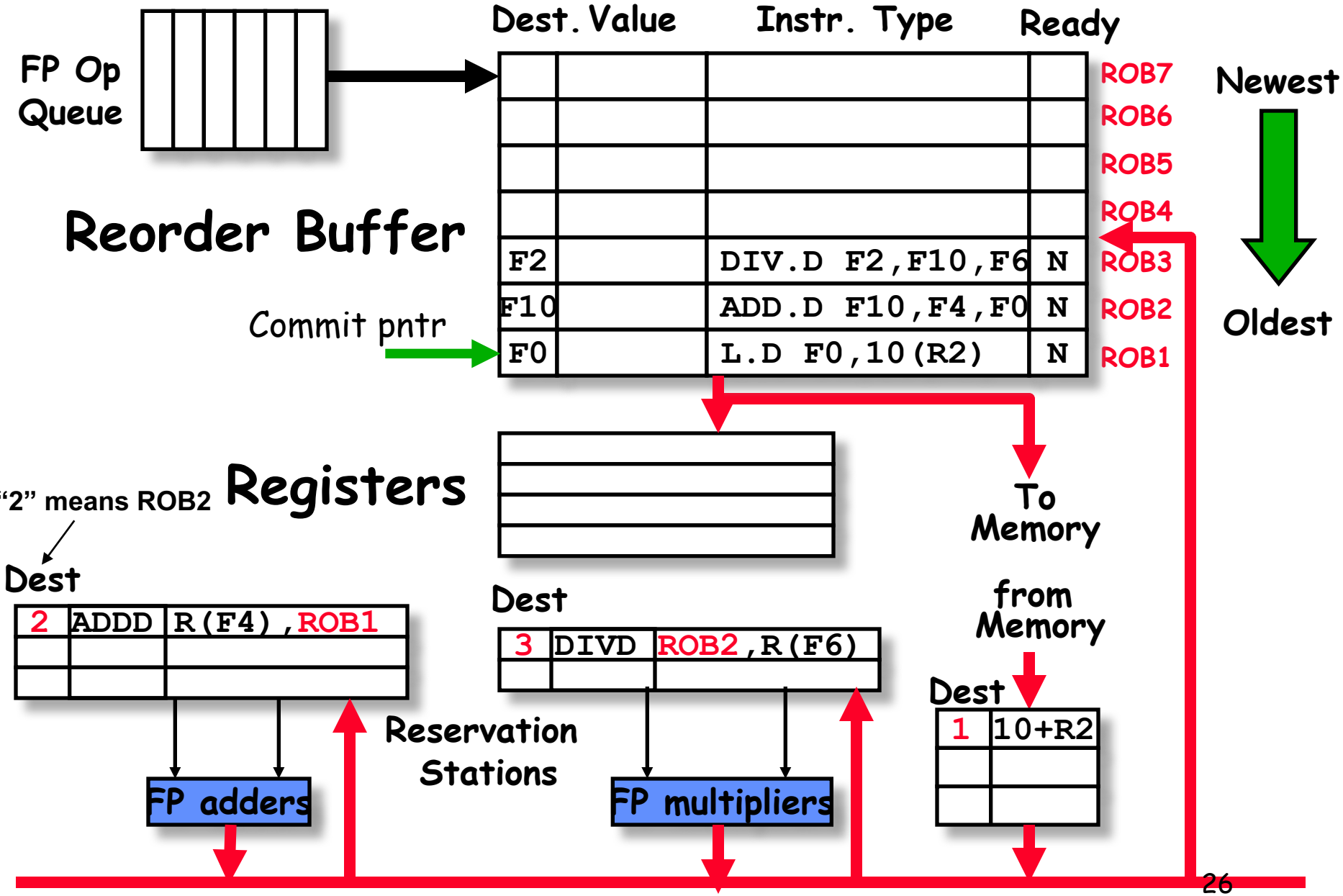
4. **Commit**—Άλλαξε τιμή του dest register με το αποτέλεσμα από το reorder buffer

   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 or exception flushes reorder buffer. (sometimes called "graduation")

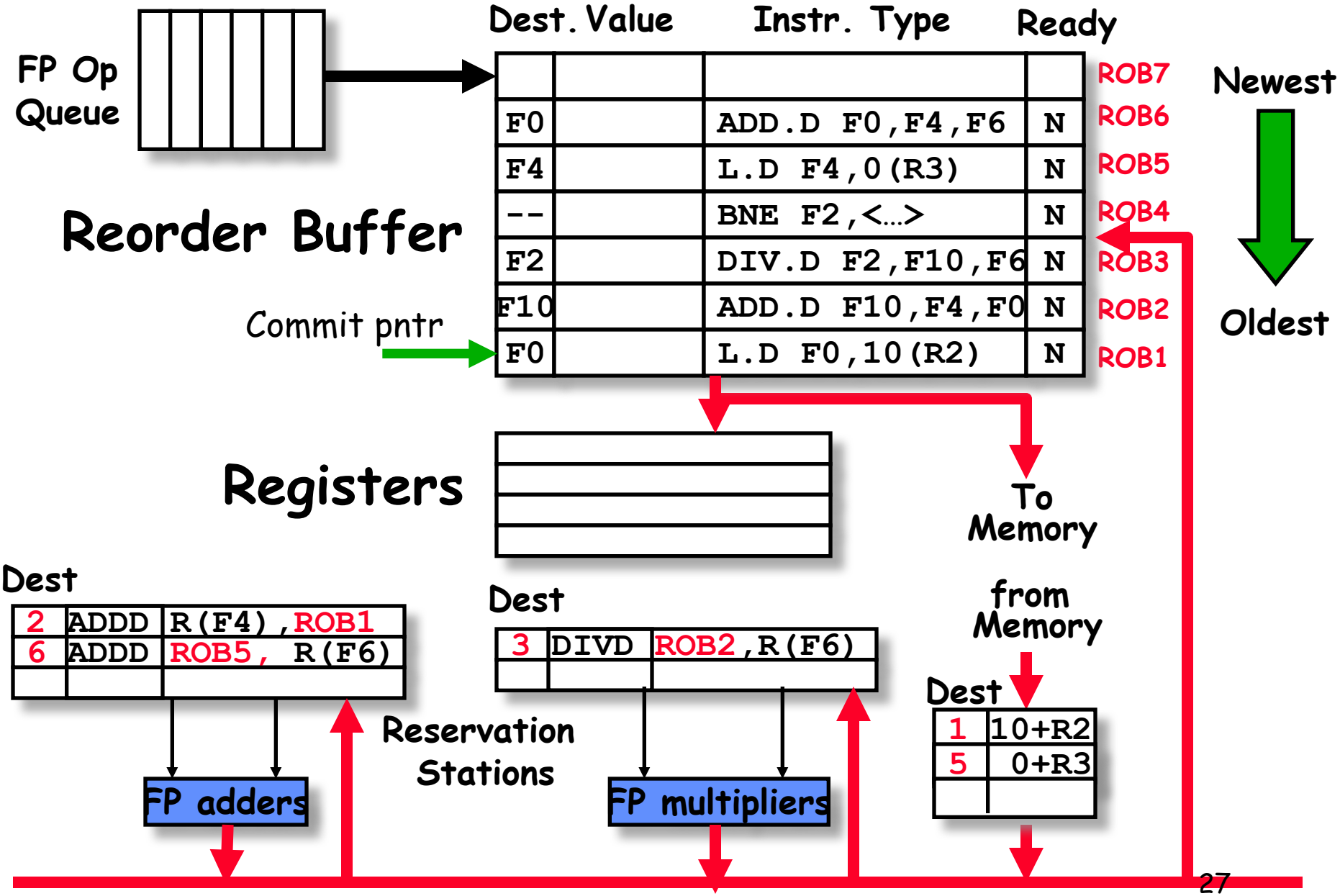
Tomasulo With Reorder buffer



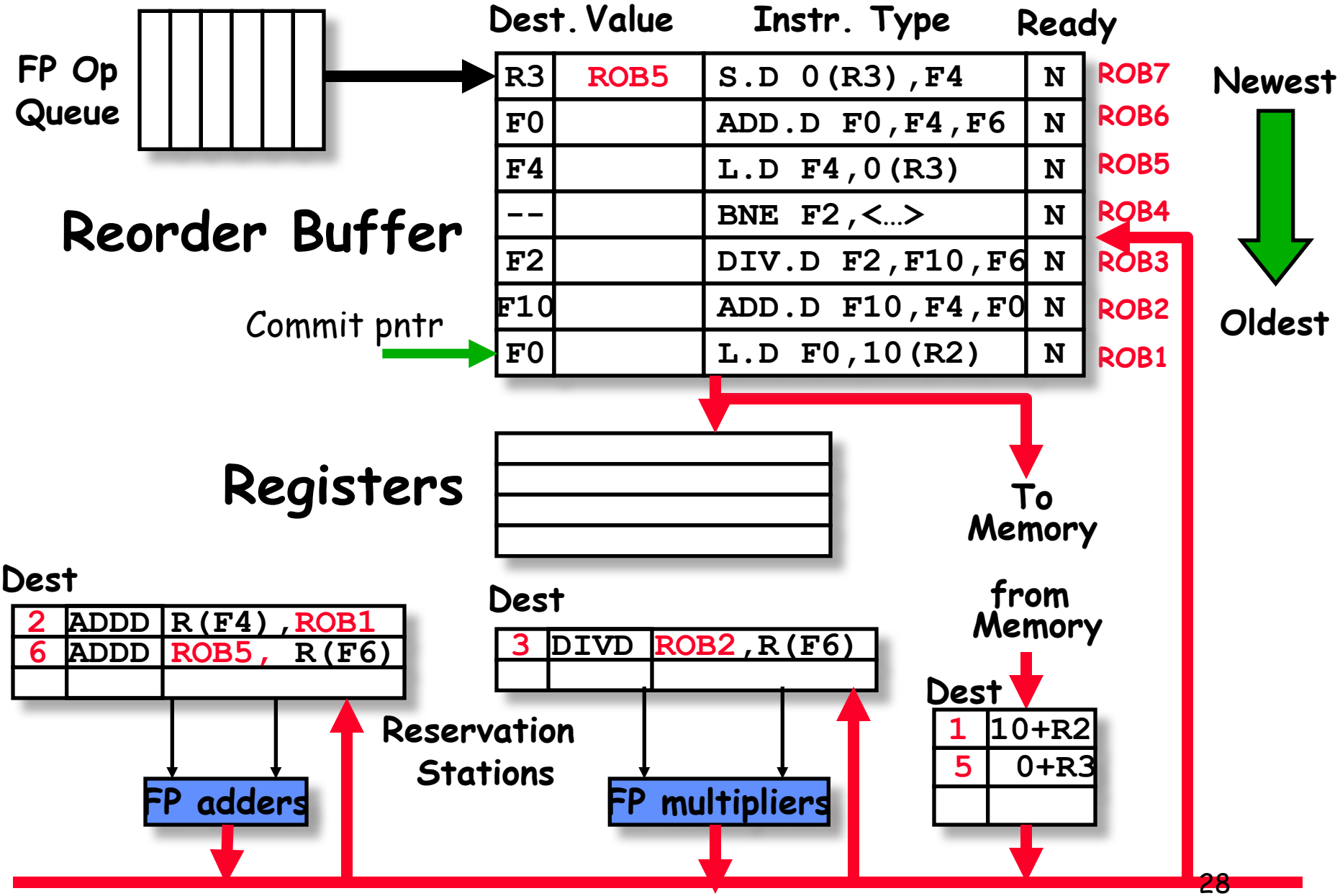
Reorder buffer (after 2 cycles)



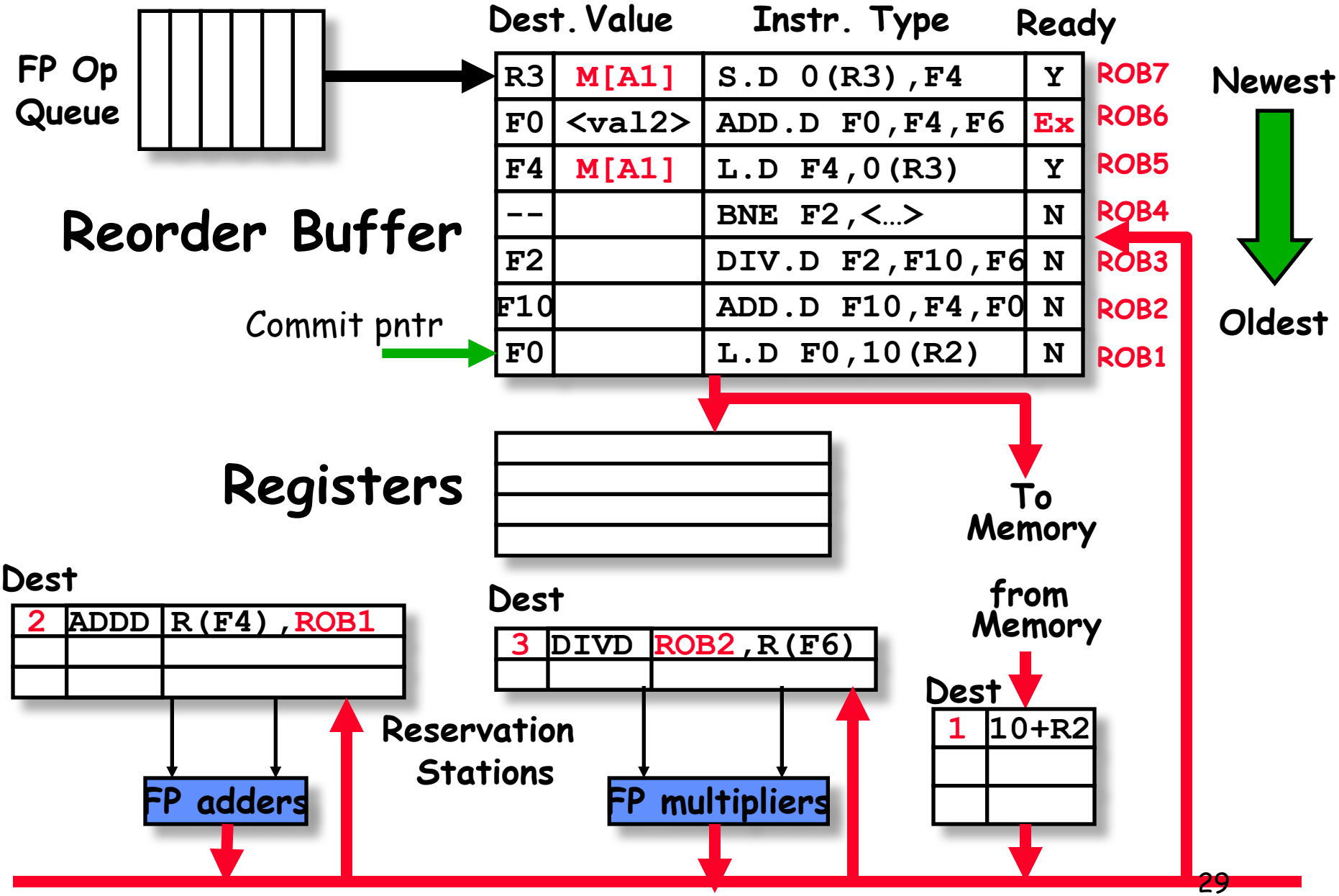
Reorder buffer (after 3 cycles)



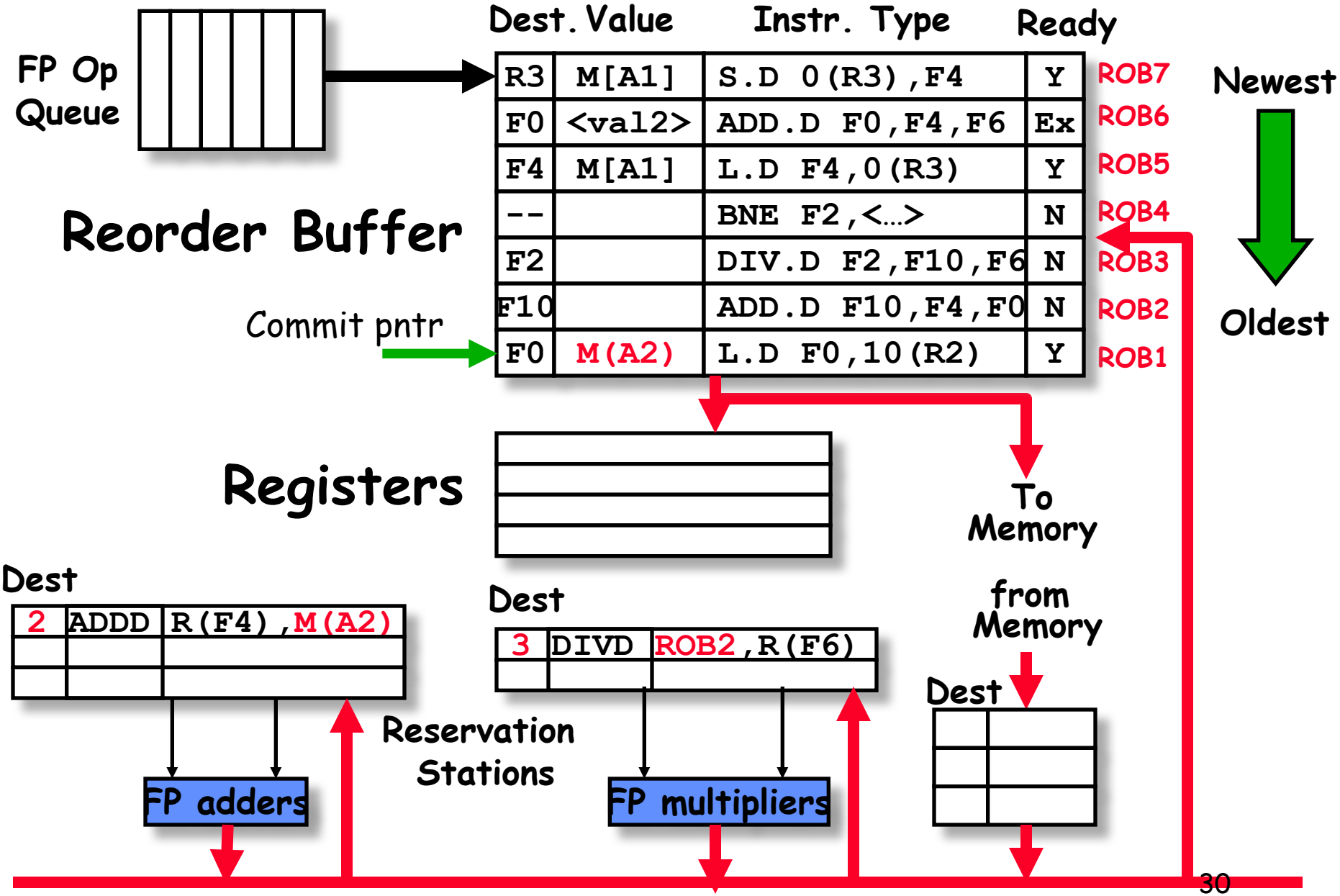
Reorder buffer (after 1 cycle)



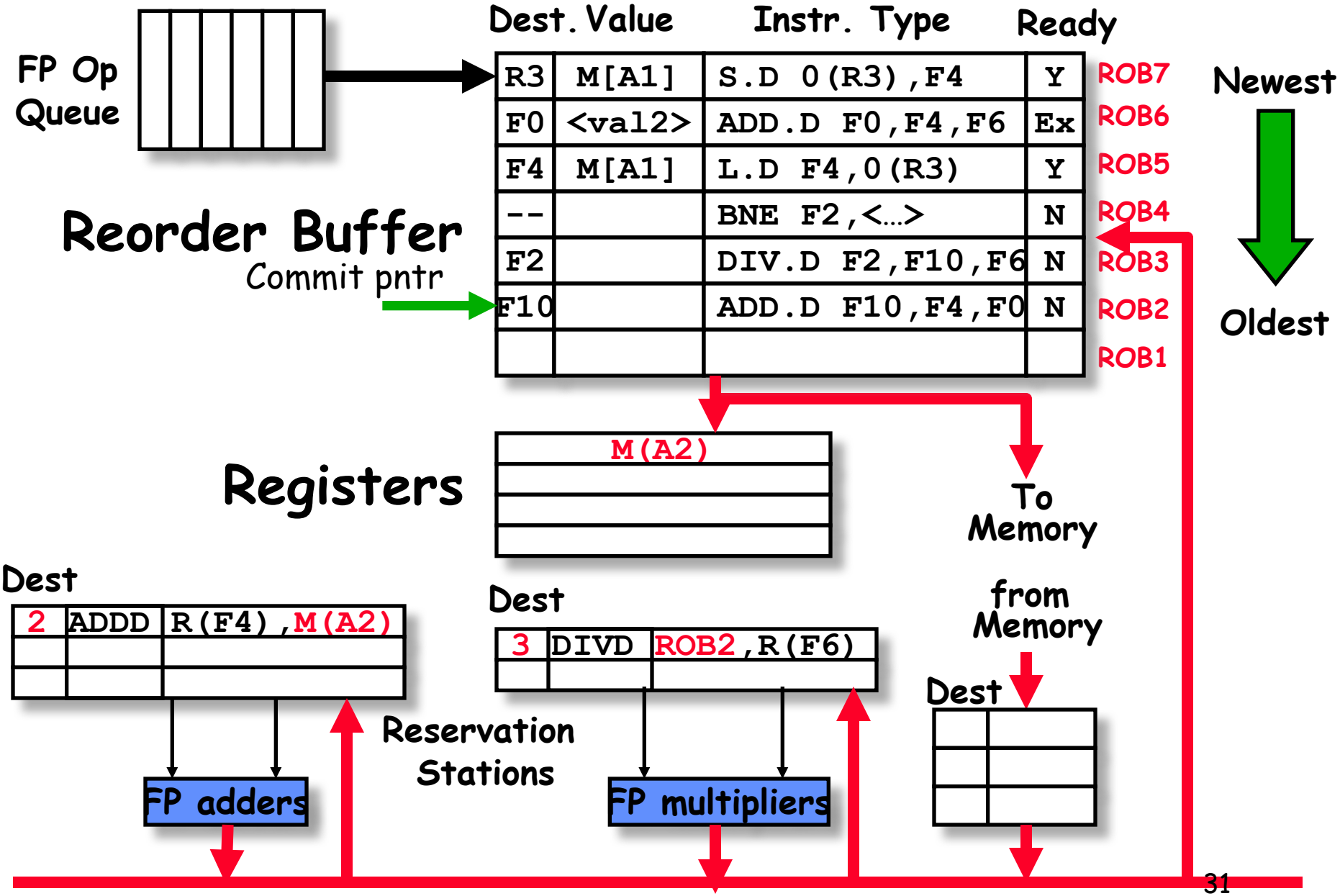
Tomasulo With Reorder buffer



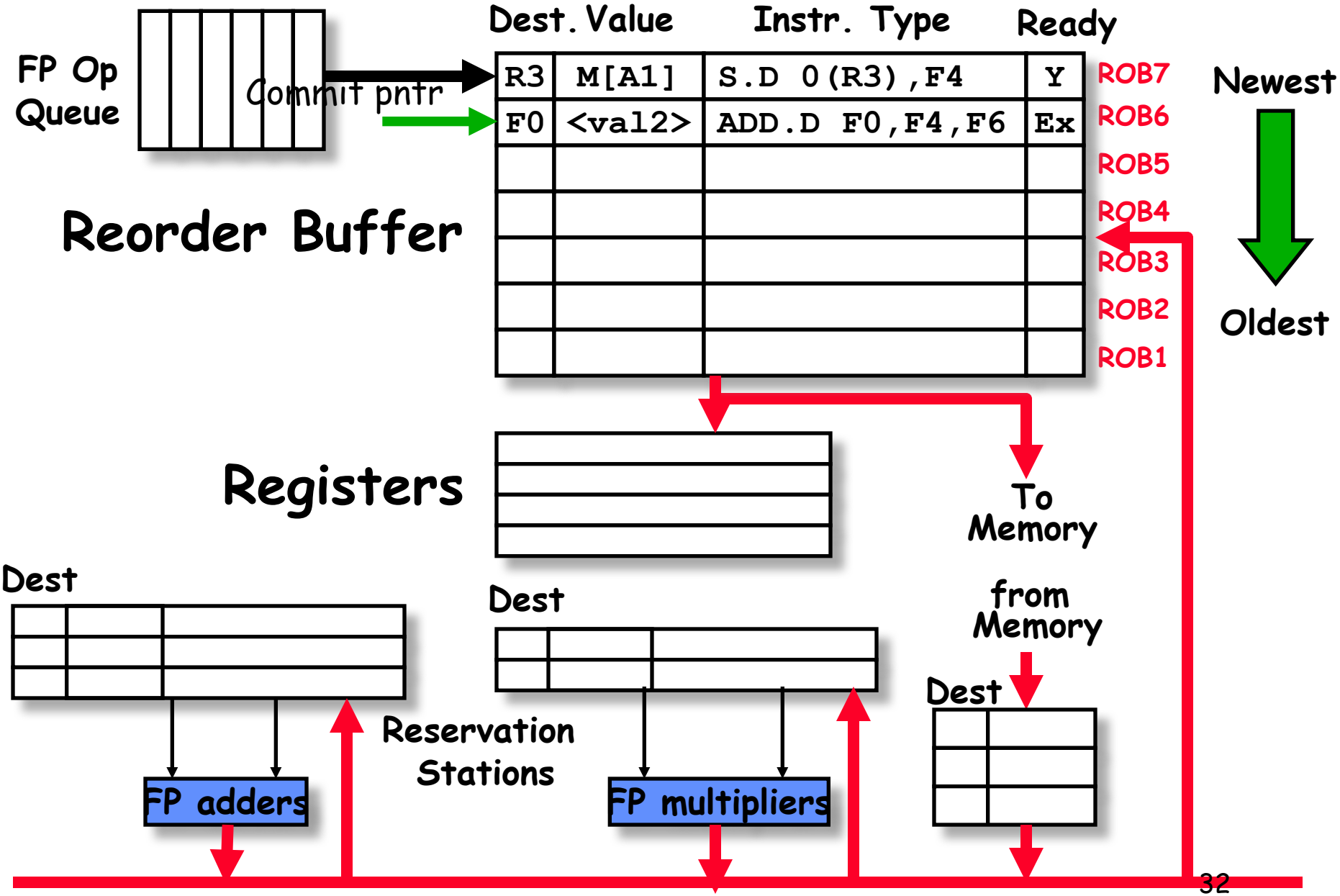
Tomasulo With Reorder buffer



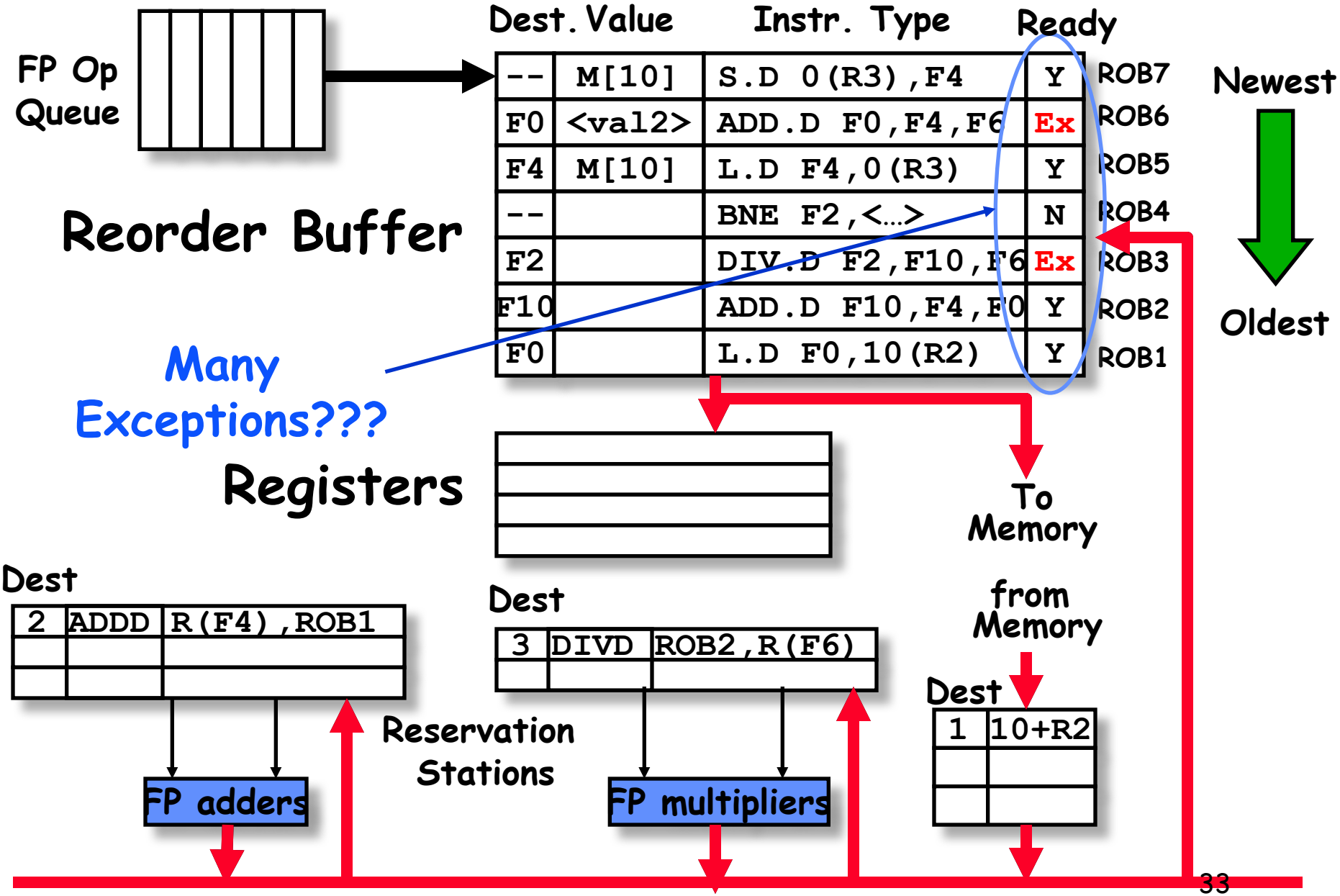
Tomasulo With Reorder buffer



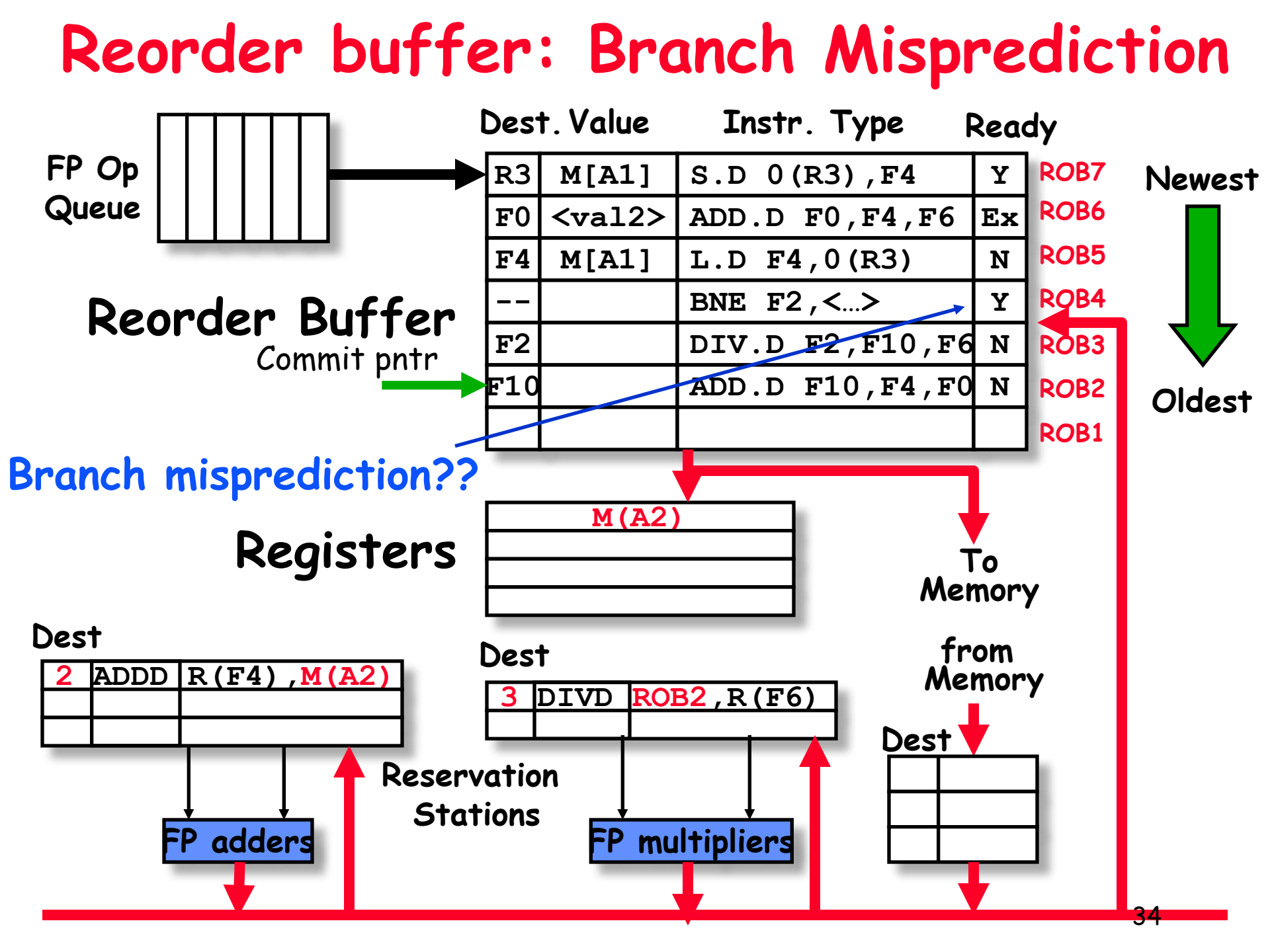
Tomasulo With Reorder buffer



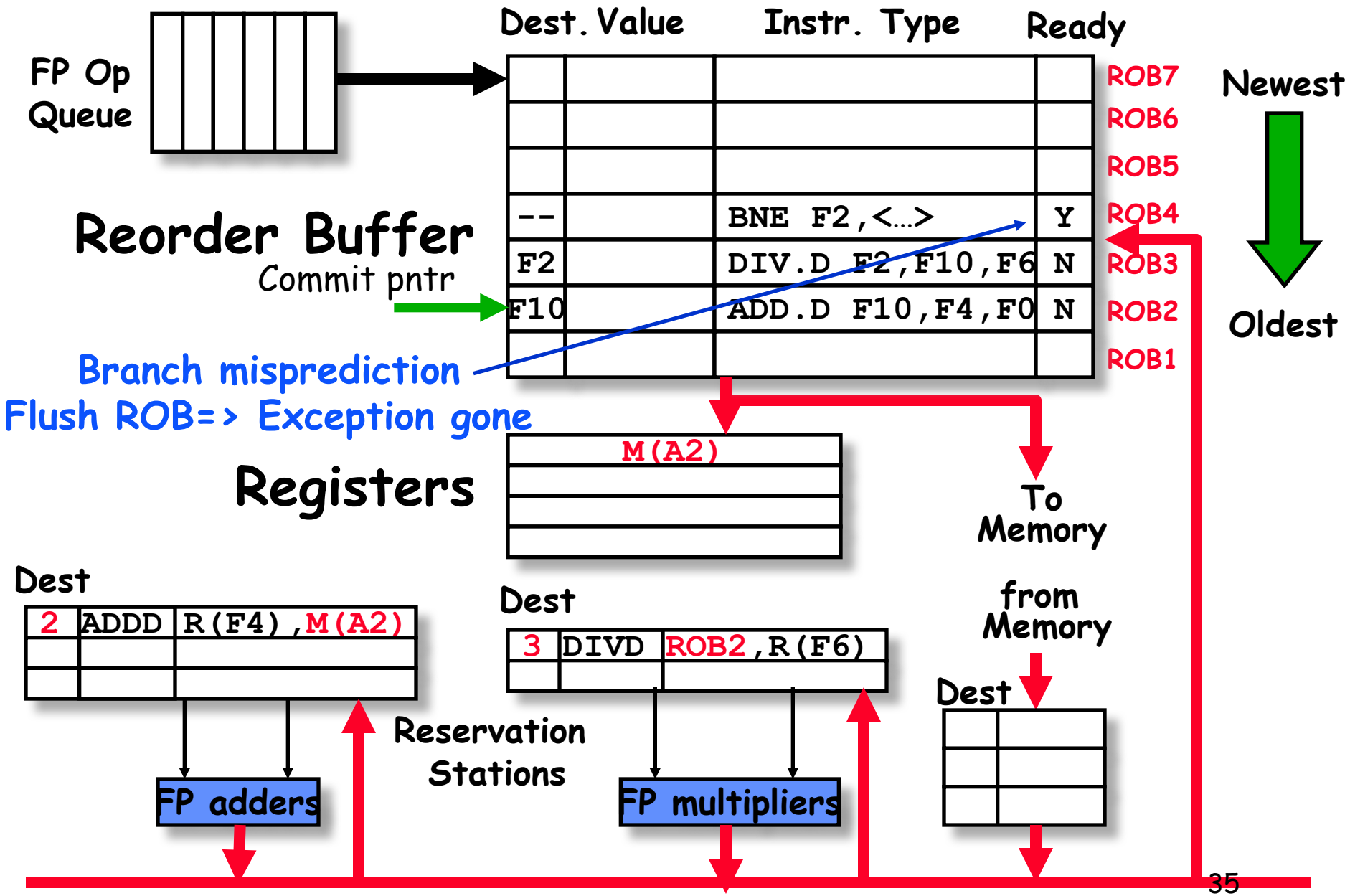
Reorder buffer : Precise Exceptions



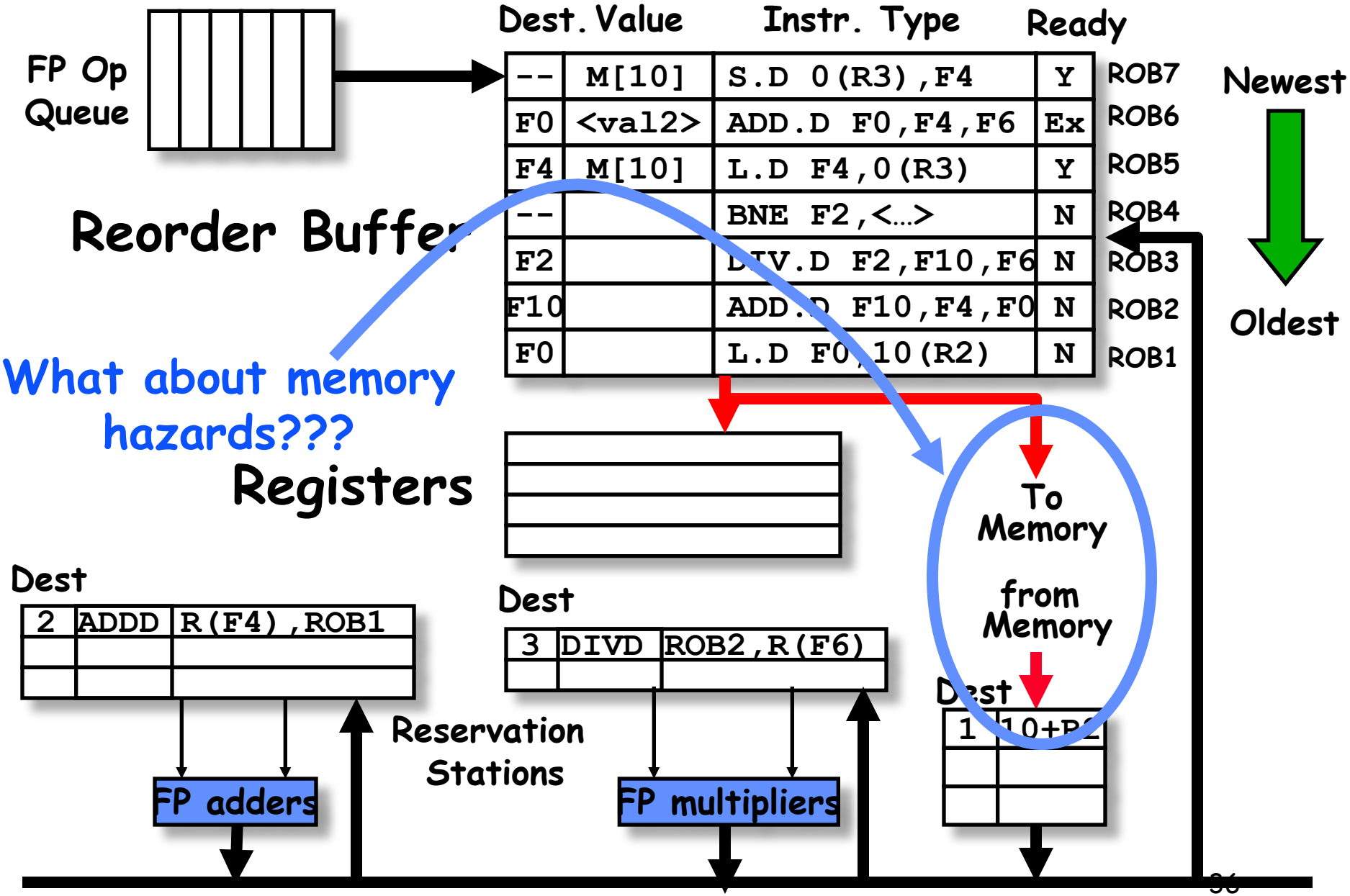
Reorder buffer: Branch Misprediction



Reorder buffer: Branch Misprediction



Tomasulo With Reorder buffer



Memory Disambiguation: WAW/WAR Hazards στη μνήμη

- Like Hazards in Register File, 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)?

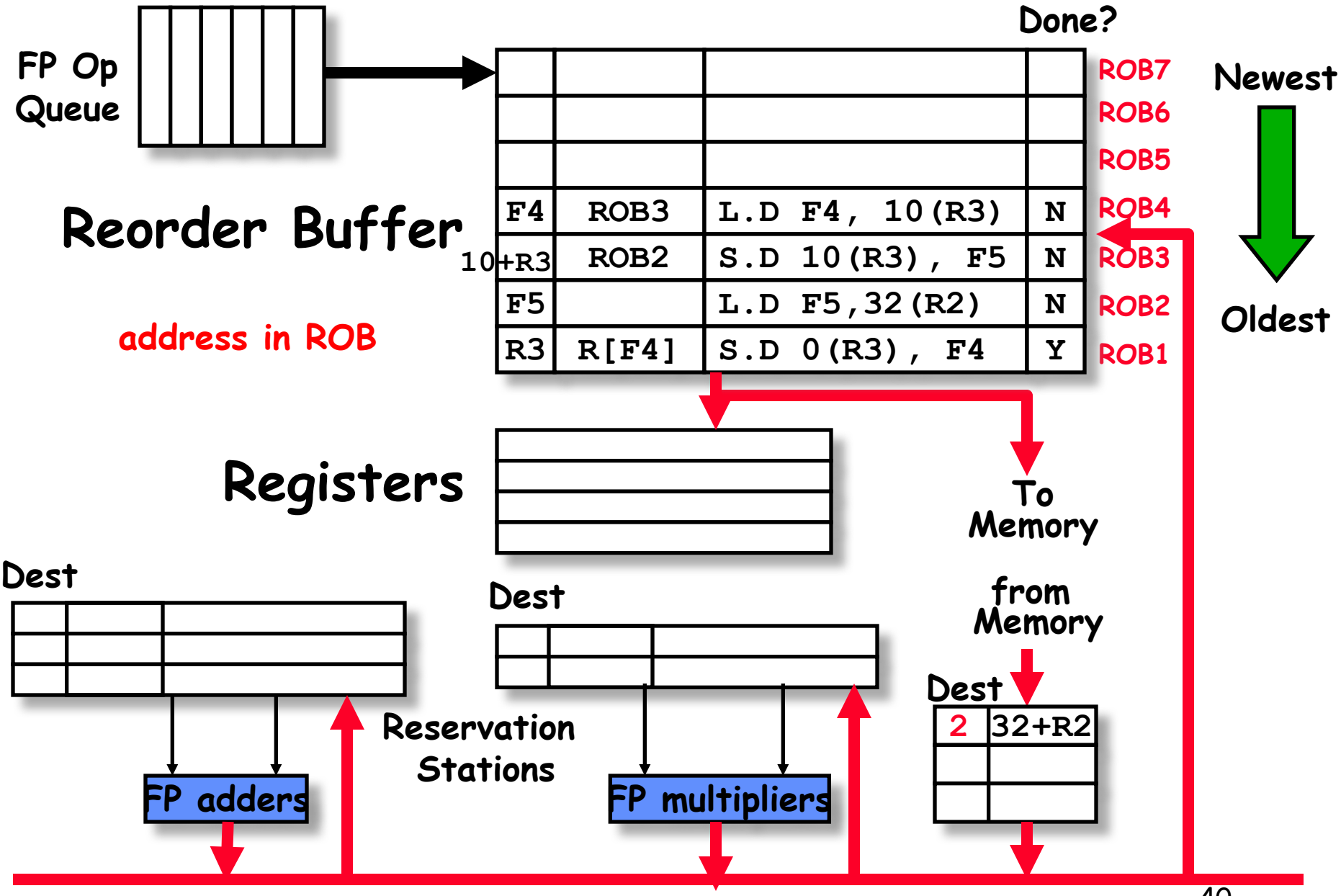
```
Eg:     st      0(R2),R5
        ld      R6,0(R3)
```

- 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.
  - Solution1: Answer is that we are not allowed to start load until we know that address 0(R2) ≠ 0(R3)
  - Solution2: 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 ⇒ return value
    - » store value not available ⇒ 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

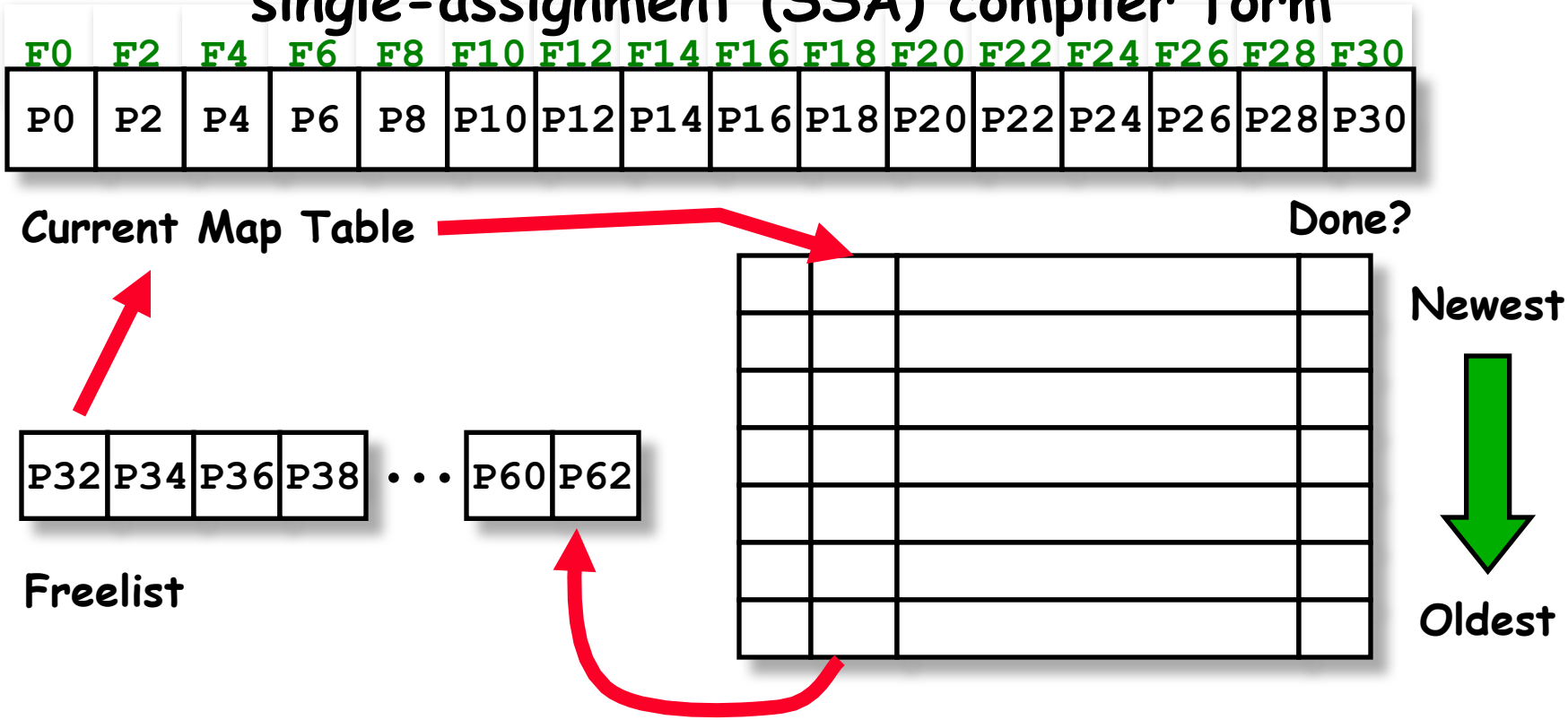


Register Renaming

- What happens with branches?
- Tomasulo can handle renaming across branches

Explicit register renaming

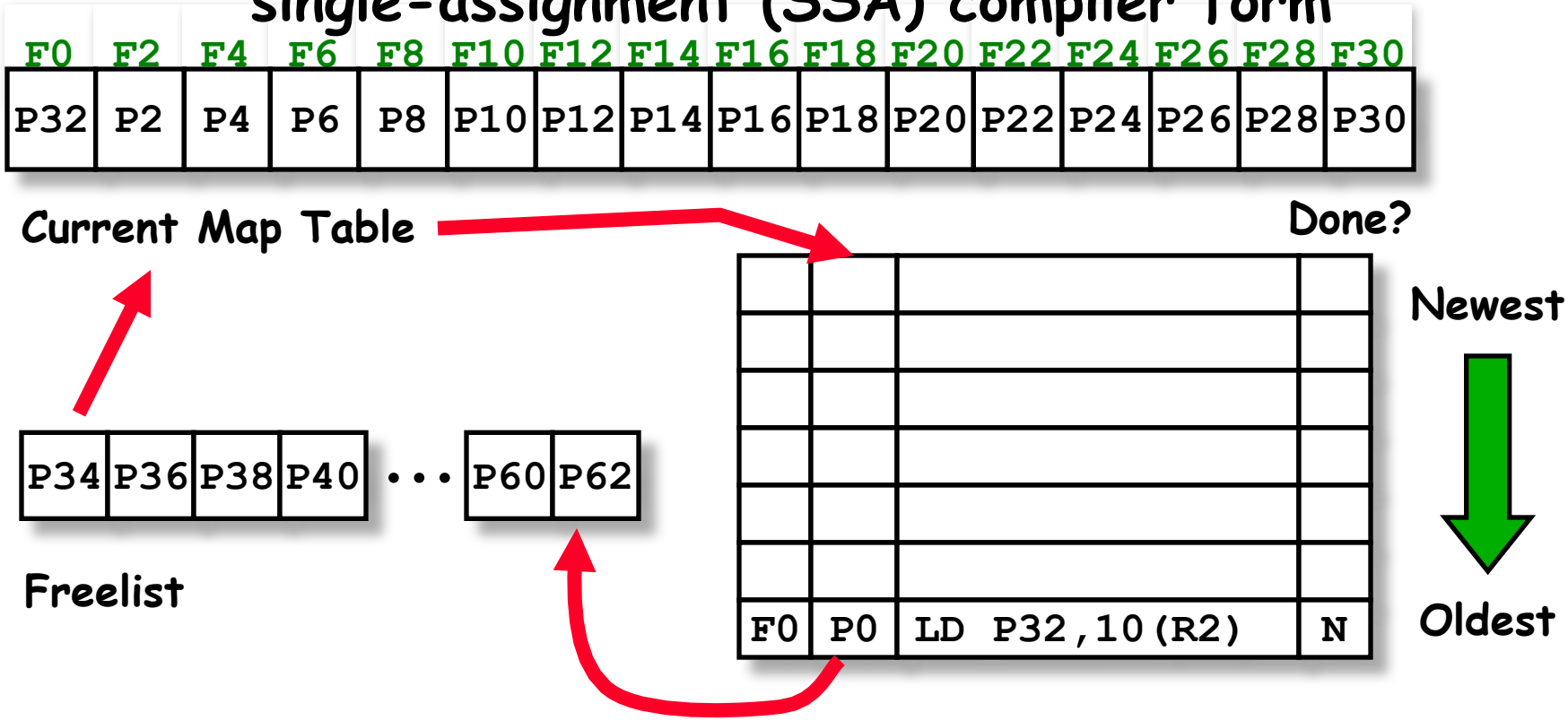
Hardware equivalent of static, single-assignment (SSA) compiler form



- 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

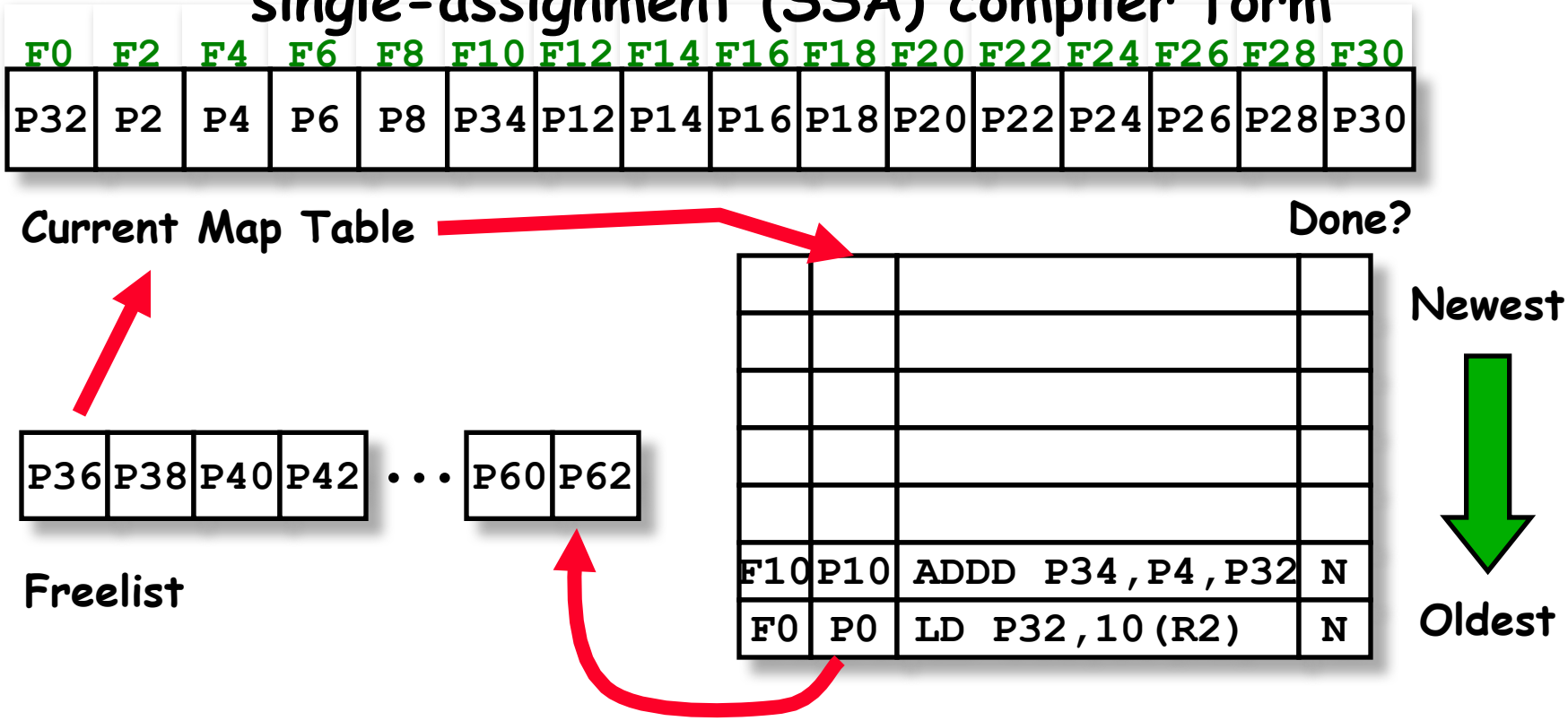


- Note that physical register P0 is "dead" (or not "live") past the point of this load.
  - When we go to commit the load, we free up

```
Issue LD F0,10(R2
```

Explicit register renaming

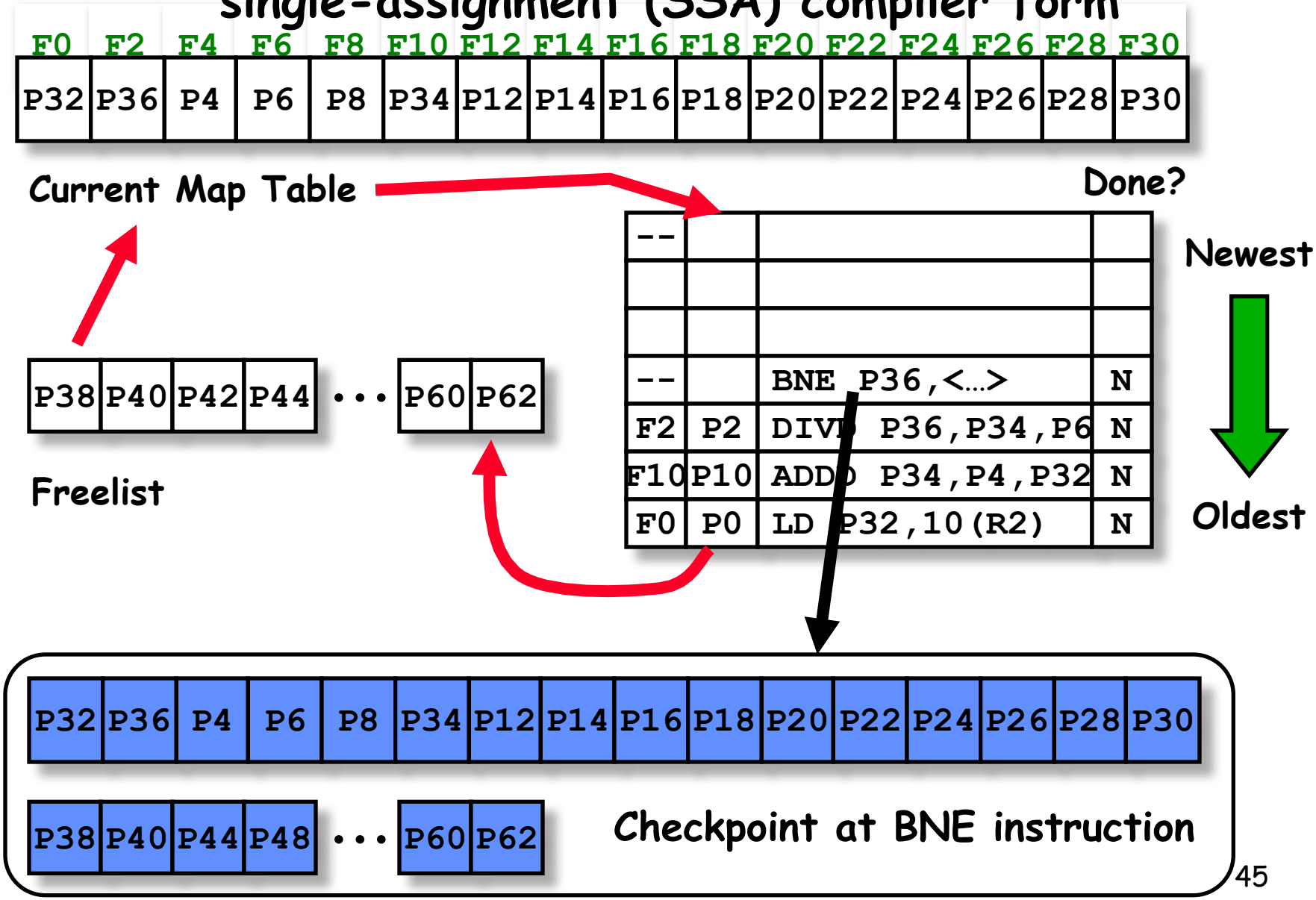
Hardware equivalent of static, single-assignment (SSA) compiler form



Issue ADD F10,F4, F0

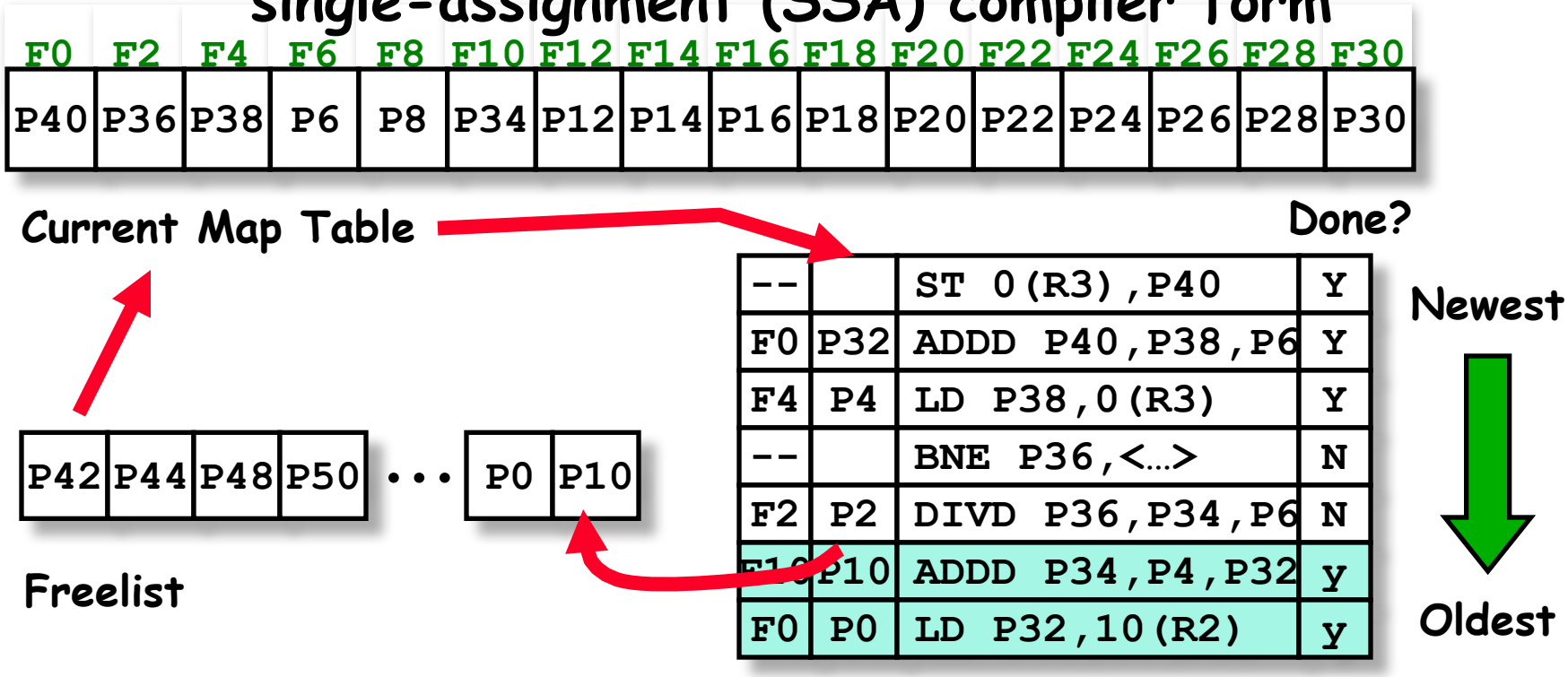
Explicit register renaming

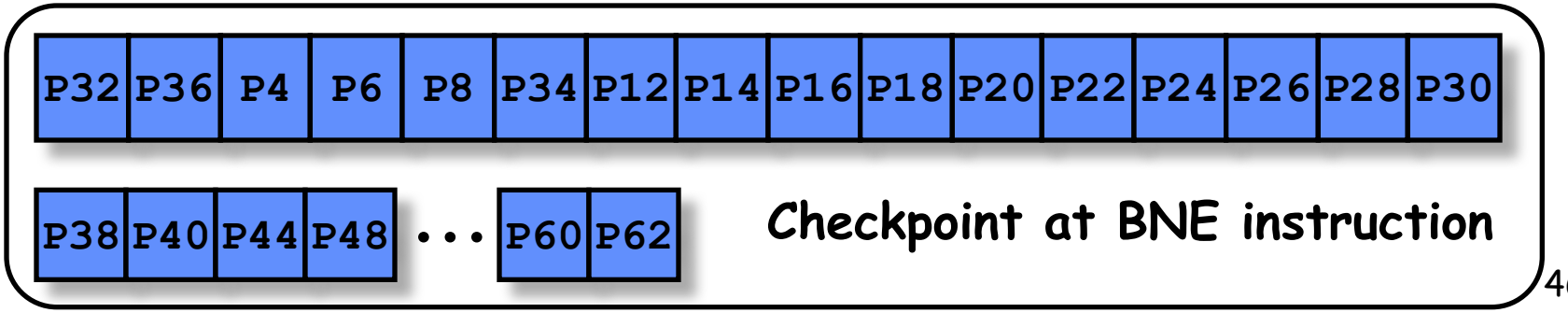
Hardware equivalent of static, single-assignment (SSA) compiler form



Explicit register renaming

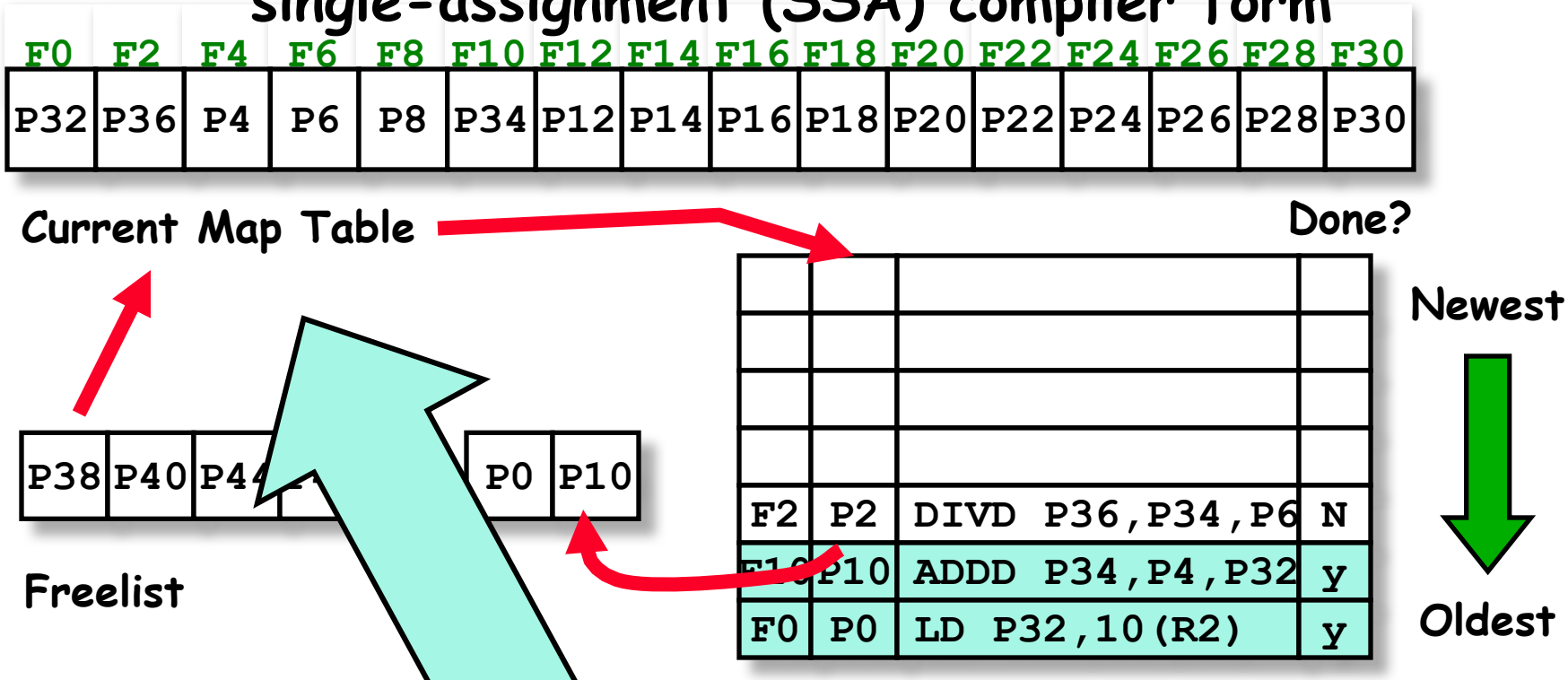
Hardware equivalent of static, single-assignment (SSA) compiler form





Explicit register renaming

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



Speculation error fixed by restoring map table and freelist