

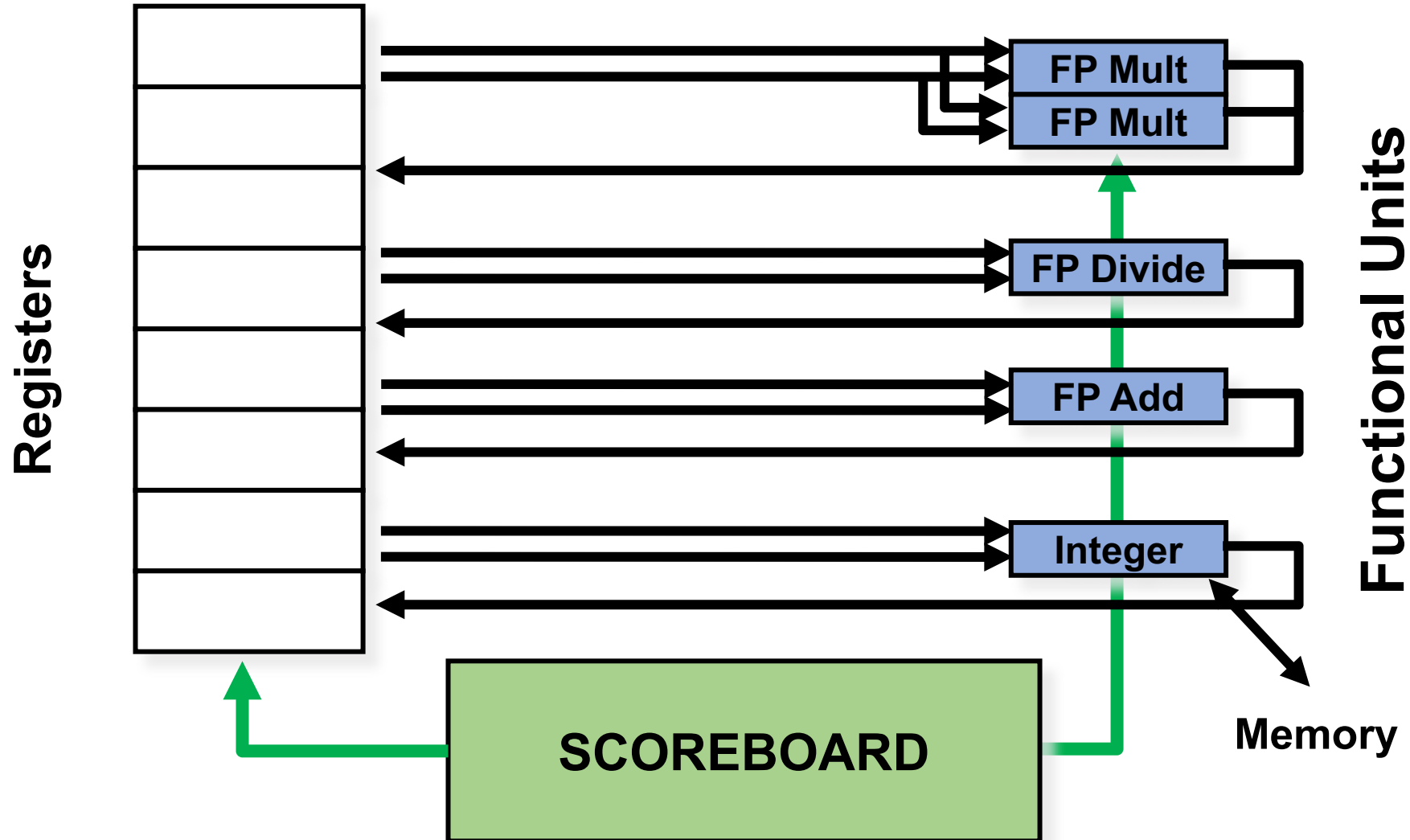
# **CS425**

# **Computer Systems Architecture**

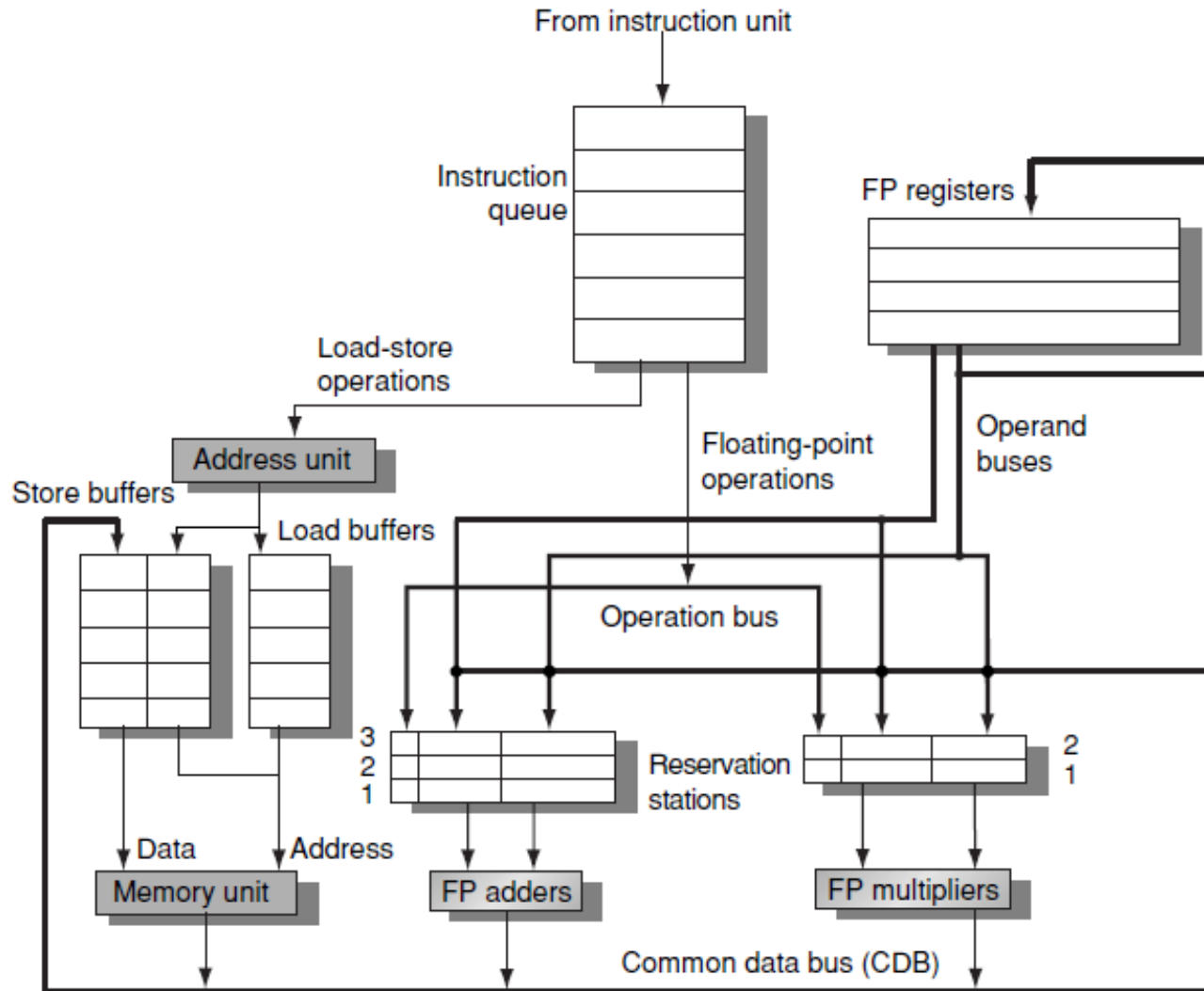
**Fall 2017**

**Re-Order Buffer:  
Precise Exceptions and Speculation**

# Scoreboard Architecture (CDC 6600)



# Tomasulo Organization



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

## Tomasulo

Pipelined Functional Units

(6 load, 3 store, 3 +, 2 x/÷)

window size:  $\leq 14$  instructions

No issue on structural hazard

WAR: renaming avoids them

WAW: renaming avoids them

Broadcast results from FU

Control: reservation stations

## Scoreboard

Multiple Functional Units

(1 load/store, 1 +, 2 x, 1 ÷)

$\leq 5$  instructions

same

stall completion

stall issue

Write/read registers

Central scoreboard



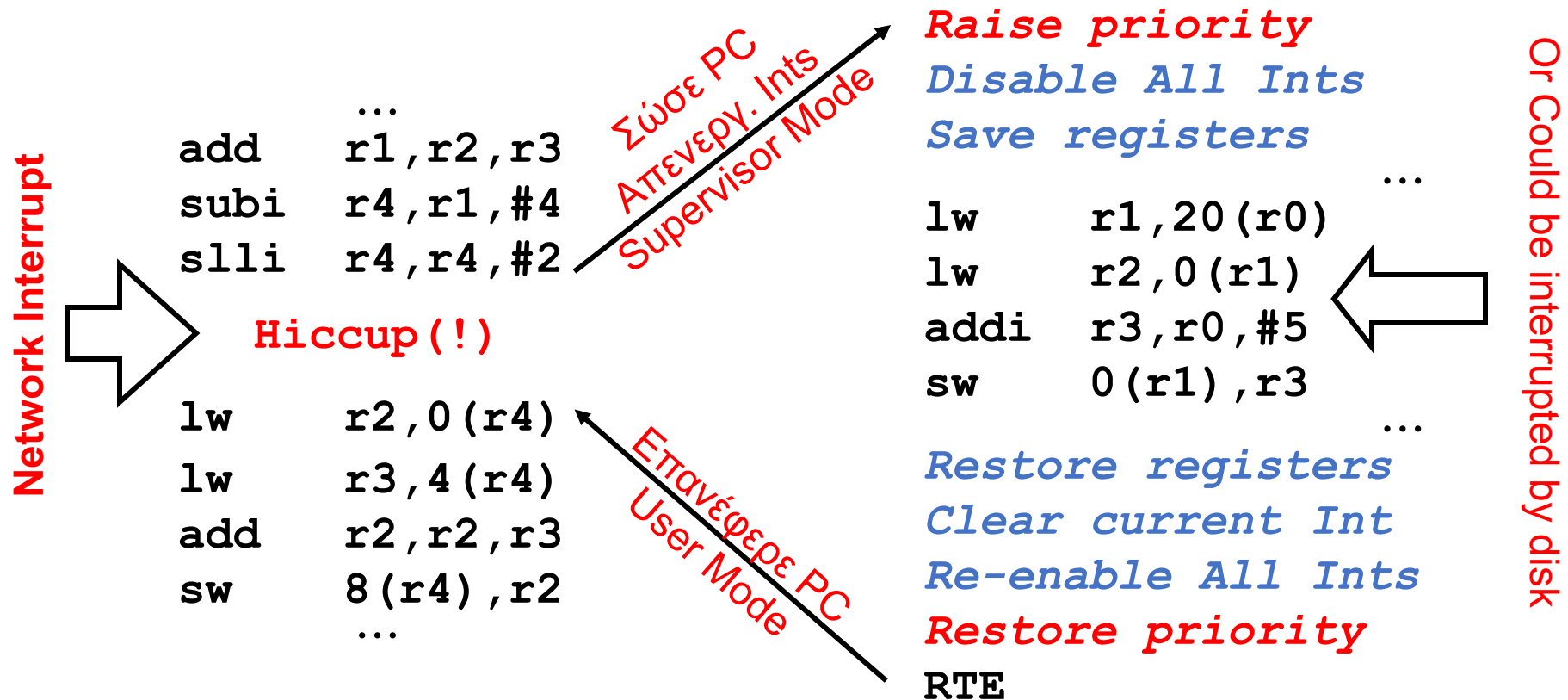
# Exception Behavior with ROB

$$\text{CPI} = \text{CPI}_{\text{IDEAL}} + \text{Stalls}_{\text{STRUC}} + \text{Stalls}_{\text{RAW}} + \text{Stalls}_{\text{WAR}} + \text{Stalls}_{\text{WAW}} + \text{Stalls}_{\text{CONTROL}}$$

- Have to maintain:
  - Data Flow
  - Exception Behavior

Dynamic instruction scheduling (HW)	Static instruction scheduling (SW/compiler)
Scoreboard (reduce RAW stalls)	Loop Unrolling
Register Renaming (reduce WAR & WAW stalls) •Tomasulo • Reorder buffer	SW pipelining
Branch Prediction (reduce control stalls)	Trace Scheduling
Multiple Issue (CPI < 1) Multithreading (CPI < 1)	

# Device Interrupt



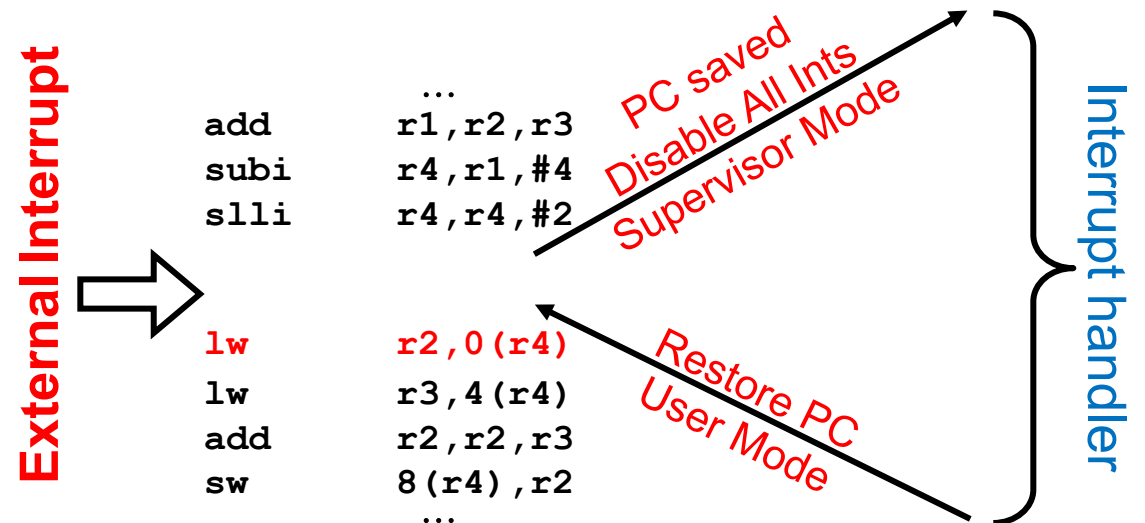
**Note that priority must be raised to avoid recursive interrupts!**

# 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 μπορούμε να επανεκκινήσουμε την εκτέλεση από το interrupt point και «να πάρουμε 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* χρειάζεται να είναι *restartable*
  - i.e. TLB faults. Πρέπει να διορθώσει *translation*, και μετά *restart load/store*
  - IEEE gradual underflow, illegal operation,

e.g: 
$$f(x) = \frac{\sin(x)}{x}$$

$$x \rightarrow 0 \quad f(0) = \frac{0}{0} \Rightarrow NaN + \textit{illegal\_operation}$$

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

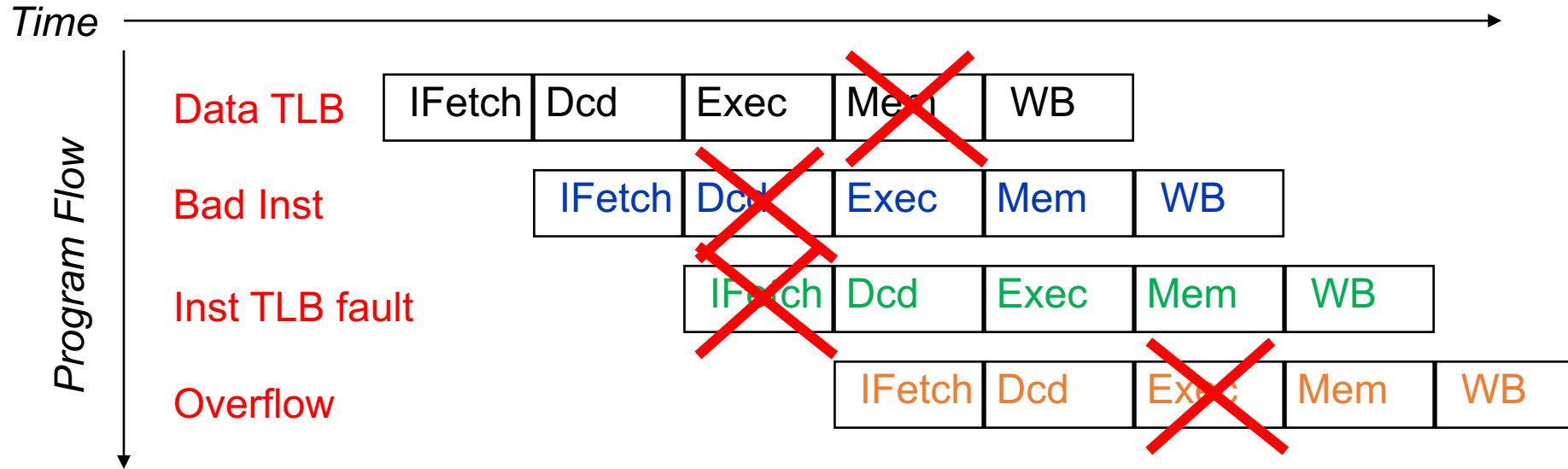
- *Restartability* δεν απαιτεί *preciseness*. Ωστόσο, με *preciseness* είναι *πολύ πιο εύκολη* η επανεκίνηση.
- Απλοποιεί το λειτουργικό σύστημα
  - *Less state* needs to be saved away if unloading process.
  - Quick to restart (for fast interrupts)

# Precise Exceptions in 5-stage DLX

- 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 do we guarantee precise exceptions? Η λύση είναι να μαρκάρουμε την εντολή ως “δημιουργεί exception ή όχι” και να περιμένουμε μέχρι το τέλος του MEM stage για το exception
  - Interrupts are marked as 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 and needs to switch to supervisor mode, 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.
  - Έλεγχε 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 fetched σε NOPs!
- Δουλεύει επειδή γίνεται in-order completion/WB

# Scoreboard Example: Cycle 62

## 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

In-order issue

Out-of-order execute

Out-of-order commit!

## Functional unit status:

<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>dest</i> <i>Fi</i>	<i>S1</i> <i>Fj</i>	<i>S2</i> <i>Fk</i>	<i>FU</i> <i>Qj</i>	<i>FU</i> <i>Qk</i>	<i>Fj?</i> <i>Rj</i>	<i>Fk?</i> <i>Rk</i>
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

## Register result status:

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
62	<i>FU</i>								

# Tomasulo Example: Cycle 57

## Instruction status:

Instruction	$j$	$k$	Issue	Exec Comp	Write Result	Busy	Address
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3	15	16	Load3
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	$S1$ $V_j$	$S2$ $V_k$	$RS$ $Q_j$	$RS$ $Q_k$
Add1		No					
Add2		No					
Add3		No					
Mult1		No					
Mult2		Yes	DIVD	M*F4	M(A1)		

In-order issue

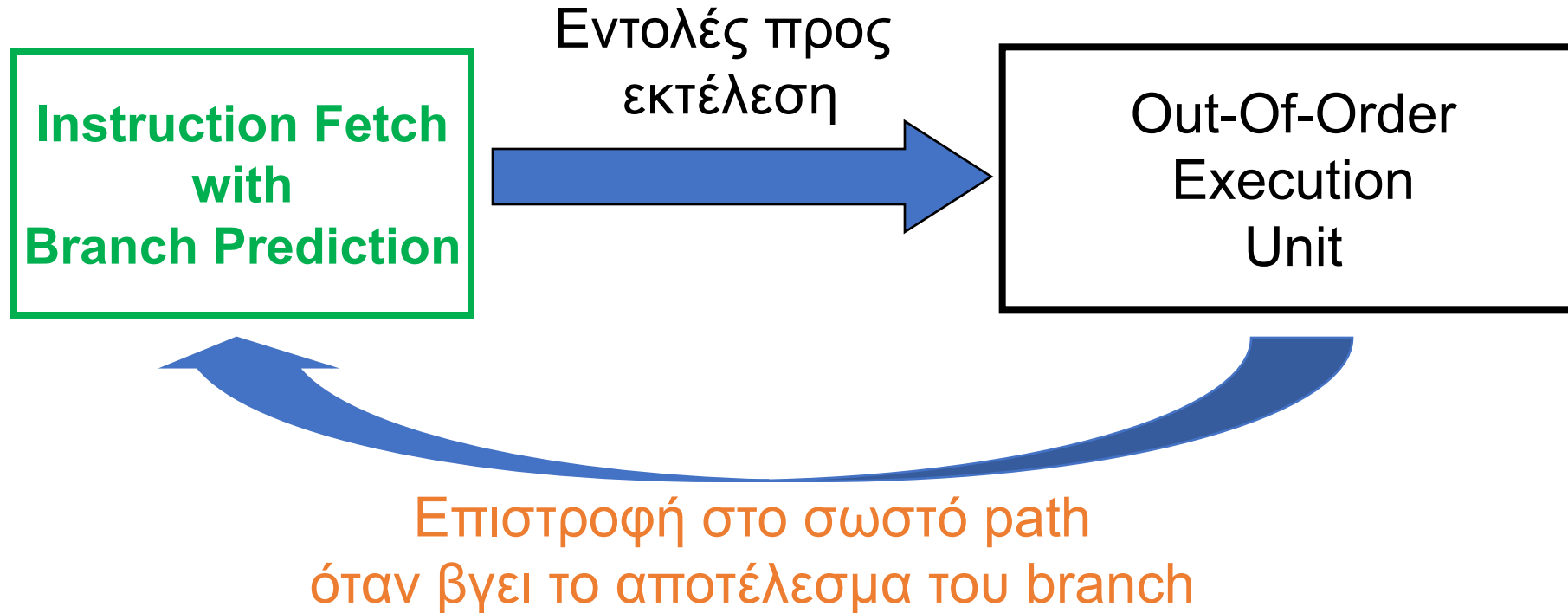
Out-of-order execute

Out-of-order commit!

## Register result status:

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

# Πρόβλημα: “Fetch” unit



- Εντολές σε πιθανώς λανθασμένο predicted path έχουν ήδη εκτελεστεί.
- Instruction fetch decoupled from execution

# Branch must execute fast for 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 μάθημα. Μοντέρνες αρχιτεκτονικές τώρα προβλέπουν πολλά πράγματα: **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**
- Χρειαζόμαστε ένα τρόπο να συγχρονίσουμε το completion στάδιο των εντολών με την σειρά στο πρόγραμμα (i.e. with issue-order)
  - Easiest way is with **in-order completion (i.e. re-order buffer)**
  - Other Techniques (Smith paper): Future File, History Buffer

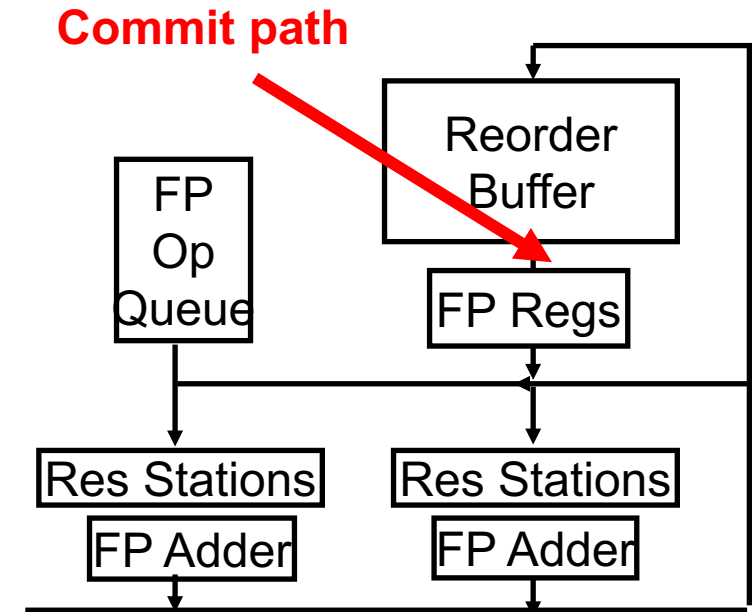
# Precise Interrupts and Speculation:

- Στο Issue στάδιο εντολών είναι σαν να προβλέπουμε 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 the 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 re-order buffer (ROB) σε όλους τους μοντέρνους out-of-order επεξεργαστές

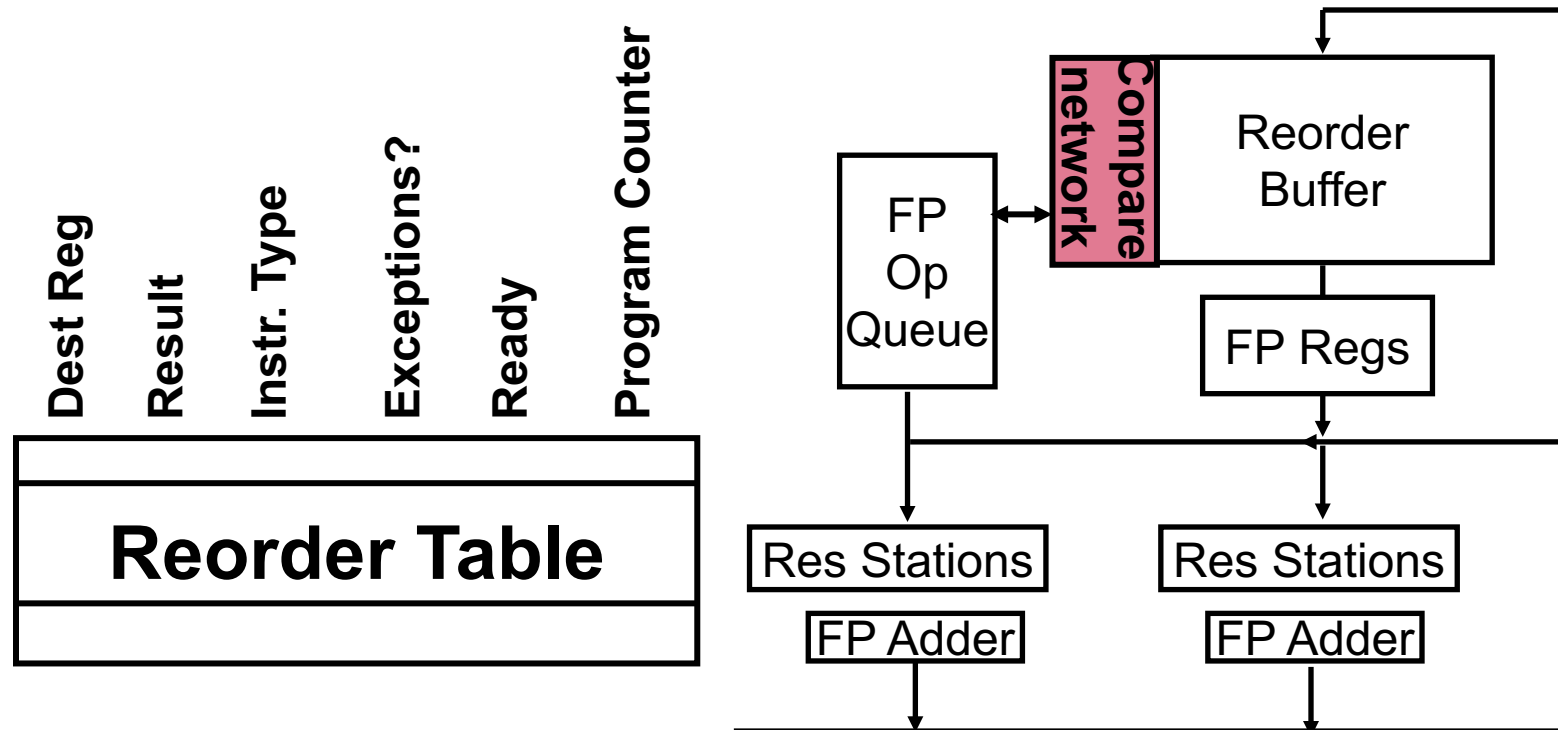


# Υποστήριξη precise interrupts από HW

- **Ιδέα του Reorder Buffer (ROB):** κρατάμε τις 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 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 are placed in registers**
- Σαν αποτέλεσμα είναι εύκολο να αναιρέσουμε speculated instructions σε **mispredicted branches** ή σε **exceptions**

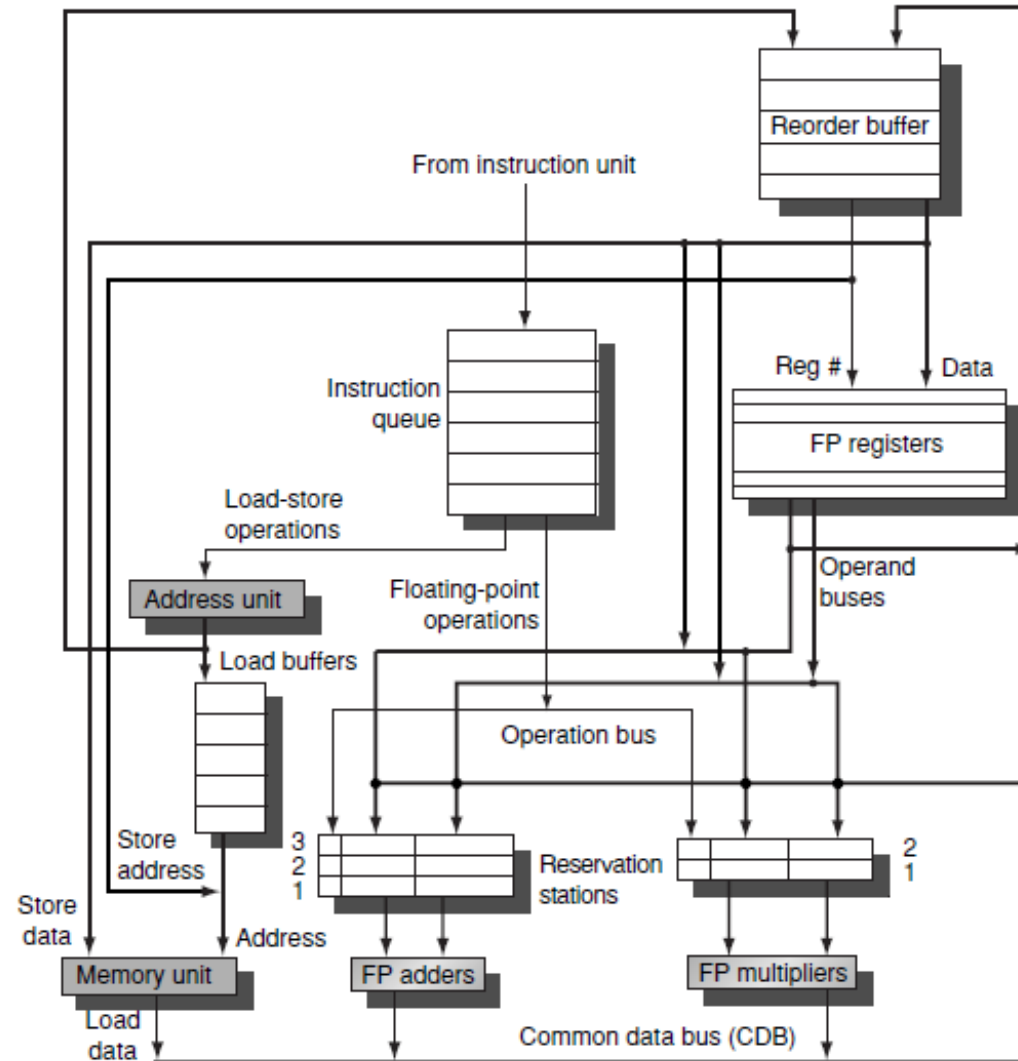


# HW με reorder buffer (ROB)?



- Πώς βρίσκουμε την τελευταία έκδοση του register?
- Πολύπορτο ROB σαν να είναι register file
- Integrate store buffer into ROB since we have in order commit. Stores use Result field for ROB tag until data ready on CDB.
- Can we also integrate the reservation stations ?

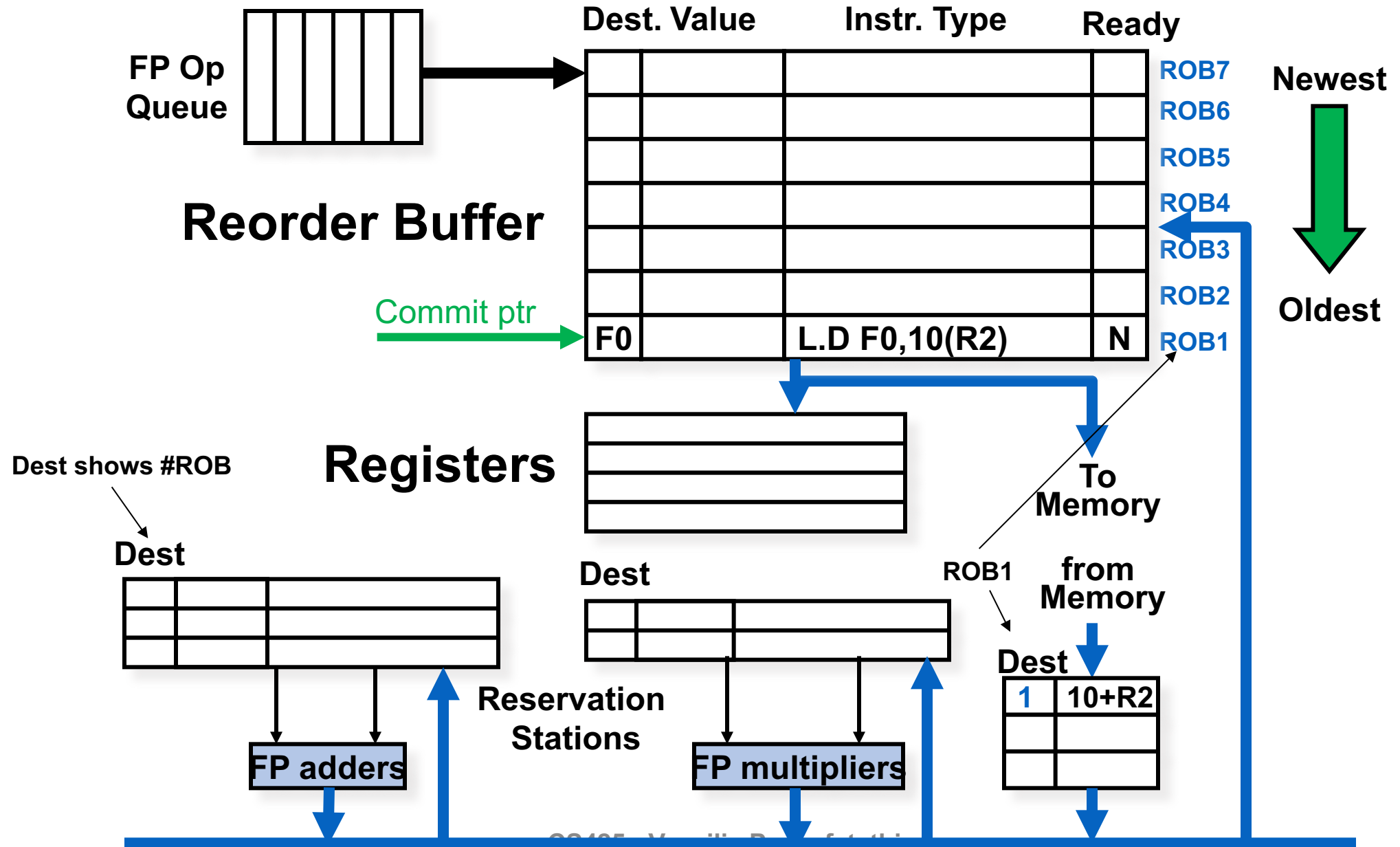
# Tomasulo with ROB: Basic Block Diagram



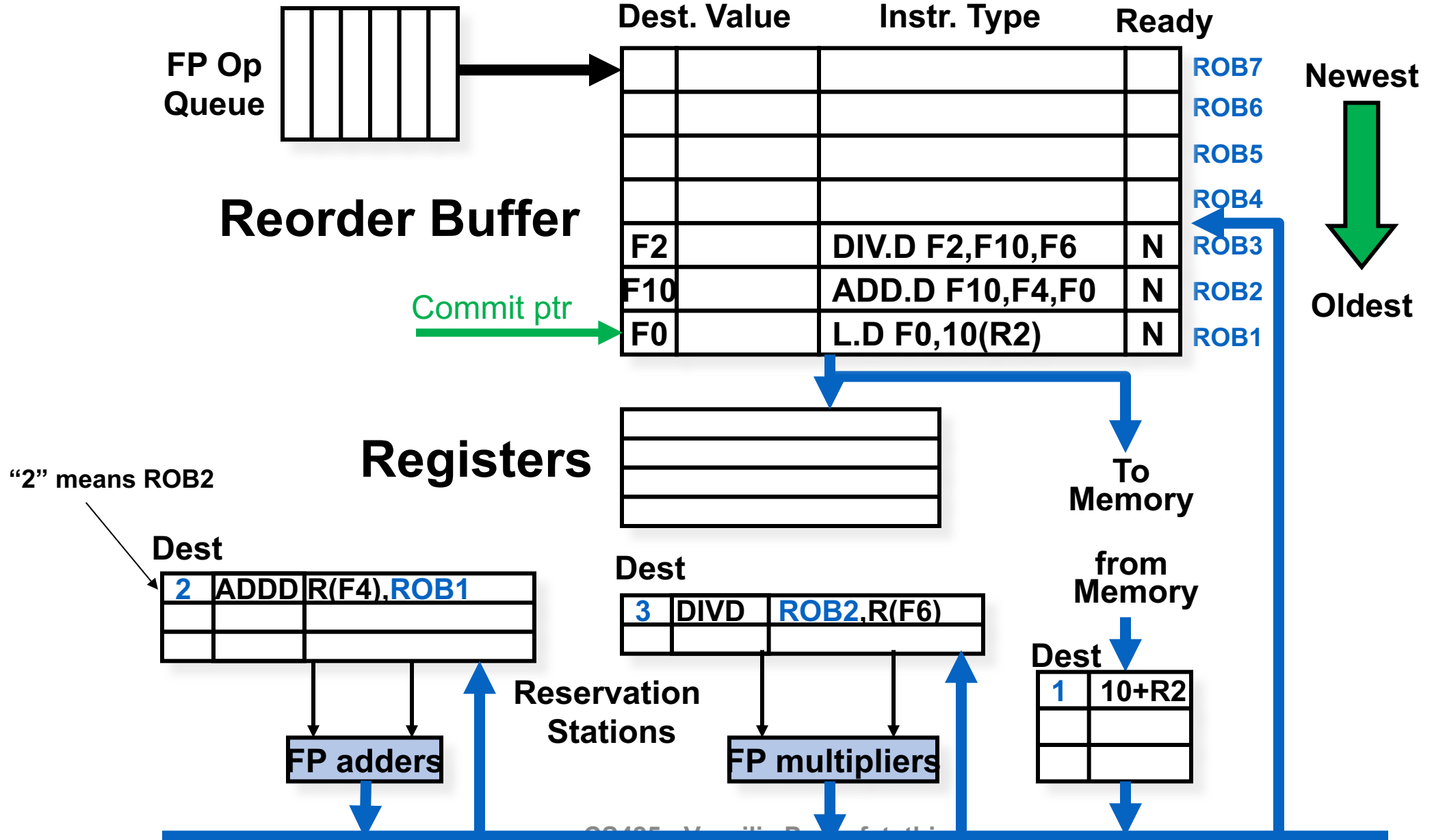
# Τέσσερα Στάδια του Tomasulo με ROB

- 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 (sometimes called “dispatch”)
- 2. Execution:** Εκτέλεσε εντολή στο Execution 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 (“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 dst reg 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”)

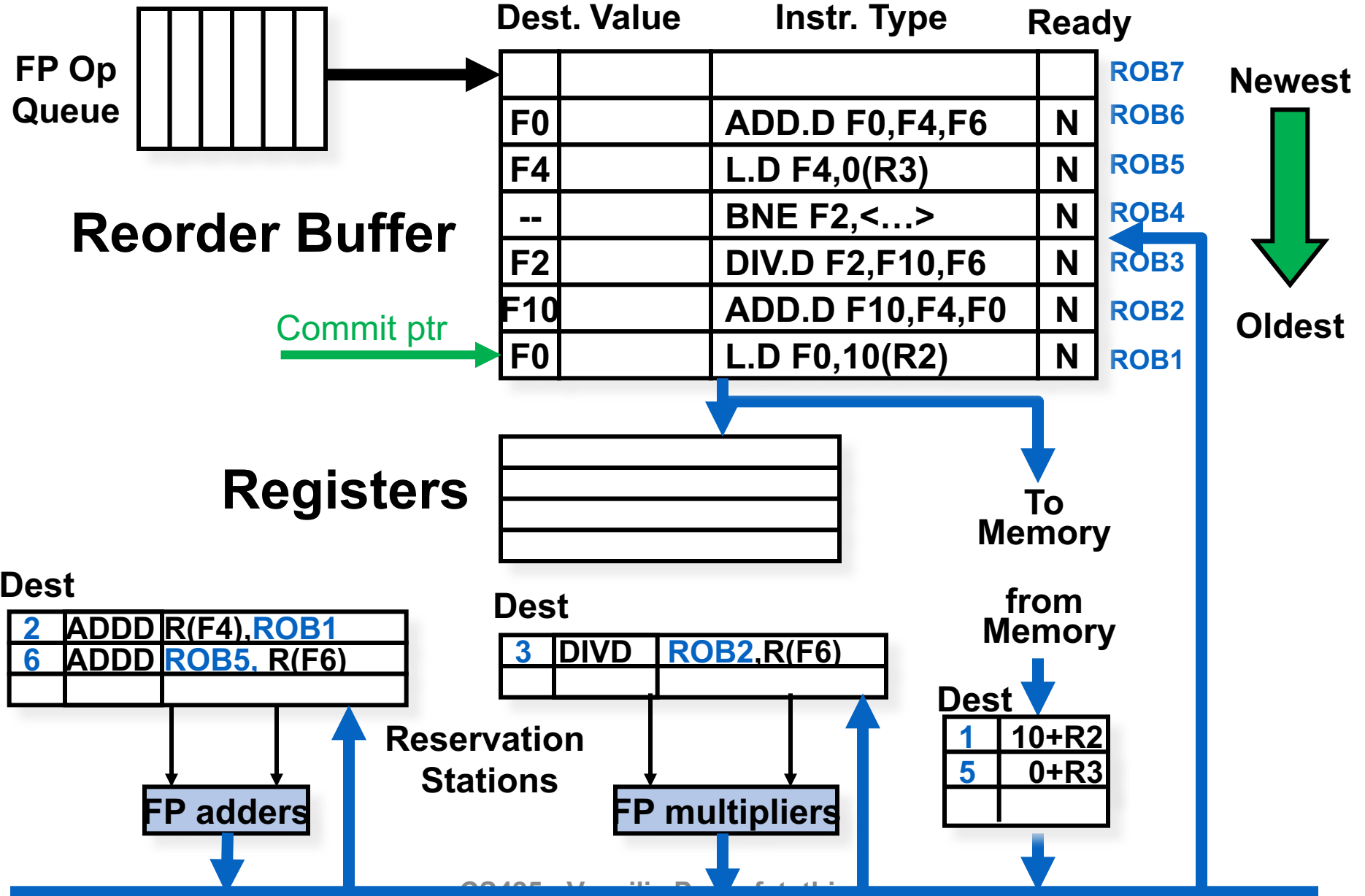
# Tomasulo With Reorder buffer



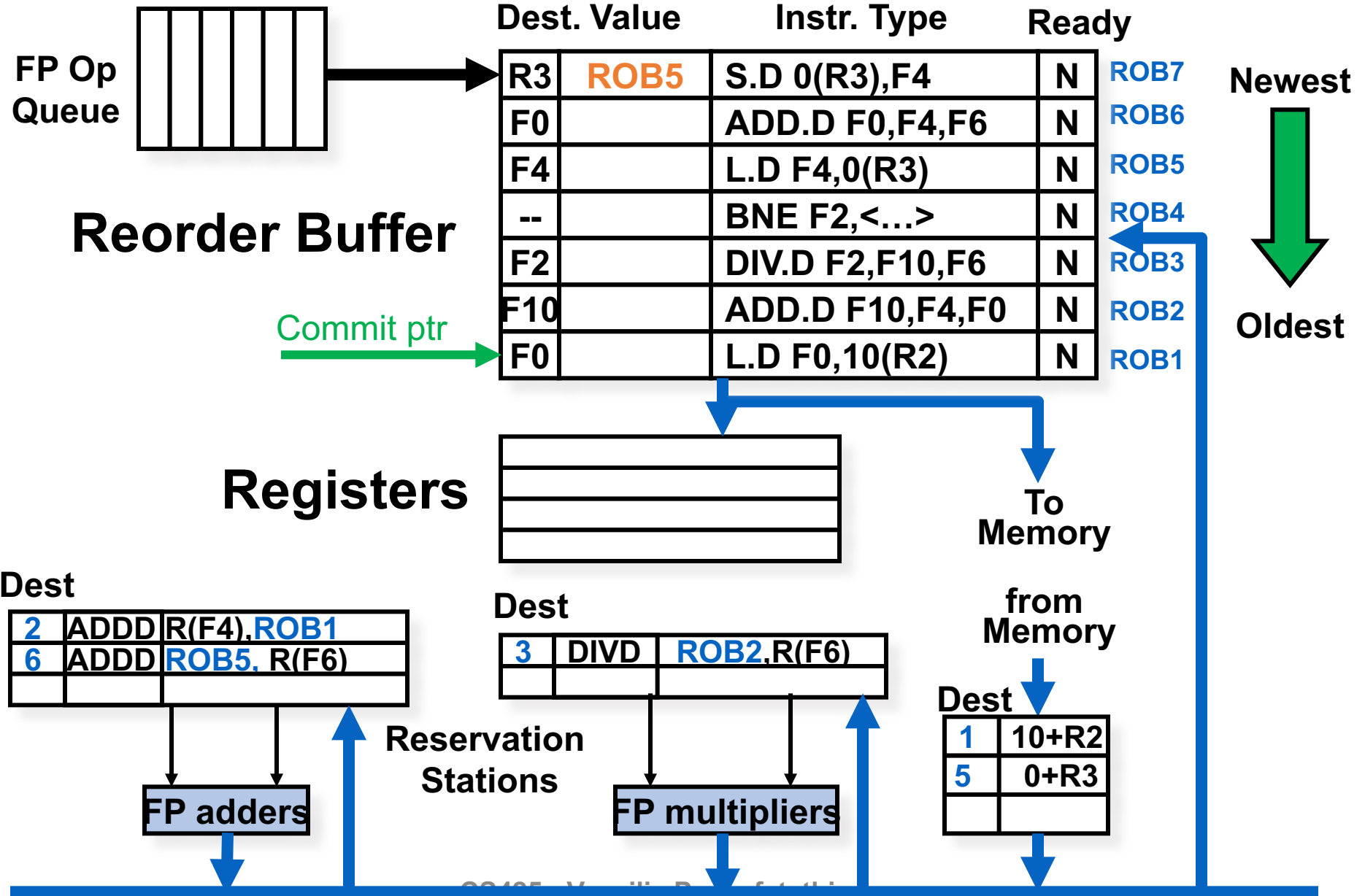
# Reorder buffer (after 2 cycles)



# Reorder buffer (after 3 cycles)

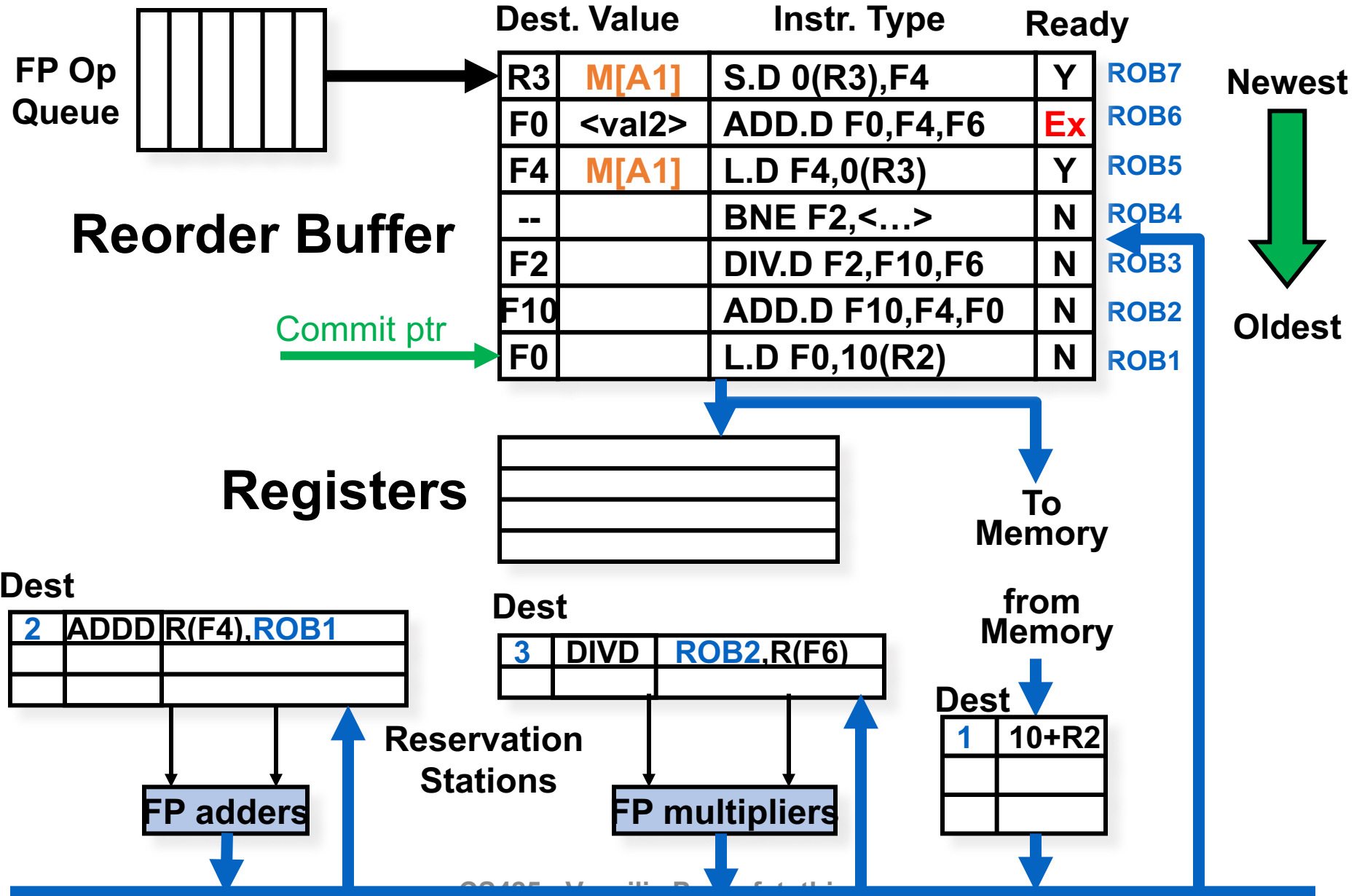


# Reorder buffer (after 1 cycle)

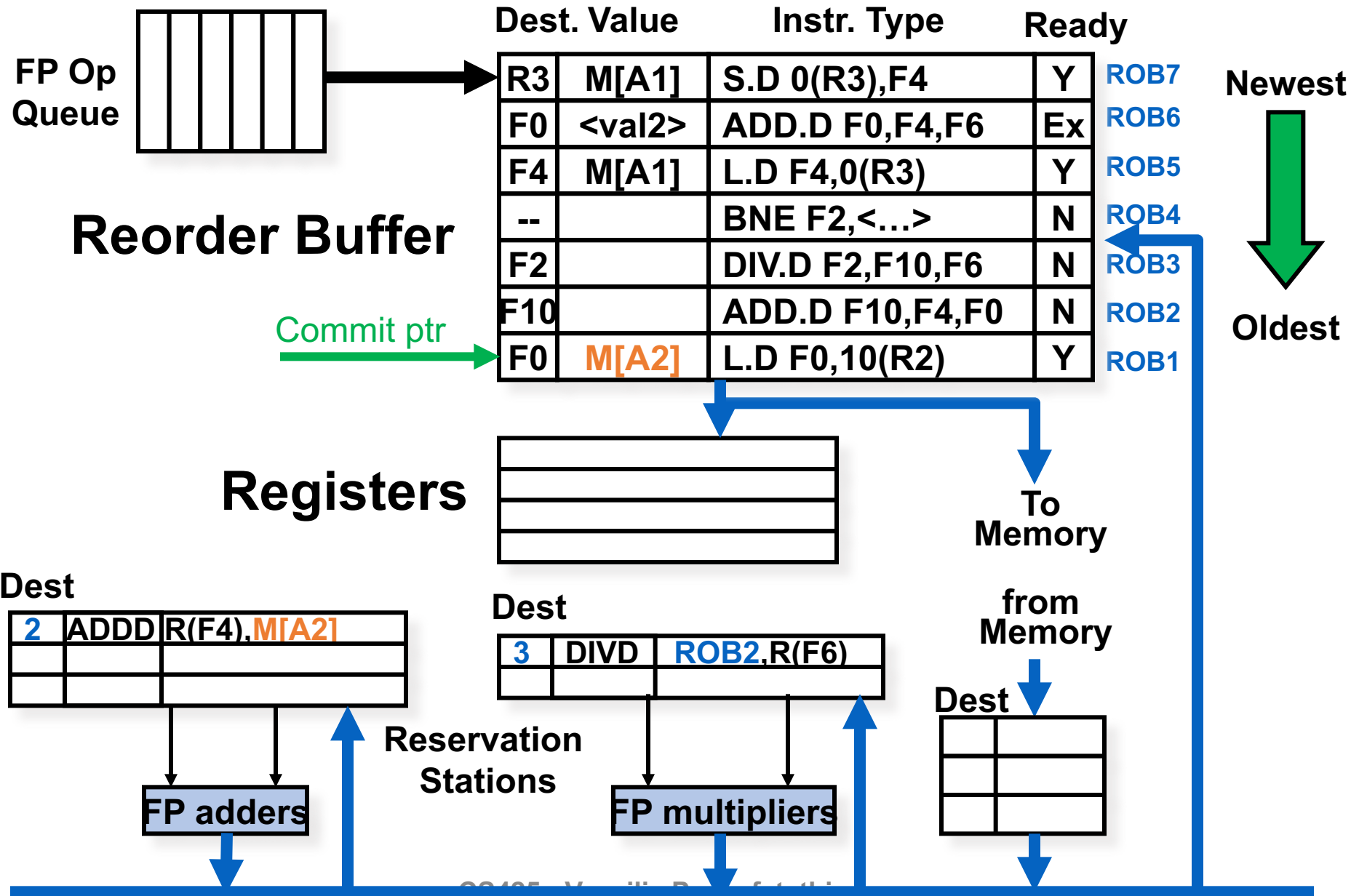




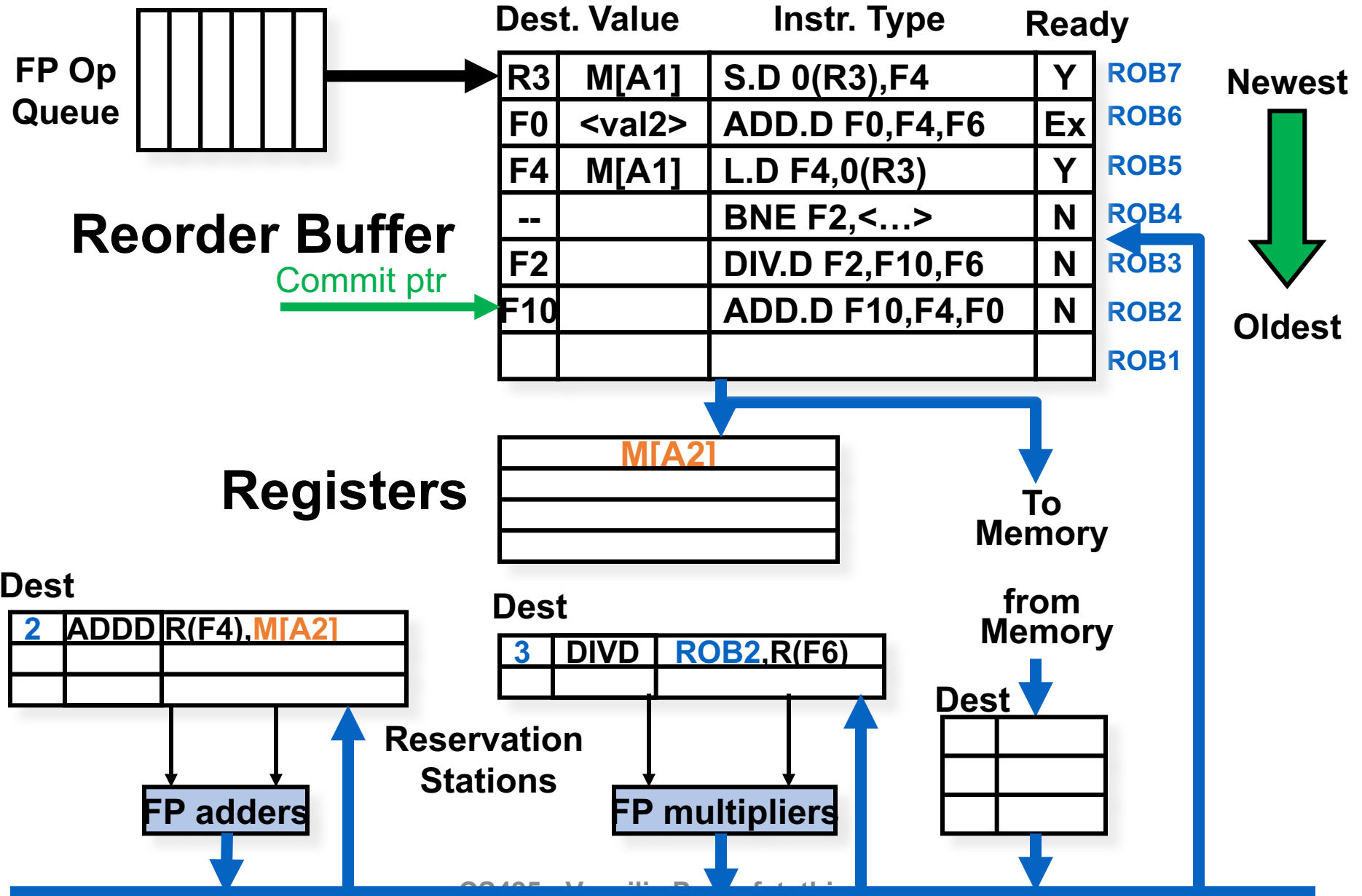
# Tomasulo With Reorder buffer



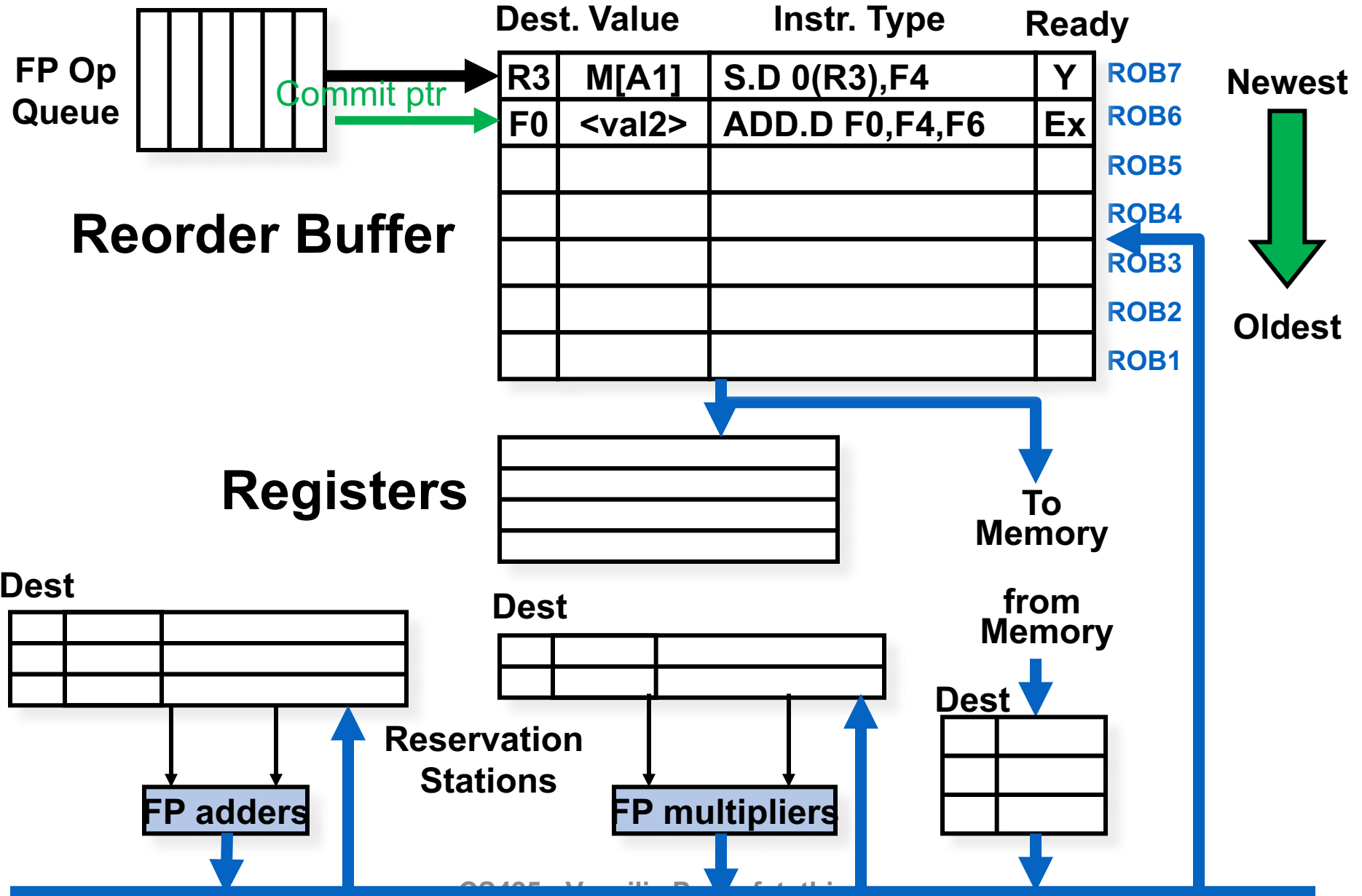
# Tomasulo With Reorder buffer



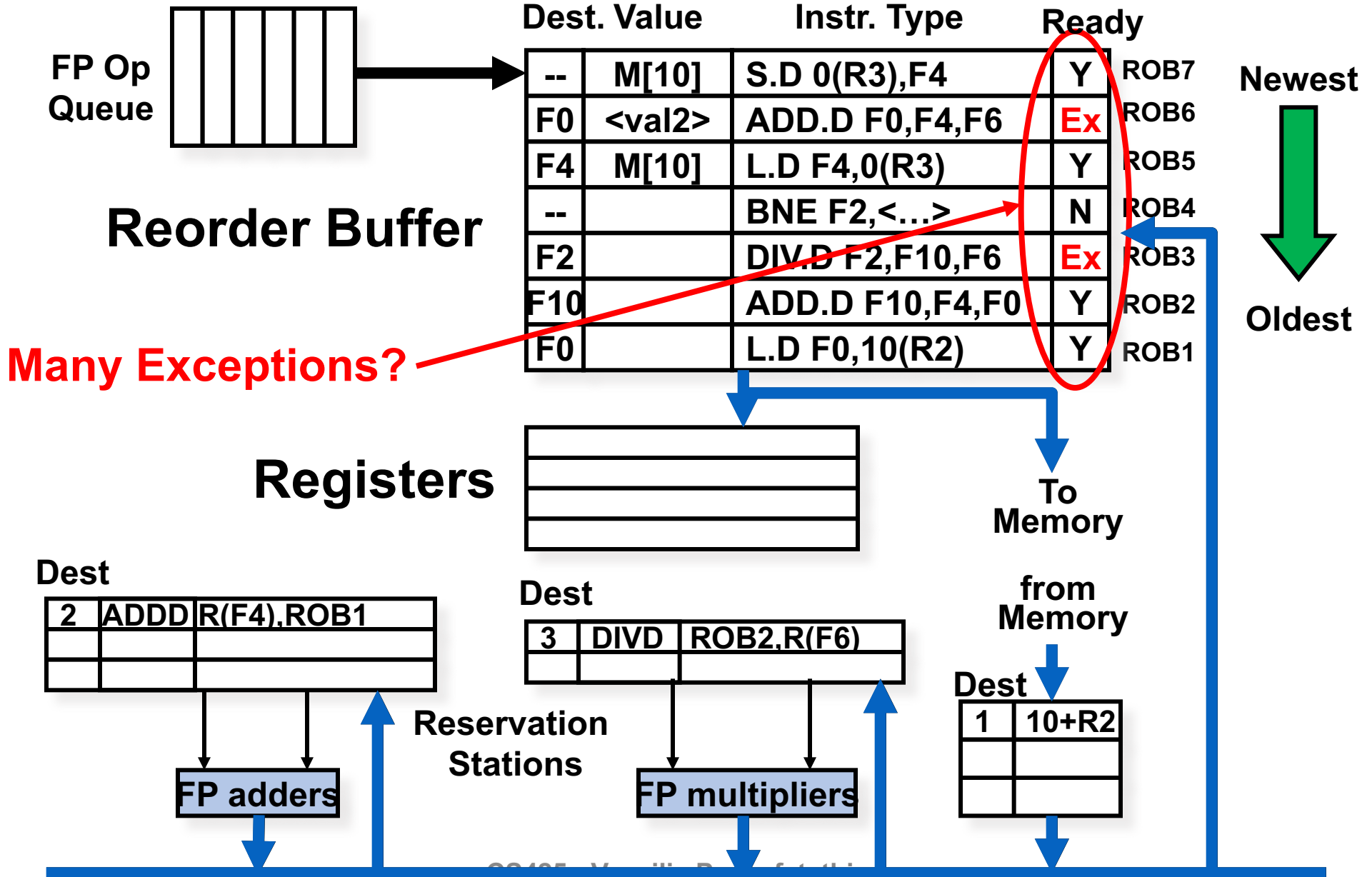
# Tomasulo With Reorder buffer



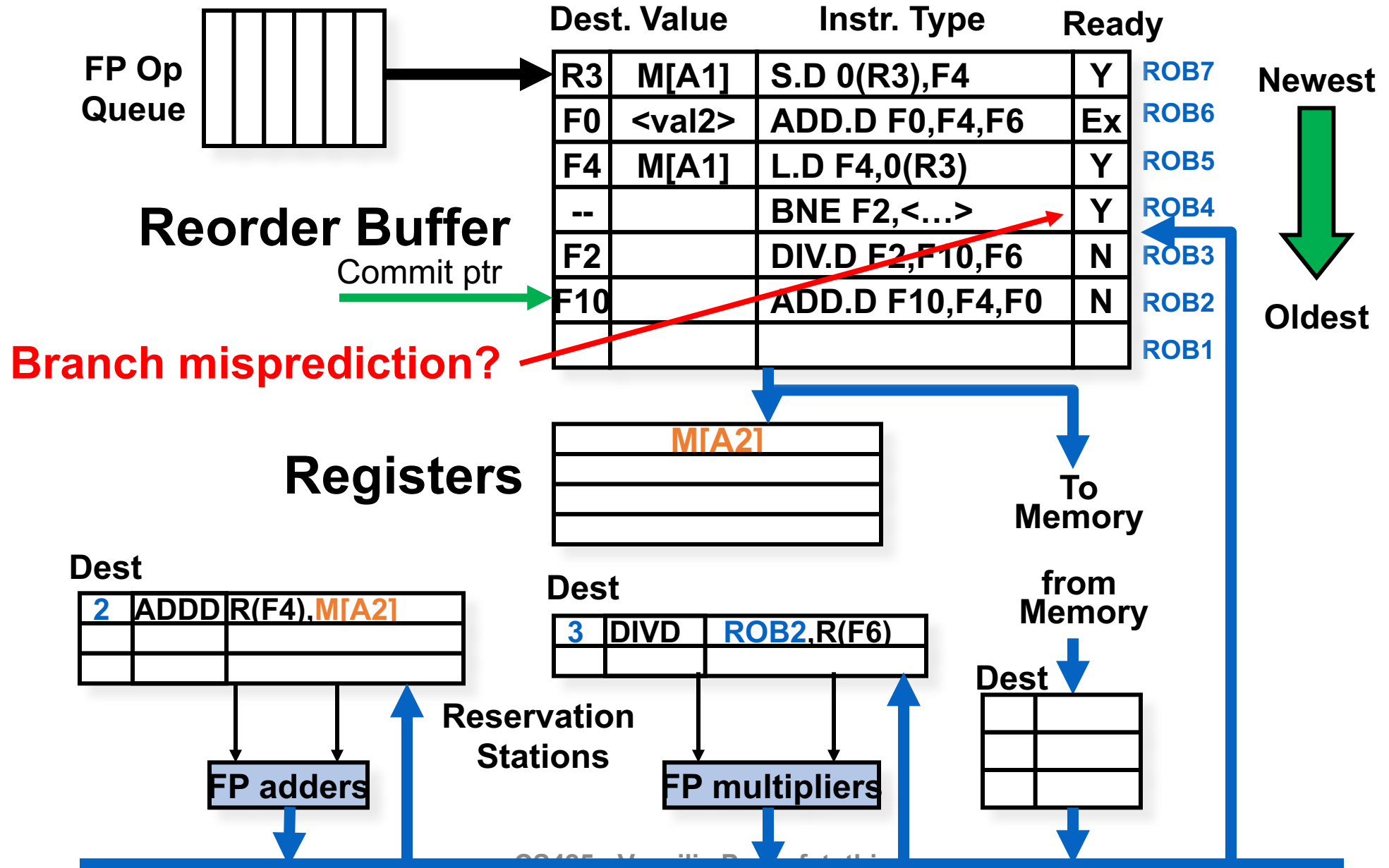
# Tomasulo With Reorder buffer



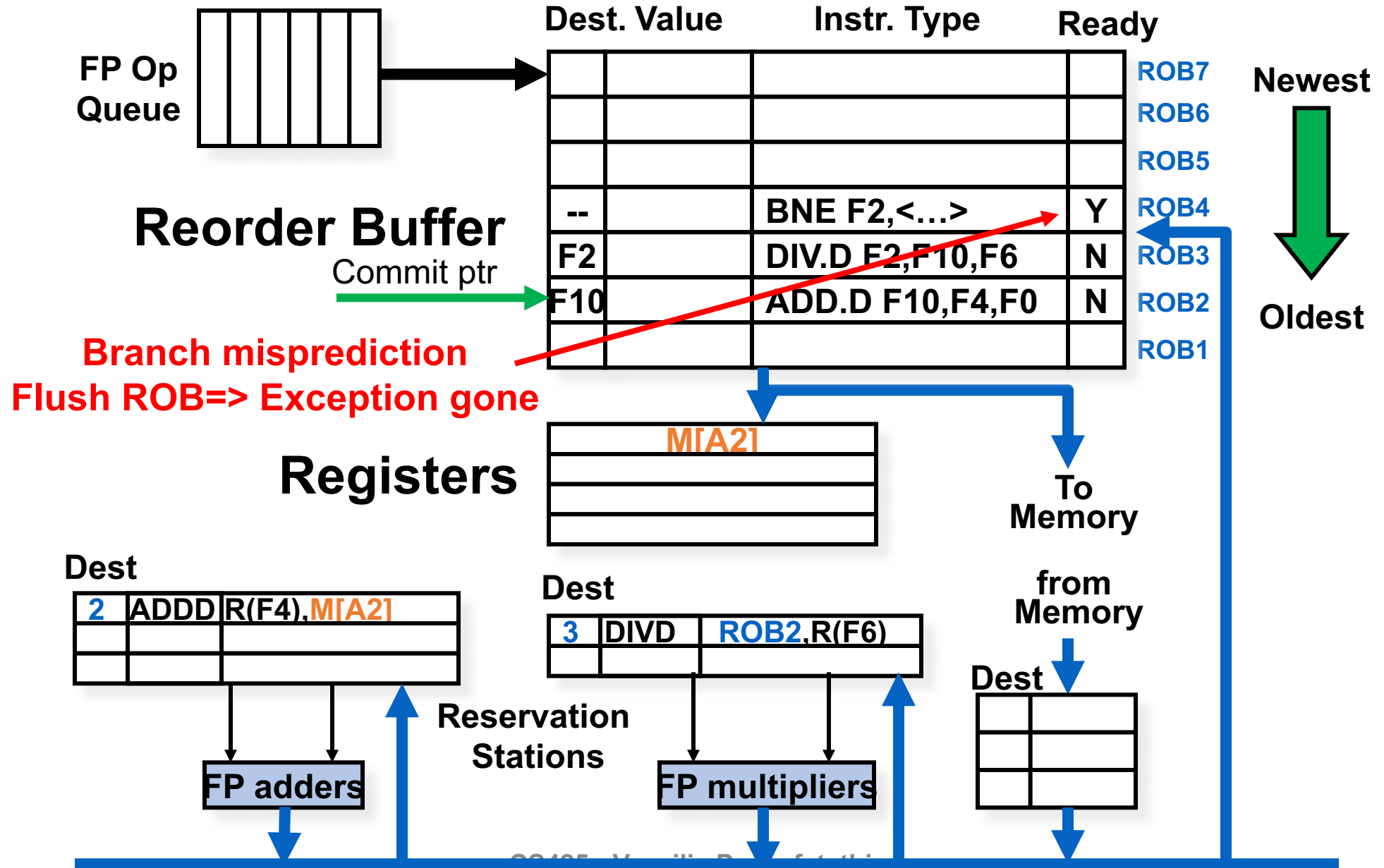
# Reorder buffer: Precise Exceptions



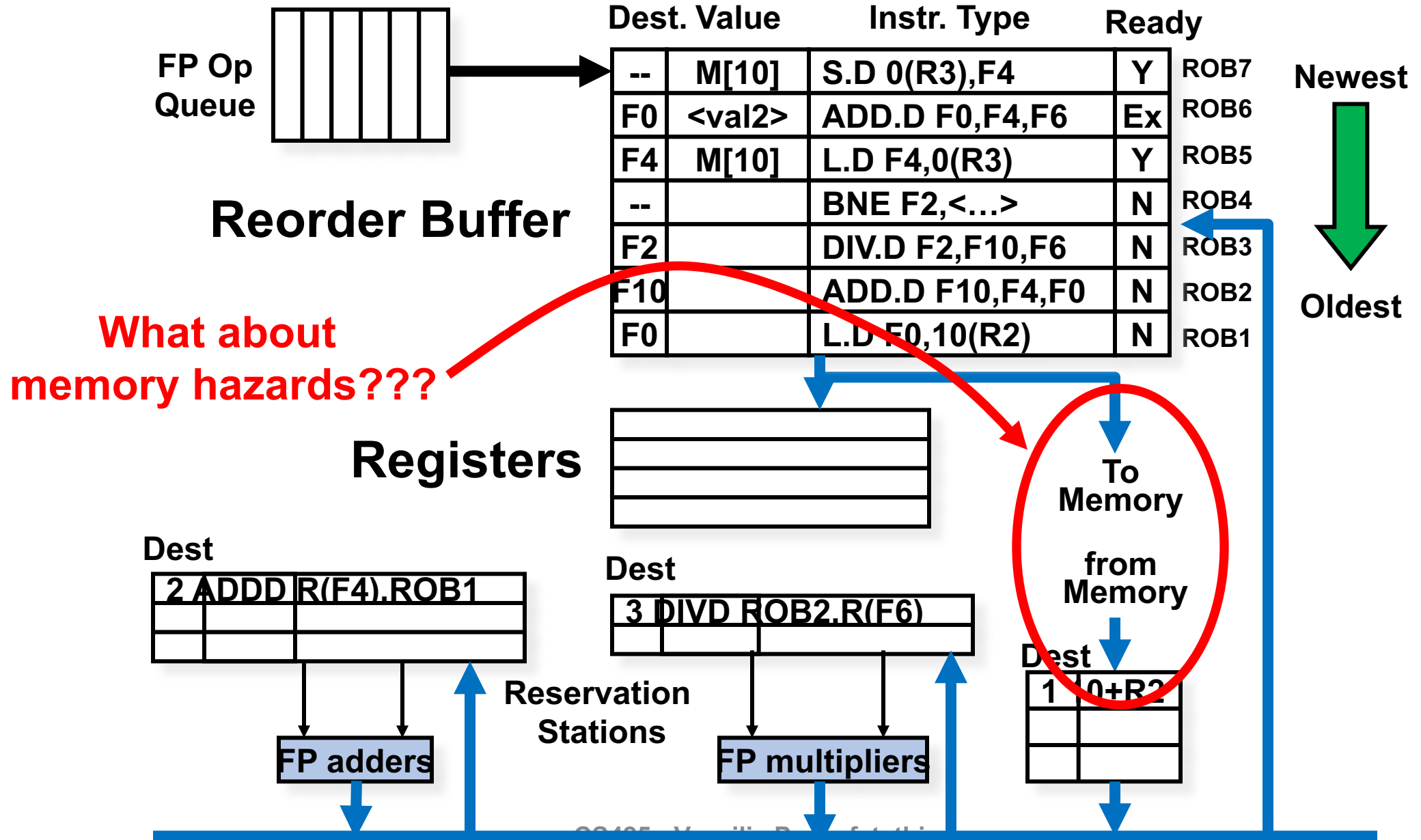
# Reorder buffer: Branch Misprediction



# Reorder buffer: Branch Misprediction



# Tomasulo With Reorder buffer





# Memory Disambiguation: WAW/WAR Hazards

- 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.

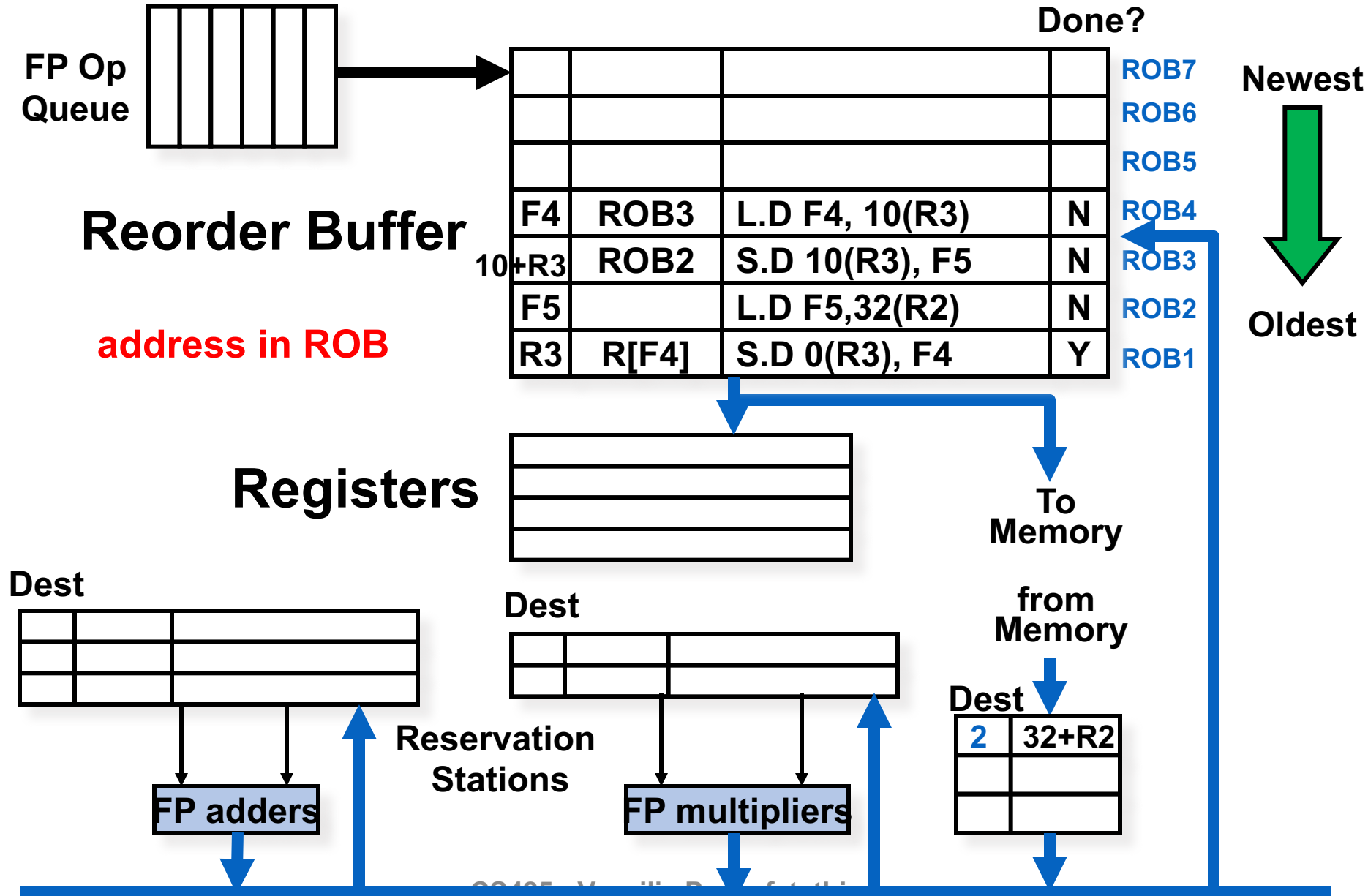
# Memory Disambiguation: RAW Hazards

- Challenge: Given a load that follows a store in program order, are these two related?
  - υπάρχει 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 (e.g. 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  $0(R2) \neq 0(R3)$
  - **Solution2**: We might guess at whether or not they are dependent (called “dependence speculation”) and use reorder buffer to fixup if we are wrong.

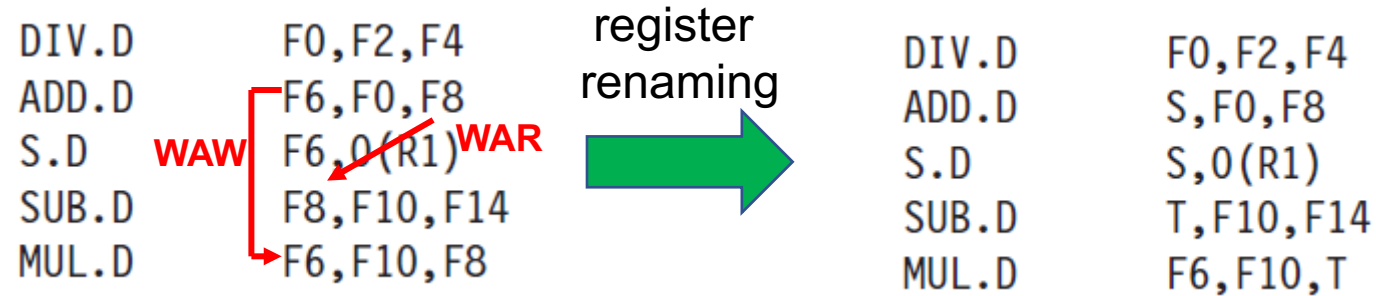
# HW support for Memory Disambiguation

- **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 κάνουμε issue ένα load, καταγράφουμε το head του store buffer (ποια stores προηγούνται)
- Όταν έχουμε την διεύθυνση του load, ελέγχουμε 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
- Stores commit in order, άρα δεν υπάρχουν WAW/WAR hazards στη μνήμη.

# Memory Disambiguation

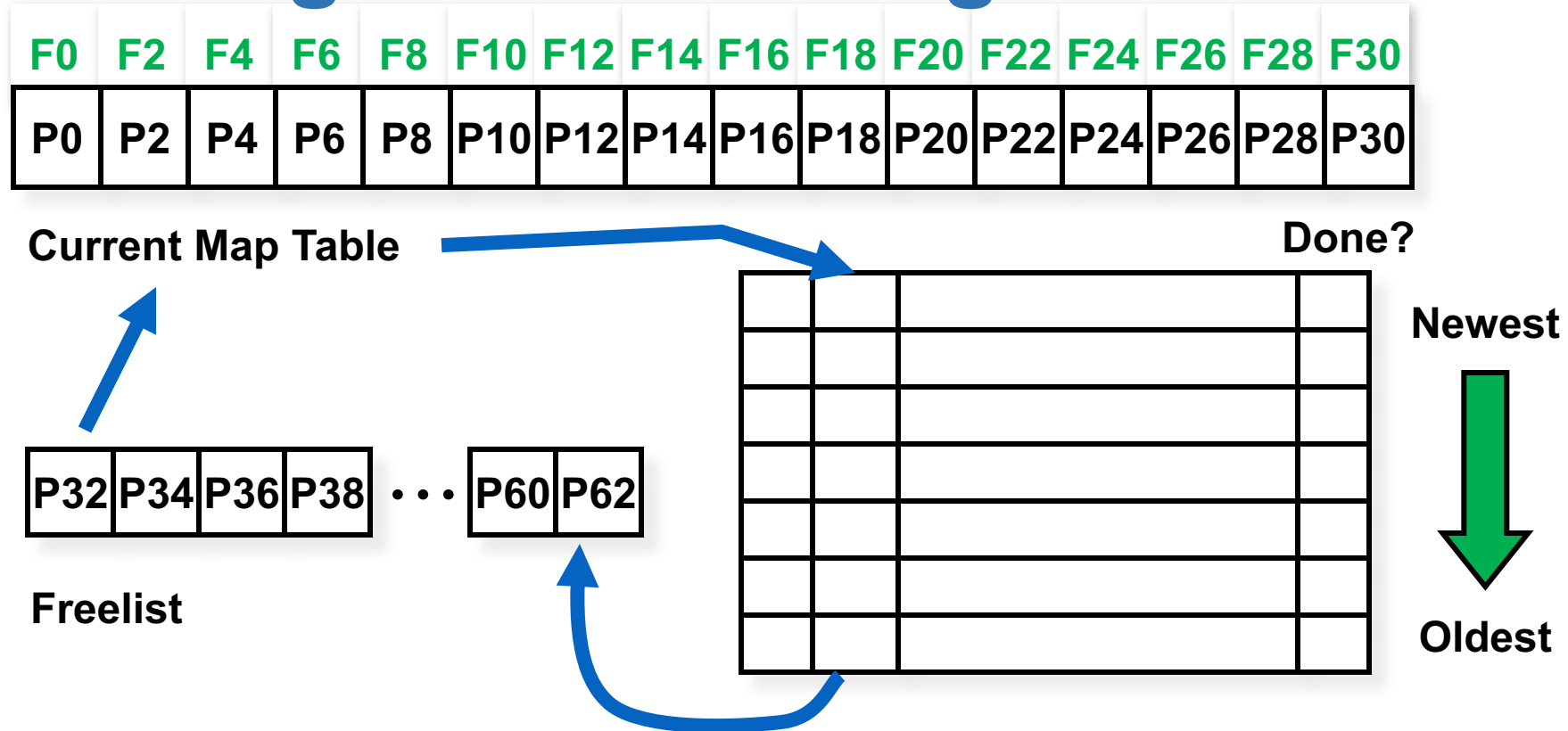


# Register Renaming



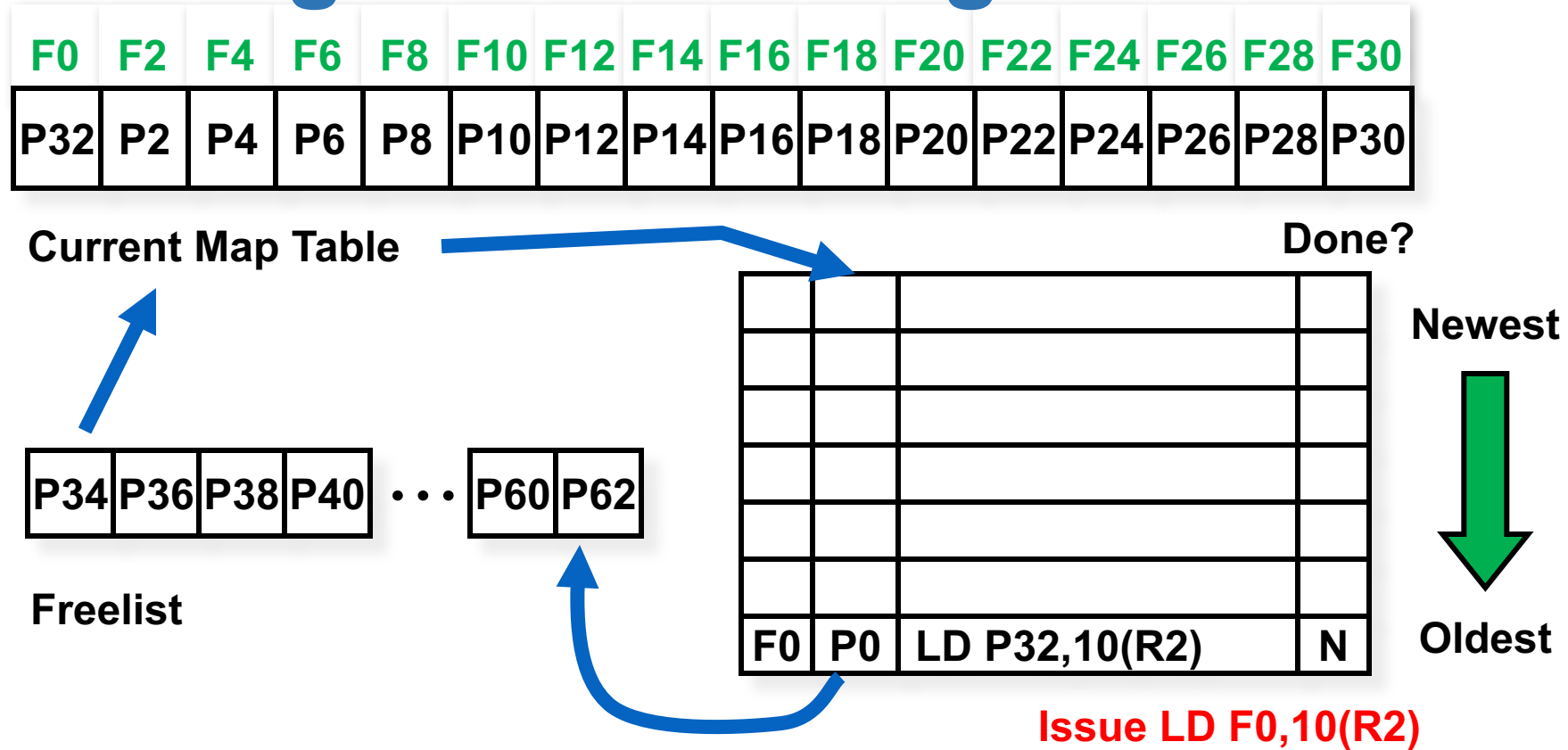
- What happens with branches?
- Tomasulo can handle renaming across branches

# 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 καινούριο physical register από την freelist

# Explicit register renaming



- 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

F0	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30
P32	P2	P4	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30

Current Map Table

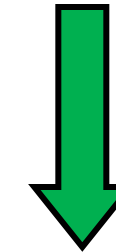
Done?

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

Freelist

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

Newest



Oldest

Issue ADD F10,F4, F0



# Explicit register renaming

F0	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30
P32	P36	P4	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30

Current Map Table

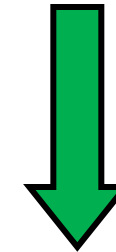
Done?

P38	P40	P42	P44	...	P60	P62
-----	-----	-----	-----	-----	-----	-----

Freelist

--			
--		BNE P36,<...>	N
F2	P2	DIV P36,P34,P6	N
F10	P10	ADD D 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

F0	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30
P40	P36	P38	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30

Current Map Table

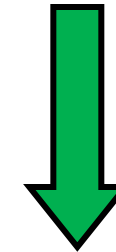
Done?

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
-----	-----	-----	-----	-----	-----	-----

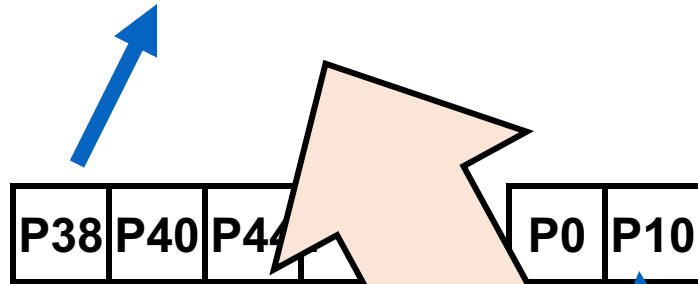
**Checkpoint at BNE instruction**

# Explicit register renaming

F0	F2	F4	F6	F8	F10	F12	F14	F16	F18	F20	F22	F24	F26	F28	F30
P32	P36	P4	P6	P8	P34	P12	P14	P16	P18	P20	P22	P24	P26	P28	P30

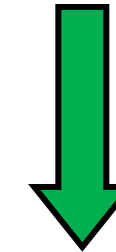
Current Map Table

Done?



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

