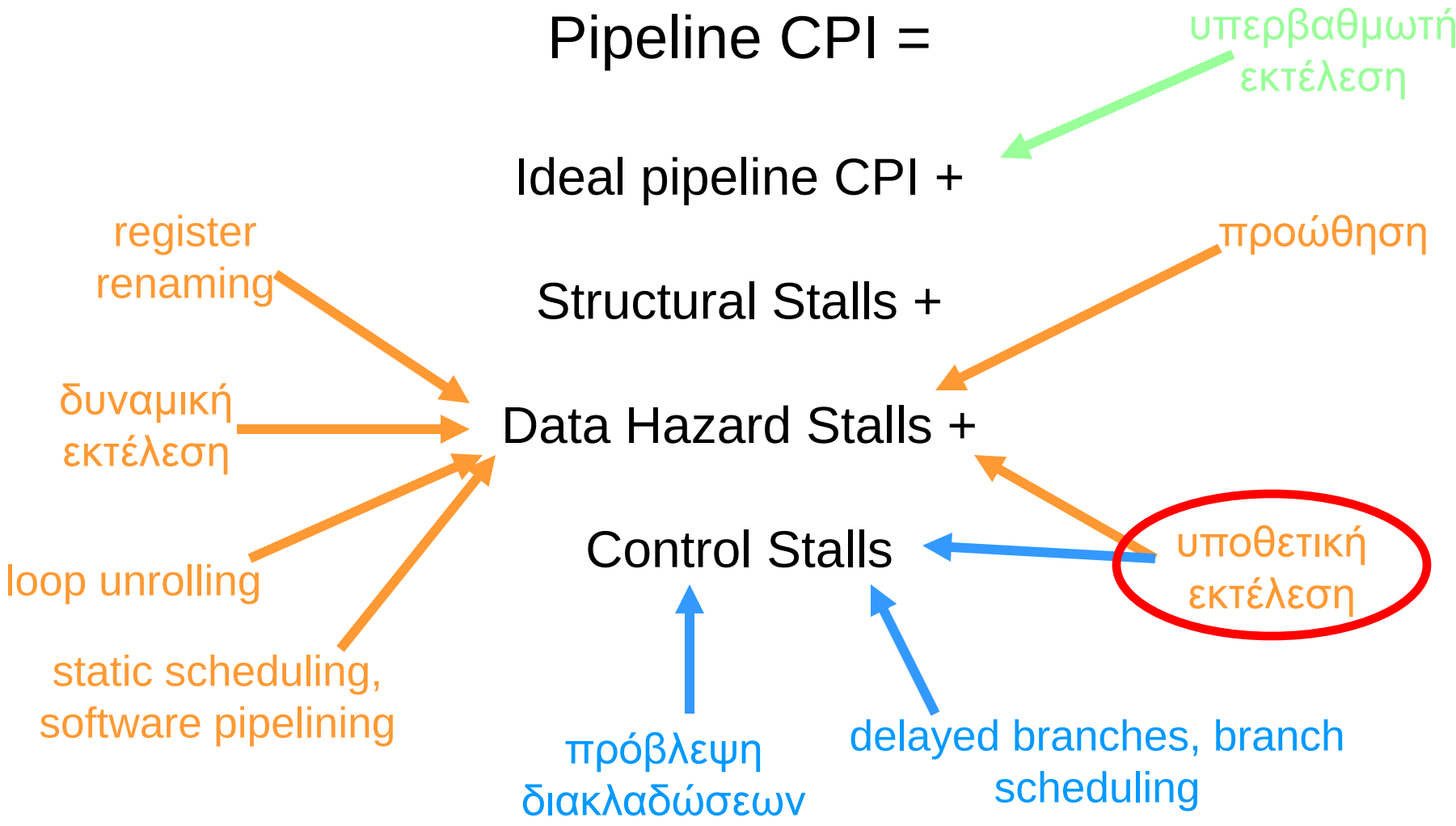
Υποθετική Εκτέλεση Εντολών (Hardware-Based Speculation)

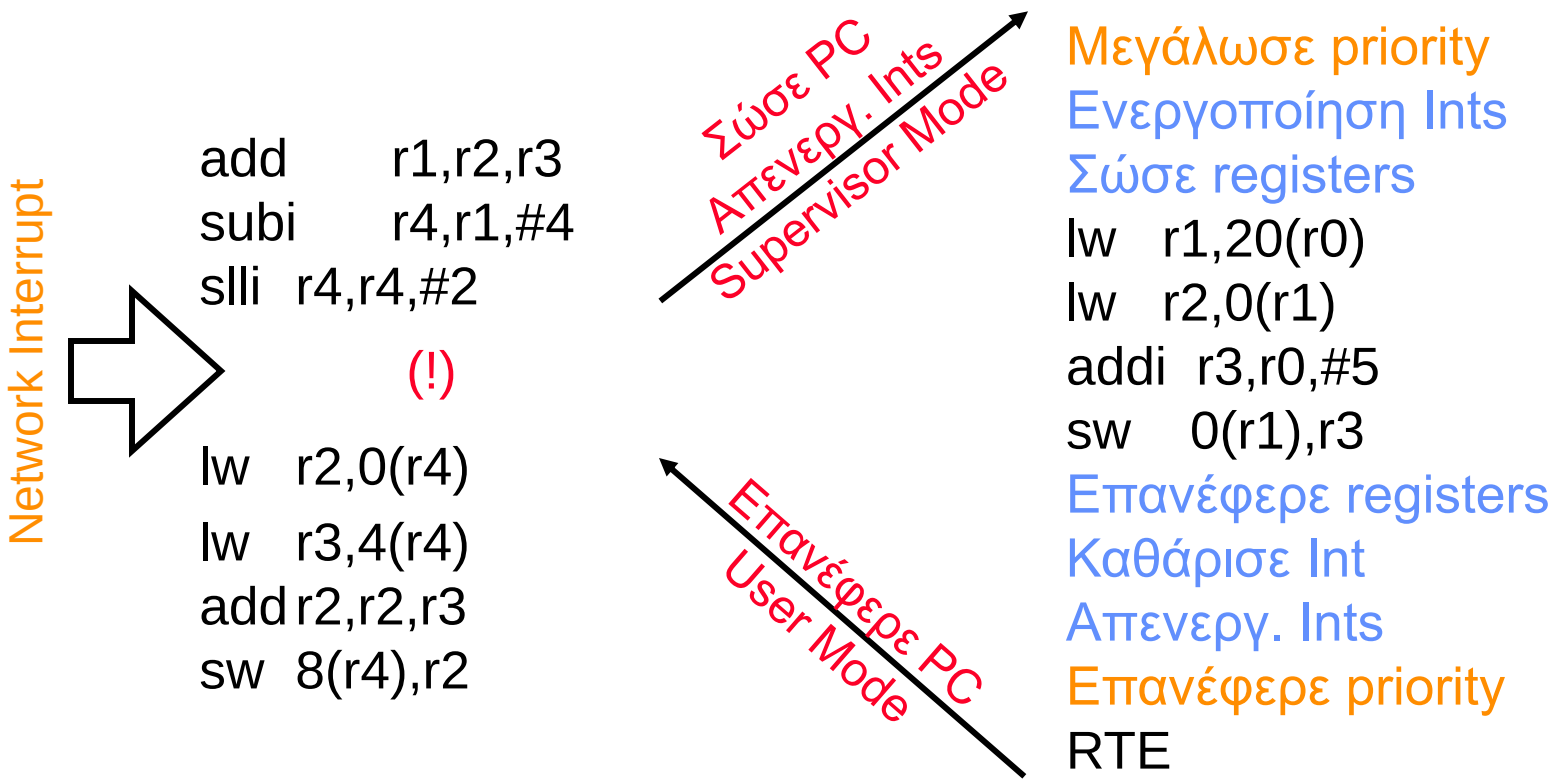
Τεχνικές βελτίωσης του CPI



Δυναμική Δρομολόγηση Εντολών

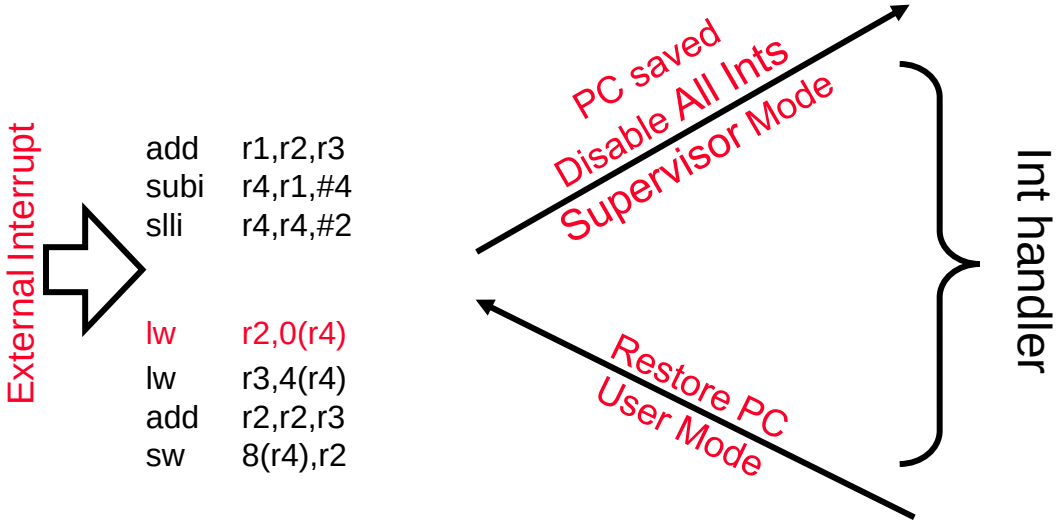
- Tomasulo, Explicit Register Renaming
  - in-order issue
  - out-of-order execution
  - out-of-order completion
- Περισσότερος παραλληλισμός (ILP)
- Βελτίωση της απόδοσης του συστήματος
- Προβλήματα
  - Interrupts/Exceptions
  - Εντολές διακλάδωσης (branches)

Device Interrupt



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 μπορούμε να επανεκκινήσουμε την εκτέλεση από το interrupt point και “να πάρουμε τα σωστά αποτελέσματα”
  - Στο παράδειγμά μας: Interrupt point είναι η lw εντολή



Γιατί χρειαζόμαστε τα precise interrupts?

- Αρκετά interrupts/exceptions χρειάζονται να είναι restartable
  - π.χ. TLB faults: διόρθωση translation και επανάληψη του load/store
  - IEEE underflows, illegal operations
- Η επανεκκίνηση δεν απαιτεί preciseness. Ωστόσο, με preciseness είναι πολύ πιο εύκολη!
- Απλοποίει το λειτουργικό σύστημα
  - το process state που χρειάζεται να αποθηκευτεί είναι μικρότερο
  - η επανεκκίνηση γίνεται γρήγορα (καλό για interrupts μικρής διάρκειας/μεγάλης συχνότητας)

Πού είναι το πρόβλημα?

- Όλες οι τεχνικές που είδαμε μέχρι τώρα (Tomasulo, Explicit Register Renaming) υλοποιούν out-of-order completion
  - Imprecise και όχι Precise Interrupts/Exceptions
    - » Είναι πιθανό όταν εμφανισθεί ένα Interrupt/Exception η κατάσταση της μηχανής (processor state) να είναι διαφορετική από ότι αν οι εντολές είχαν εκτελεστεί σειριακά ⇒ Imprecise

- Πρέπει να αλλάξουμε τη λογική του out-of-order completion, ώστε να μπορούμε να ορίζουμε precise interrupt points μέσα στο instruction stream

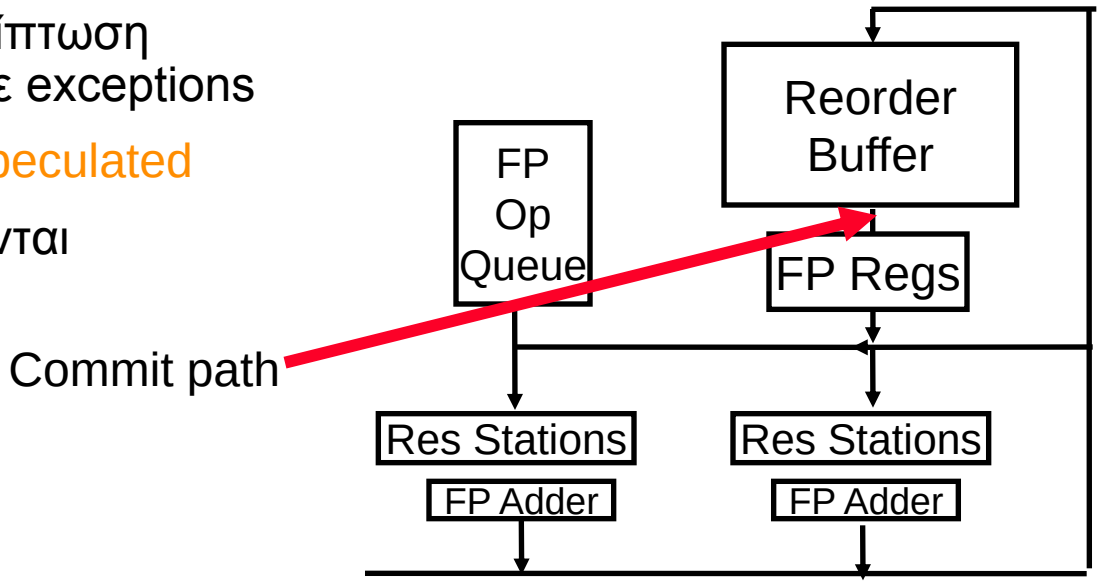
- Πρέπει να βρούμε τρόπο ώστε να *συγχρονίσουμε* το completion στάδιο των εντολών με την σειρά στο πρόγραμμα (issue-order)
  - Ο απλούστερος τρόπος: in-order completion

Εντολές διακλάδωσης (branches)

- Για να βελτιώσουμε την απόδοση του συστήματος χρησιμοποιούμε branch prediction
- Επομένως κατά το ooo execution εκτελούμε εντολές οι οποίες εξαρτώνται από το αποτέλεσμα της εντολής διακλάδωσης
- Αν η πρόβλεψη δεν επαληθευτεί, θα πρέπει να κάνουμε rollback στο σημείο όπου κάναμε την πρόβλεψη, διότι οι εντολές στο λανθασμένο predicted path έχουν ήδη εκτελεστεί
  - Αυτό ακριβώς είναι το πρόβλημα και με τα precise exceptions!
- Λύση: in-order completion

Hardware Support

- Ιδέα του Reorder Buffer (ROB):
  - Κράτα εντολές σε μία FIFO, ακριβώς με την σειρά που γίνονται issue.
    - » Κάθε ROB entry περιέχει: instruction type (branch/store/register op), destination, value, ready field
  - Όταν η εντολή τελειώσει την εκτέλεση, τοποθέτησε τα αποτελέσματα στον ROB.
    - » Παρέχει operands σε άλλες εντολές στο διάστημα μεταξύ execution complete & commit
  - Η εντολή αλλάζει την κατάσταση της μηχανής στο commit στάδιο όχι στο WB → in-order commit → οι τιμές στην κεφαλή του ROB αποθηκεύονται στο register file
  - Εύκολη η αναίρεση σε περίπτωση mispredicted branches ή σε exceptions
    - » Απλά διαγραφή των speculated instructions που βρίσκονται στο ROB



Αλγόριθμος Tomasulo με ROB

Issue — Πάρε εντολή από FP Op Queue

Αν υπάρχει ελεύθερο reservation station & reorder buffer entry, issue instr & send operands & reorder buffer no. for destination

Execution — Εκτέλεσε εντολή στο Ex Unit (EX)

Όταν και οι τιμές και των 2 source regs είναι έτοιμες εκτέλεσε την εντολή; Αν όχι, παρακολούθησε το CDB για το αποτέλεσμα; Όταν και οι 2 τιμές βρίσκονται στο RS, εκτέλεσε την εντολή; (checks RAW)

Write result — Τέλος εκτέλεσης (WB)

Γράψε το αποτέλεσμα στο CDB προς όλες τις μονάδες που το περιμένουν & στον reorder buffer; Σημείωσε τον RS ως διαθέσιμο.

Commit — Ανανέωσε τον dest register με το αποτέλεσμα από τον reorder buffer

Όταν η εντολή φτάσει στην κεφαλή του reorder buffer & το result είναι διαθέσιμο, ανανέωσε τον dest register με αυτό (ή, αντίστοιχα, αποθήκευσε στη μνήμη) και βγάλε την εντολή από τον reorder buffer. Όταν η εντολή είναι mispredicted branch, καθάρισε (flush) τον reorder buffer και επανεκκίνησε την εκτέλεση στο σωστό path.

Status	Wait until	Action or bookkeeping

Issue all instructions	Reservation station (r) and ROB (b) both available	`if (RegisterStat[rs].Busy)/*in-flight instr. writes rs*/`
		`  {h ← RegisterStat[rs].Reorder;`
		`  if (ROB[h].Ready)/* Instr completed already */`
		`  {RS[r].Vj ← ROB[h].Value; RS[r].Qj ← 0;}`
		`  else {RS[r].Qj ← h;} /* wait for instruction */`
		`} else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0;};`
		`RS[r].Busy ← yes; RS[r].Dest ← b;`
		`ROB[b].Instruction ← opcode; ROB[b].Dest ← rd;ROB[b].Ready ← no;`

FP operations and stores		`if (RegisterStat[rt].Busy) /*in-flight instr writes rt*/`
		`  {h ← RegisterStat[rt].Reorder;`
		`  if (ROB[h].Ready)/* Instr completed already */`
		`  {RS[r].Vk ← ROB[h].Value; RS[r].Qk ← 0;}`
		`  else {RS[r].Qk ← h;} /* wait for instruction */`
		`} else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0;};`

FP operations		`RegisterStat[rd].Reorder ← b; RegisterStat[rd].Busy ← yes;` `ROB[b].Dest ← rd;`
Loads		`RS[r].A ← imm; RegisterStat[rt].Reorder ← b;` `RegisterStat[rt].Busy ← yes; ROB[b].Dest ← rt;`
Stores		`RS[r].A ← imm;`

Status	Wait until	Action or bookkeeping
Execute FP op	(RS[r].Qj == 0) and (RS[r].Qk == 0)	Compute results—operands are in Vj and Vk
Load step 1	(RS[r].Qj == 0) and there are no stores earlier in the queue	`RS[r].A ← RS[r].Vj + RS[r].A;`
Load step 2	Load step 1 done and all stores earlier in ROB have different address	Read from Mem[RS[r].A]
Store	(RS[r].Qj == 0) and store at queue head	`ROB[h].Address ← RS[r].Vj + RS[r].A;`

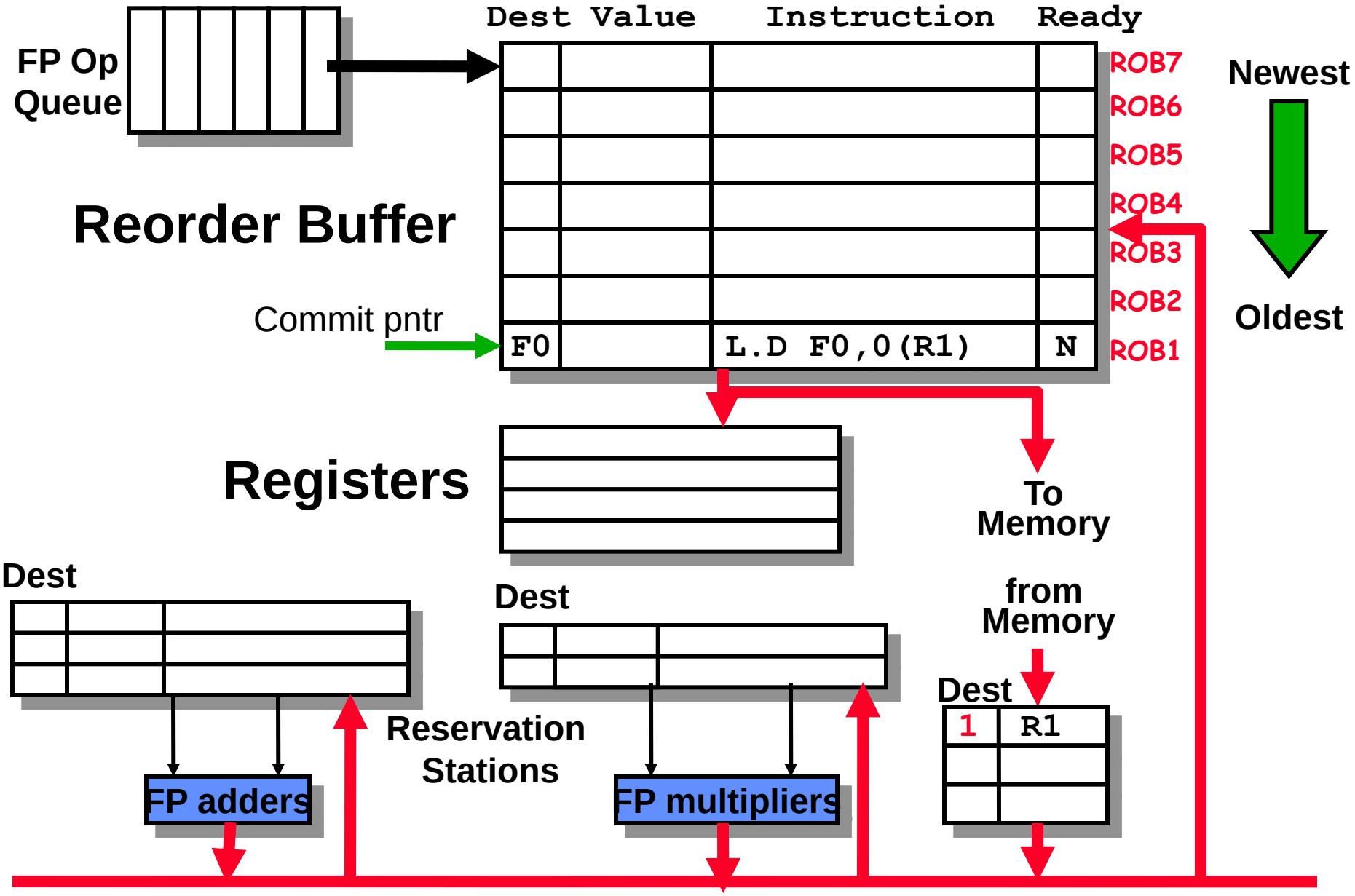
Status	Wait until	Action or bookkeeping

Write result all but store	Execution done at *r* and CDB available	`b ← RS[r].Dest; RS[r].Busy ← no;`
		`∀x(if (RS[x].Qj==b) {RS[x].Vj ← result; RS[x].Qj ← 0});`
		`∀x(if (RS[x].Qk==b) {RS[x].Vk ← result; RS[x].Qk ← 0});`
		`ROB[b].Value ← result; ROB[b].Ready ← yes;`

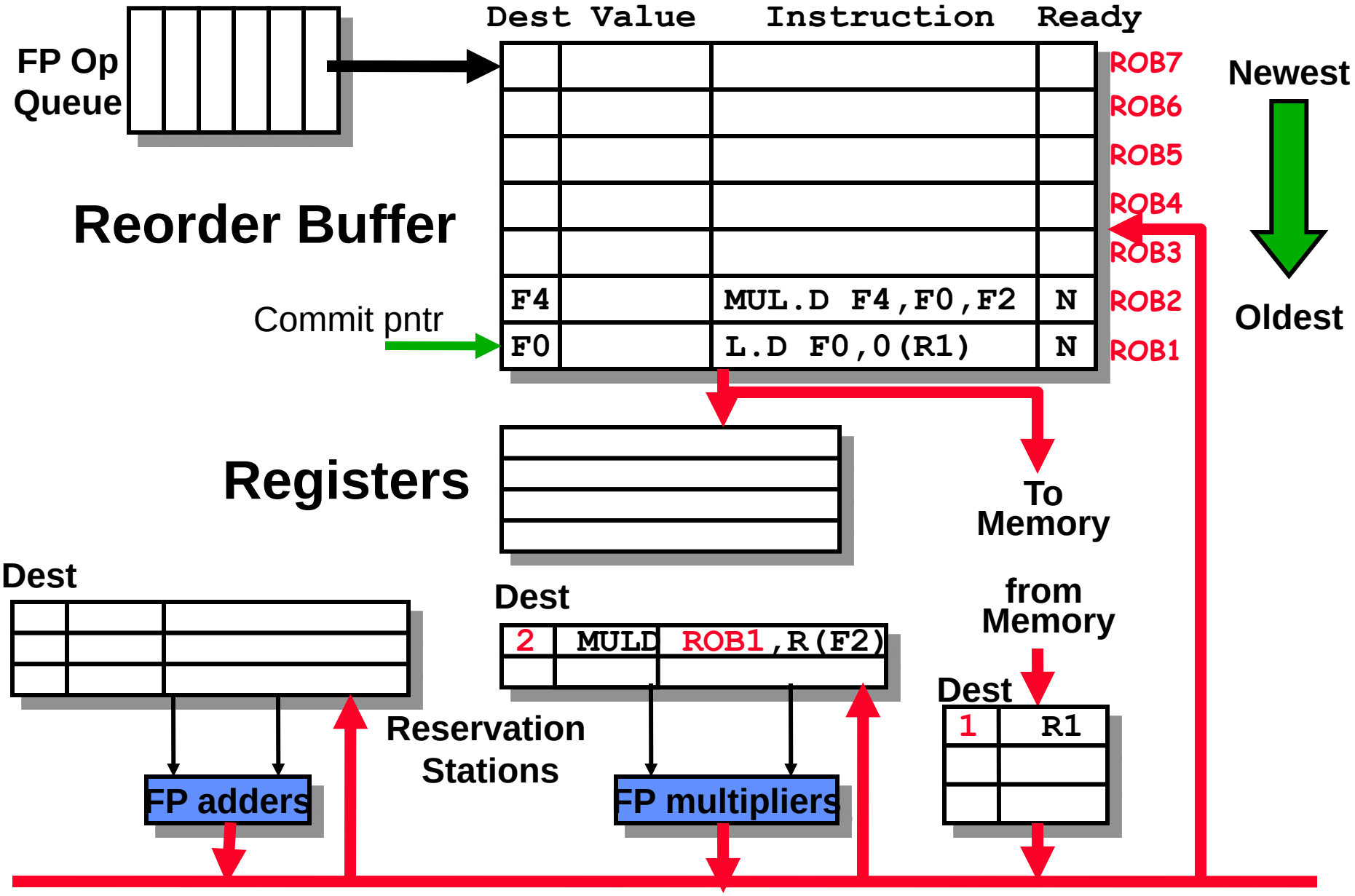
Store	Execution done at *r* and (RS[r].Qk == 0)	`ROB[h].Value ← RS[r].Vk;`

Commit	Instruction is at the head of the ROB (entry h) and ROB[h].ready == yes	`d ← ROB[h].Dest; /* register dest, if exists */`
		`if (ROB[h].Instruction==Branch)`
		`{if (branch is mispredicted)`
		`{clear ROB[h], RegisterStat; fetch branch dest;};}`
		`else if (ROB[h].Instruction==Store)`
		`{Mem[ROB[h].Destination] ← ROB[h].Value;}`
		`else /* put the result in the register destination */`
		`{Regs[d] ← ROB[h].Value;};`
		`ROB[h].Busy ← no; /* free up ROB entry */`
		`/* free up dest register if no one else writing it */`
		`if (RegisterStat[d].Reorder==h) {RegisterStat[d].Busy ← no;};`

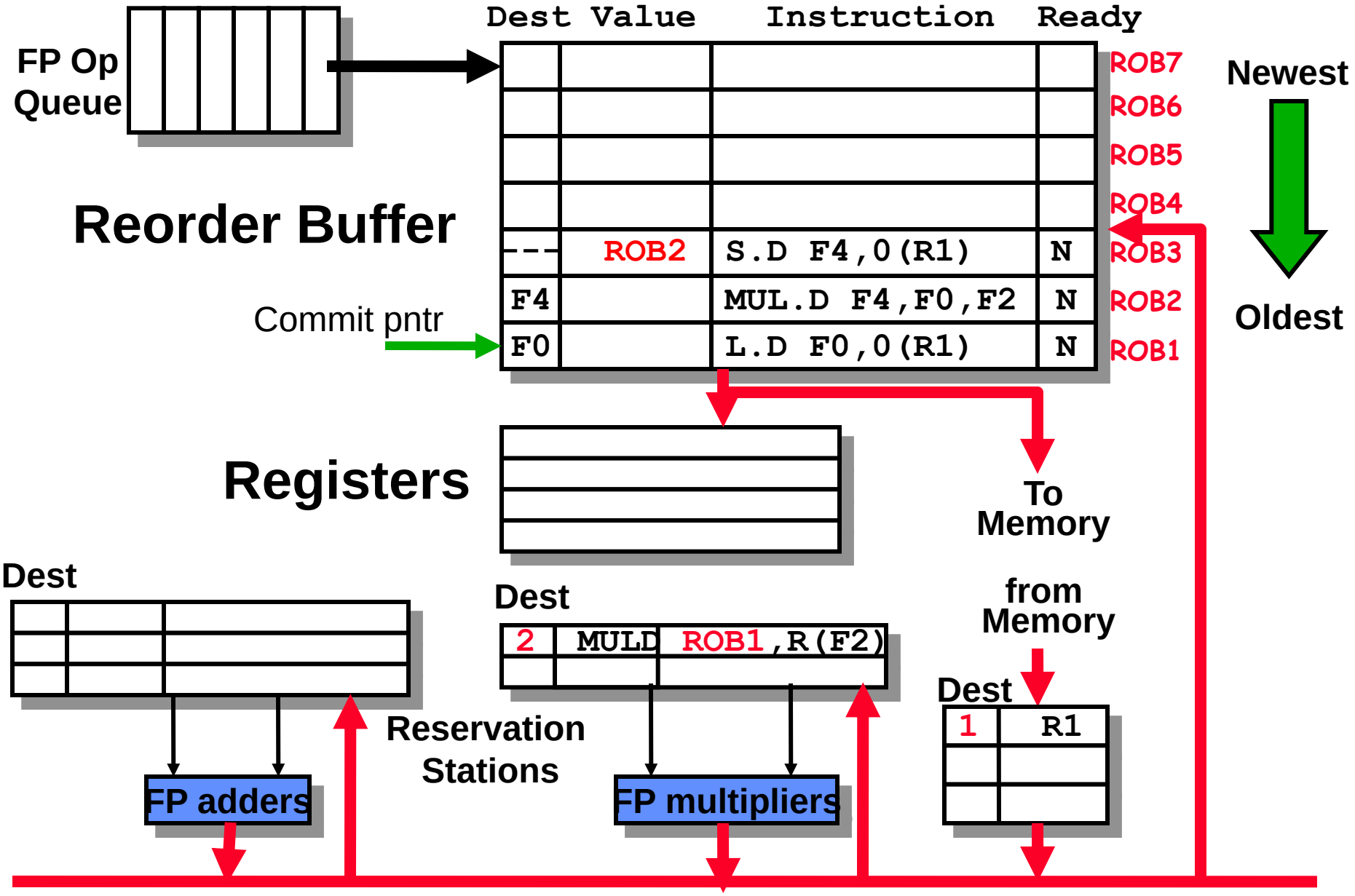
Tomasulo With Reorder buffer(1)



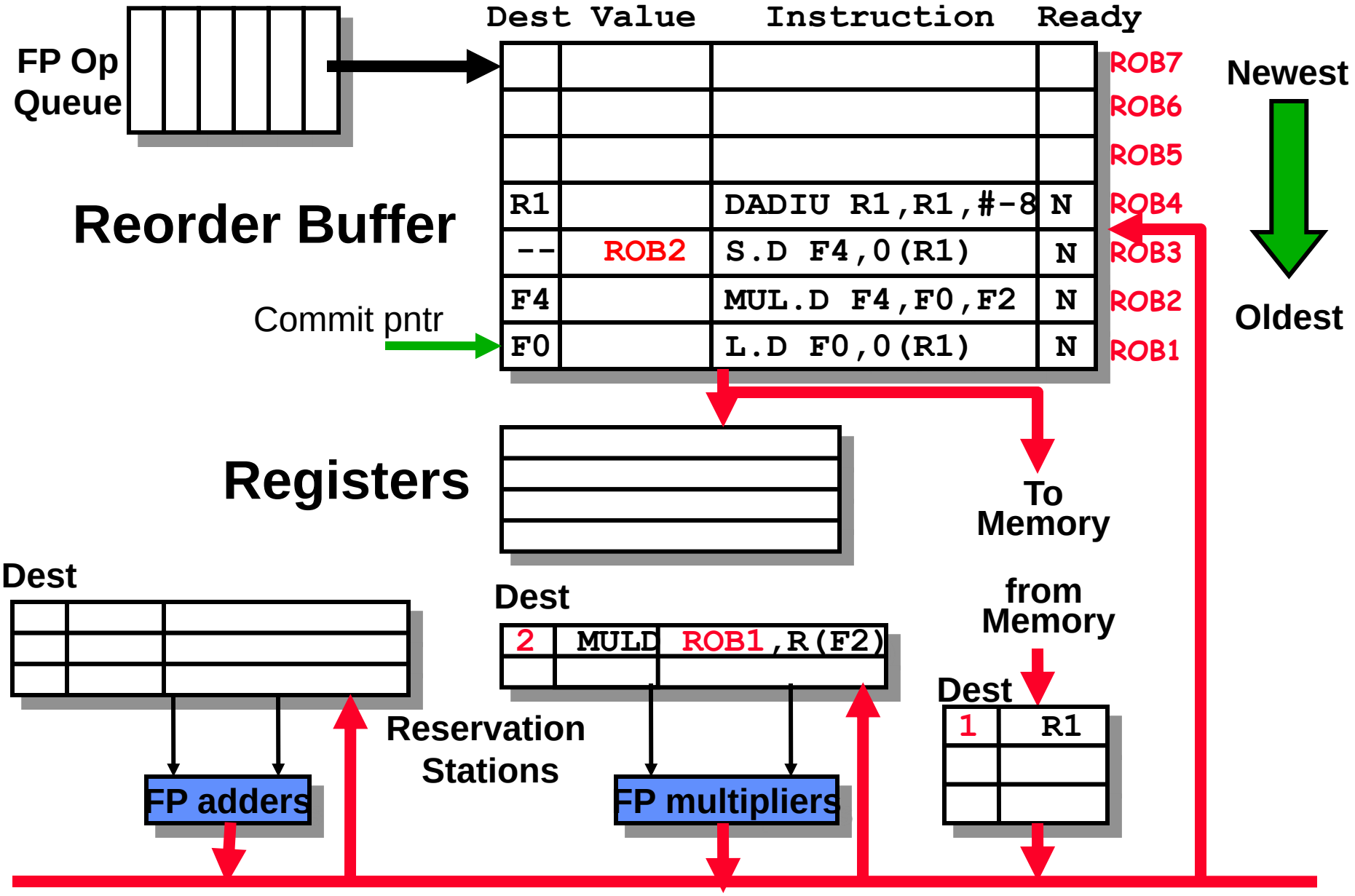
Tomasulo With Reorder buffer(2)



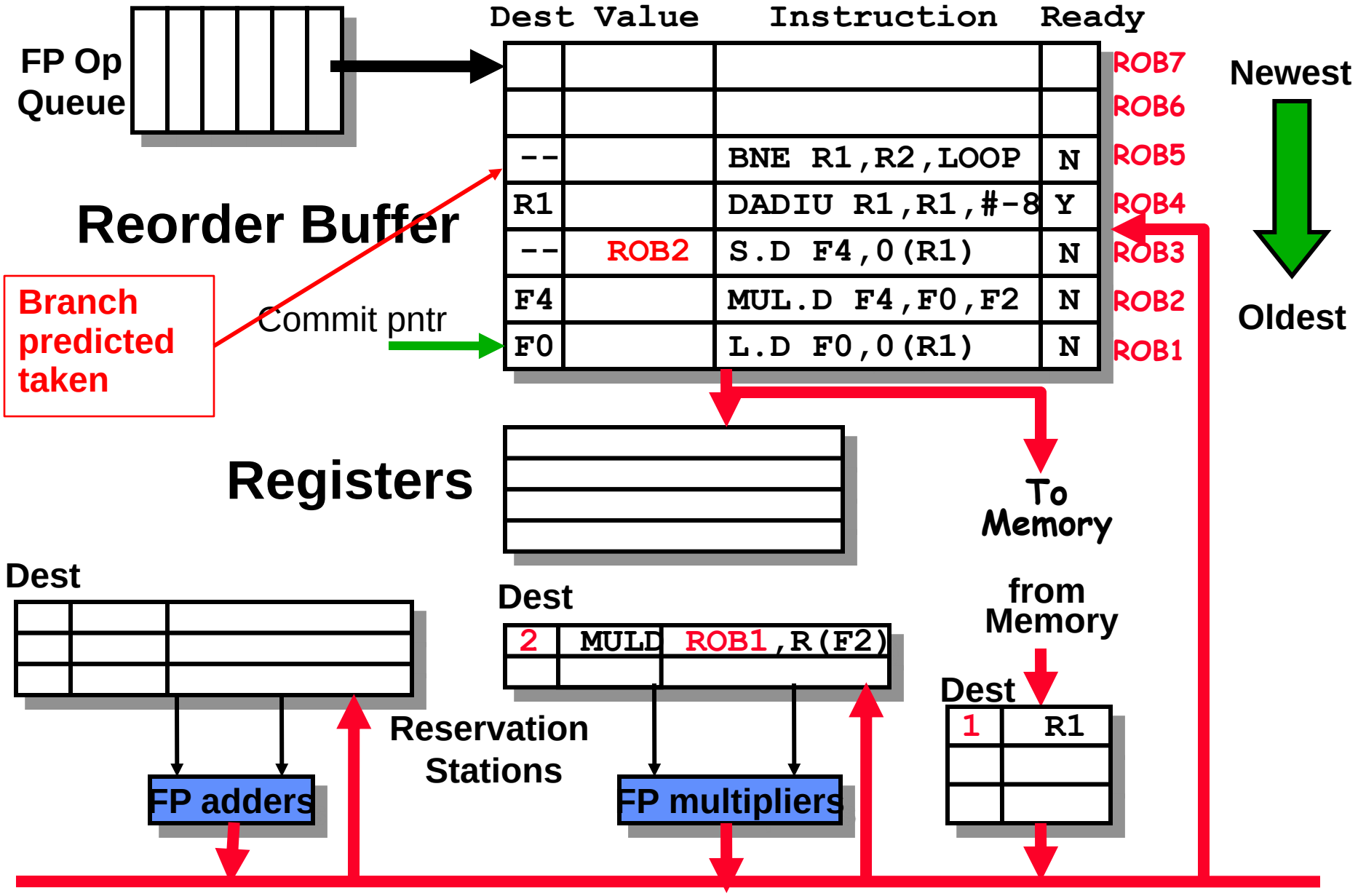
Tomasulo With Reorder buffer(3)



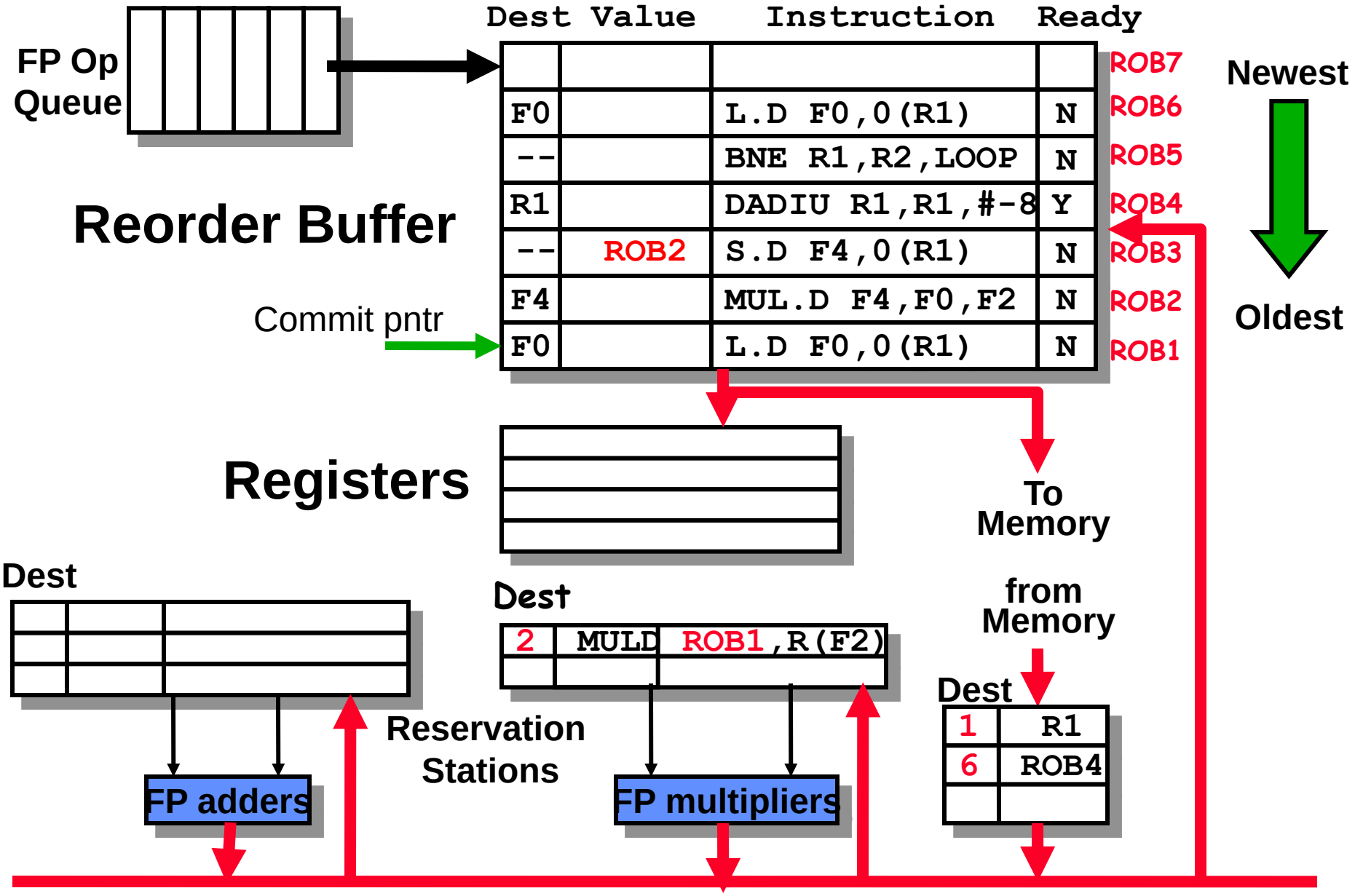
Tomasulo With Reorder buffer(4)



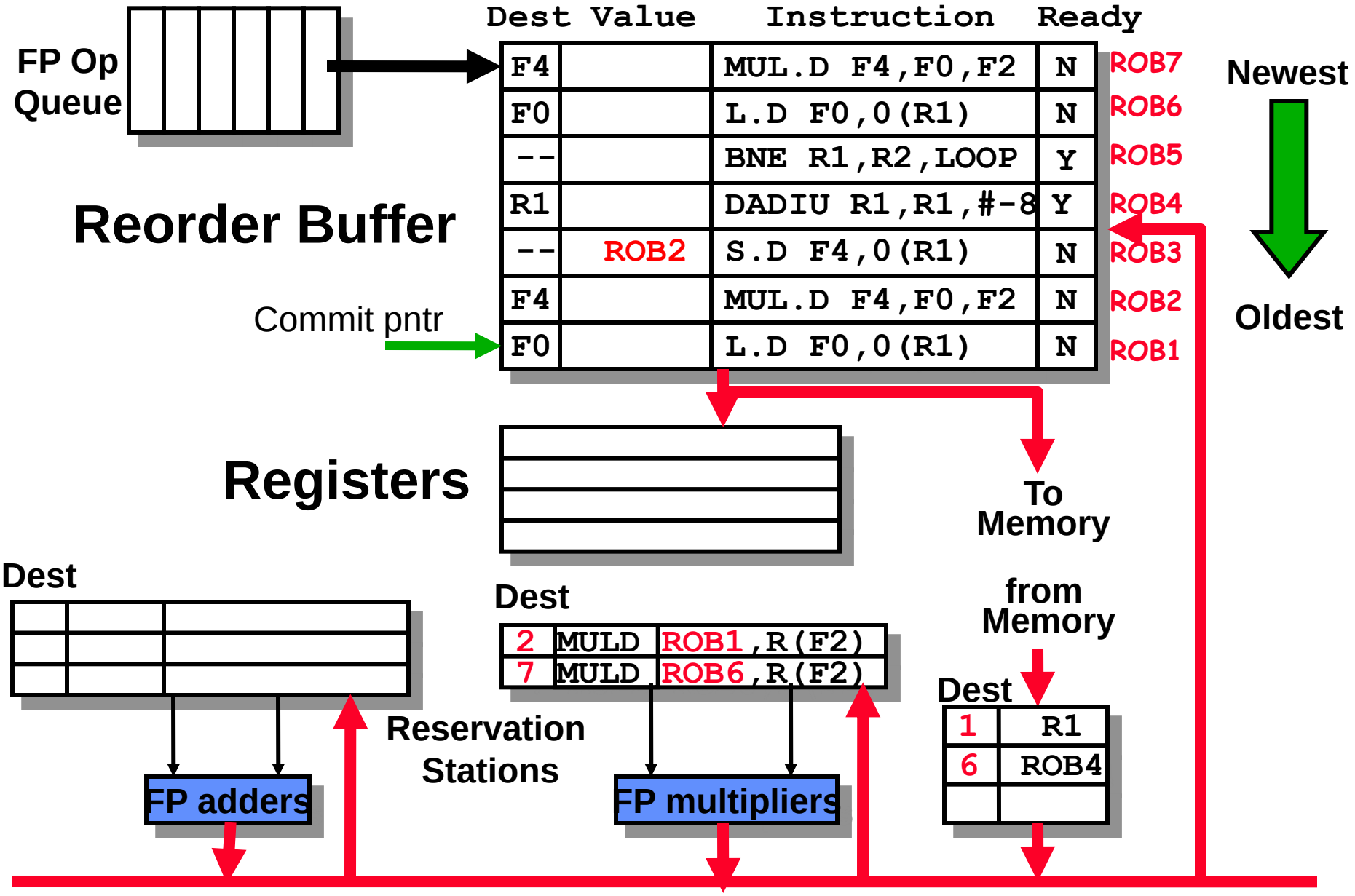
Tomasulo With Reorder buffer(5)



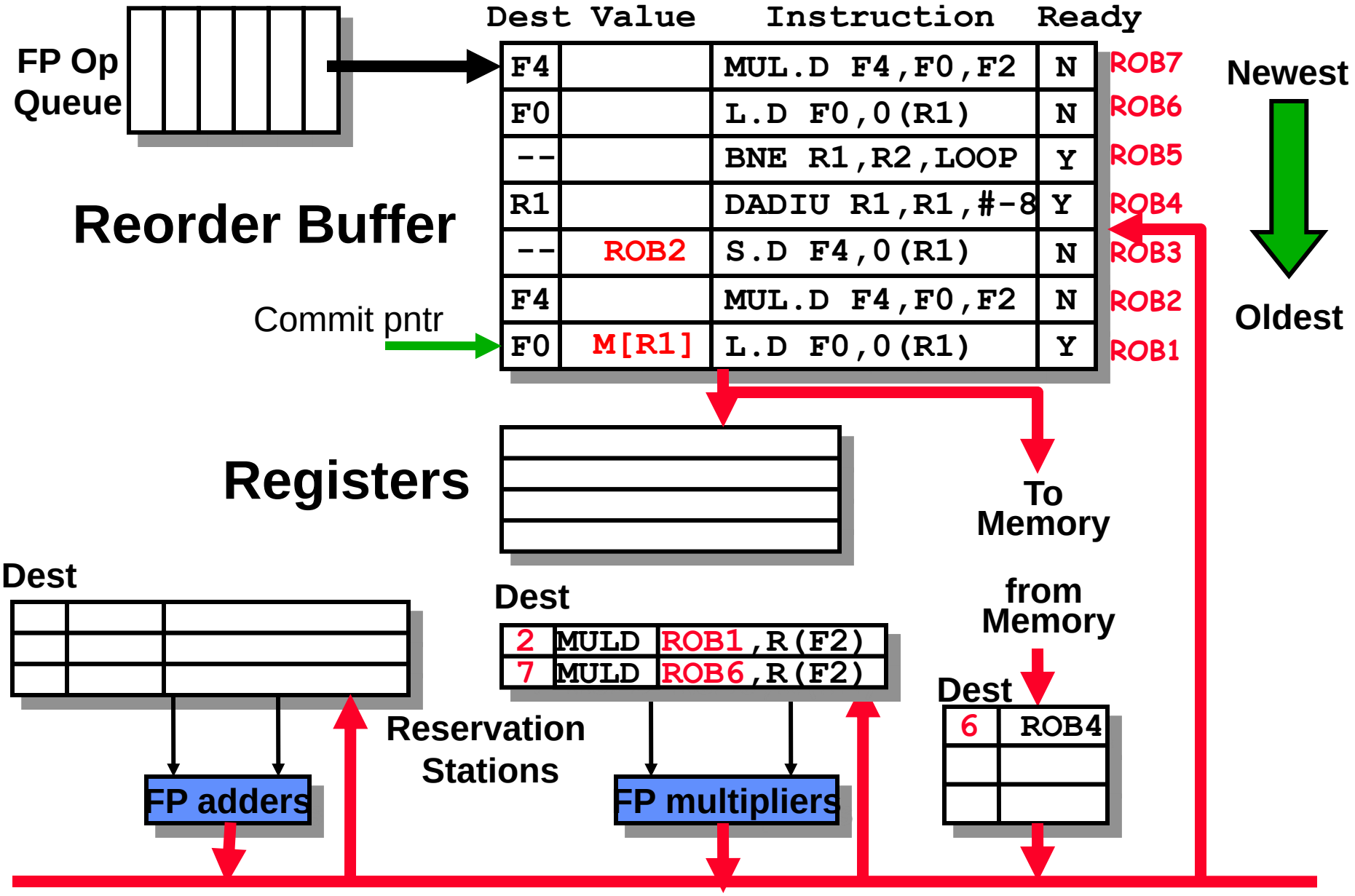
Tomasulo With Reorder buffer(6)



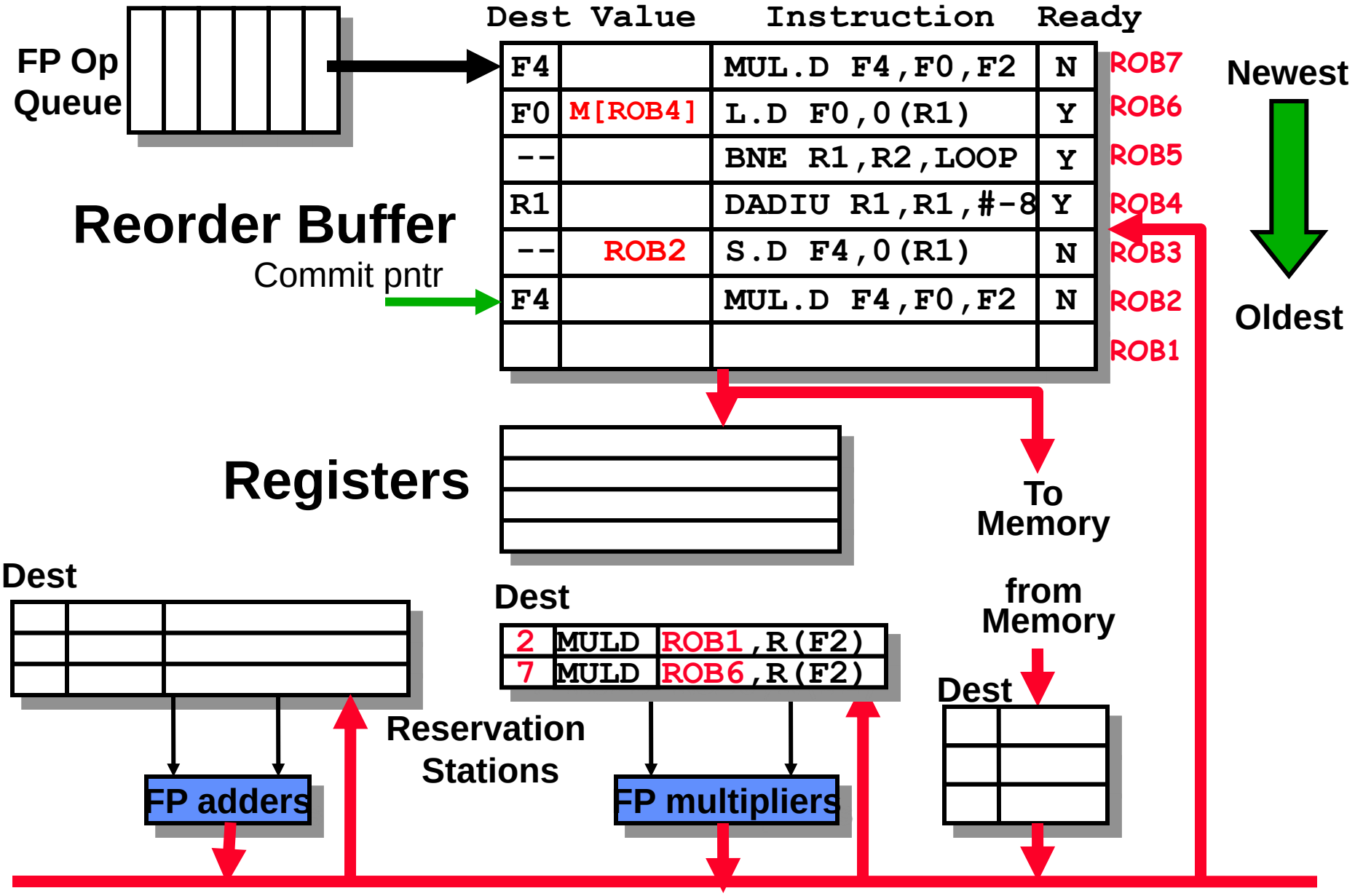
Tomasulo With Reorder buffer(7)



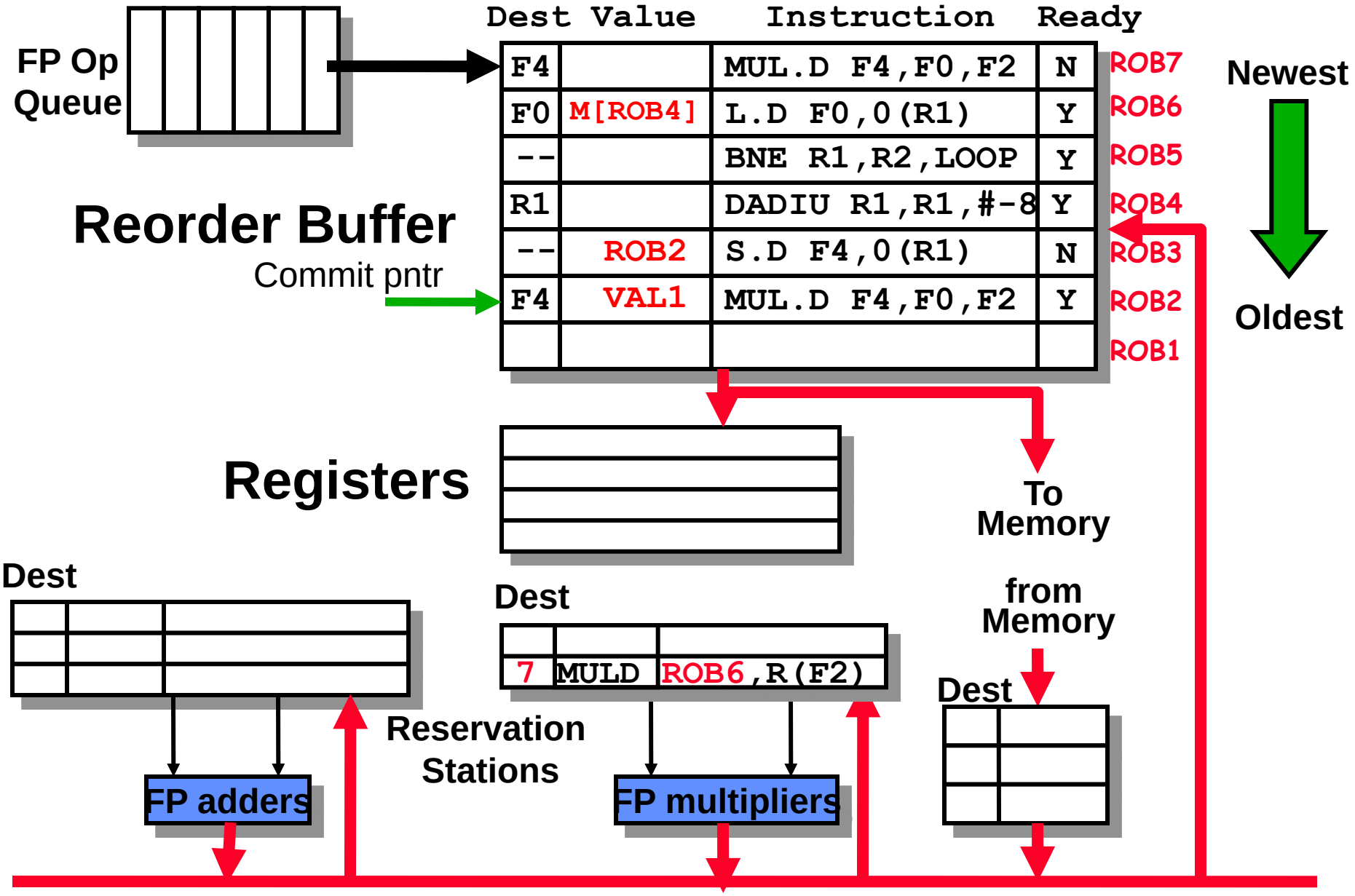
Tomasulo With Reorder buffer(8)



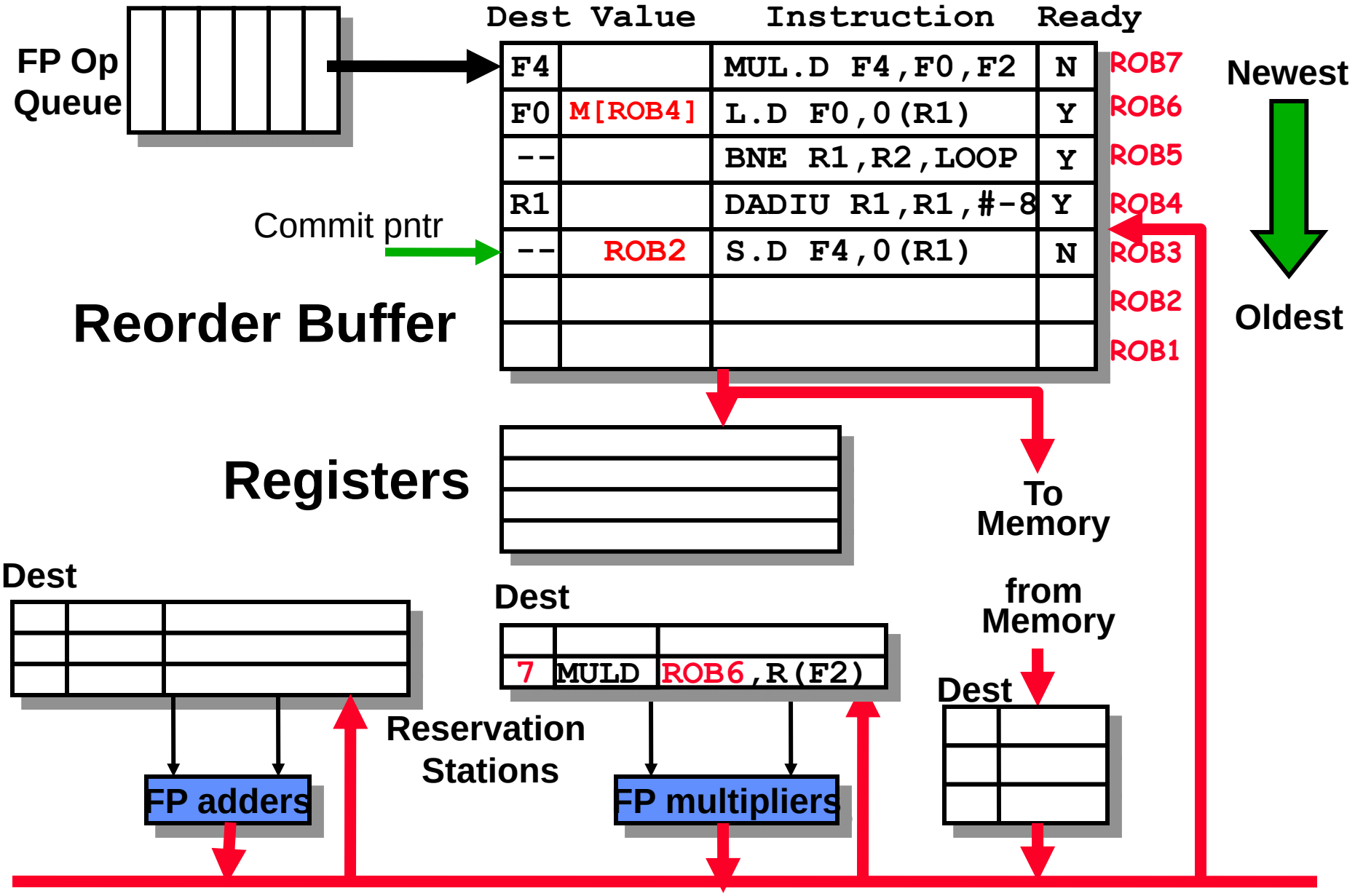
Tomasulo With Reorder buffer(9)



Tomasulo With Reorder buffer(12)



Tomasulo With Reorder buffer(13)



Tomasulo With Reorder buffer(13)

- Στο σημείο αυτό γίνεται commit το SD, ενώ καθώς αδειάζει ο ROB εισέρχονται καινούριες εντολές.
- Οι άλλες εντολές (DADDIU,BNE,LD) ενώ έχουν εκτελεστεί δεν έχουν γίνει ακόμα commit.
- Αν η πρόβλεψη taken για το BNE ήταν σωστή, οι επόμενες εντολές που έχουν εκτελεστεί γίνονται κανονικά commit.
- Αν ανακαλύψουμε όμως ότι ήταν λάθος, τότε απλά διαγράφουμε τις εντολές από το BNE και πάνω μέσα στον ROB. Η εκτέλεση συνεχίζεται από το σωστό σημείο.
  - Aggressive συστήματα θα μπορούσαν να κάνουν τη διαγραφή πρίν ο commit_pntr φτάσει στο BNE

Hazards

- Δεν υπάρχουν WAW, WAR hazards
- Για την αποφυγή RAW hazards ακολουθούμε τα εξής βήματα:
  - Αν κάποιο store προηγείται ενός load, καθυστερούμε το load μέχρι το store να υπολογίσει το address του
  - Αν load_address = store_address, τότε υπάρχει πιθανότητα RAW και :
    - Αν η τιμή του store είναι γνωστή, την περνάμε στο load
    - Αν η τιμή του store δεν είναι γνωστή, τότε το load χρησιμοποίει ως source το νούμερο του ROB που περιέχει το store
  - Αποστολή του request στη μνήμη

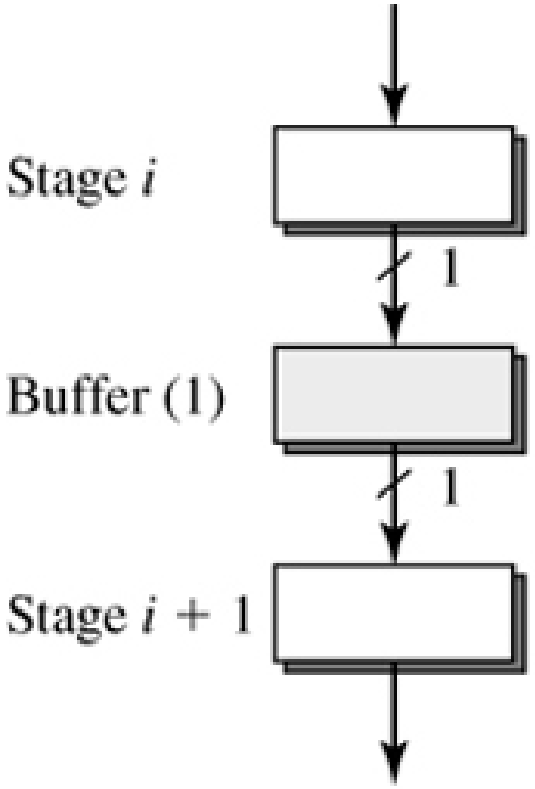
Out-of-order εκτέλεση:

επανεξέταση υπερβαθμωτών αρχιτεκτονικών...

Βαθμωτή αρχιτεκτονική αγωγού

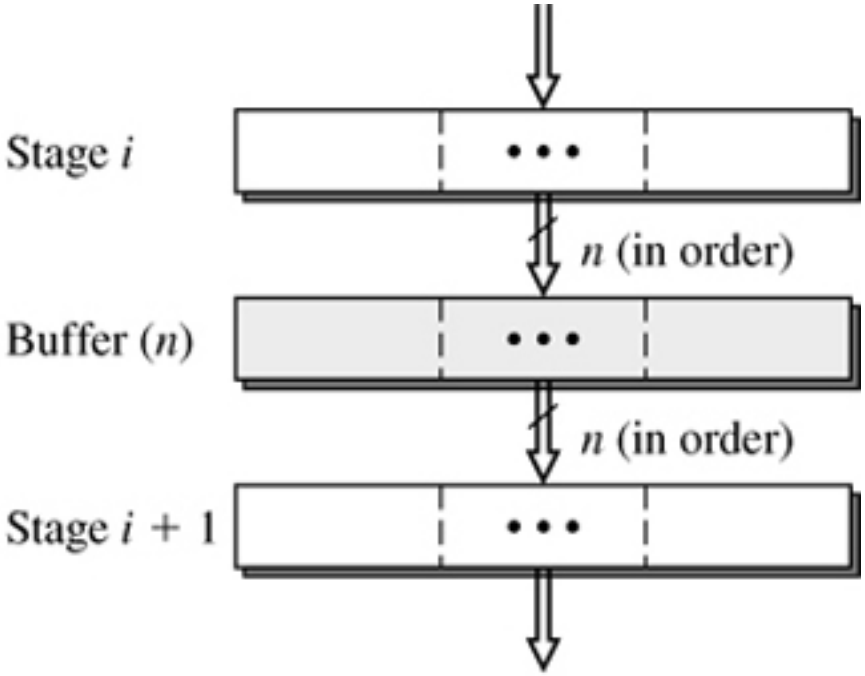
- μεταξύ δύο σταδίων του pipeline υπάρχει πάντα ένας ενδιάμεσος καταχωρητής δεδομένων (pipeline register ή buffer)

- στις βαθμωτές σωληνώσεις, κάθε buffer έχει μία μόνο θύρα εισόδου και μία εξόδου



Υπερβαθμωτή αρχιτεκτονική αγωγού

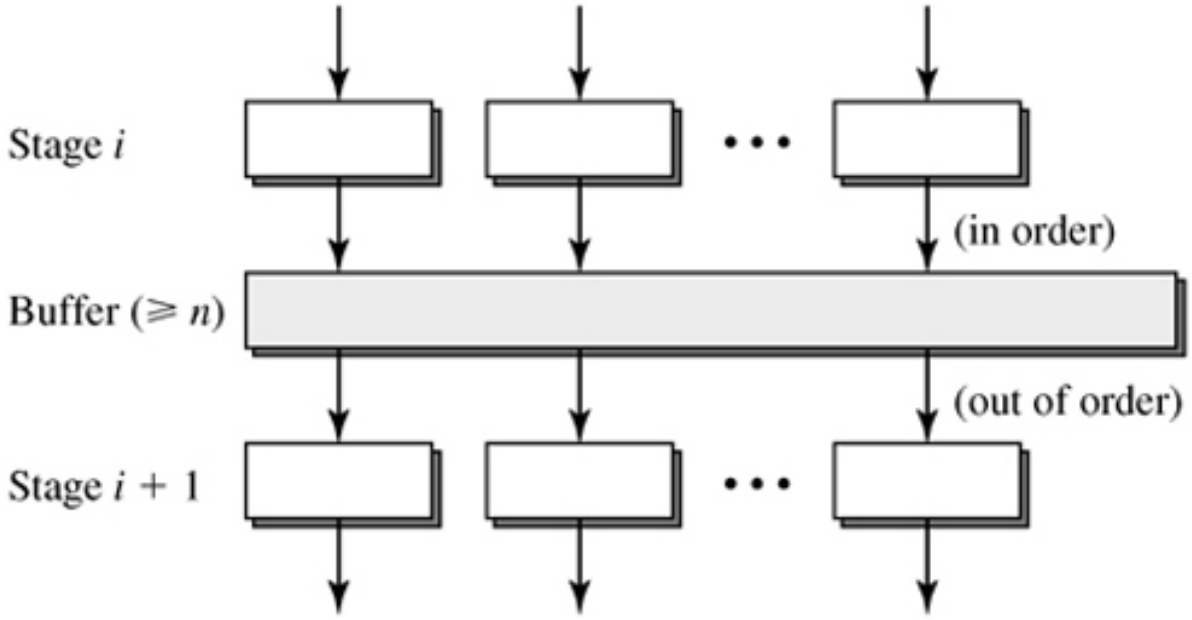
- κάθε buffer έχει πολλαπλές θύρες εισόδου και εξόδου, και αποτελείται απο πολλαπλούς ενδιάμεσους καταχωρητές
- κάθε ενδιάμεσος καταχωρητής μπορεί να λάβει δεδομένα μόνο από μία θύρα
- δεν υπάρχει δυνατότητα διακίνησης των δεδομένων μεταξύ των καταχωρητών του ίδιου buffer



Υπερβαθμωτή αρχιτεκτονική αγωγού – εκτέλεση εκτός σειράς (out-of-order)

- οι εντολές που καθυστερούν μέσα στη σωλήνωση (stall) παρακάμπτονται και συνεχίζεται η ροή εκτέλεσης με τις επόμενες εντολές

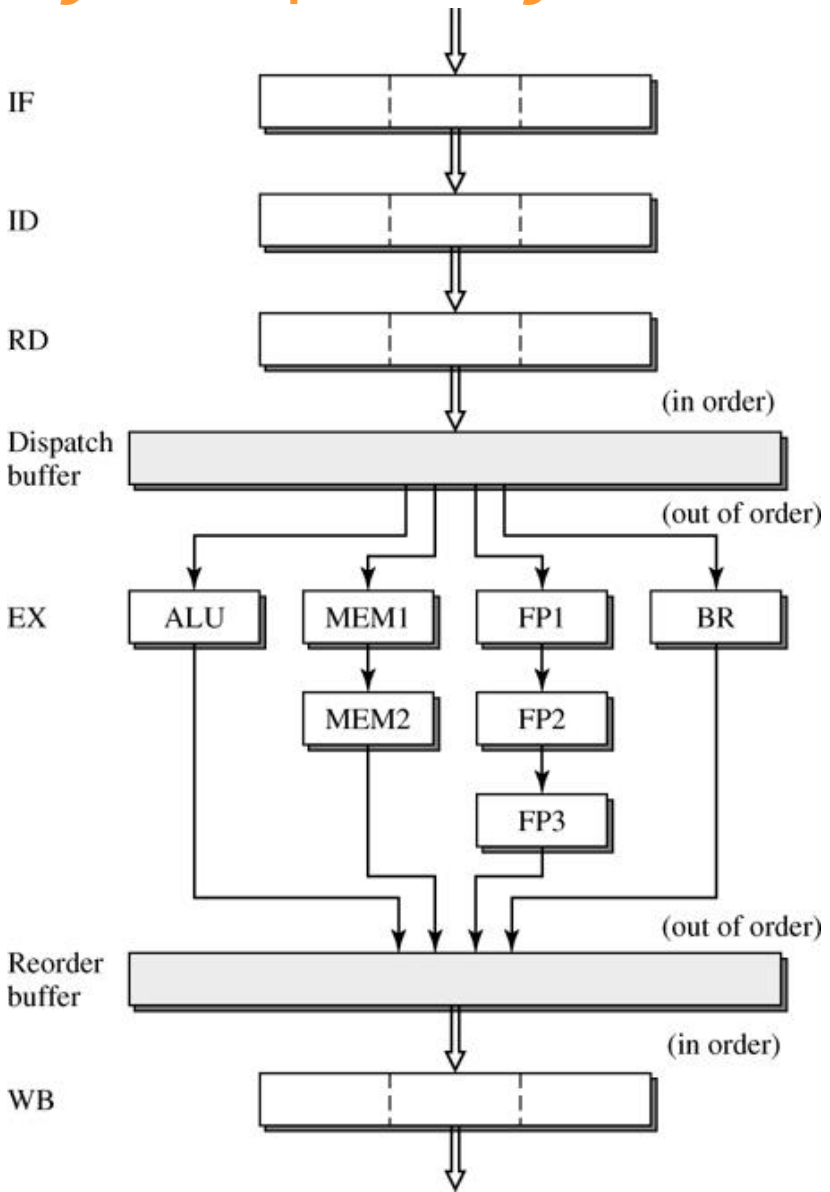
- out-of-order execution ή dynamic scheduling



Δυναμική αρχιτεκτονική αγωγού, με πολλαπλές, ετερογενείς σωληνώσεις

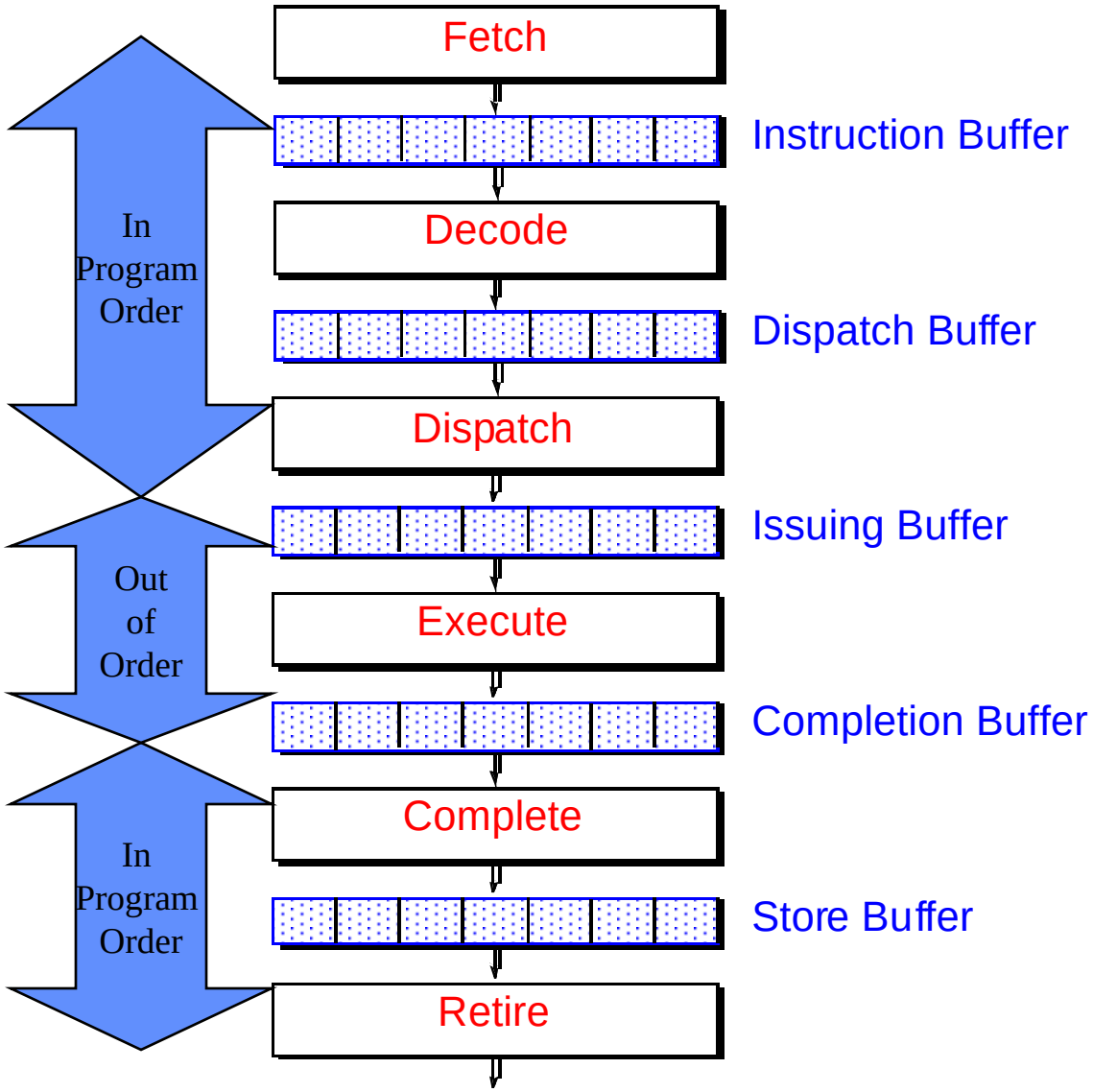
- ο dispatch buffer λαμβάνει τις εντολές εν σειρά, και τις κατανέμει στις λειτουργικές μονάδες εκτός σειράς

- όταν οι εντολές ολοκληρώσουν την εκτέλεσή τους, ο reorder buffer τις αναδιατάσσει, σύμφωνα με τη σειρά που υπαγορεύει το πρόγραμμα, προκειμένου να πιστοποιήσει τη σωστή ολοκλήρωσή τους



Γενική μορφή υπερβαθμωτής αρχιτεκτονικής αγωγού

- το στάδιο εκτέλεσης (execute) μπορεί να περιλαμβάνει πολλαπλές διαφορετικού τύπου σωληνώσεις, με διαφορετικό latency η κάθε μία
- αναγκαία τα στάδια dispatch και complete, για την αναδιάταξη και επαναφορά των εντολών σε σειρά
- παρεμβολή ενδιάμεσων buffers ανάμεσα σε διαδοχικά στάδια



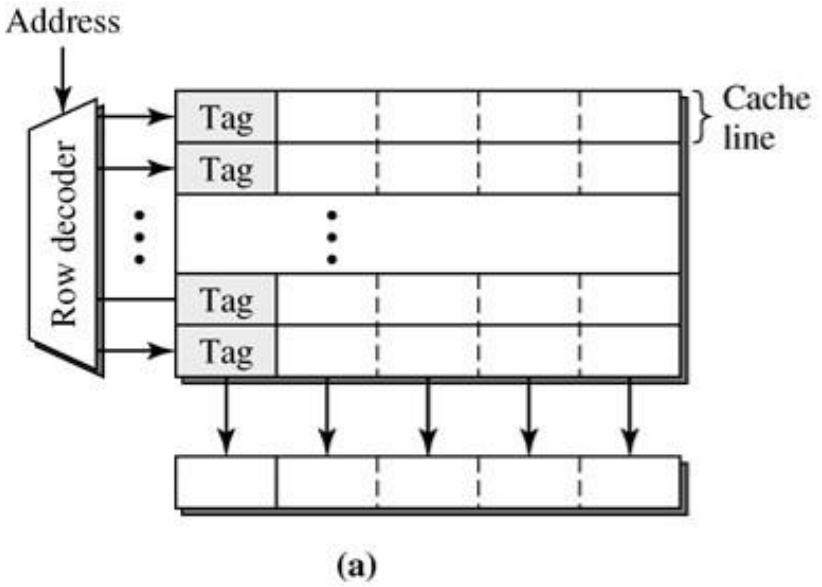
Superscalar Pipeline Design

- Instruction Fetching Issues
- Instruction Decoding Issues
- Instruction Dispatching Issues
- Instruction Execution Issues
- Instruction Completion & Retiring Issues

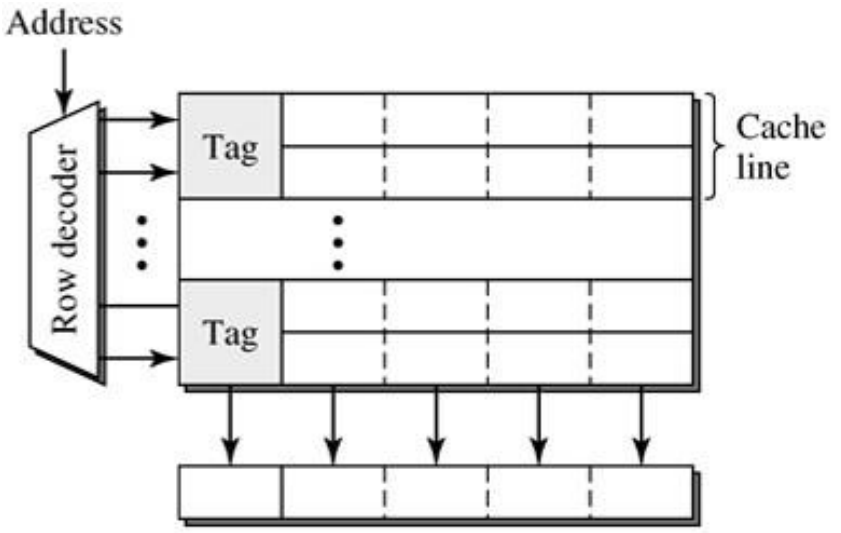
Ανάγνωση εντολής (Instruction Fetch)

- Σε μία αρχιτεκτονική πλάτους *s*, πρέπει διαβάζονται *s* εντολές σε κάθε κύκλο μηχανής από την κρυφή μνήμη εντολών (I-cache)

- Σε κάθε προσπέλαση της I-cache διαβάζεται μία ολόκληρη γραμμή (*s* εντολές ανά γραμμή): 1 cache line ανά γραμμή της I-cache (a). Θα μπορούσε μία cache line να επεκτείνεται σε περισσότερες της μίας γραμμής (b).



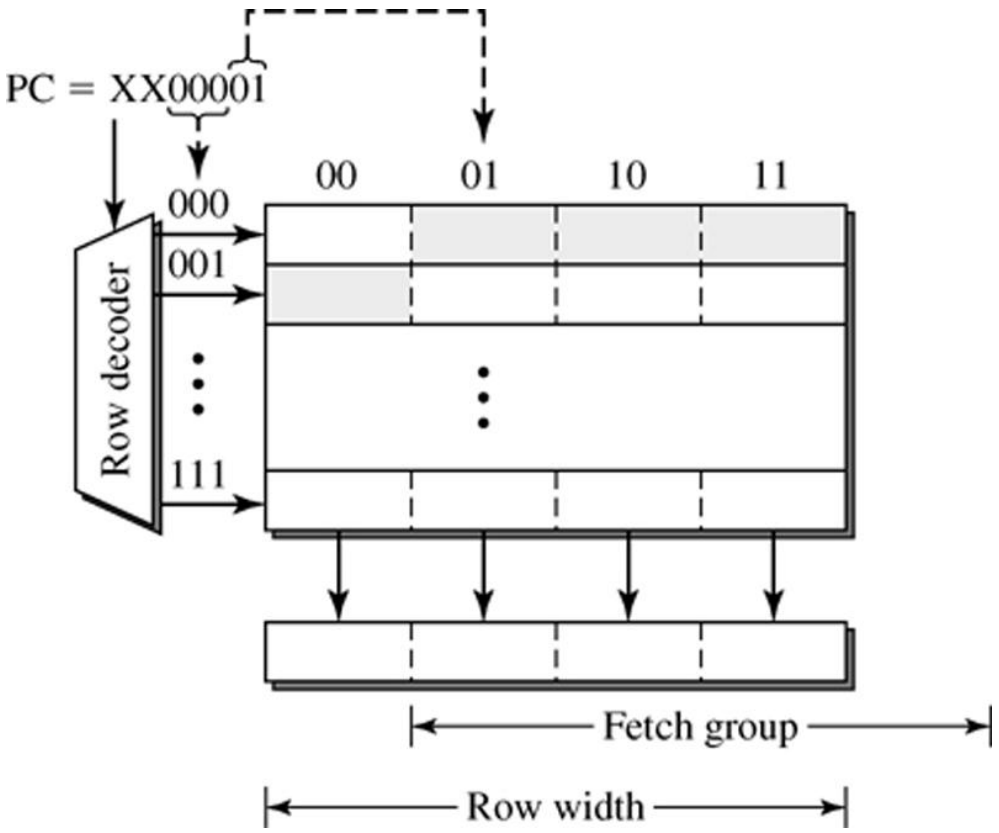
(a)



(b)

Ανάγνωση εντολής (Instruction Fetch)

- Για την ανάγνωση *s* εντολών σε κάθε κύκλο μηχανής θα πρέπει:
  - Οι *s* εντολές (που ανήκουν στο ίδιο fetch group) να είναι ευθυγραμμισμένες (aligned) στην I-cache



Ανάγνωση εντολής (Instruction Fetch)

- Εναλλακτικά, υπάρχουν μηχανισμοί ελέγχου της ροής των εντολών, που μπορούν να φροντίσουν για την ευθυγράμμιση των εντολών που ανήκουν στο ίδιο fetch group
  - Software (compiler static alignment)
  - Hardware (run time technique)

Αποκωδικοποίηση εντολών (Instruction Decode)

- Αναγνώριση των όποιων εξαρτήσεων μεταξύ εντολών – προώθηση των μη εξαρτημένων εντολών στο επόμενο στάδιο
- Αναγνώριση των εντολών άλματος
- Αποτελεί το κρισιμότερο στάδιο για την επίδοση ολόκληρης της αρχιτεκτονικής αγωγού
- Ακόμη μεγαλύτερη δυσκολία στις CISC εντολές: κάθε εντολή μεταφράζεται σε περισσότερες μικρο-εντολές (1,5-2 μops / CISC εντολή για την Intel)

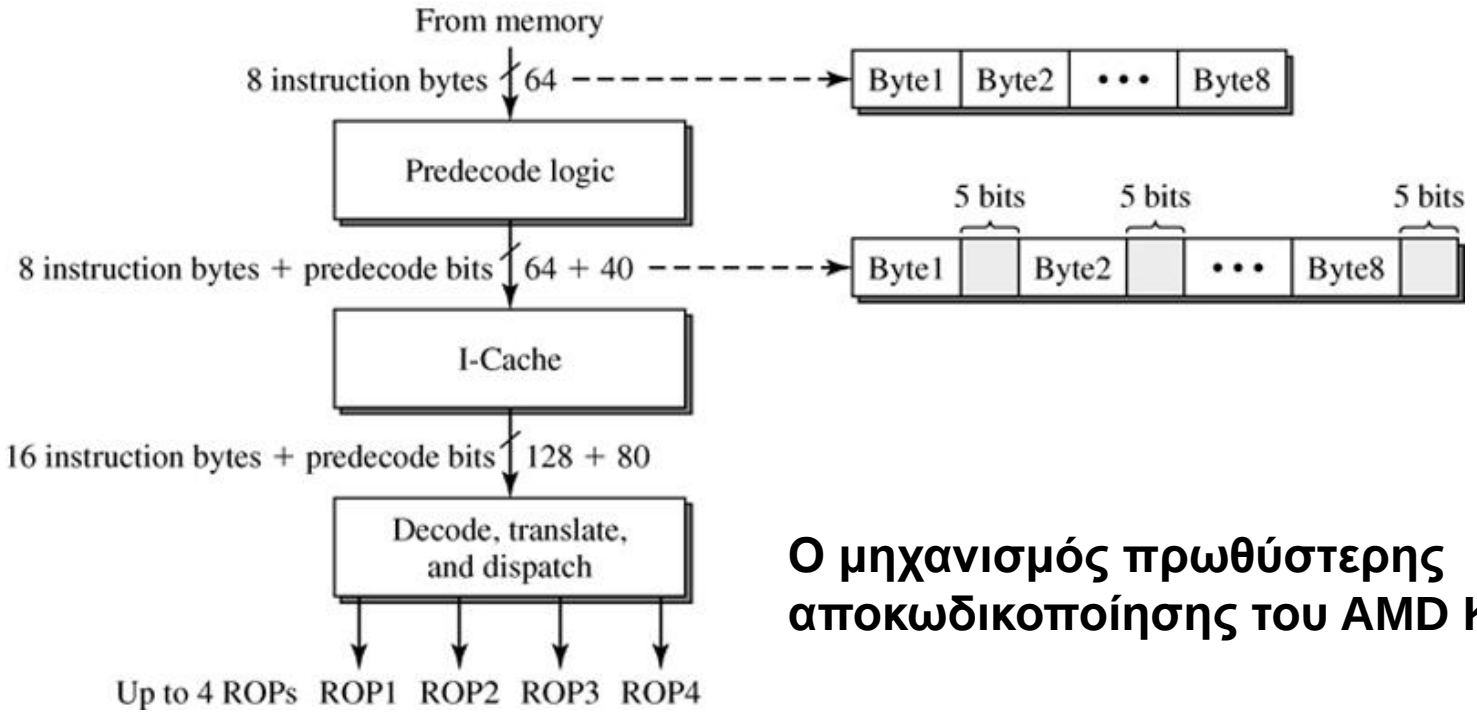
Αποκωδικοποίηση εντολών (Instruction Decode)

- Η αποκωδικοποίηση των εντολών CISC σε παράλληλες αρχιτεκτονικές αγωγού ή οι πολύ πλατιές σωληνώσεις
  - Απαιτεί πολλαπλά στάδια αγωγού
  - Αυξάνει το branch penalty

- *Πώς μπορεί να αποφευχθεί η αύξηση του βάθους του τμήματος αποκωδικοποίησης στη σωλήνωση;*

Αποκωδικοποίηση εντολών (Instruction Decode)

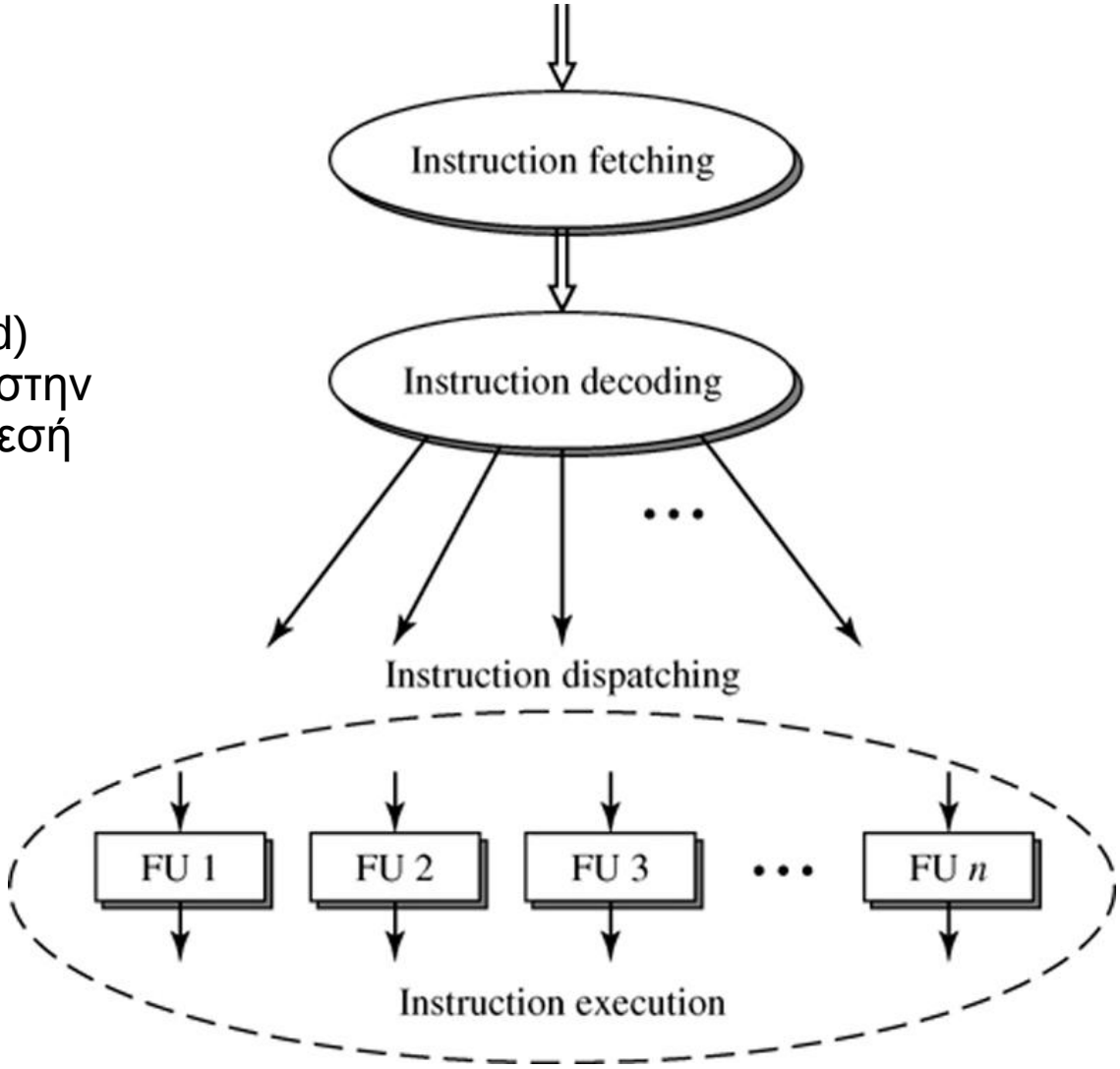
- Λύση:
  - Μερική αποκωδικοποίηση των εντολών πριν την είσοδό τους στην I-cache (predecoding)
- Αύξηση του I-cache miss penalty
- Αύξηση του μεγέθους της I-cache (για να συμπεριληφθούν τα predecoded bits)



Ο μηχανισμός πρωθύστερης αποκωδικοποίησης του AMD K5

Διανομή των εντολών – Instruction Dispatch

- Το στάδιο αυτό αναλαμβάνει τη μεταγωγή από την κεντρική (centralized) διαχείριση εντολών στην κατανεμημένη εκτέλεσή τους

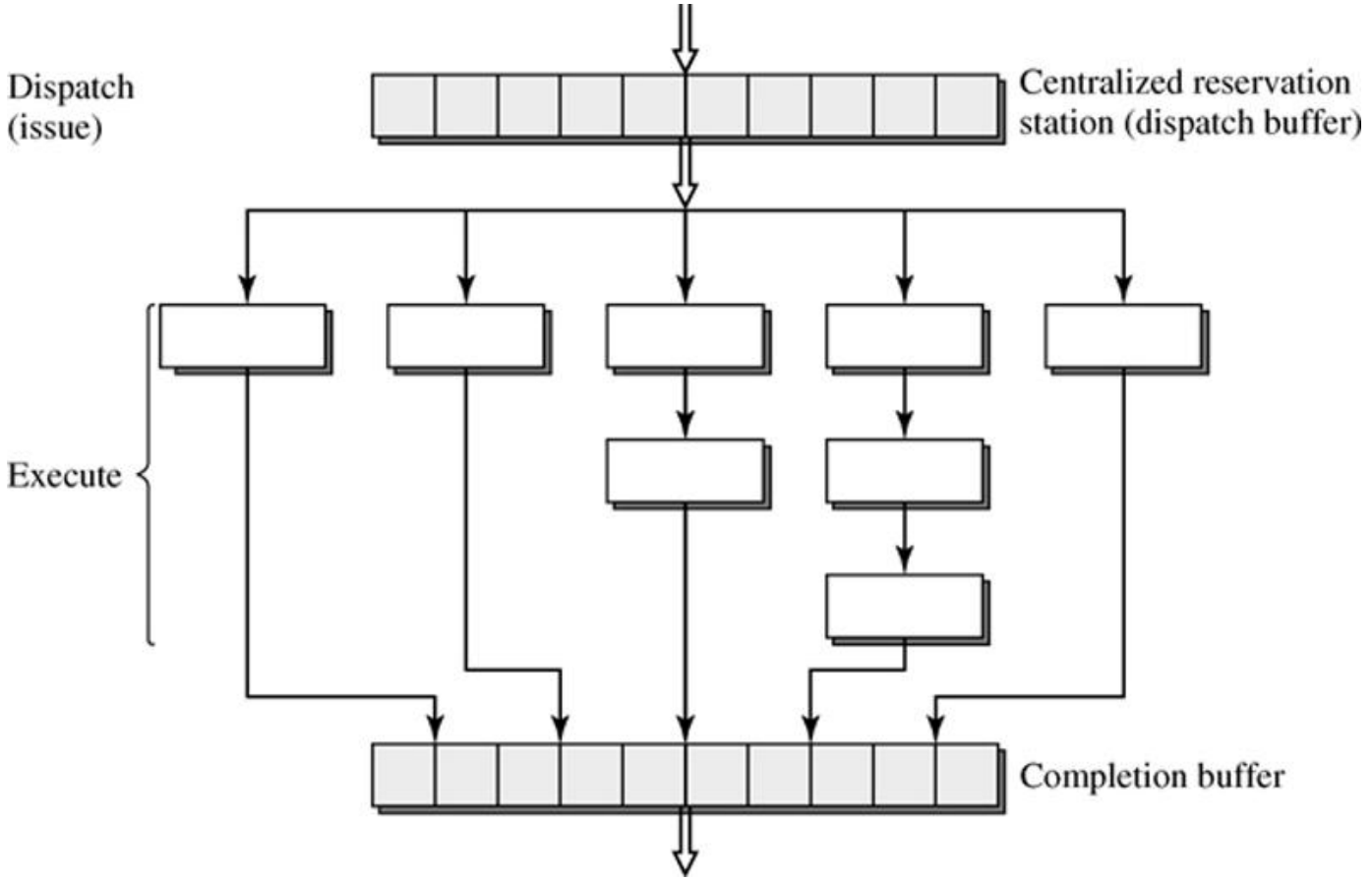


Διανομή των εντολών – Instruction Dispatch

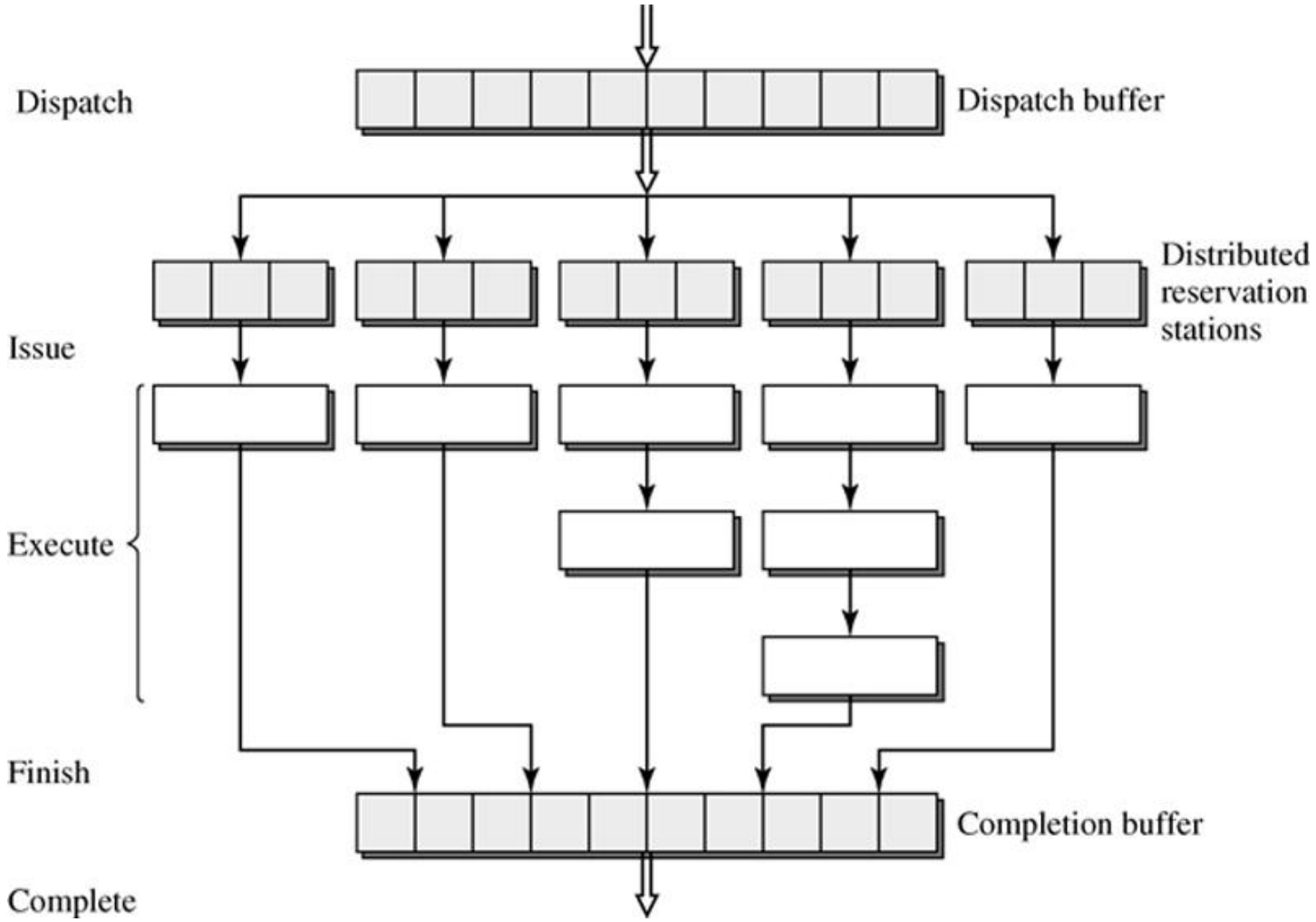
- Reservation station: προσωρινός καταχωρητής αποθήκευσης των αποκωδικοποιημένων εντολών που δεν έχουν διαθέσιμα όλα τα ορίσματά τους
  - Κεντρικός καταχωρητής (centralized reservation station)
    - » Για παράλληλες σωληνώσεις
  - Κατανεμημένοι καταχωρητές (distributed reservation station)
    - » Για ετερογενείς σωληνώσεις

Centralized reservation station

- Ενοποίηση των σταδίων dispatch και issue



Distributed reservation station



Εκτέλεση εντολών – Instruction Execution

- Σύγχρονες τάσεις:
  - Πολλές παράλληλες σωληνώσεις (δύσκολη η out-of-order εκτέλεση με bypassing εντολών)
  - Διαφοροποιημένες μεταξύ τους σωληνώσεις
  - Βαθιές σωληνώσεις

- Συνήθης καταμερισμός (δεν ακολουθεί τη στατιστική αναλογία των προγραμμάτων σε τύπους εντολών):
  - 4 μονάδες ALU
  - 1 μονάδα διακλάδωσης (μπορεί να εκτελέσει θεωρητικά (speculatively) > 1 εντολές διακλάδωσης
  - 1 μονάδα ανάγνωσης/εγγραφής στη μνήμη (πολύ πολύπλοκη η υλοποίηση μνημών πολλαπλών εισόδων-εξόδων, μόνο με πολλαπλά banks)
  - Περισσότερες ειδικευμένες (και πιο αποδοτικές στην επίδοση) λειτουργικές μονάδες

Ολοκλήρωση και Αποδέσμευση εντολών – Instruction Completion and Retiring

- Instruction Completion: ενημέρωση της κατάστασης του μηχανήματος (machine state update)
- Instruction Retiring: ενημέρωση της μνήμης (memory state update)
- Αν η εντολή δεν περιλαμβάνει ενημέρωση της μνήμης, μετά το στάδιο ολοκλήρωσης, η εντολή αποδεσμεύεται
- Τα στάδια ολοκλήρωσης και αποδέσμευσης των εντολών πρέπει να εκτελέσουν τις εντολές εν σειρά
  - συνεπές arch. state
  - συνεπής κατάσταση μνήμης

Ολοκλήρωση και Αποδέσμευση εντολών – Instruction Completion and Retiring

- Τα στάδια εντολών:
  - Fetch
  - Decode
  - Dispatch
  - Issue
  - Execute
  - Finish
  - Complete
  - Retire