Αρχιτεκτονική Υπολογιστών

Κεφάλαιο #4 (β)

Multi-Cycle & Pipelined Datapath Design

Διονύσης Πνευματικάτος

pnevmati@cslab.ece.ntua.gr

5ο εξάμηνο ΣΗΜΜΥ – Ακαδημαϊκό Έτος: 2019-20

Τμήμα 3 (ΠΑΠΑΔ-Ω)

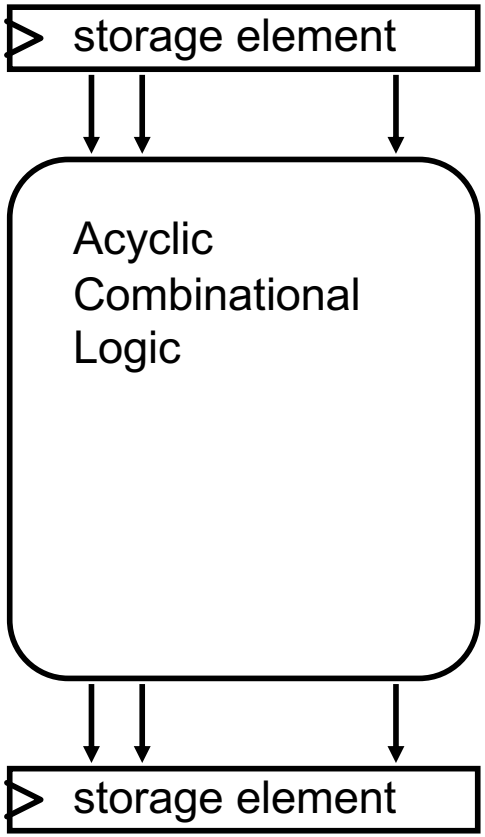
http://www.cslab.ece.ntua.gr/courses/comparch/

Ζητήματα απόδοσης #1

- CPI = 1 αλλά ένας μεγάλος κύκλος
- Η μεγαλύτερη καθυστέρηση καθορίζει την περίοδο ρολογιού
  - Κρίσιμη διαδρομή (critical path): εντολή load
  - Μνήμη εντολών → αρχείο καταχωρητών → ALU → μνήμη δεδομένων → αρχείο καταχωρητών
- Δεν είναι εφικτή διαφορετική περίοδος για διαφορετικές εντολές
- Παραβιάζει τη σχεδιαστική αρχή: «Κάνε τη συνηθισμένη περίπτωση γρήγορη»
- Προσπάθεια #1: Θα βελτιώσουμε την απόδοση με εκτέλεση σε *πολλούς* κύκλους

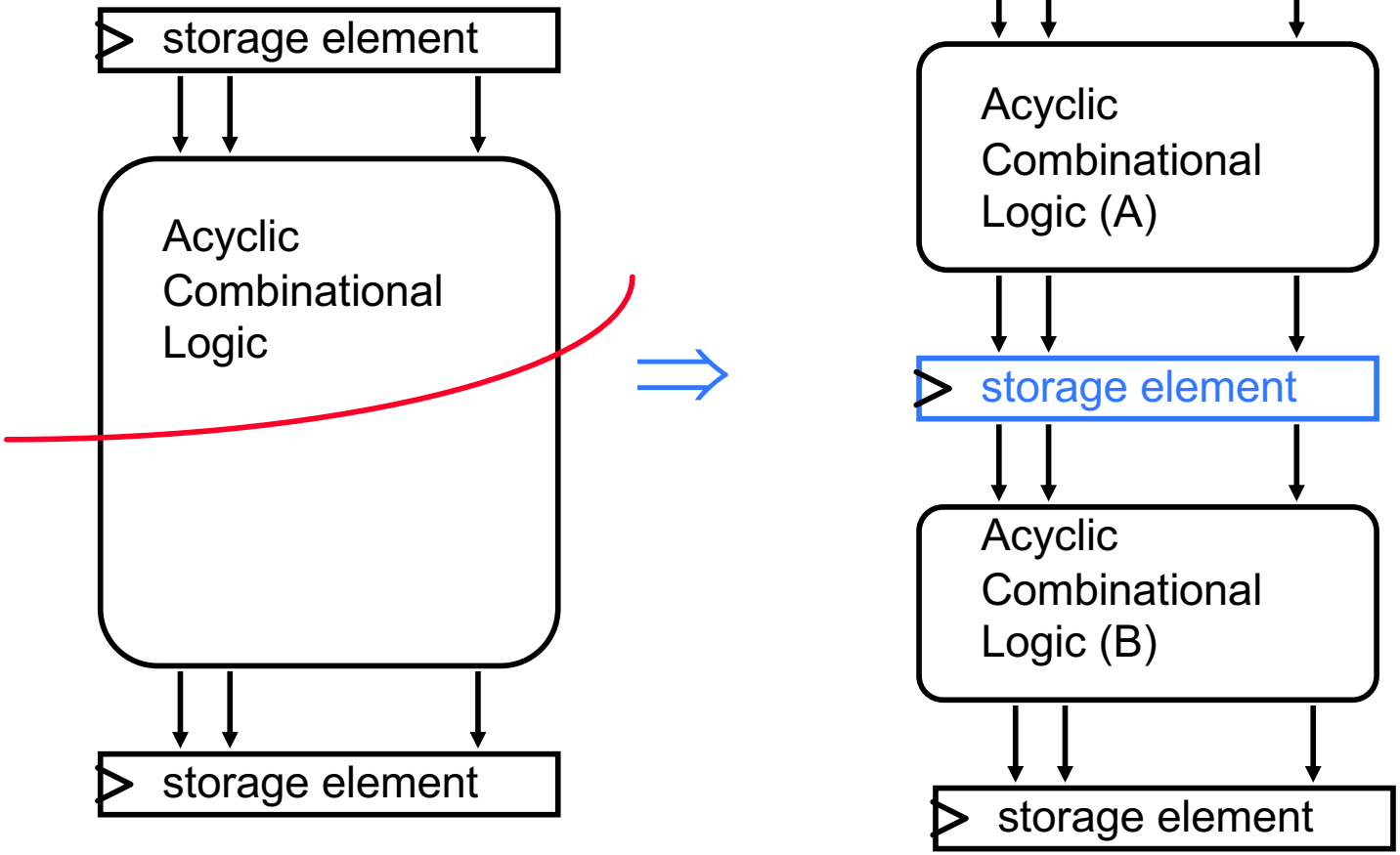
Reducing Cycle Time

- Cut combinational dependency graph and insert register / latch
- Do same work in two fast cycles, rather than one slow one
- May be able to short-circuit path and remove some components for some instructions!



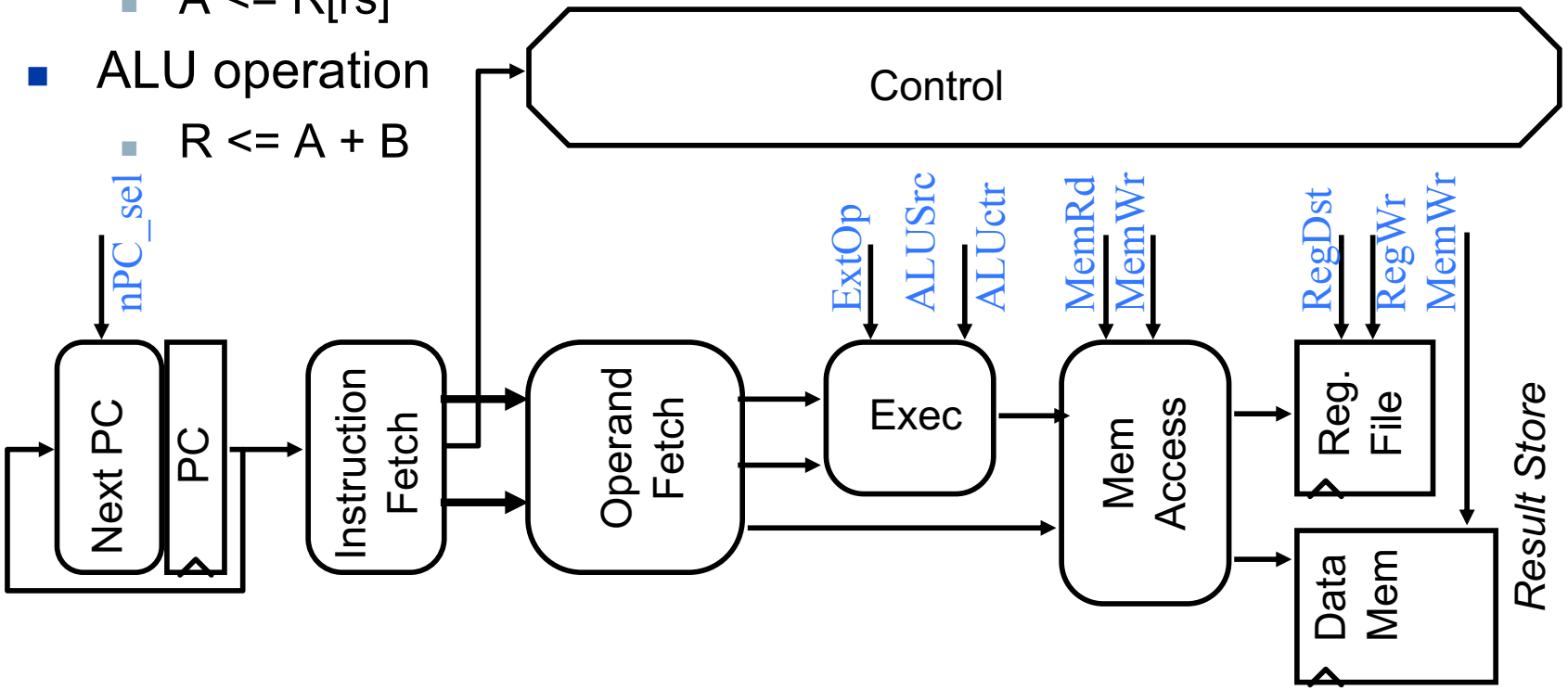
Μείωση της διάρκειας του κύκλου

- Cut combinational dependency graph and insert register / latch
- Do same work in two fast cycles, rather than one slow one
- May be able to short-circuit path and remove some components for some instructions!



Βασικά όρια στην διάρκεια του κύκλου

- Next address logic
  - PC <= branch ? PC + offset : PC + 4
- Instruction Fetch
  - InstructionReg <= Mem[PC]
- Register Access
  - A <= R[rs]
- ALU operation
  - R <= A + B

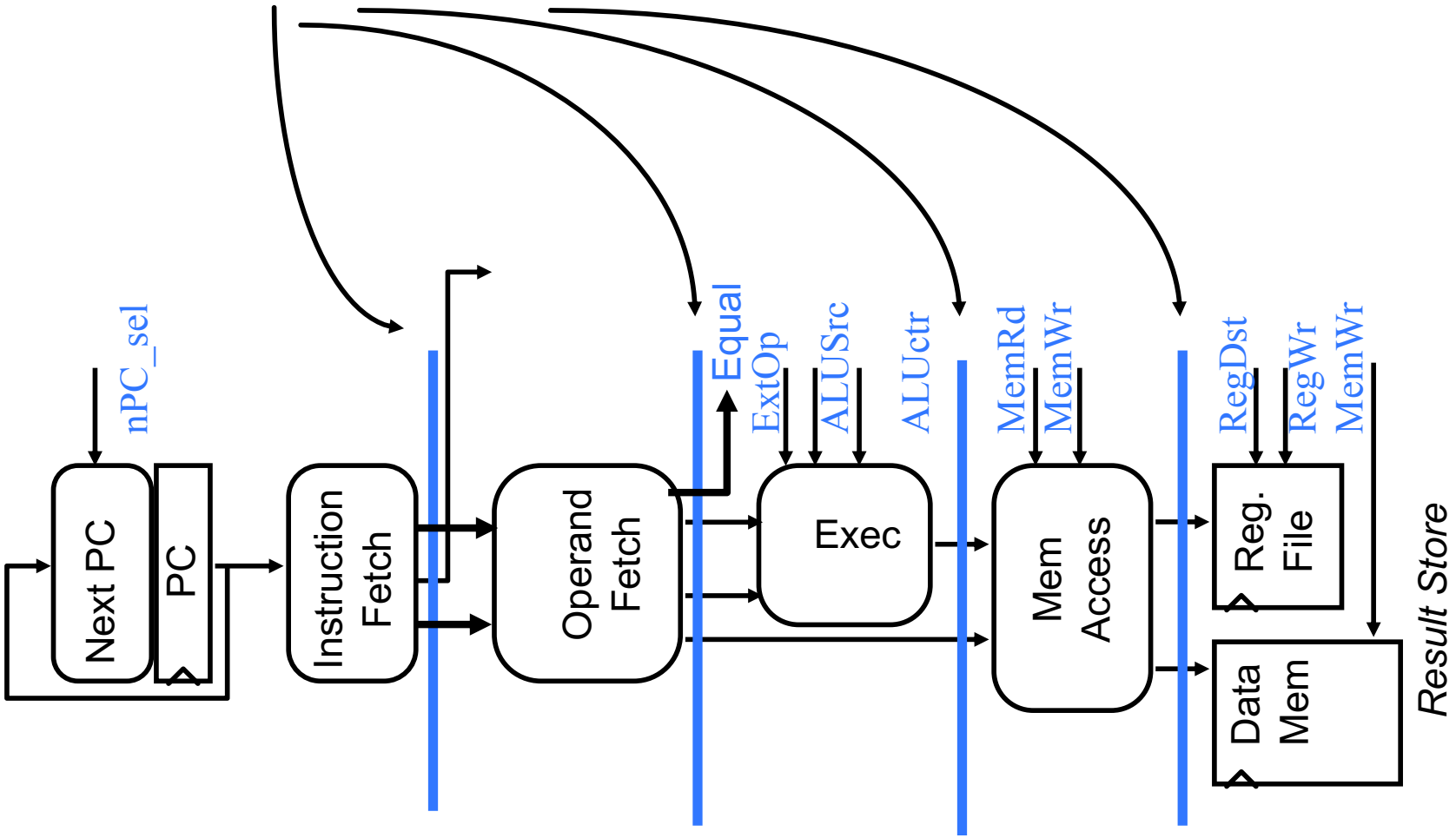


Υλοποίηση πολλαπλών κύκλων

- Διαιρούμε την εκτέλεση της κάθε εντολής σε βήματα ανάλογα με τον αριθμό των functional units που χρειάζεται
  - Κάθε βήμα και ένας ξεχωριστός παλμός ρολογιού
- Όταν έχουμε multicycle υλοποίηση, μπορούμε το ίδιο functional unit να το χρησιμοποιήσουμε πολλές φορές στην ίδια εντολή, σε διαφορετικούς όμως κύκλους (οικονομία hardware)
  - Οι εντολές διαρκούν μεταβλητό αριθμό κύκλων, άρα μπορούμε να κάνουμε την συνηθισμένη περίπτωση πιο γρήγορη.

Κατάτμηση του Datapath ενός κύκλου

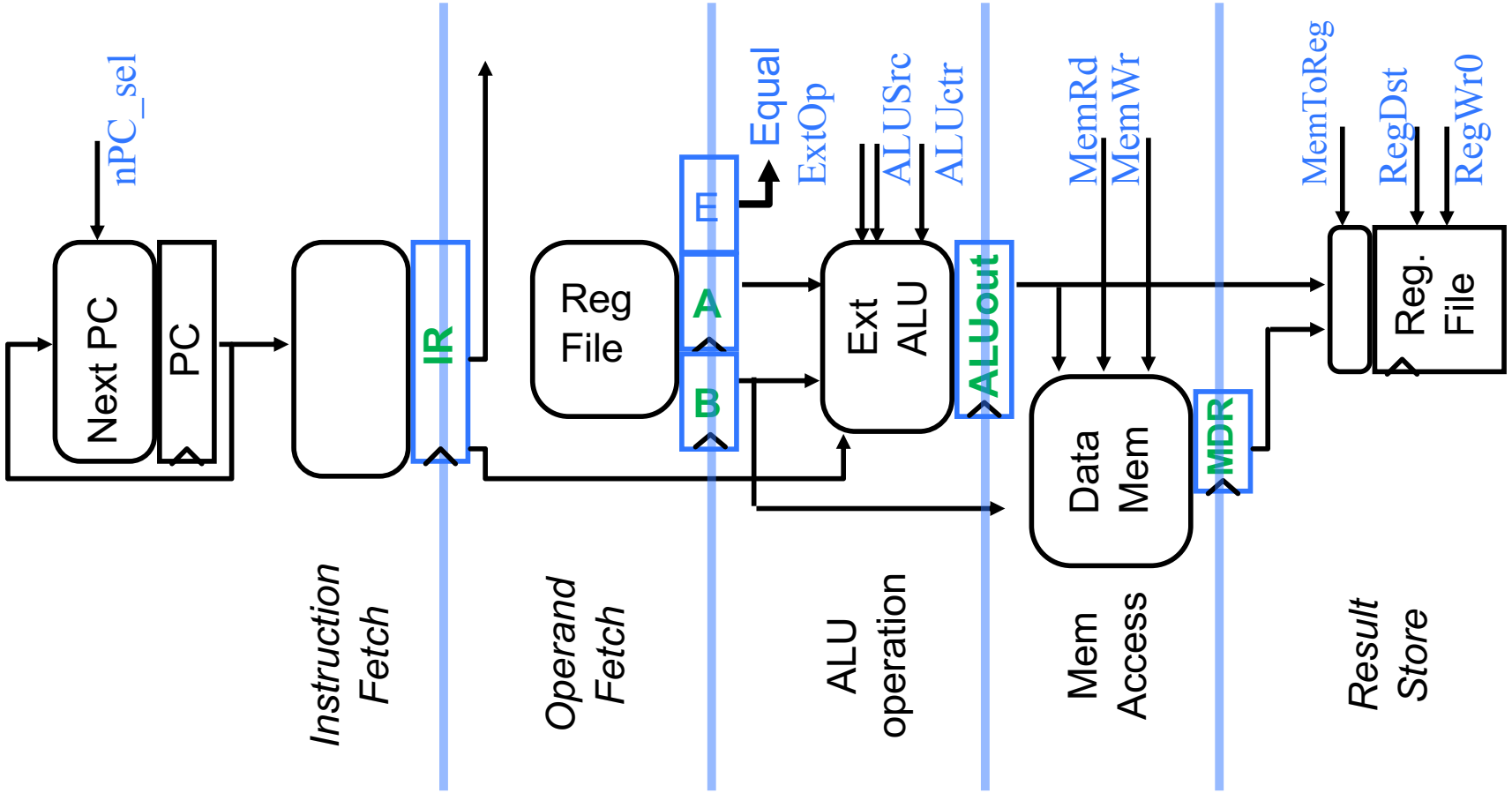
- Add registers between smallest steps



- Add enables on all registers

Multi-Cycle Datapath

- Ποιο είναι το κρίσιμο (μακρύτερο) μονοπάτι;

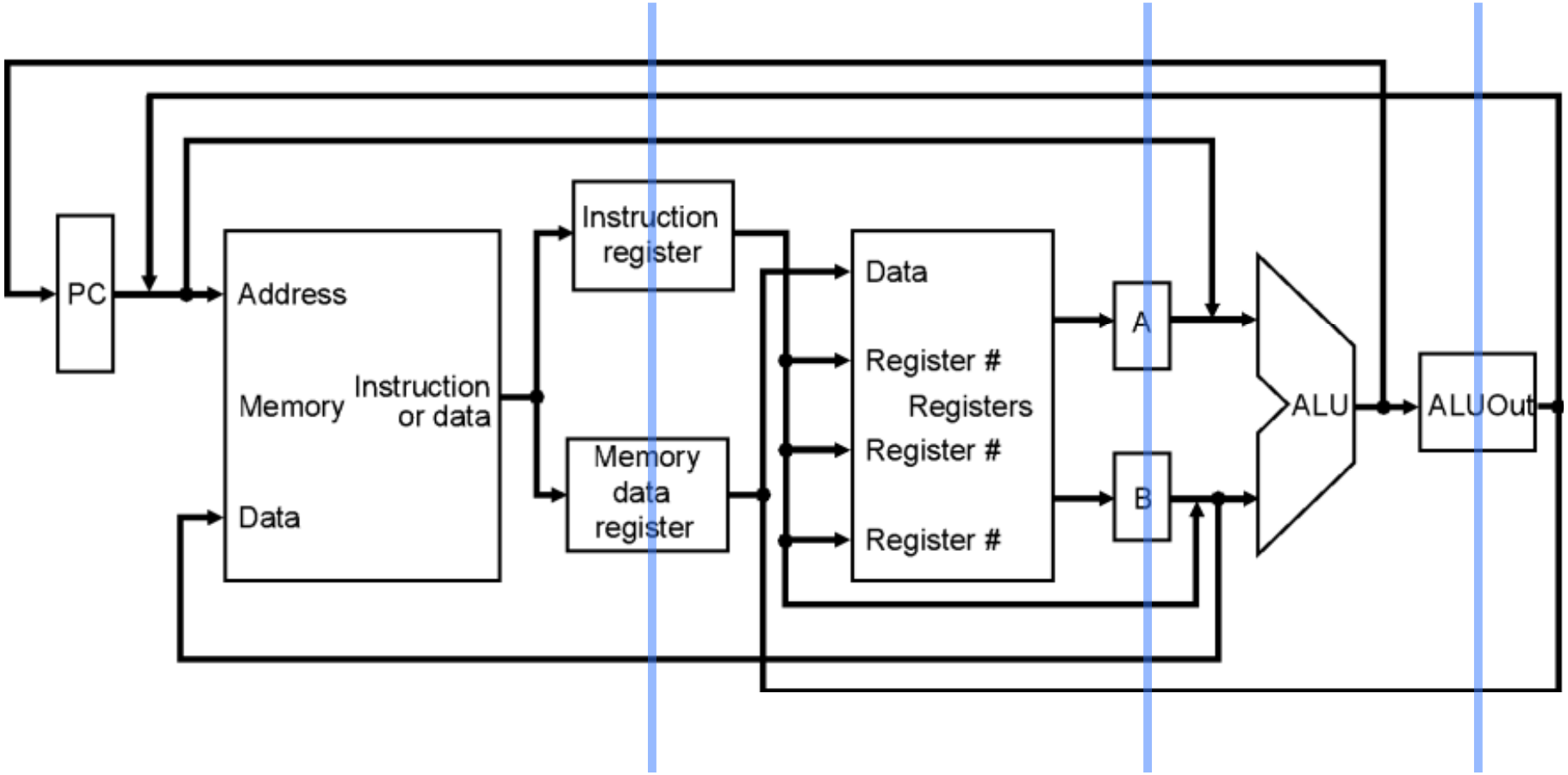


Υλοποίηση του Datapath πολλαπλών κύκλων

- Χρησιμοποιούμε την ίδια memory unit τόσο για instructions όσο και για data
- Χρησιμοποιούμε την ίδια ALU (αντί για μια ALU και δύο αθροιστές PC+4 και PC+4+address_offset)
- Μετά από κάθε functional unit υπάρχουν καταχωρητές που κρατάνε το αποτέλεσμα μέχρις ότου το πάρει το επόμενο functional unit (στον επόμενο κύκλο)

Υλοποίηση του Datapath πολλαπλών κύκλων

- Επιπλέον καταχωρητές: IR, MDR, A, B και ALUOut

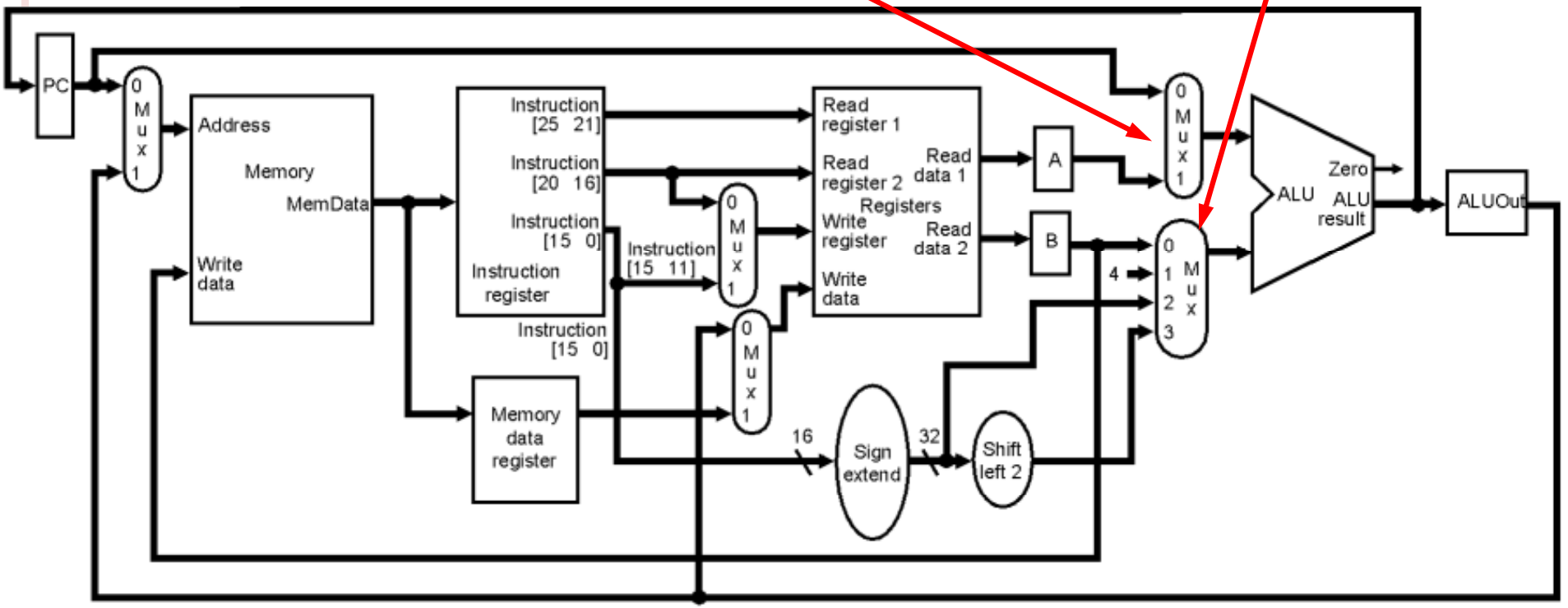


- Που είναι η 5η βαθμίδα;

Υλοποίηση του Datapath πολλαπλών κύκλων

Επιλογή μεταξύ PC (για εντολή branch, PC+4) και A (για R-Type)

Επιλογή μεταξύ 4 (PC+4), B (R-Type), sign_extend offset για I-Type (lw, sw) και branch offset



Βήματα εκτέλεσης εντολών σε πολλαπλούς κύκλους #1

1. Instruction Fetch

- «Φέρε την εντολή από τη μνήμη και υπολόγισε τα διεύθυνση ανάκλησης για την επόμενη εντολή»
  - IR = Memory[PC];
  - PC = PC + 4;

2. Instruction decode and register fetch (RF read)

- «Διάβασε τους καταχωρητές rs και rt και αποθήκευσέ τους στους A και B αντίστοιχα»
  - A = Reg[IR[25-21]];
  - B = Reg[IR[20-16]];
  - ALUOut=PC + (sign-extend (IR[15-0] << 2) ;

... #2

2. Instruction decode and register fetch (reg. File read)

- Οι A και B «γεμίζουν» σε κάθε κύκλο! Πάντα ο IR περιέχει την εντολή από την αρχή μέχρι το τέλος!
- Στο βήμα αυτό υπολογίζεται και η διεύθυνση «πιθανού» άλματος και αποθηκεύεται στο καταχωρητή ALUOut (αν πρόκειται για εντολή branch)
- Οι δύο παραπάνω λειτουργίες γίνονται ταυτόχρονα

... #3

3. Execution, memory address computation or branch completion

Εδώ για πρώτη φόρα, παίζει ρόλο τι είδους εντολή έχουμε

a) Memory Reference:

ALUOut=A+sign-extend (IR[15-0]);

b) Arithmetic-Logical:

ALUOut=A op B;

c) Branch:

If (A==B) PC = ALUOut;

d) Jump:

PC = PC[31-28] || (IR[25-0] <<2);

... #4

4. Memory Access or R-Type instruction completion

α) Memory Reference:

MDR = Memory [ALUOut];

ή

Memory [ALUOut] = B;

«διάβασε από τη διεύθυνση που έχει σχηματιστεί στον ALUOut και αποθήκευσε στον MDR (load)»

ή

«διάβασε το B (που πάντα έχει τον destination reg rt) και αποθήκευσέ το στη μνήμη με δνση ALUOut

β) Arithmetic-Logical:

Reg[IR[15-11]] = ALUOut;

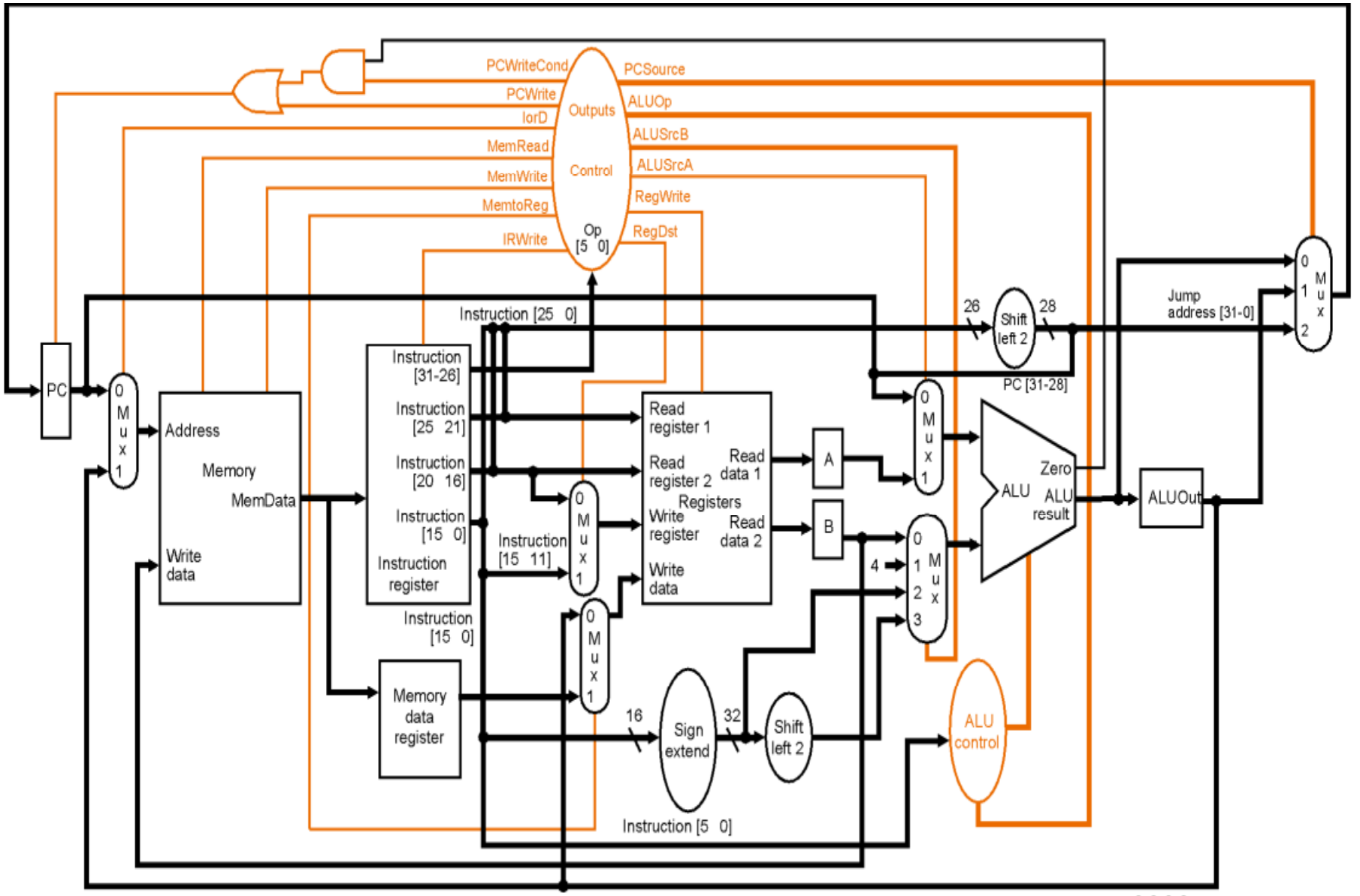
… #5

5. Memory read completion (write back step)

Reg[IR[20-16]] = MDR;

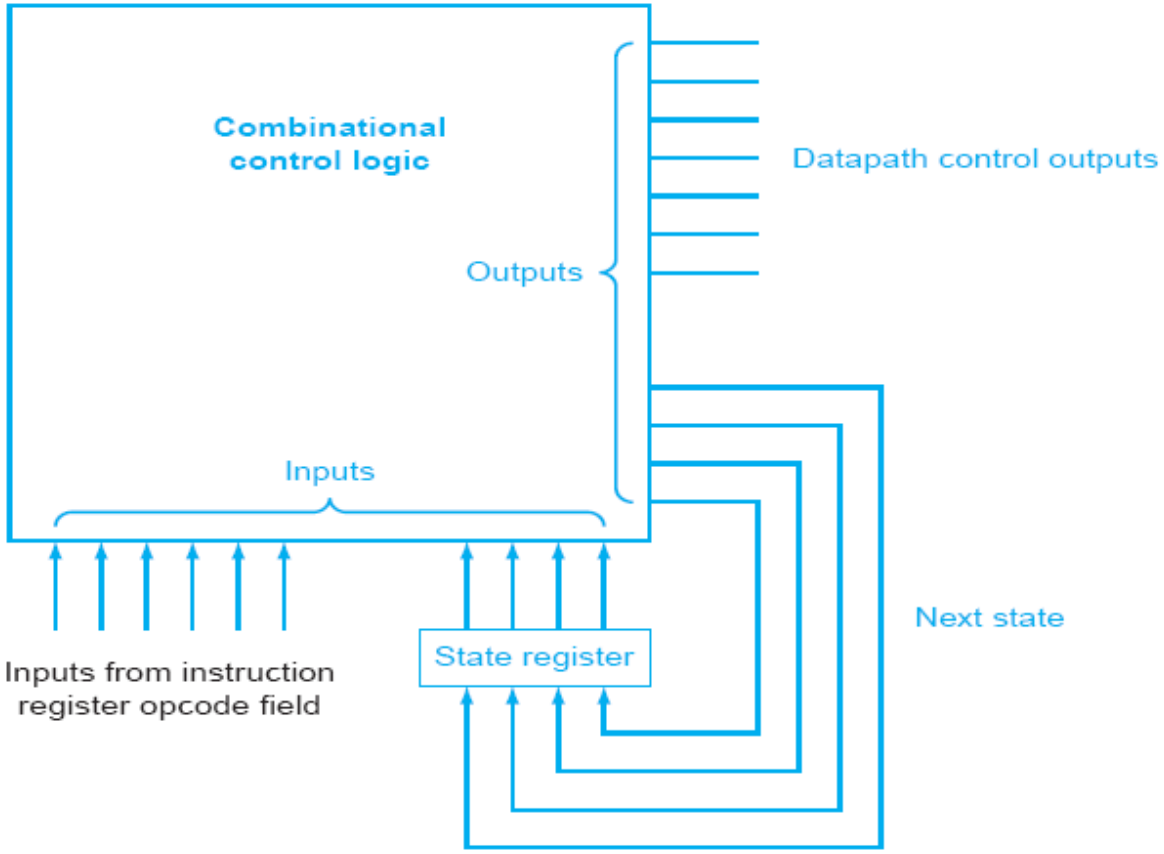
«Γράψε πίσω τα data που είχαν την προηγούμενη φάση αποθηκευτεί στον MDR, στο register file»

Πλήρες Datapath πολλαπλών κύκλων

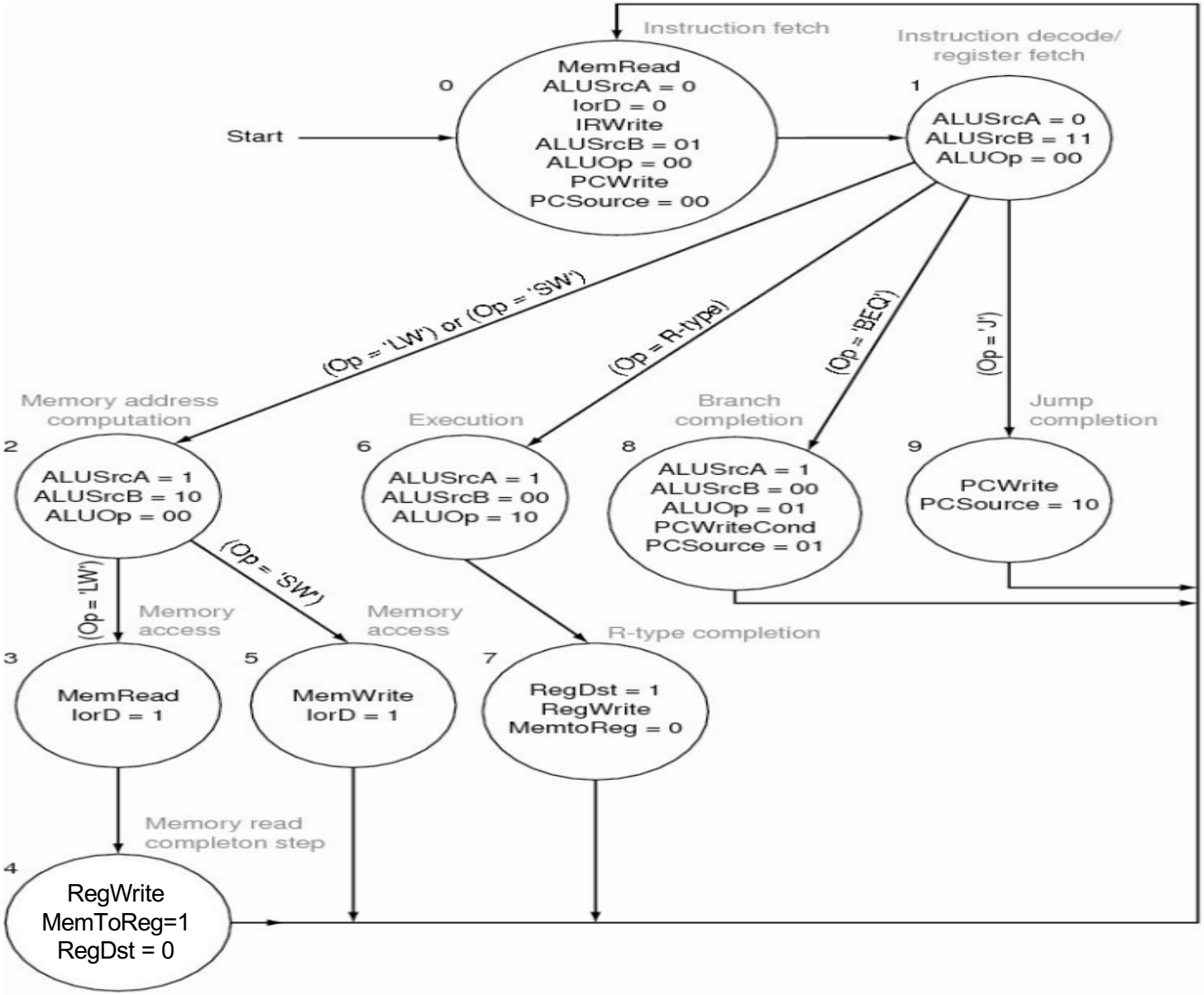


Σχεδιασμός του ελέγχου

- Είσοδοι
  - Πεδία του instruction
- Έξοδοι
  - Σήματα ελέγχου ALU
  - Σήματα ελέγχου μνήμης
  - Πολυπλέκτες
- Finite State Machine



Σχεδιασμός FSM ελέγχου του datapath πολλαπλών κύκλων



Απόδοση datapath πολλαπλών κύκλων

Θυμηθήτε: load/store ~= 1/3 των εντολών (εκ των οποίων 2/3 load, 1/3 store), branch ~= 1/5 των εντολών

- $CPI_{load} = 5$, $CPI_{store} = 4$, $CPI_{branch} = 3$, $CPI_{R\text{-}type} = 4$
- Μέσο CPI_{mc} ~= 0,22*5 + 0,11*4 + 0,2*3 + 0,47*4 =
  1,1 + 0,44 + 0,6 + 1,88 = 4,02 ~= 4

Στην καλύτερη περίπτωση $T_{mc} = 1/5*T_{sc}$

$Χρόνος_{sc} \sim 1 * T_{sc}$

$Χρόνος_{mc} \sim T_{mc} * CPI_{mc} = 1/5*T_{sc} * 4 = 4/5*T_{sc}$

$Speedup_{mc} = Χρόνος_{sc} / Χρόνος_{mc} = 5/4 = 1,25$

Πρακτικά, λόγω μη ιδανικής κατάτμησης, η βελτίωση στο χρόνο θα είναι ελάχιστη!

Ζητήματα απόδοσης #2

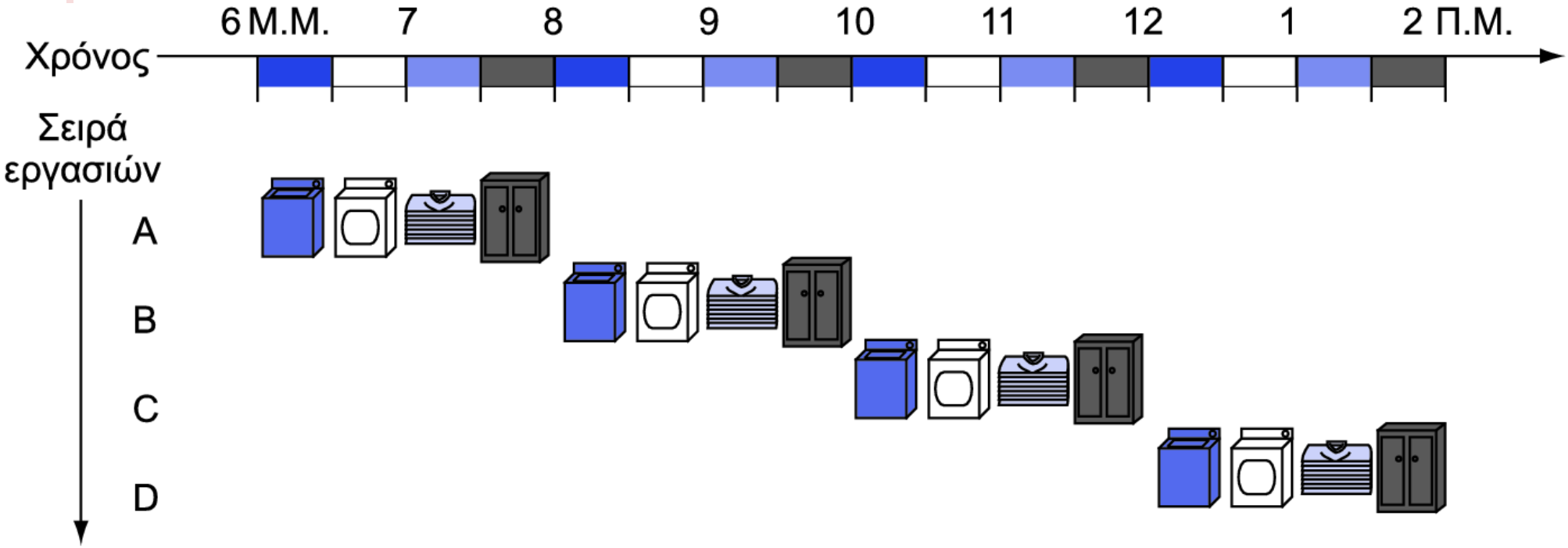
- Single-cycle: CPI = 1 αλλά ένας μεγάλος κύκλος
- Multi-cycle: λογική διάρκεια κύκλου, αλλά CPI~=4!
- Πλεονεκτήματα multi-cycle:
  - Μείωση στο κόστος (μία ALU, μία μνήμη)
  - Η μνήμη είναι υλοποιήσιμη: δεν απαιτείται συνδυαστική ανάγνωση και ακμοπυροδότητη εγγραφή

Τι κάνω για καλύτερες επιδόσεις;

- Προσπάθεια #2: Βελτίωση της απόδοσης με διοχέτευση (pipelining)

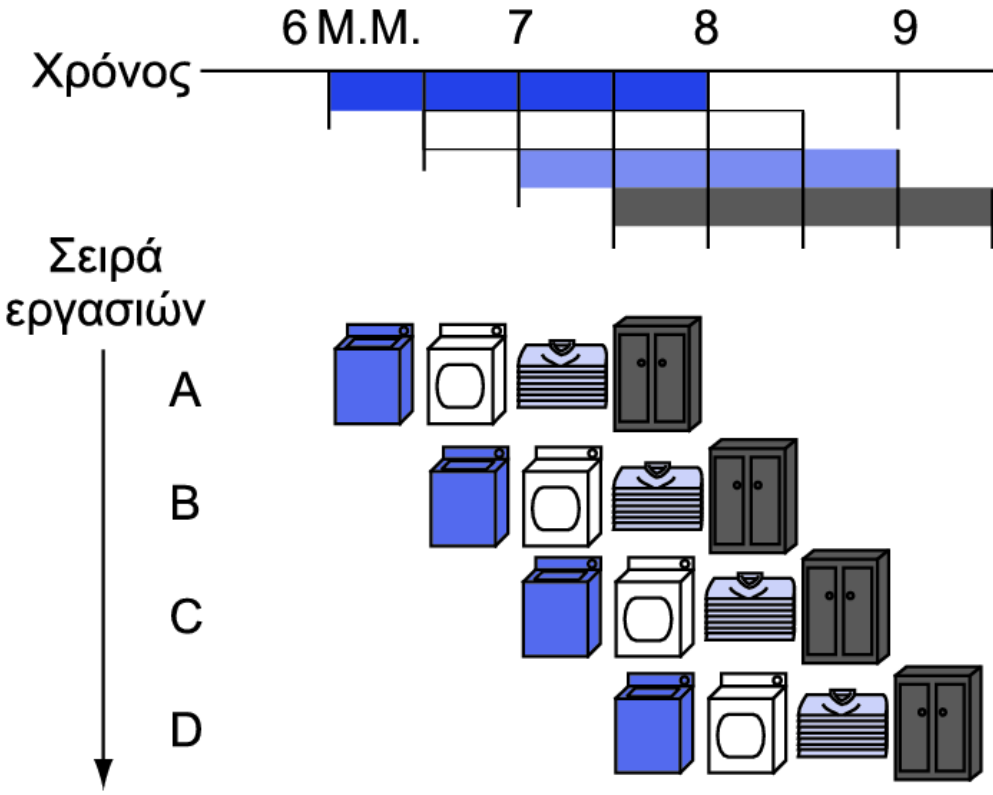
Αναλογία διοχέτευσης

- Μπουγάδα, 4 φορτία, 4 «στάδια» επεξεργασίας από 30’ έκαστο
- Συνολικός χρόνος 4*4*0.5 = 8 ώρες



Αναλογία διοχέτευσης

- Προφανώς μπορούμε καλύτερα με επικάλυψη των βημάτων (παραλληλία)



- Τέσσερα φορτία:
  - Επιτάχυνση = 8/3.5 = 2.3
- Ασταμάτητα:
  - Επιτάχυνση = 2n/0.5n + 1.5 ≈ 4 = αριθμός σταδίων

Διοχέτευση στον MIPS

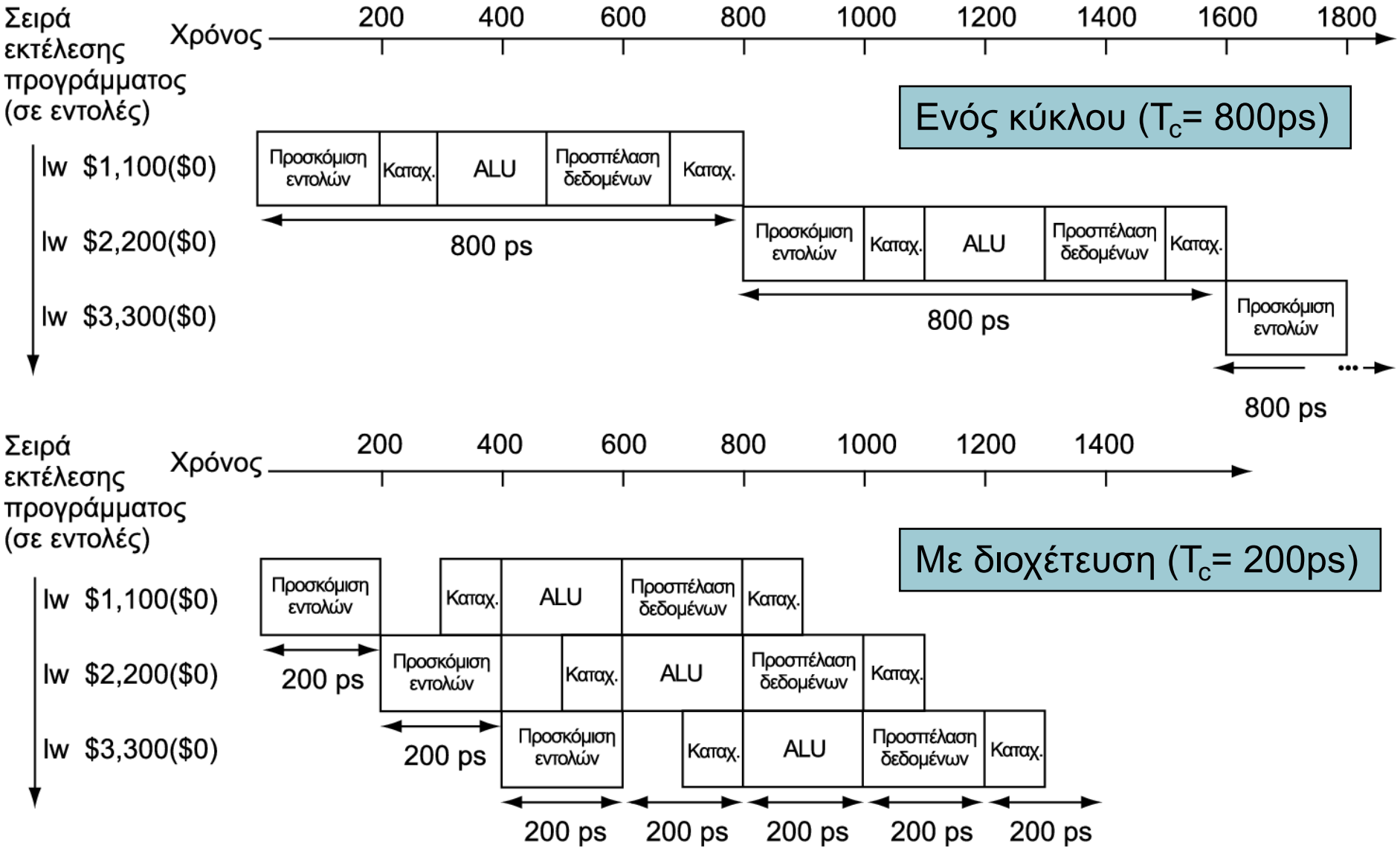
- Πέντε στάδια (stages), ένα βήμα σε κάθε στάδιο
  1. **IF**: Instruction fetch from memory (προσκόμιση εντολής από τη μνήμη)
  2. **ID**: Instruction decode & register read (αποκωδικοποίηση εντολής & ανάγνωση καταχωρητών)
  3. **EX**: Execute operation or calculate address (εκτέλεση λειτουργίας ή υπολογισμός δ/νσης)
  4. **MEM**: Access memory operand (προσπέλαση τελεστέου μνήμης)
  5. **WB**: Write result back to register (επανεγγραφή αποτελέσματος σε καταχωρητή)

Απόδοση διοχέτευσης

- Υποθέστε ότι ο χρόνος των σταδίων είναι
  - 100ps για ανάγνωση ή εγγραφή καταχωρητή
  - 200ps για τα άλλα στάδια
- Σύγκριση της διαδρομής δεδομένων με διοχέτευση με τη διαδρομή δεδομένων ενός κύκλου

Εντολή	Instruction fetch	Register read	ALU op	Memory access	Register write	Συνολικός χρόνος
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Απόδοση διοχέτευσης



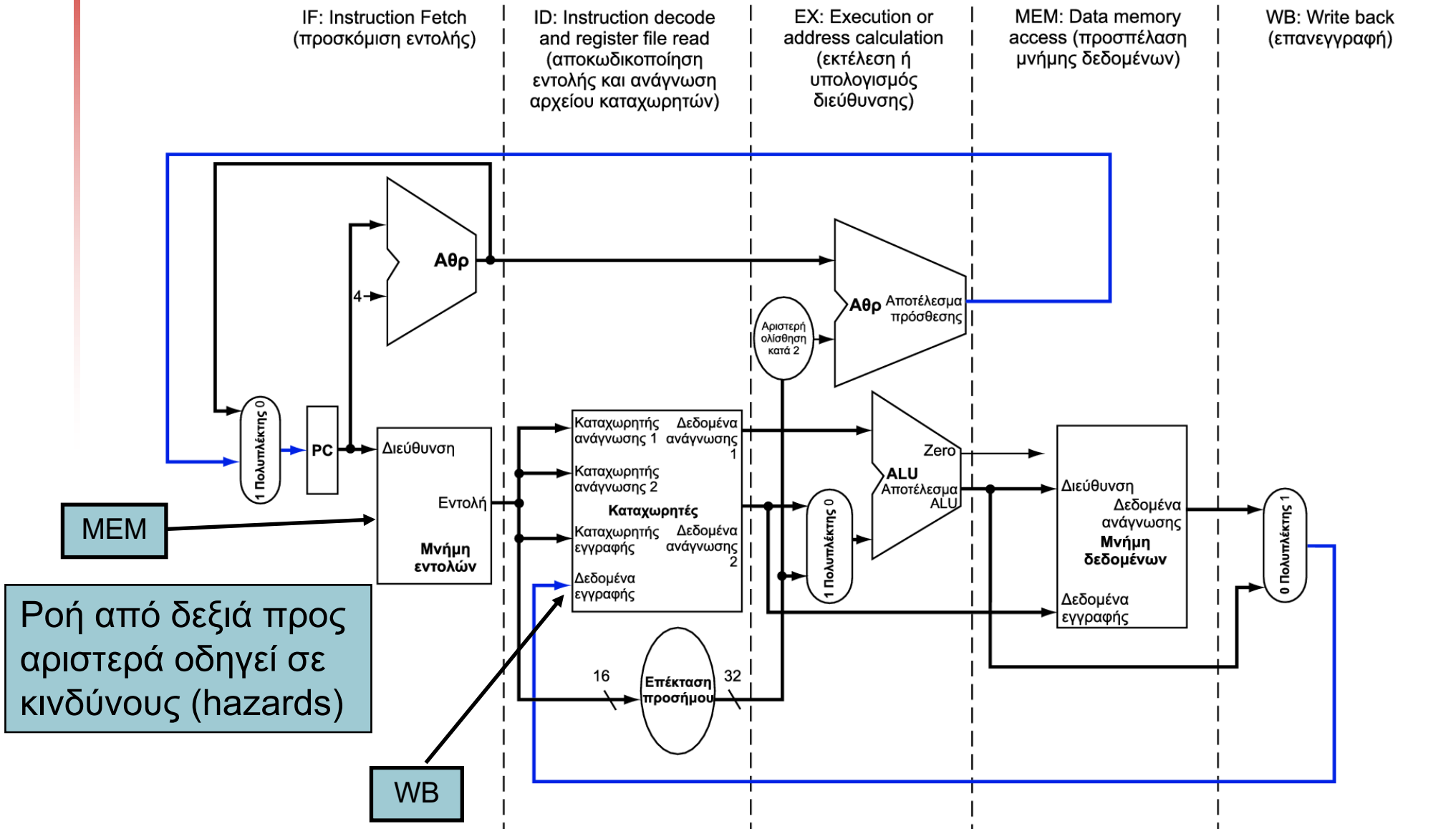
Επιτάχυνση λόγω διοχέτευσης

- Αν είναι ισορροπημένα όλα τα στάδια
  - Δηλαδή, όλα διαρκούν τον ίδιο χρόνο
  - Χρόνος μεταξύ εντολών$_{\text{με διοχέτευση}}$
    $= \dfrac{\text{Χρόνος μεταξύ εντολών}_{\text{χωρίς διοχέτευση}}}{\text{Αριθμός σταδίων}}$
- Αν δεν είναι ισορροπημένα, η επιτάχυνση είναι μικρότερη
- Επιτάχυνση λόγω αυξημένης διεκπεραιωτικής ικανότητας (throughput)
  - Λανθάνων χρόνος – latency (χρόνος για κάθε εντολή) δε μειώνεται

Διοχέτευση και σχεδίαση συνόλου εντολών

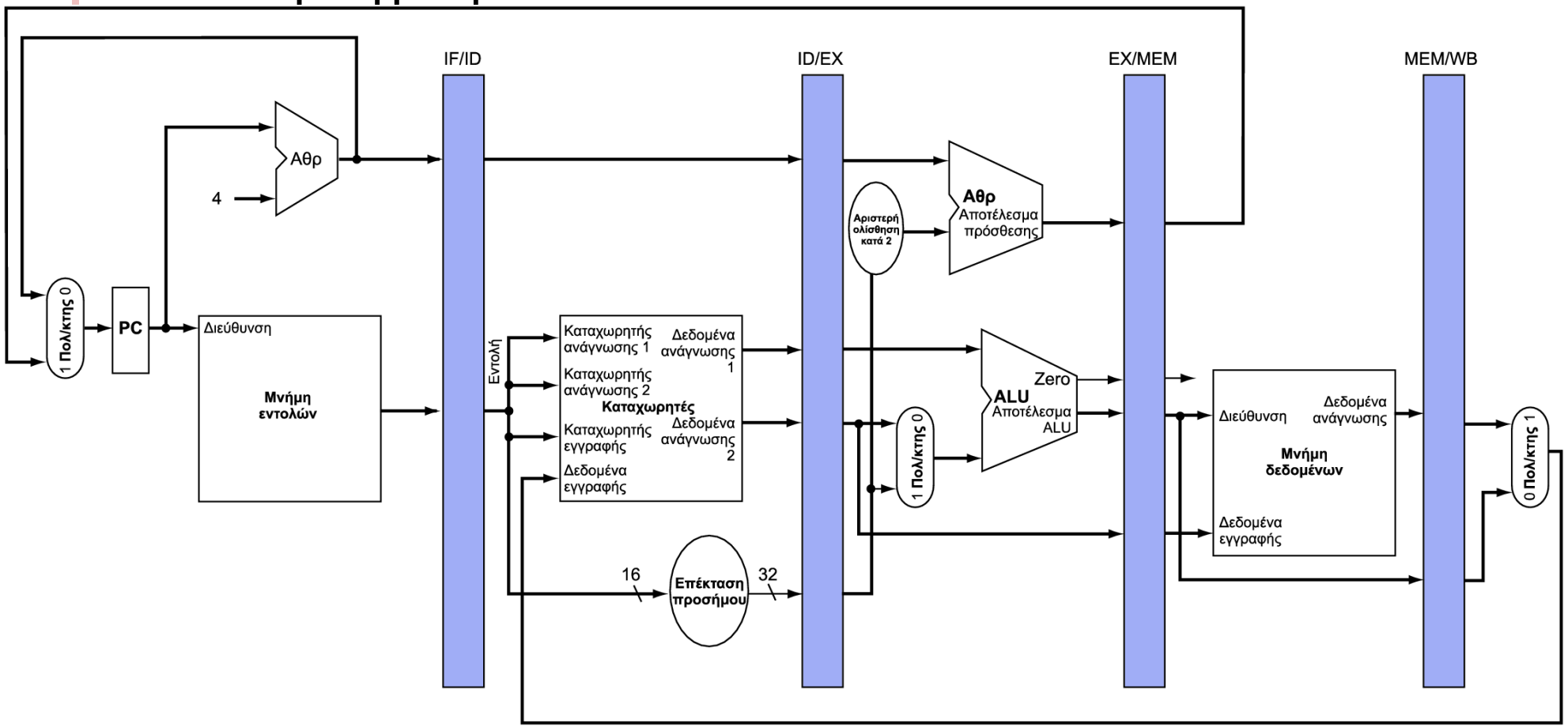
- Το σύνολο εντολών του MIPS είναι σχεδιασμένο για διοχέτευση
  - Όλες οι εντολές είναι των 32 bit
    - Ευκολότερη προσκόμιση και αποκωδικοποίηση σε έναν κύκλο
    - σύγκριση με x86: εντολές 1 έως 17 byte
  - Λίγες και κανονικές μορφές εντολών
    - Μπορεί να αποκωδικοποιήσει και να διαβάσει καταχωρητές σε ένα βήμα
  - Διευθυνσιοδότηση Load/store
    - Μπορεί να υπολογίσει τη δ/νση στο τρίτο στάδιο, και να προσπελάσει τη μνήμη στο τέταρτο στάδιο
  - Ευθυγράμμιση των τελεστέων μνήμης
    - Προσπέλαση μνήμης διαρκεί μόνο έναν κύκλο

Διαδρομή δεδομένων MIPS με διοχέτευση



Καταχωρητές διοχέτευσης

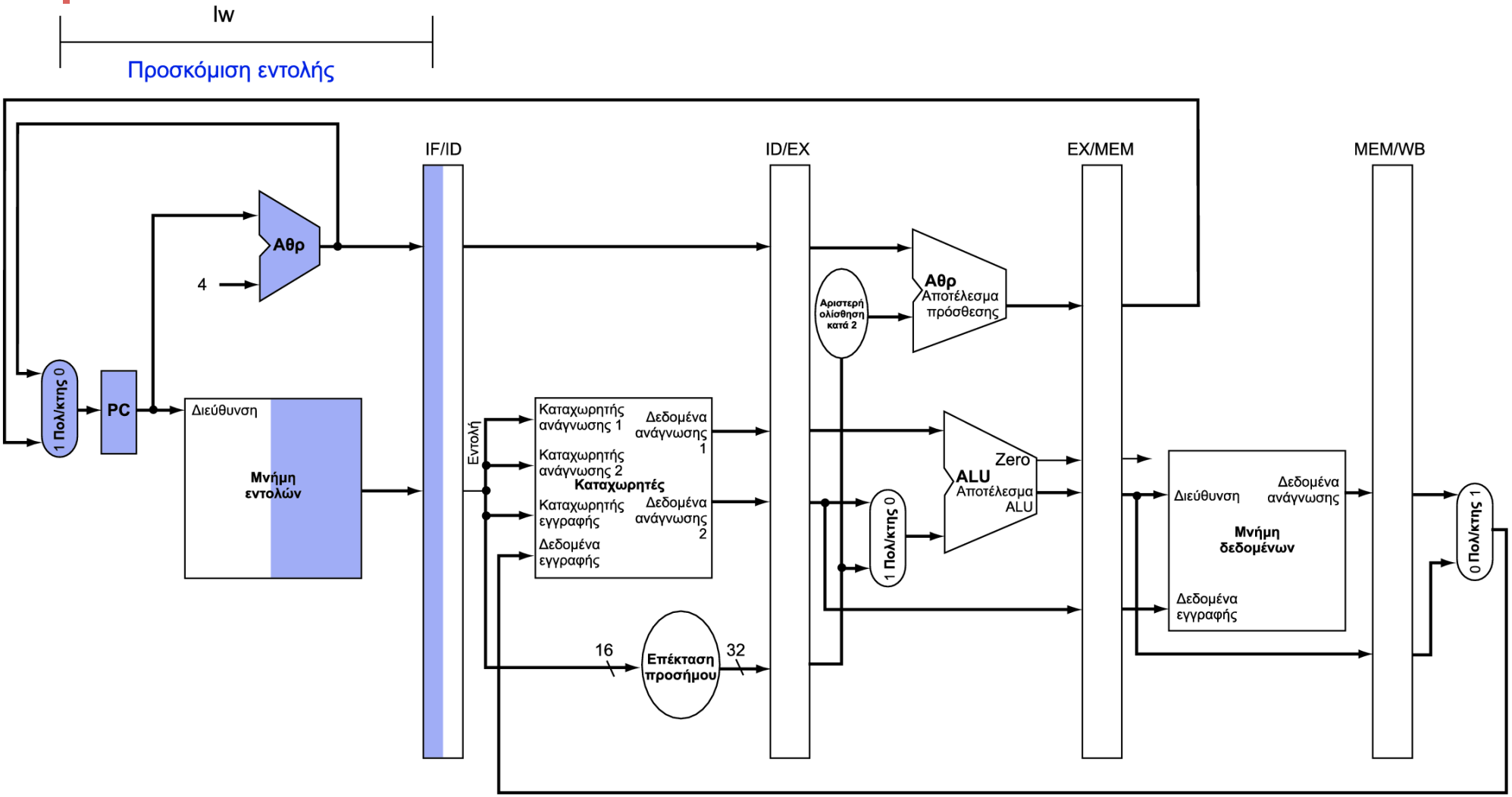
- Χρειάζονται καταχωρητές ανάμεσα στα στάδια
  - Για να κρατήσουν πληροφορίες που παράγονται στον προηγούμενο κύκλο



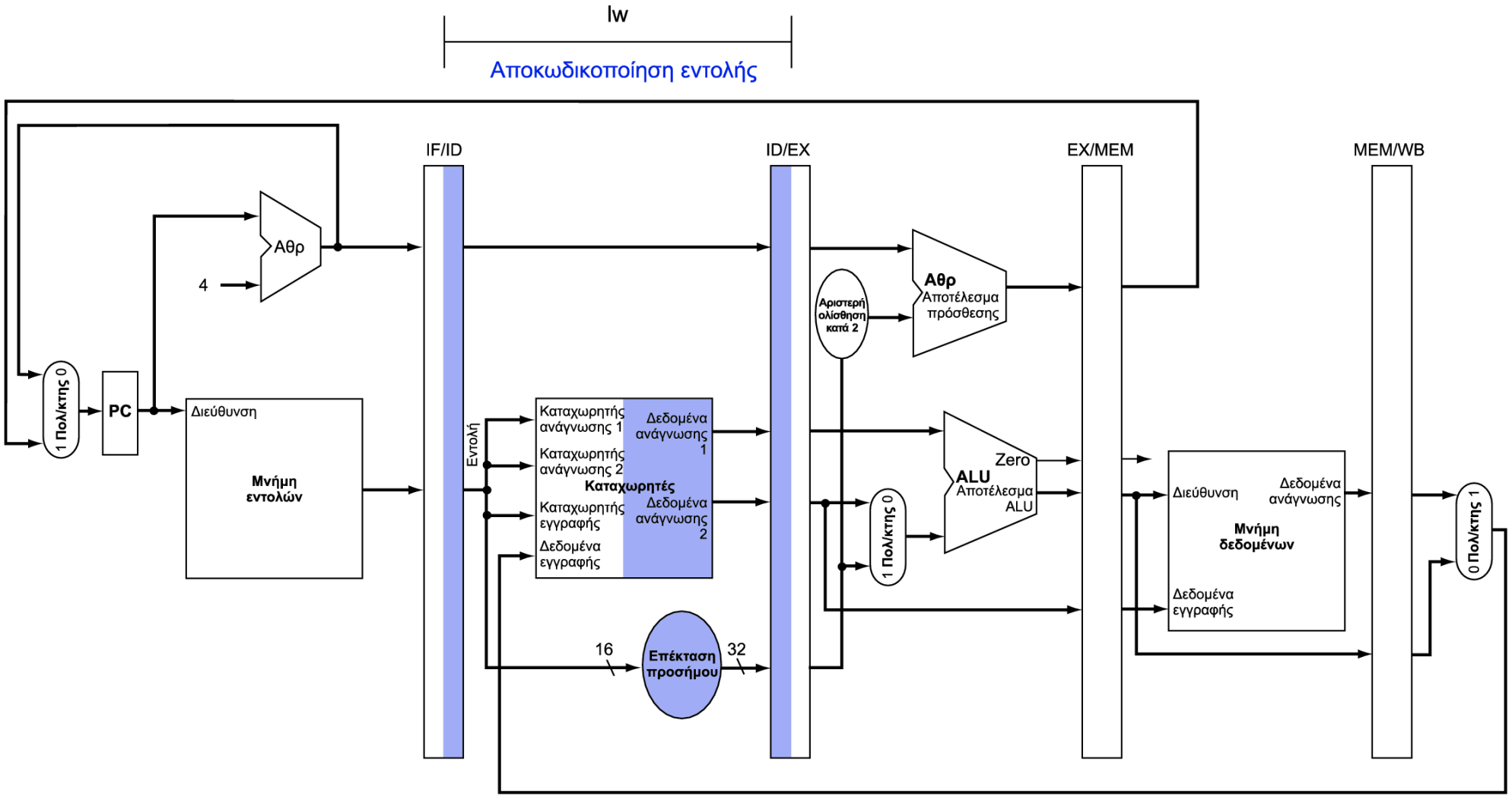
Λειτουργία διοχέτευσης

- Ανά κύκλο ροή των εντολών μέσα από τη διαδρομή δεδομένων με διοχέτευση
  - Διάγραμμα διοχέτευσης «ενός κύκλου ρολογιού» (“single-clock-cycle”)
    - Δείχνει τη χρήση της διοχέτευσης σε ένα μόνο κύκλο
    - Τονίζει τους πόρους που χρησιμοποιούνται
  - Σύγκριση με διάγραμμα «πολλών κύκλων ρολογιού» (“multi-clock-cycle”)
    - Γράφημα της λειτουργίας στο χρόνο
- Θα δούμε διαγράμματα «ενός κύκλου ρολογιού» για εντολές load και store

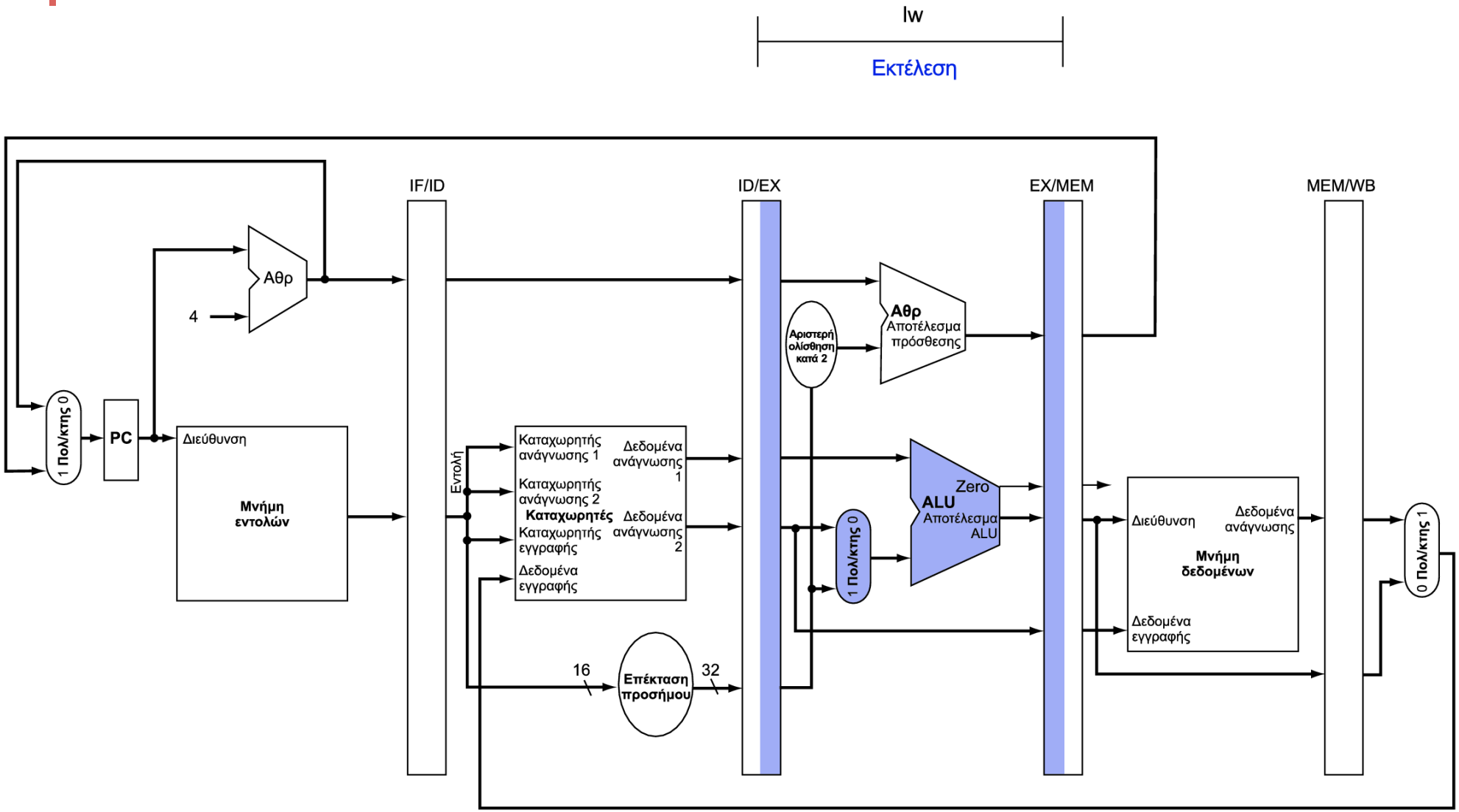
Στάδιο IF για Load, Store, …



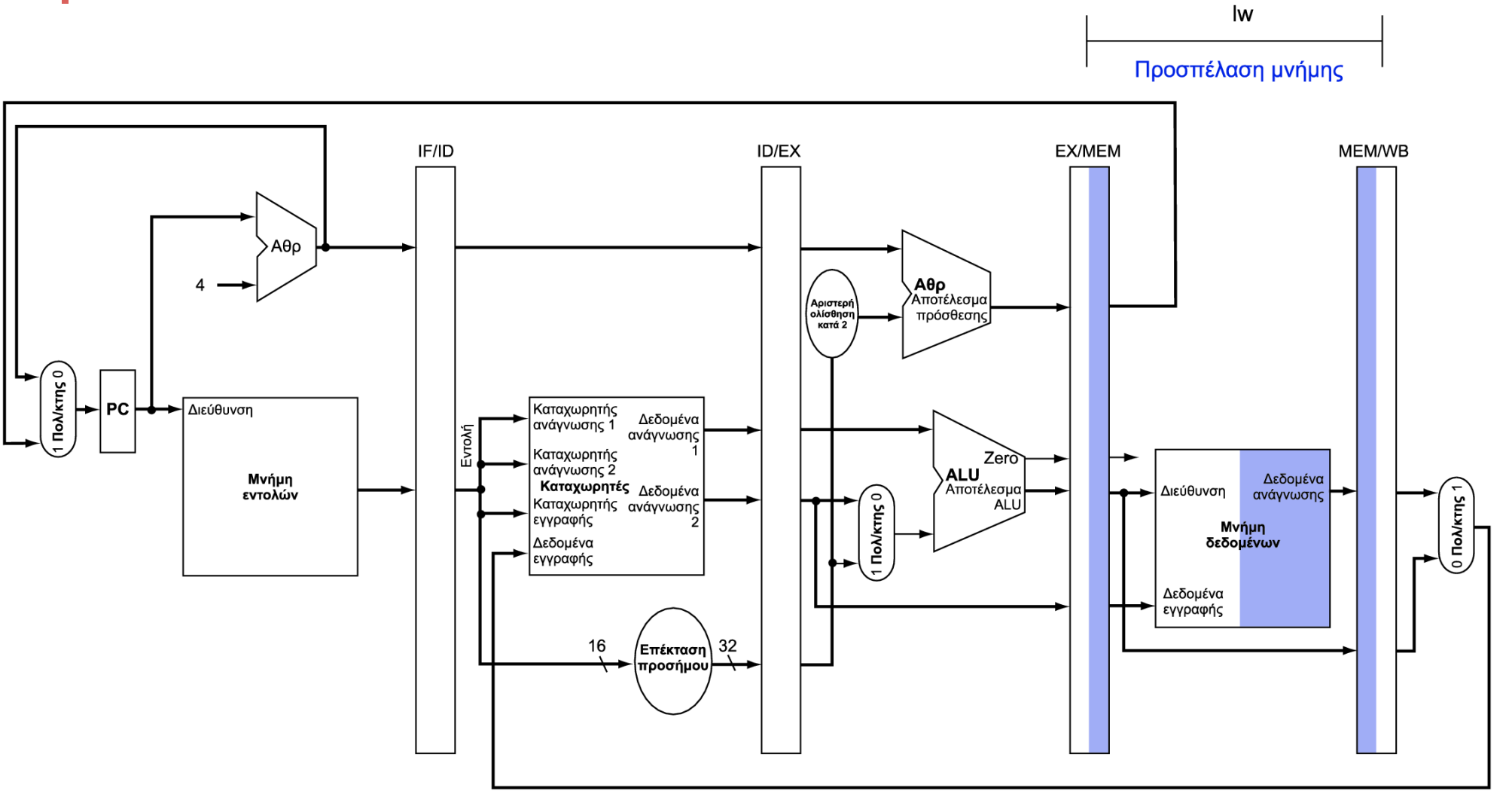
Στάδιο ID για Load, Store, …



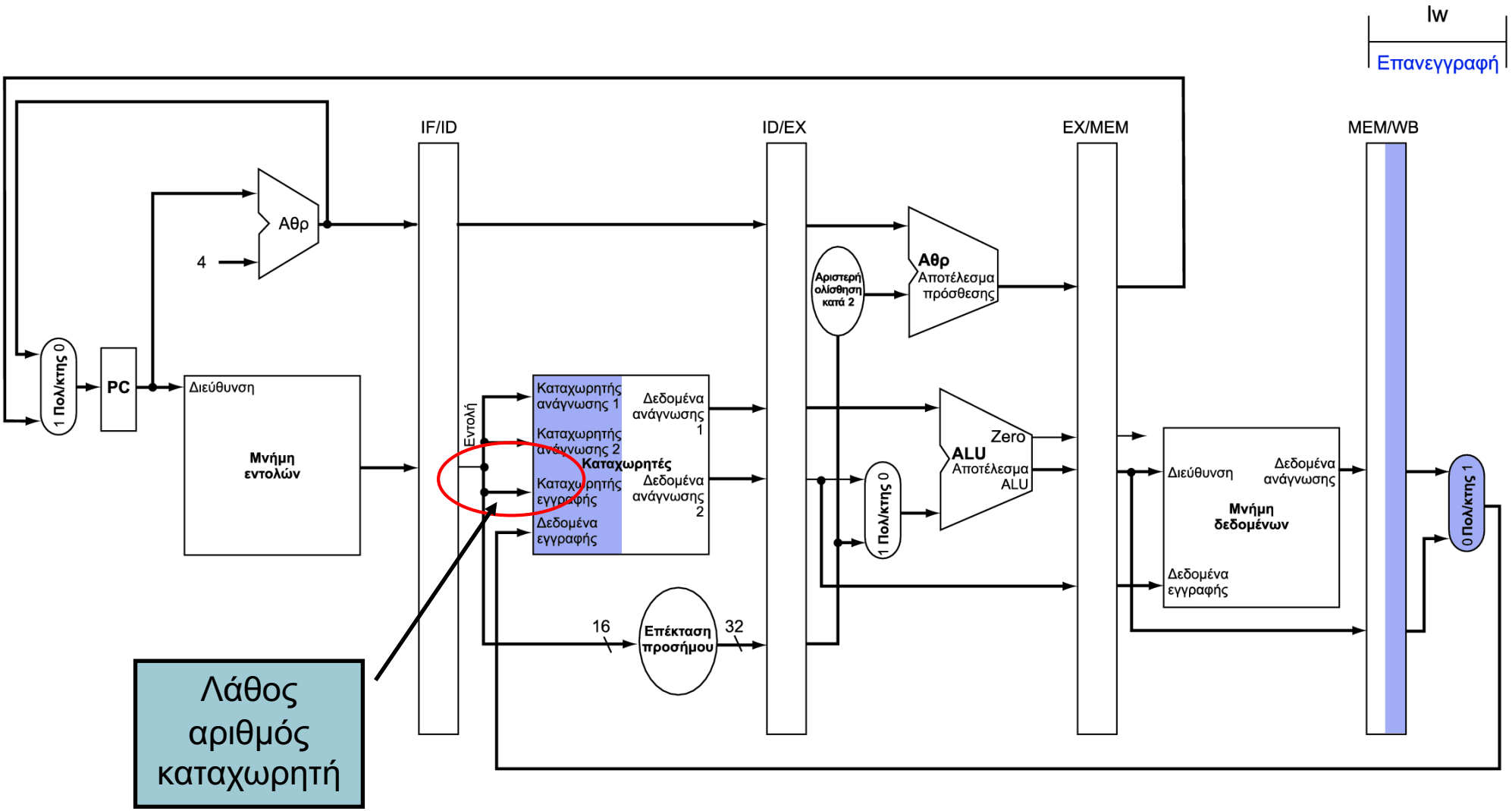
Στάδιο EX για Load



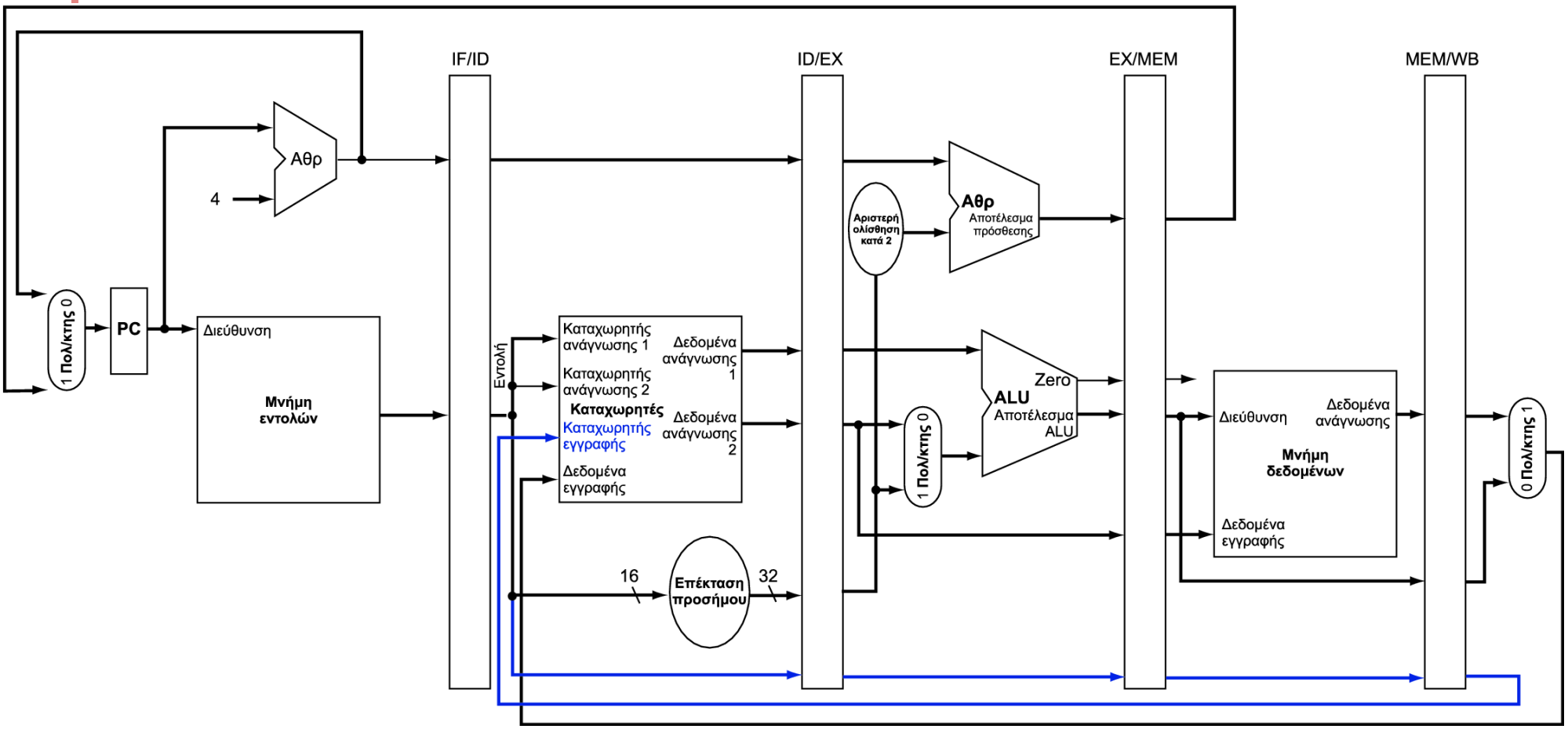
Στάδιο MEM για Load



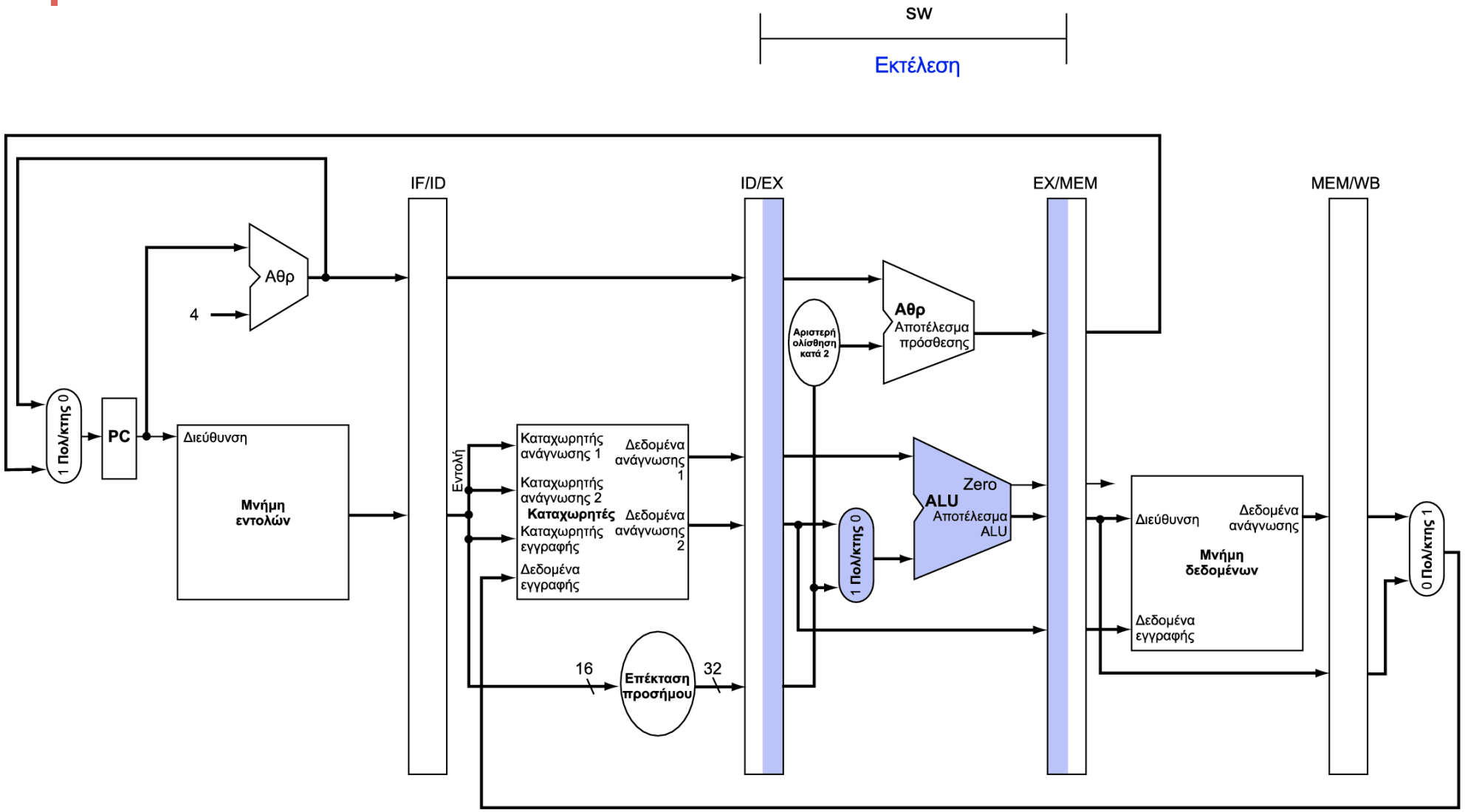
Στάδιο WB για Load



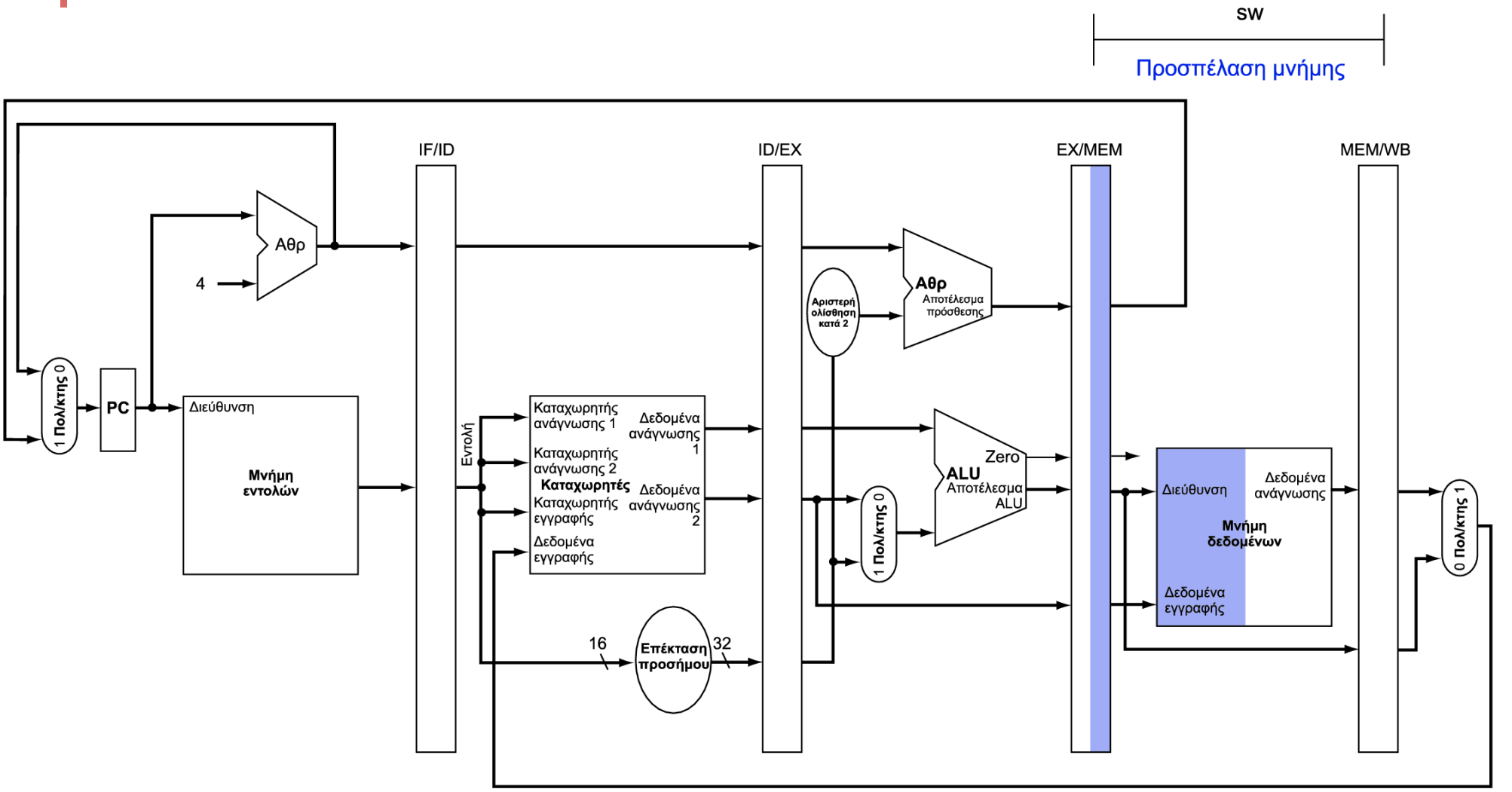
Διορθωμένη διαδρομή Load



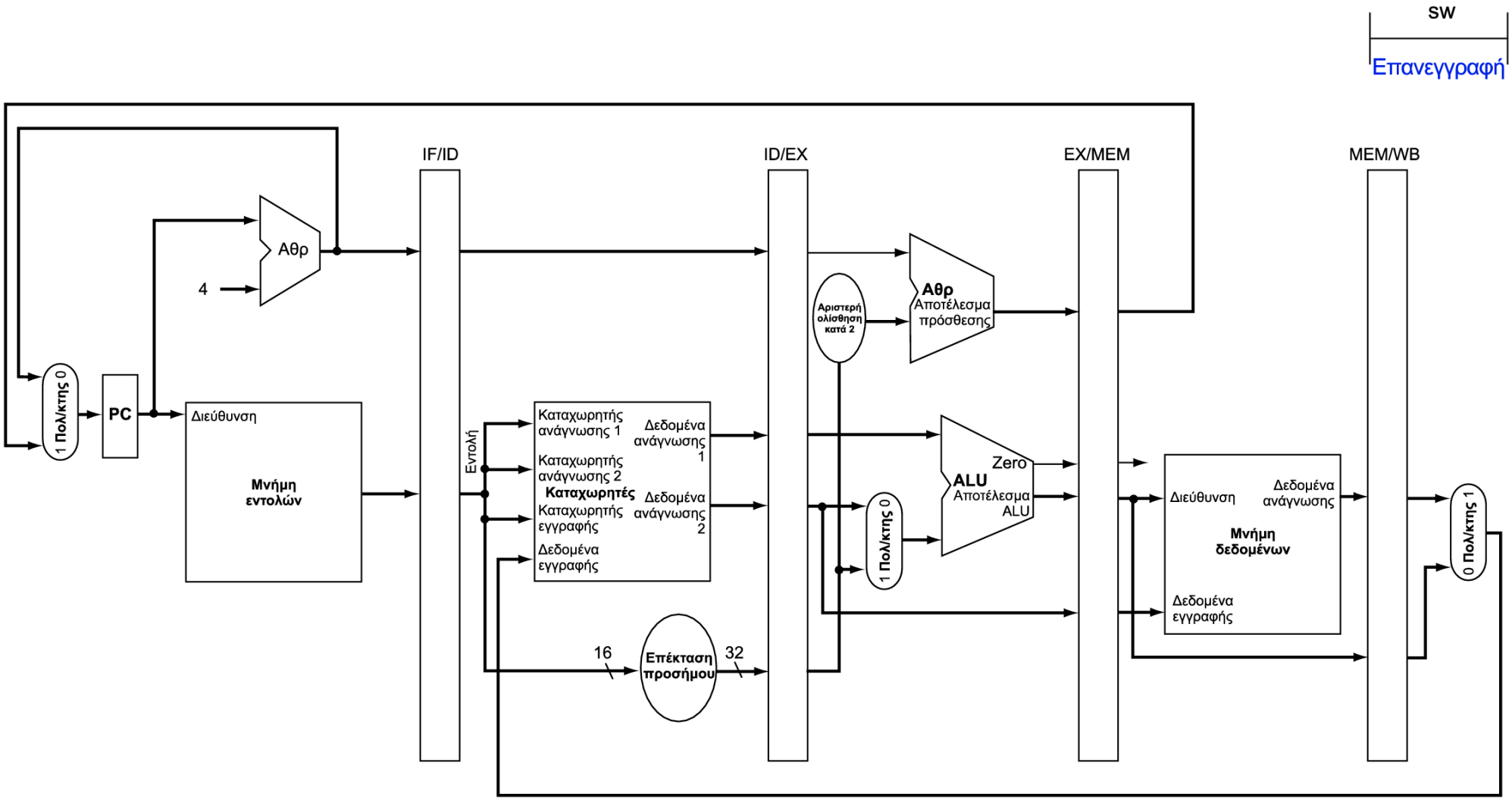
Στάδιο EX για Store



Στάδιο MEM για Store

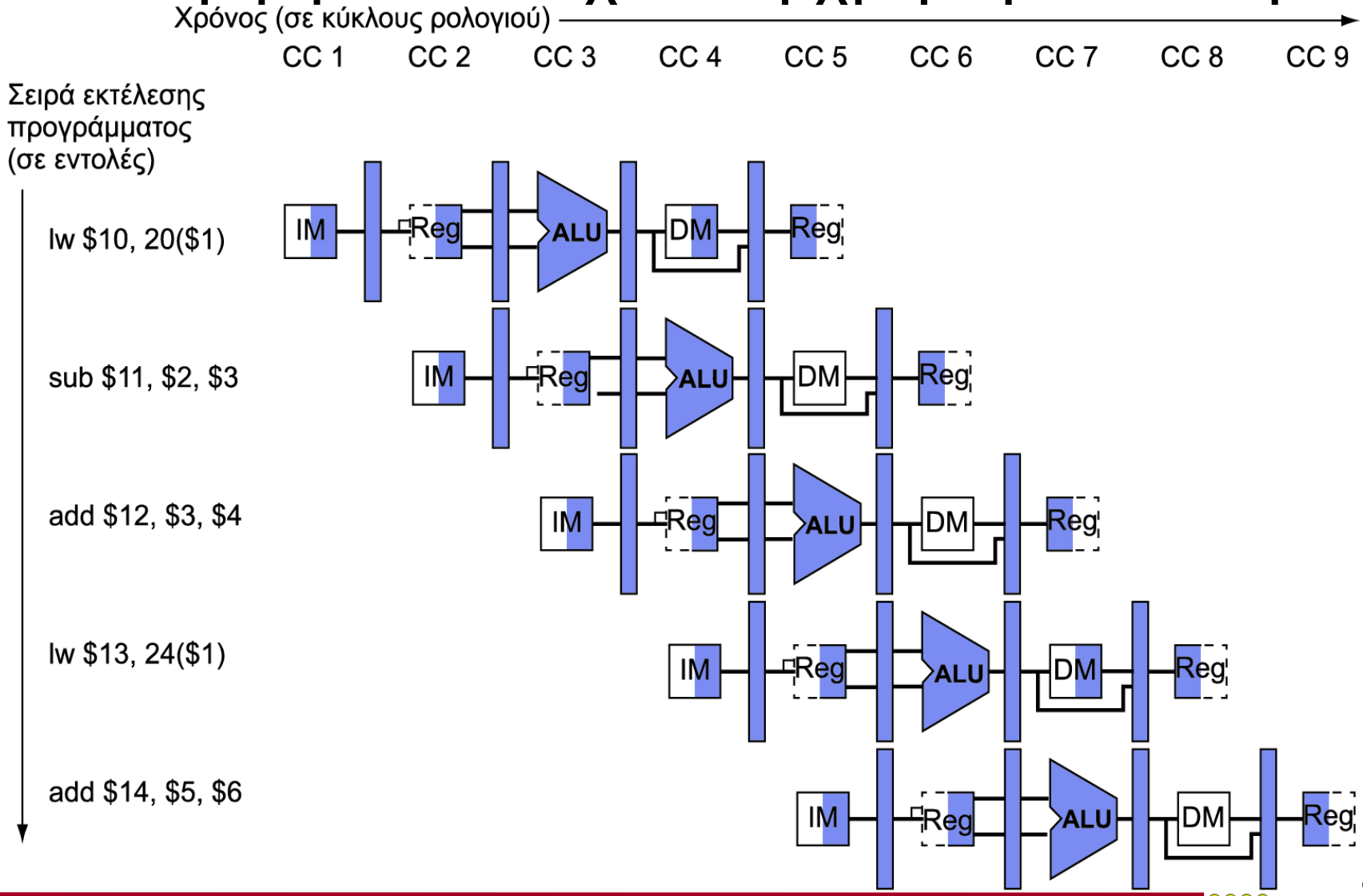


Στάδιο WB για Store



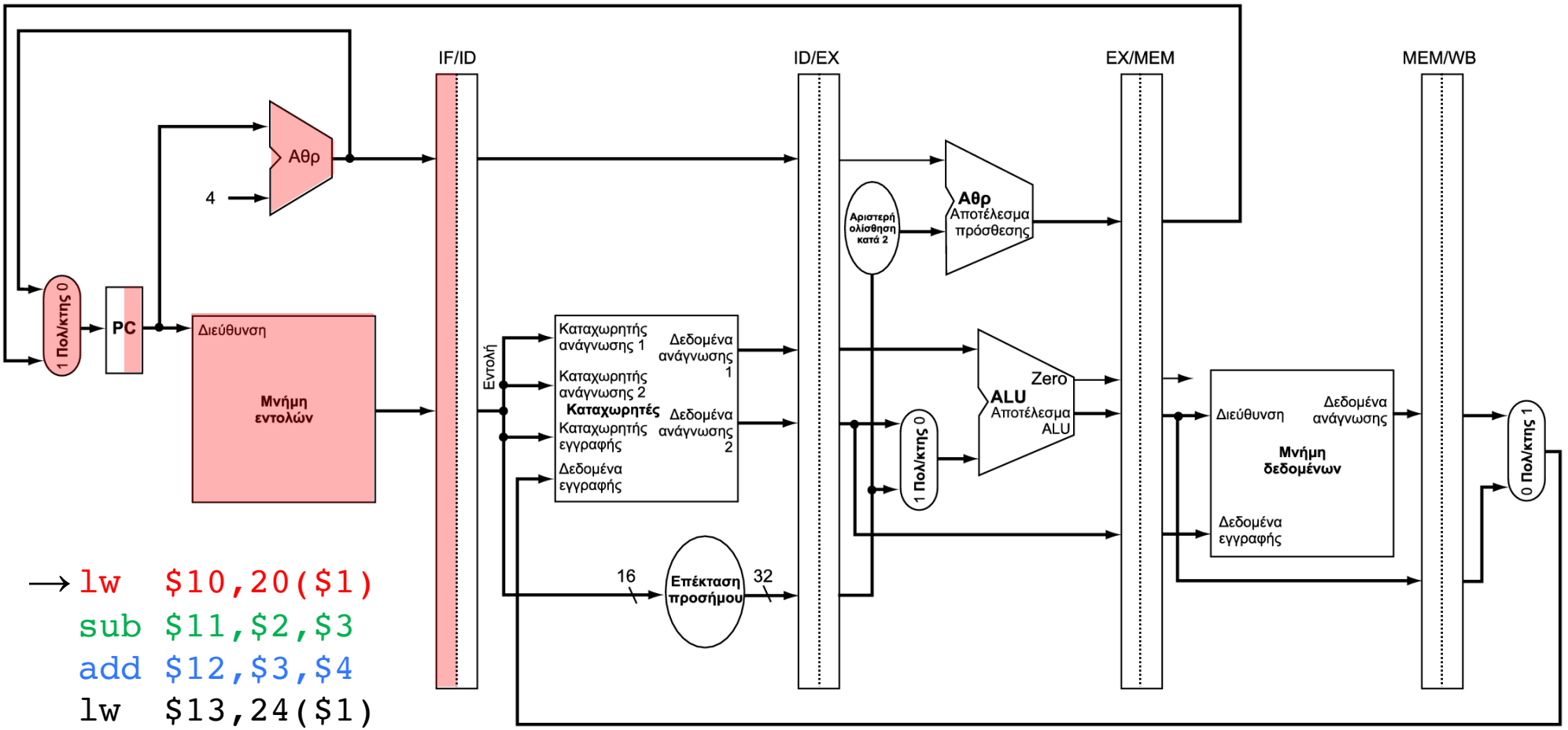
Διάγραμμα διοχέτευσης πολλών κύκλων

- Μορφή που δείχνει τη χρήση των πόρων



Παράδειγμα - Κύκλος #1

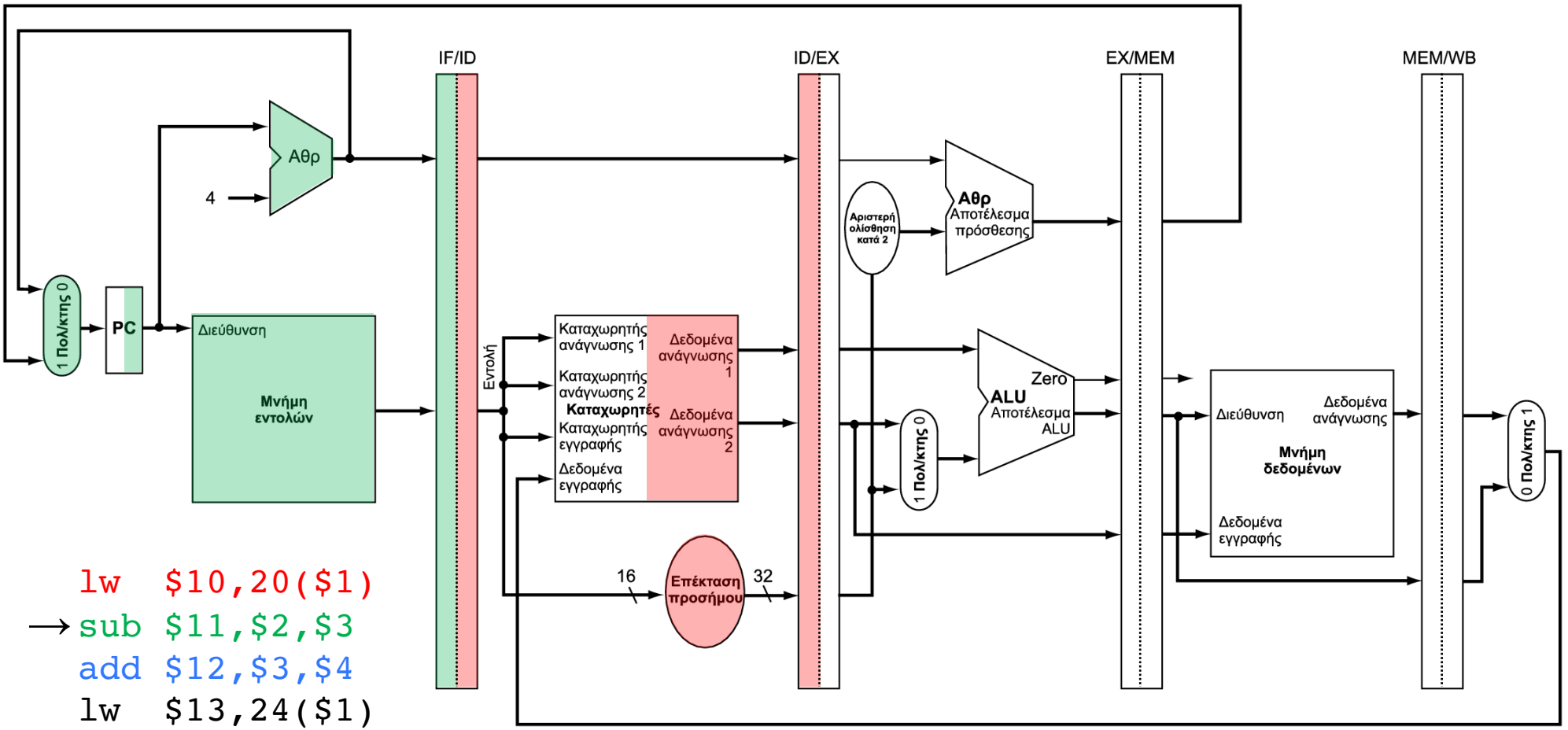
lw $10,20($1)	—	—	—	—




```
→ lw  $10,20($1)
  sub $11,$2,$3
  add $12,$3,$4
  lw  $13,24($1)
  add $14,$5,$6
```

Παράδειγμα - Κύκλος #2

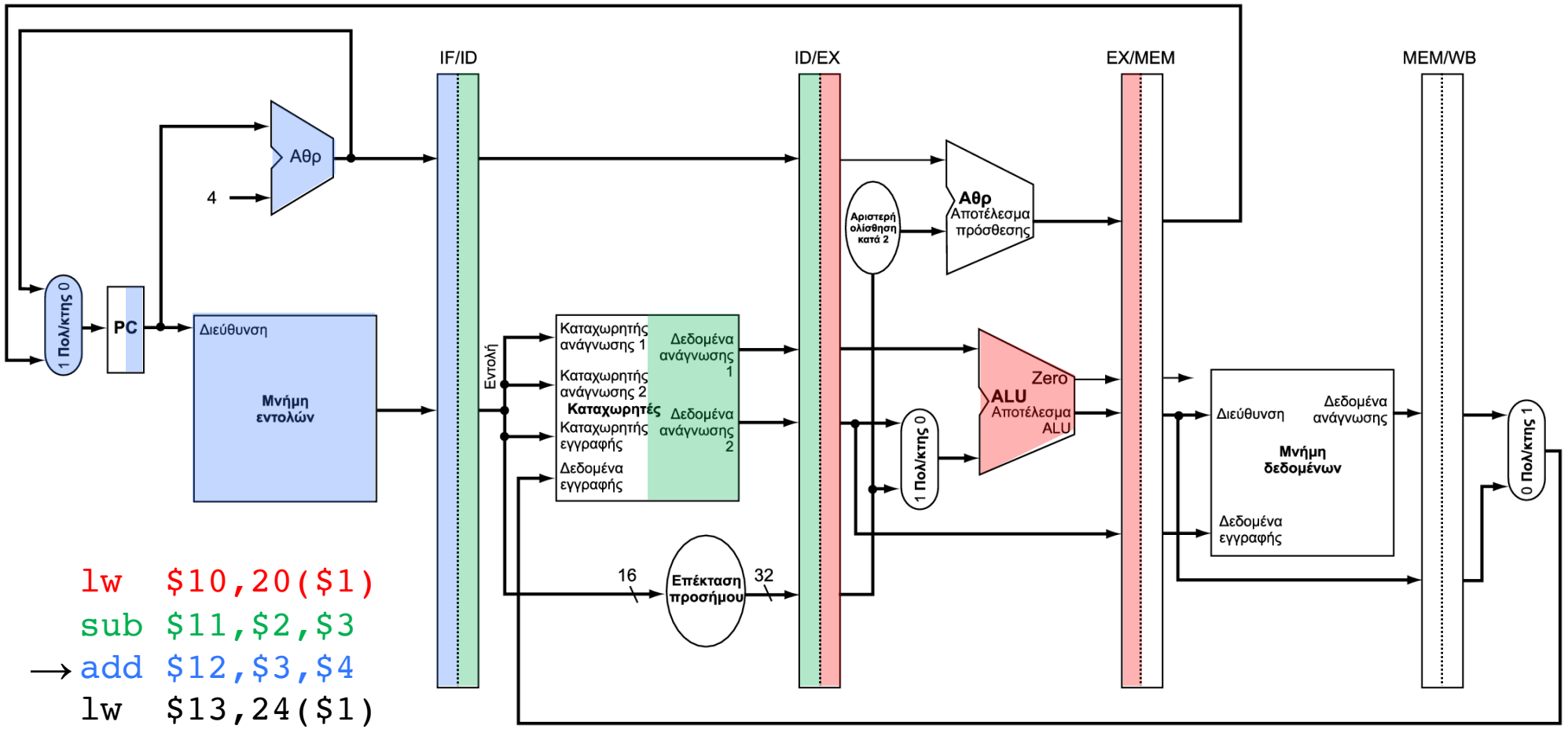
sub $11,$2,$3	lw $10,20($1)	—	—	—




```
   lw  $10,20($1)
→  sub $11,$2,$3
   add $12,$3,$4
   lw  $13,24($1)
   add $14,$5,$6
```

Παράδειγμα - Κύκλος #3

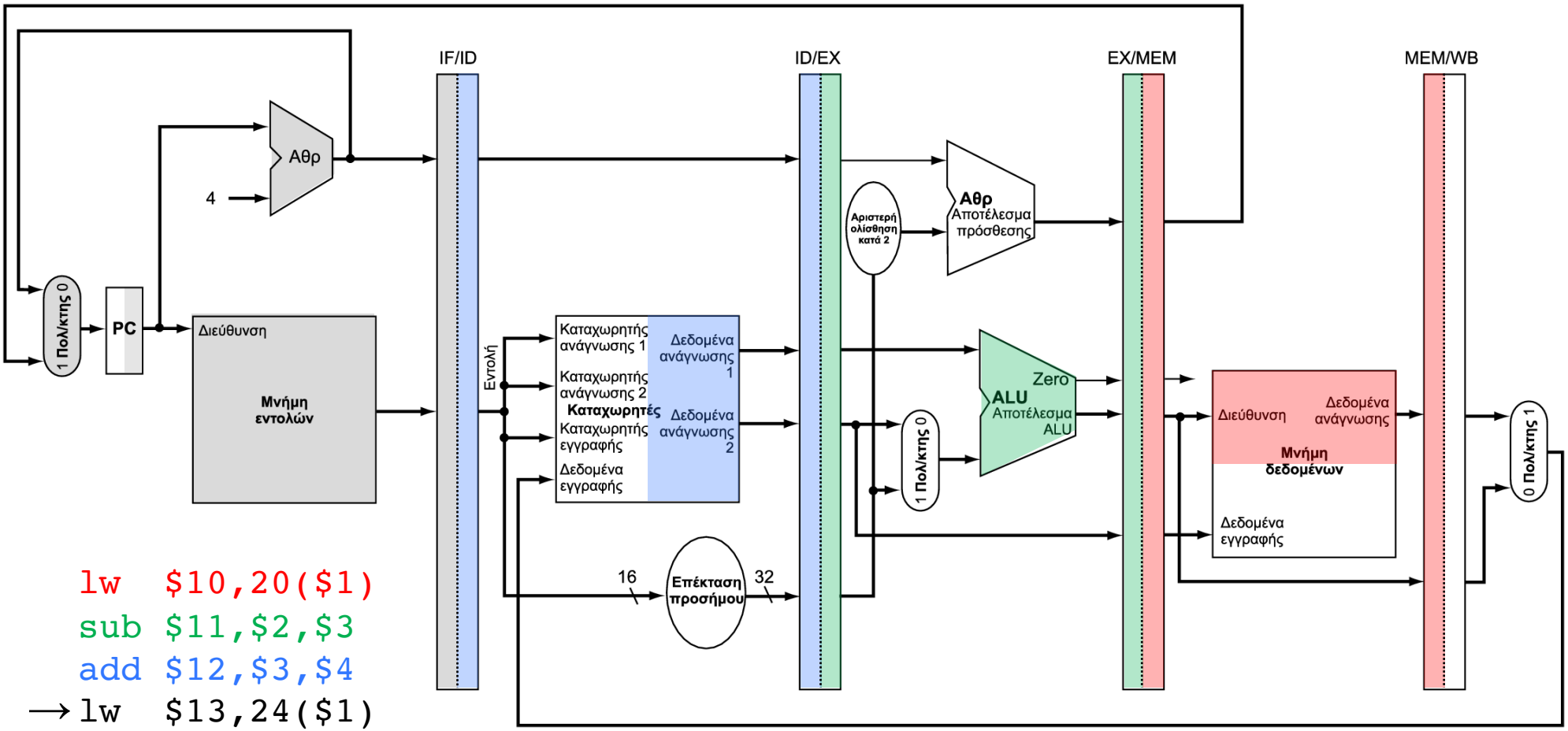
add $12,$3,$4	sub $11,$2,$3	lw $10,20($1)	—	—




```
   lw  $10,20($1)
   sub $11,$2,$3
→  add $12,$3,$4
   lw  $13,24($1)
   add $14,$5,$6
```

Παράδειγμα - Κύκλος #4

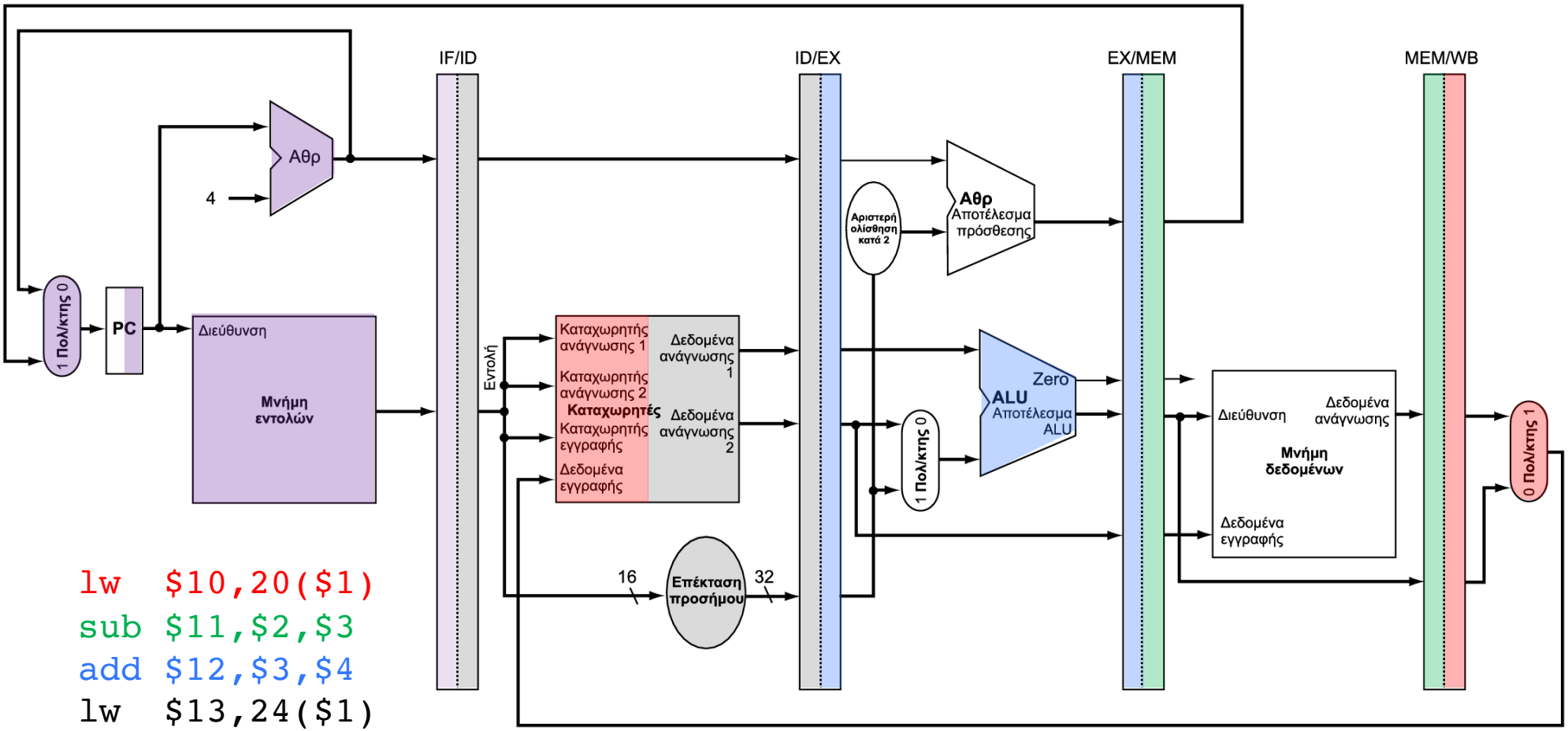
lw $13,24($1)	add $12,$3,$4	sub $11,$2,$3	lw $10,20($1)	—




```
   lw  $10,20($1)
   sub $11,$2,$3
   add $12,$3,$4
→  lw  $13,24($1)
   add $14,$5,$6
```

Παράδειγμα - Κύκλος #5

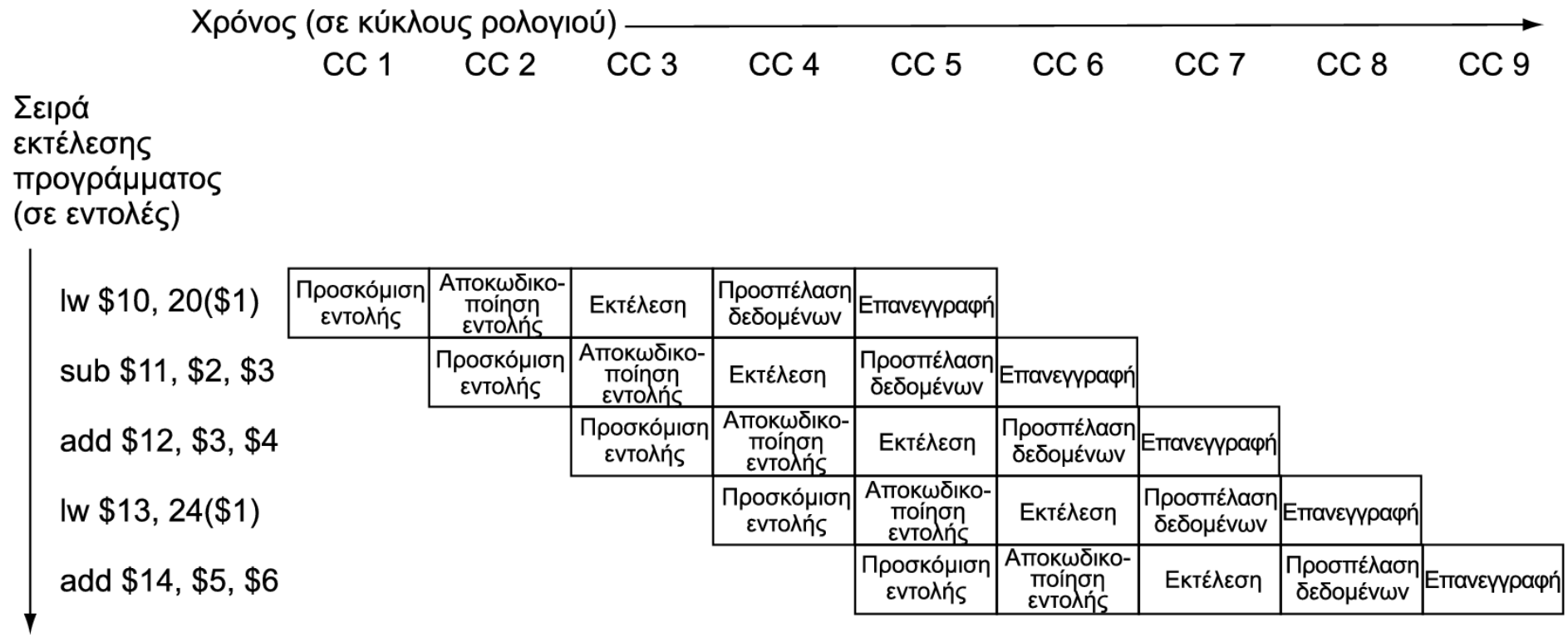
add $14,$5,$6	lw $13,24($1)	add $12,$3,$4	sub $11,$2,$3	lw …




```
   lw  $10,20($1)
   sub $11,$2,$3
   add $12,$3,$4
   lw  $13,24($1)
→  add $14,$5,$6
```

Διάγραμμα διοχέτευσης πολλών κύκλων

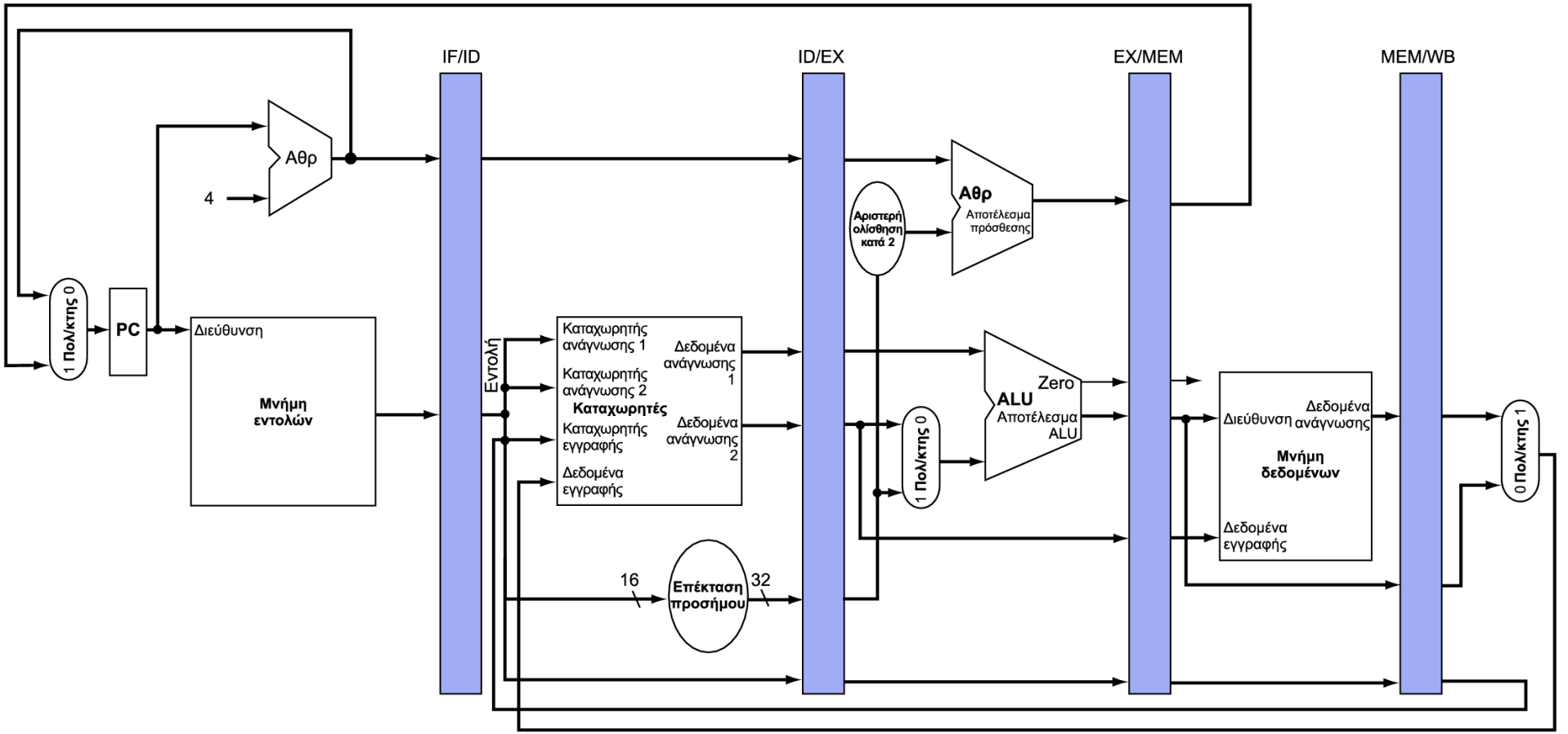
- Παραδοσιακή μορφή



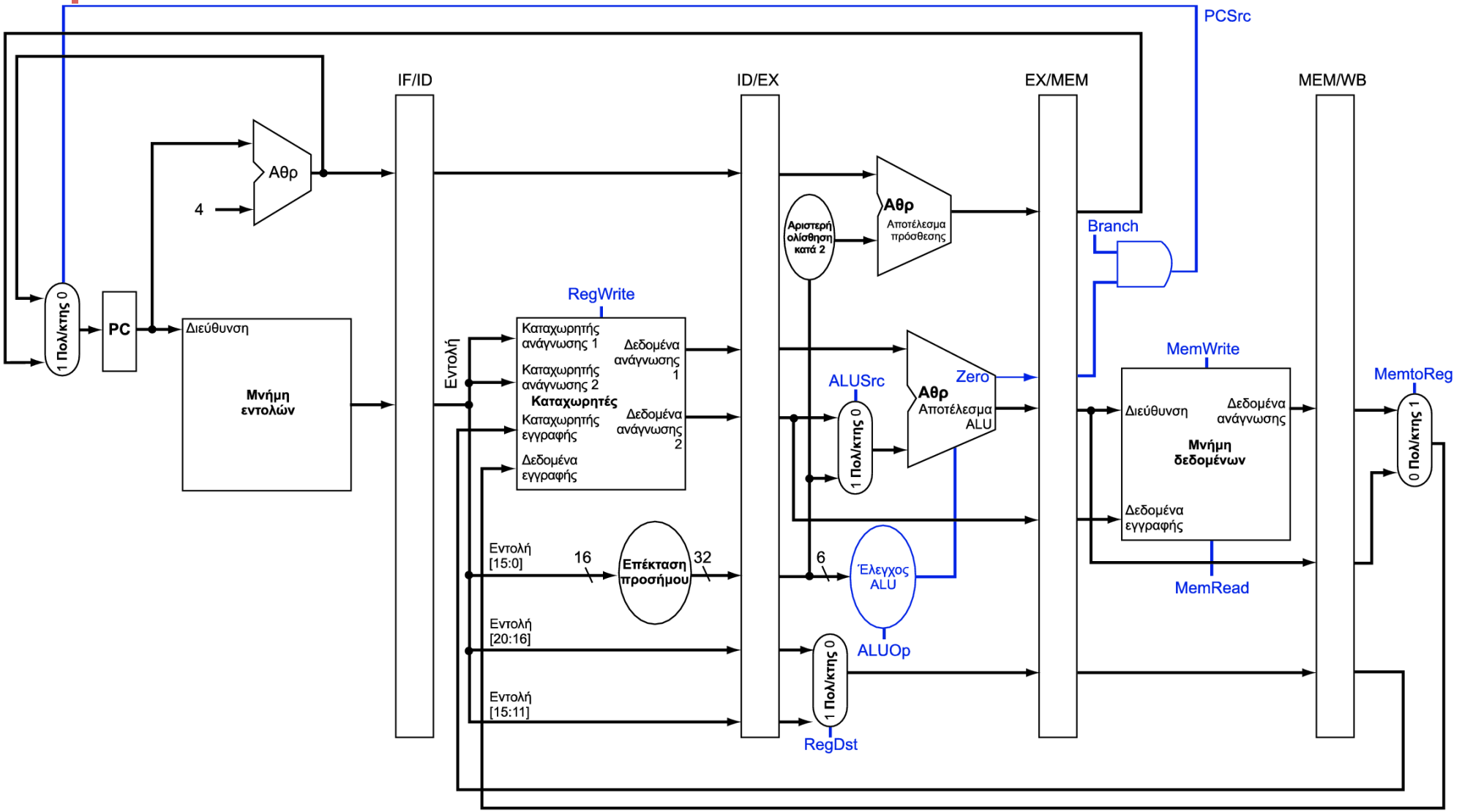
Διάγραμμα διοχέτευσης ενός κύκλου

- Κατάσταση της διοχέτευσης σε δεδομένο κύκλο

add $14, $5, $6	lw $13, 24 ($1)	add $12, $3, $4	sub $11, $2, $3	lw$10, 20($1)
Προσκόμιση εντολής	Αποκωδικοποίηση εντολής	Εκτέλεση	Προσπέλαση μνήμης	Επανεγγραφή

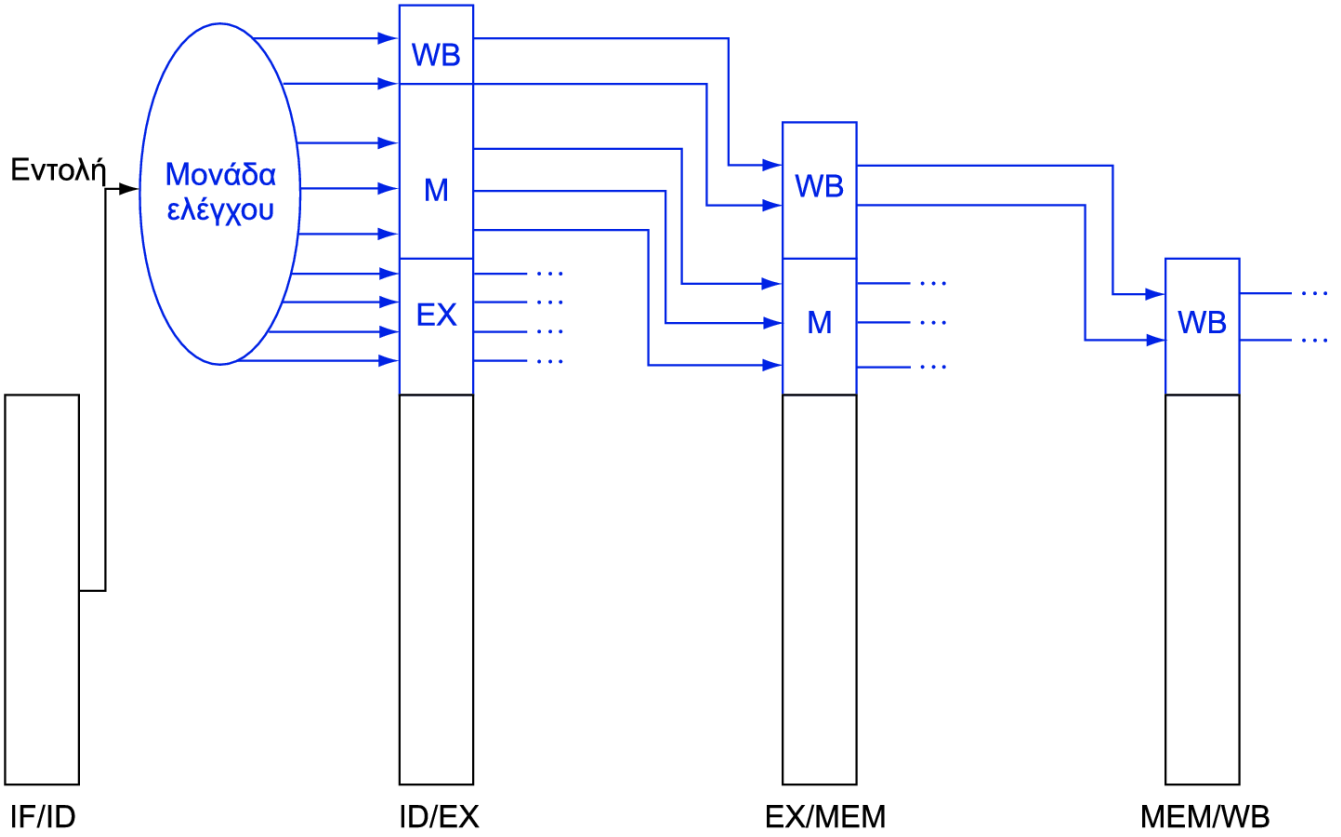


Έλεγχος διοχέτευσης (απλουσ.)

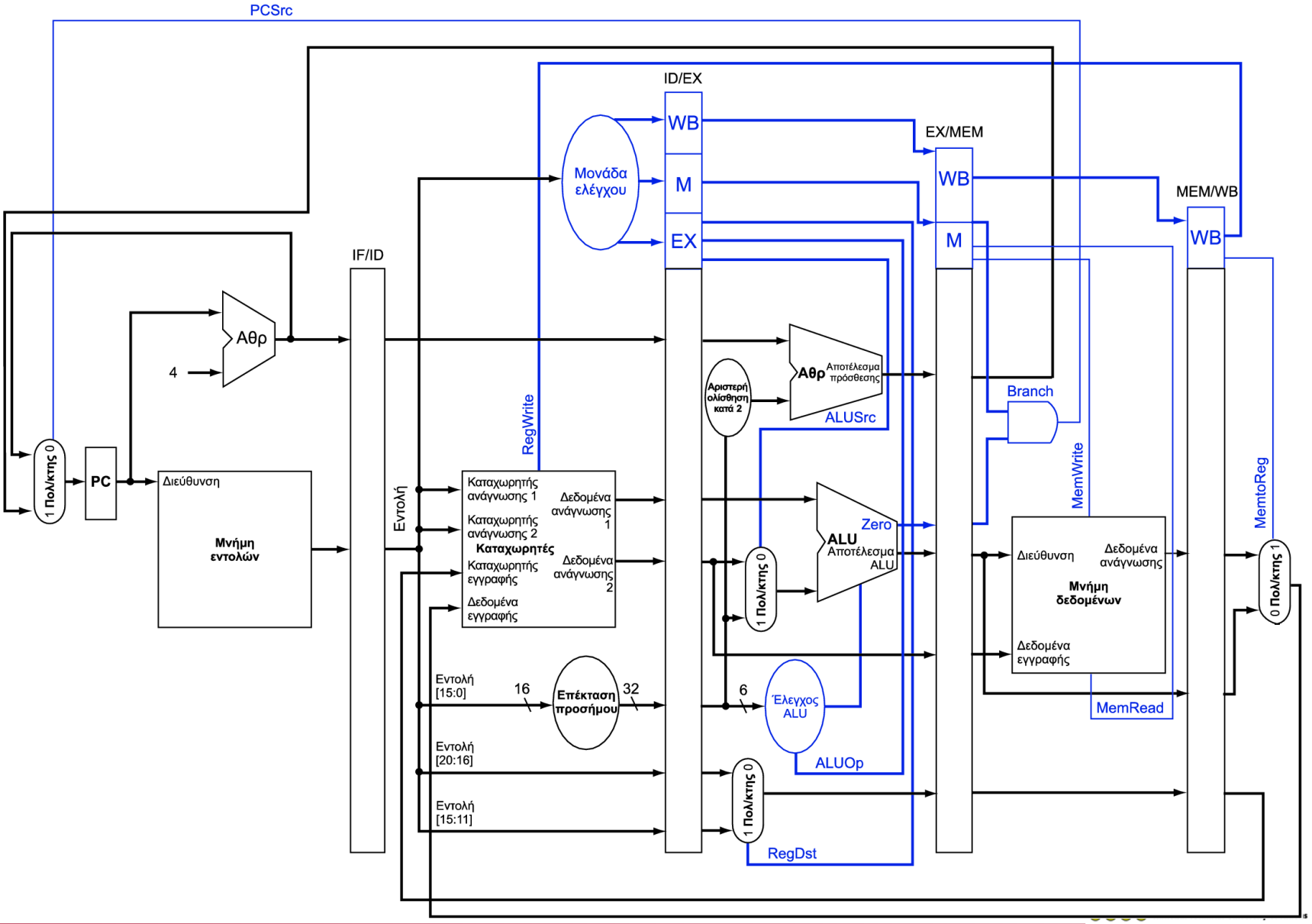


Έλεγχος διοχέτευσης

- Σήματα ελέγχου εξάγονται από την εντολή
  - Όπως και στην υλοποίηση ενός κύκλου



Έλεγχος διοχέτευσης



Κίνδυνοι δεδομένων σε εντολές ALU

- Θεωρήστε την ακολουθία:

```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- Πως θα εκτελεστεί σωστά;
- Μπορούμε να επιλύσουμε τους κινδύνους (ναι, με προώθηση-forwarding)
  - Πώς ανιχνεύουμε πότε πρέπει να γίνει προώθηση;

Κίνδυνοι (hazards)

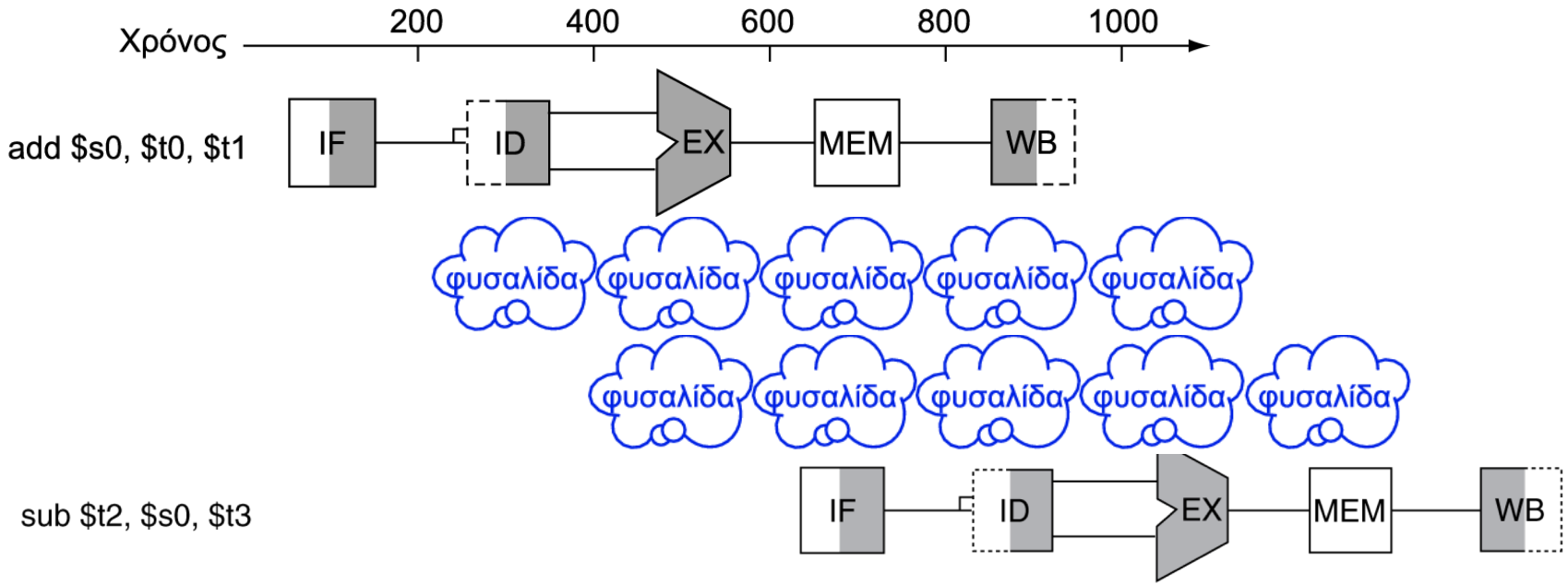
- Καταστάσεις που αποτρέπουν την εκκίνηση της επόμενης εντολής στον επόμενη κύκλο
- Κίνδυνος δομής (structure hazards)
  - Ένας απαιτούμενος πόρος είναι απασχολημένος
- Κίνδυνος δεδομένων (data hazard)
  - Πρέπει να περιμένει την προηγούμενη εντολή να ολοκληρώσει την ανάγνωση/εγγραφή δεδομένων της
- Κίνδυνος ελέγχου (control hazard)
  - Η απόφαση σε μια ενέργεια ελέγχου εξαρτάται από προηγούμενη εντολή

Κίνδυνοι δομής

- Διένεξη στη χρήση ενός πόρου
- Στη διοχέτευση του MIPS με μία μοναδική μνήμη
  - Οι εντολές load/store απαιτούν προσπέλαση μνήμης
  - Η προσκόμιση εντολής πρέπει να *καθυστερήσει* (*stall*) σε εκείνο το κύκλο
    - Θα προκαλούσε «φυσαλίδα» της διοχέτευσης (pipeline “bubble”)
- Έτσι, οι διαδρομές δεδομένων με διοχέτευση απαιτούν ξεχωριστές μνήμες εντολών/δεδομένων
  - Ή ξεχωριστές κρυφές μνήμες (cache memories) εντολών/δεδομένων

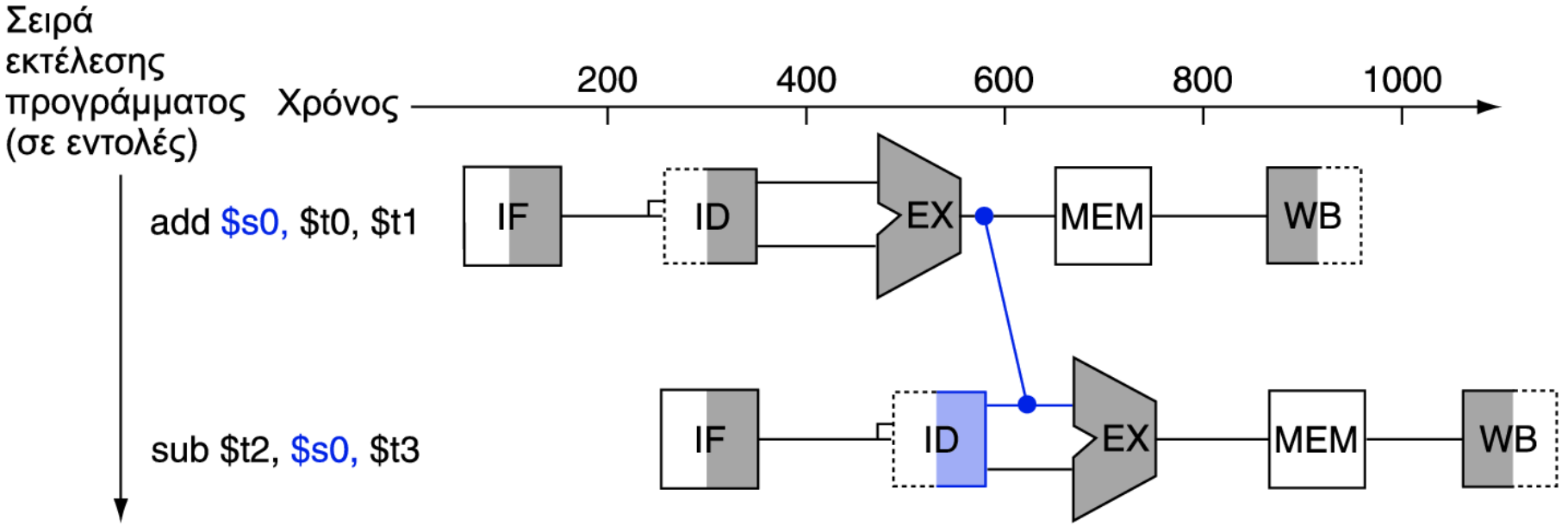
Κίνδυνοι δεδομένων

- Μια εντολή εξαρτάται από την ολοκλήρωση μιας προσπέλασης δεδομένων μιας προηγούμενης εντολής
  - `add  $s0, $t0, $t1`
    `sub  $t2, $s0, $t3`



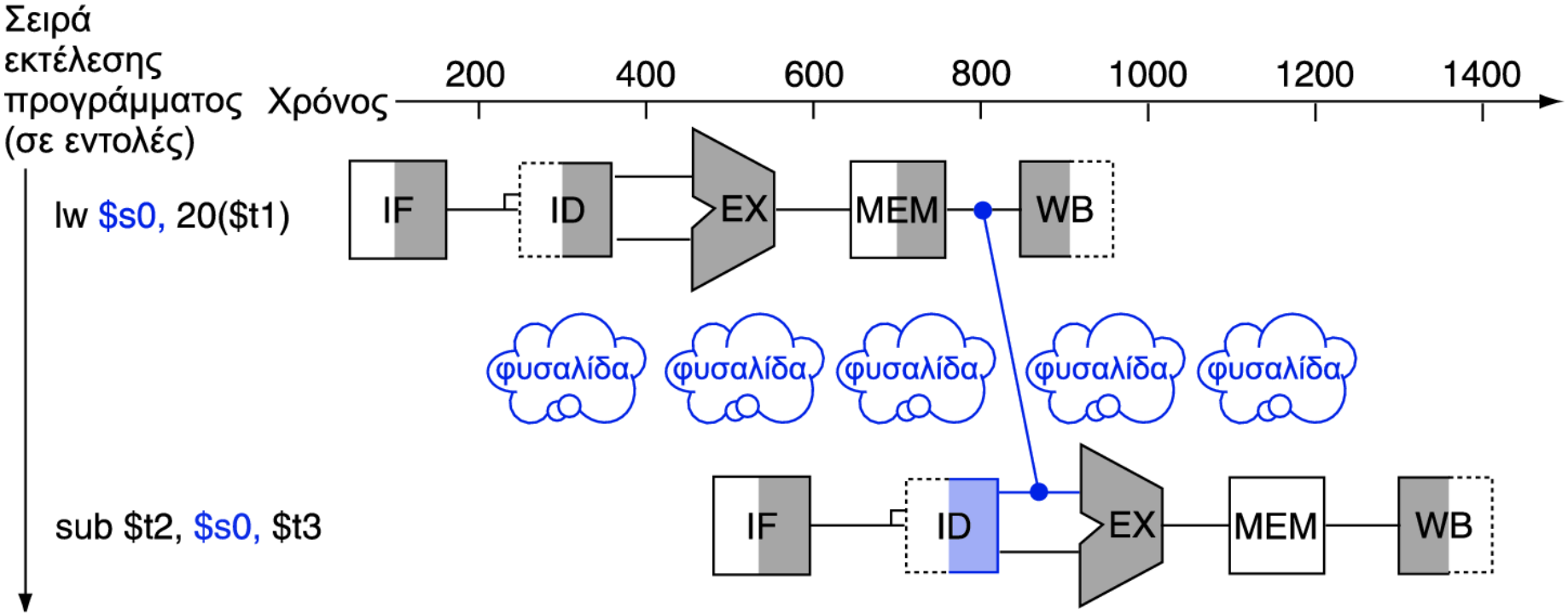
Προώθηση (Forwarding)

- Λέγεται επίσης Παράκαμψη (Bypassing)
- Χρήση του αποτελέσματός όταν δημιουργηθεί
  - Μην περιμένεις να αποθηκευτεί σε καταχωρητή
  - Απαιτεί επιπλέον συνδέσεις στη διαδρομή δεδομένων

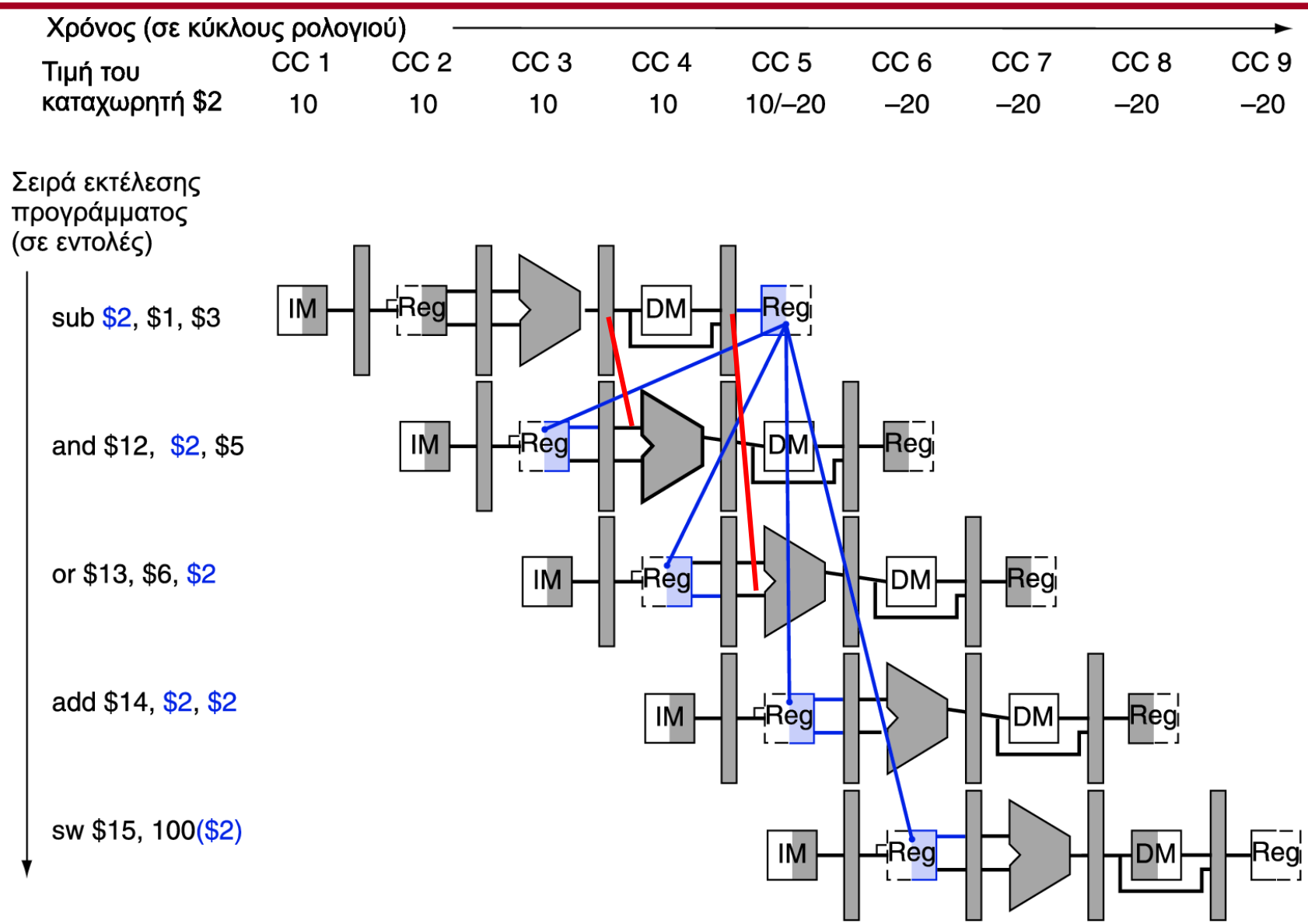


Κίνδυνος δεδομένων Φόρτωσης/Χρήσης

- Φόρτωση/Χρήση (Load/Use)
- Δεν μπορούμε να αποφύγουμε πάντα τις καθυστερήσεις με την προώθηση
  - Αν η τιμή δεν έχει υπολογιστεί όταν χρειάζεται
  - Δεν μπορεί να γίνει προώθηση προς τα πίσω στο χρόνο!



Εξαρτήσεις και προώθηση



Ανίχνευση ανάγκης για προώθηση

- Μεταβίβαση αριθμών καταχωρητών μέσα στη διοχέτευση
  - π.χ., ID/EX.RegisterRs = αριθμός καταχωρητή για το Rs που βρίσκεται στον καταχωρητή διοχέτευσης ID/EX
- Οι αριθμοί καταχωρητών των τελεστέων της ALU στο στάδιο EX δίνονται από τα
  - ID/EX.RegisterRs, ID/EX.RegisterRt
- Κίνδυνοι δεδομένων όταν
  1α. EX/MEM.RegisterRd = ID/EX.RegisterRs
  1β. EX/MEM.RegisterRd = ID/EX.RegisterRt
  2α. MEM/WB.RegisterRd = ID/EX.RegisterRs
  2β. MEM/WB.RegisterRd = ID/EX.RegisterRt

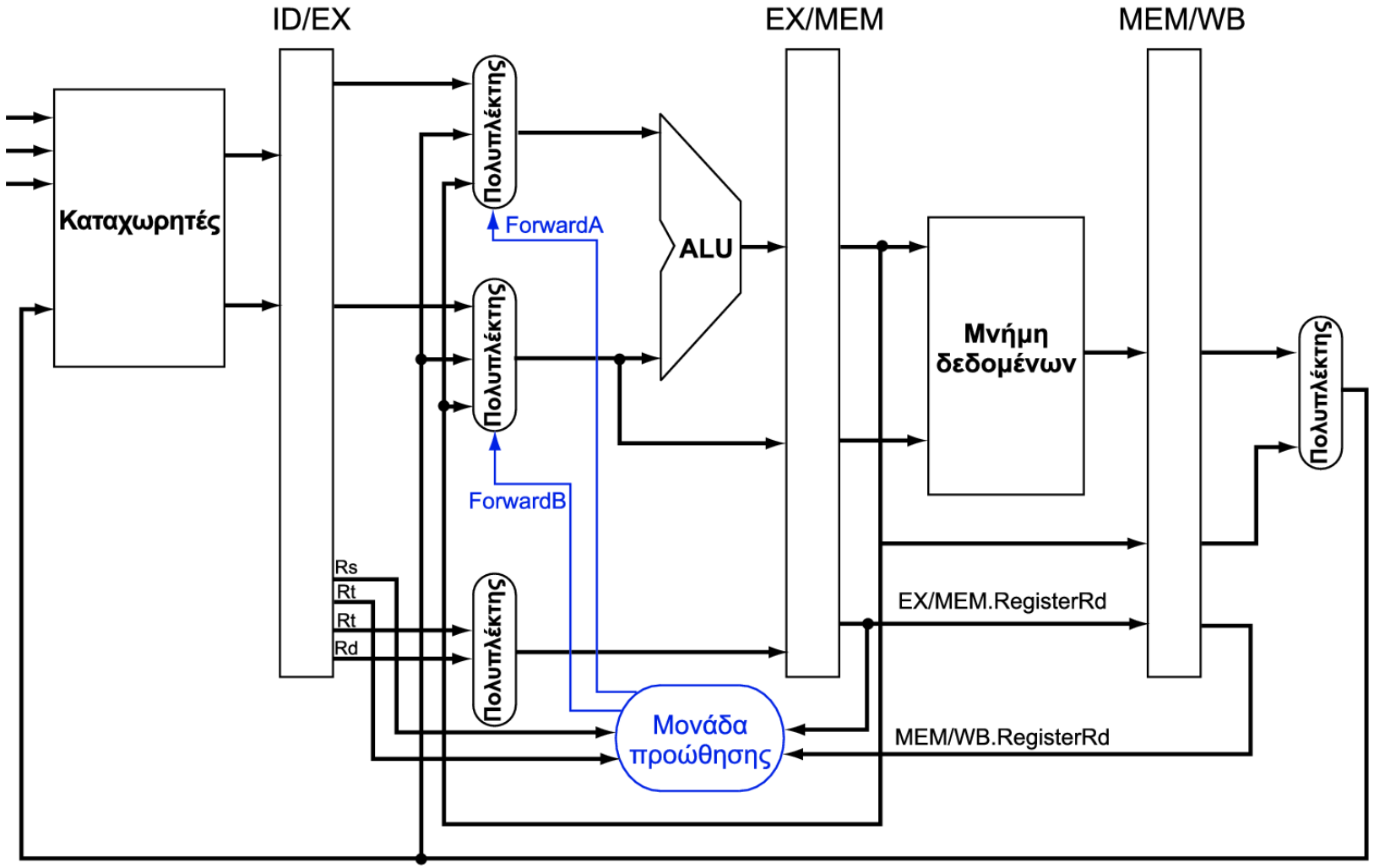
Προωθ. από EX/MEM καταχ. διοχ.

Προώθ. από MEM/WB καταχ. διοχ.

Ανίχνευση ανάγκης για προώθηση

- Αλλά μόνο αν η εντολή που προωθεί πρόκειται να γράψει σε κάποιο καταχωρητή!
  - EX/MEM.RegWrite, MEM/WB.RegWrite
- Και μόνο αν το Rd της εντολής δεν είναι $zero
  - EX/MEM.RegisterRd ≠ 0, MEM/WB.RegisterRd ≠ 0

Διαδρομές προώθησης



β. Με προώθηση

Συνθήκες προώθησης

- Κίνδυνος στο στάδιο EX
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 10
  - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 10
- Κίνδυνος στο στάδιο MEM
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

Διπλός κίνδυνος δεδομένων

- Θεωρήστε την ακολουθία:

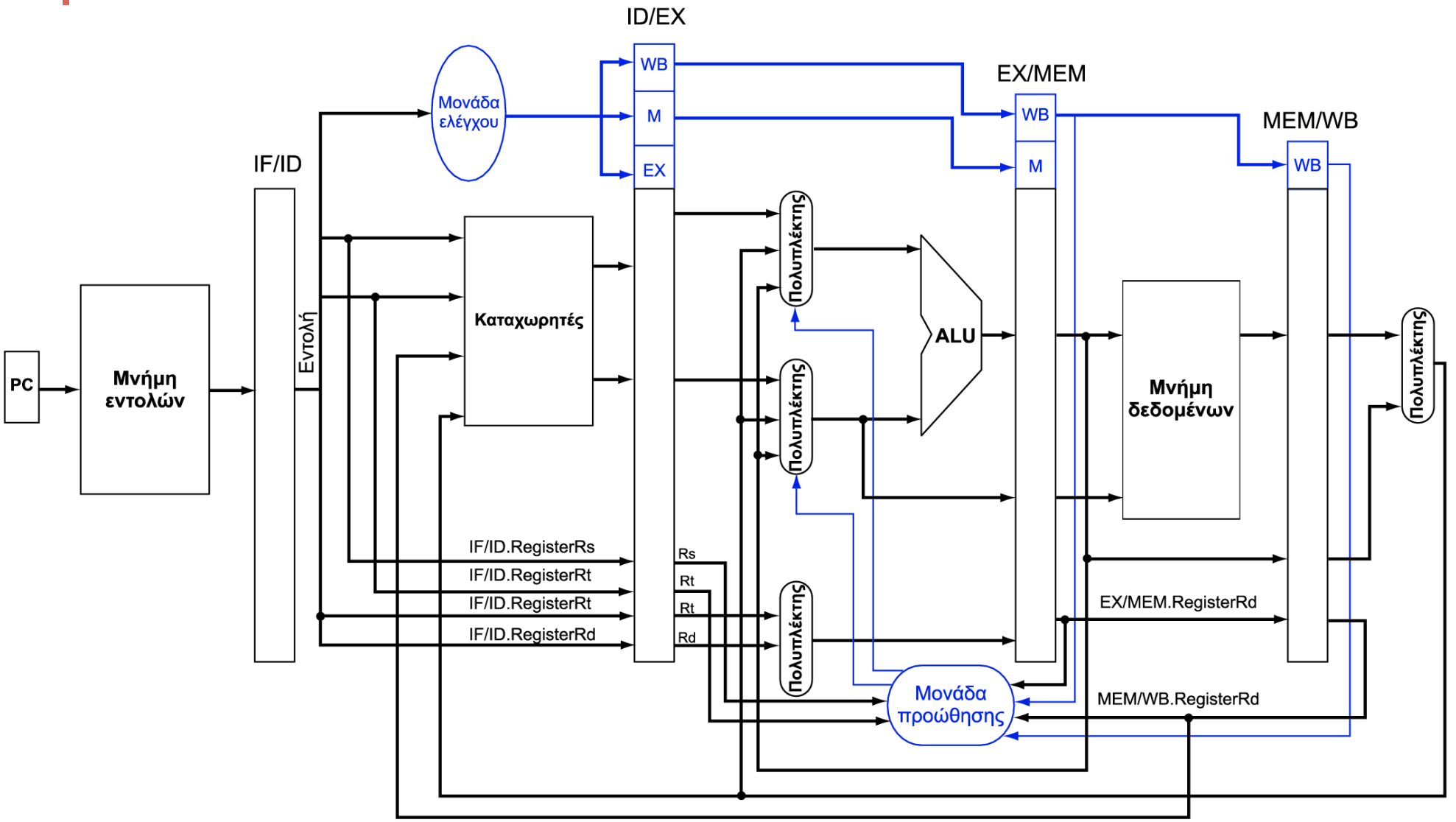
```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

- Συμβαίνουν και οι δύο κίνδυνοι
  - Θέλουμε να χρησιμοποιήσουμε τον πιο πρόσφατο
- Αναθεώρηση της συνθήκης προώθησης του MEM
  - Προώθηση μόνο αν η συνθήκη κινδύνου του σταδίου EX δεν είναι αληθής

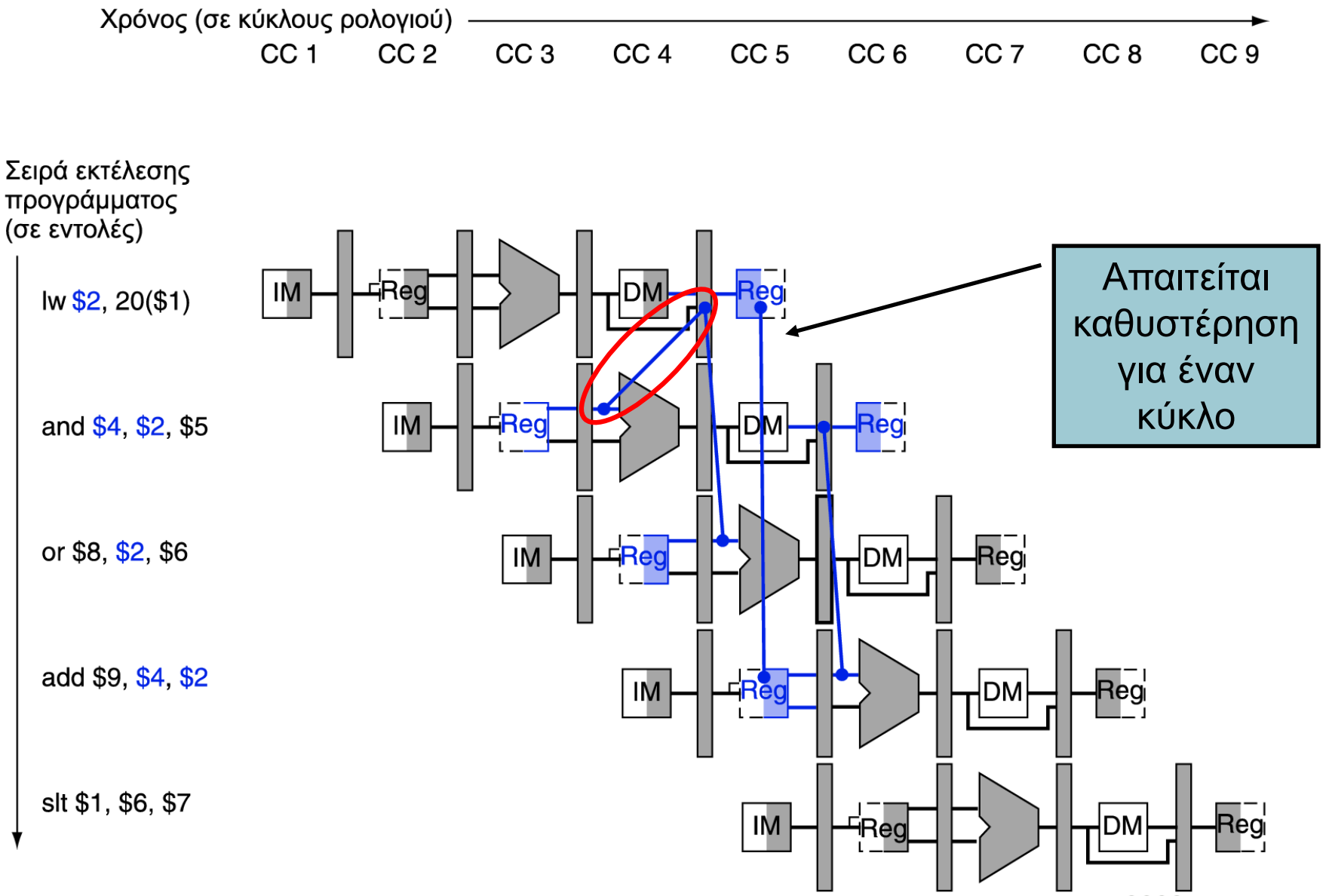
Αναθεωρημένη συνθήκη προώθησης

- Κίνδυνος στο στάδιο MEM
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
    ForwardA = 01
  - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
    and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    ForwardB = 01

Διαδρομή δεδομένων με προώθηση



Κίνδυνος δεδομένων φόρτωσης-χρήσης



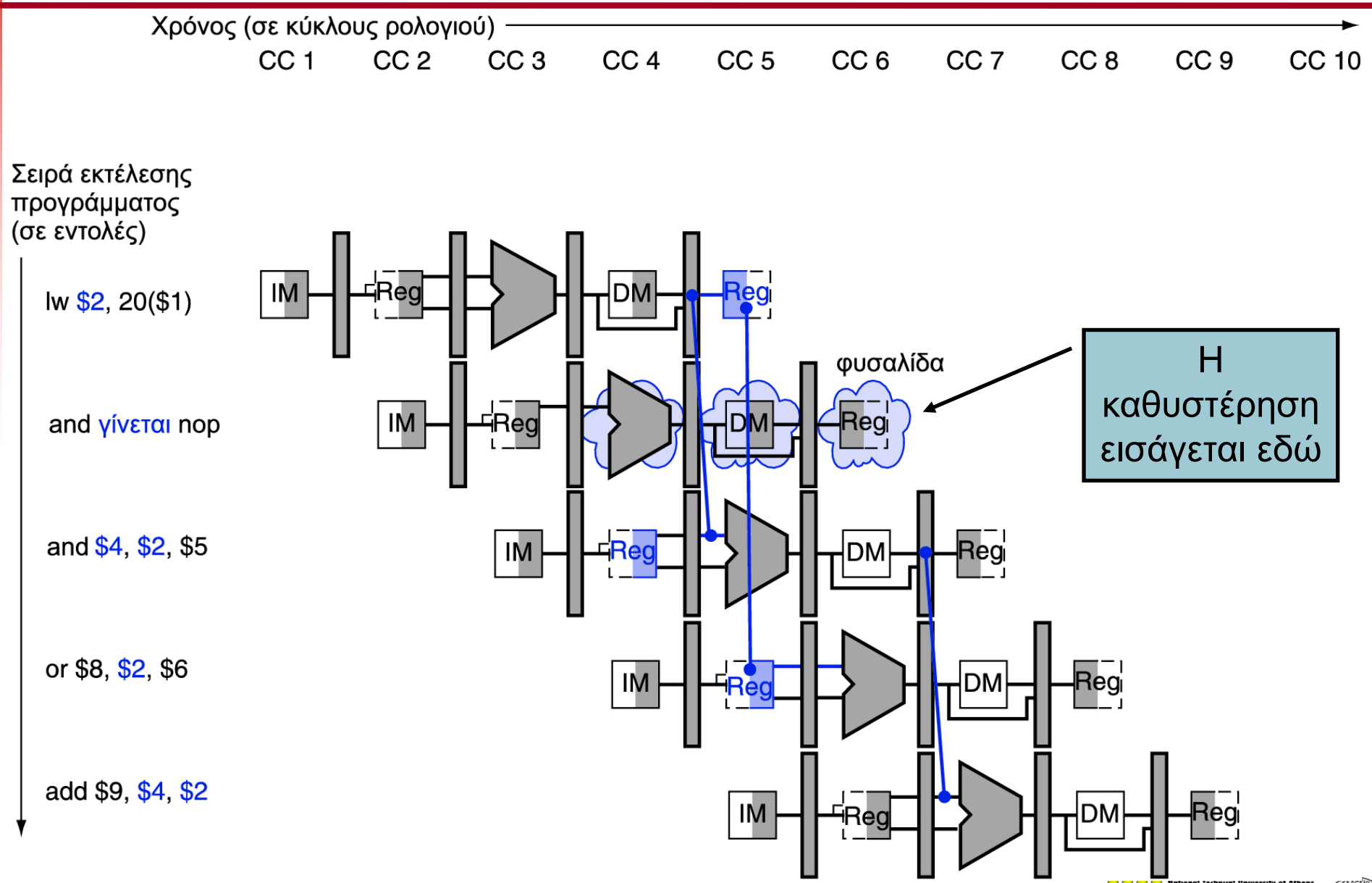
Ανίχνευση κινδύνου φόρτωσης-χρήσης

- Έλεγχος όταν η εντολή που κάνει τη χρήση αποκωδικοποιείται στο στάδιο ID
- Οι αριθμοί των καταχωρητών τελεστέων της ALU στο στάδιο ID δίνονται από τα
  - IF/ID.RegisterRs, IF/ID.RegisterRt
- Κίνδυνος φόρτωσης-χρήσης όταν
  - ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt))
- Αν ανιχνευθεί, γίνεται καθυστέρηση της διοχέτευσης και εισάγεται φυσαλίδα

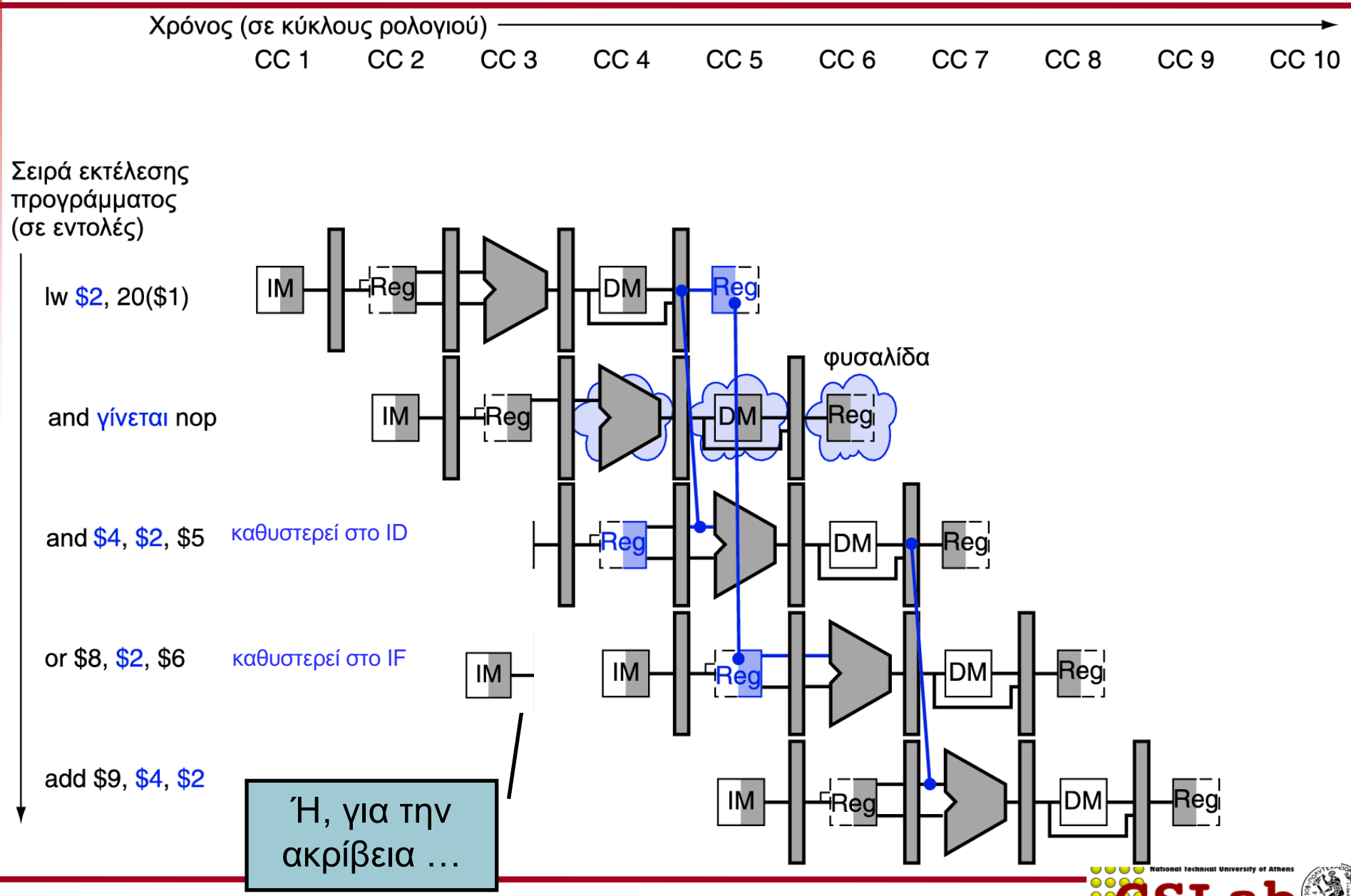
Πώς καθυστερεί η διοχέτευση

- Οι τιμές ελέγχου στον καταχωρητή ID/EX γίνονται 0
  - Τα EX, MEM και WB εκτελούν nop (no-operation, απραξία)
- Αποτρέπεται η ενημέρωση του PC και του καταχωρητή IF/ID
  - Η εντολή που κάνει τη χρήση αποκωδικοποιείται και πάλι
  - Η επόμενη εντολή προσκομίζεται και πάλι
  - Η καθυστέρηση 1 κύκλου επιτρέπει στο MEM να διαβάσει τα δεδομένα για την εντολή lw
    - Μπορεί στη συνέχεια να κάνει προώθηση στο στάδιο EX

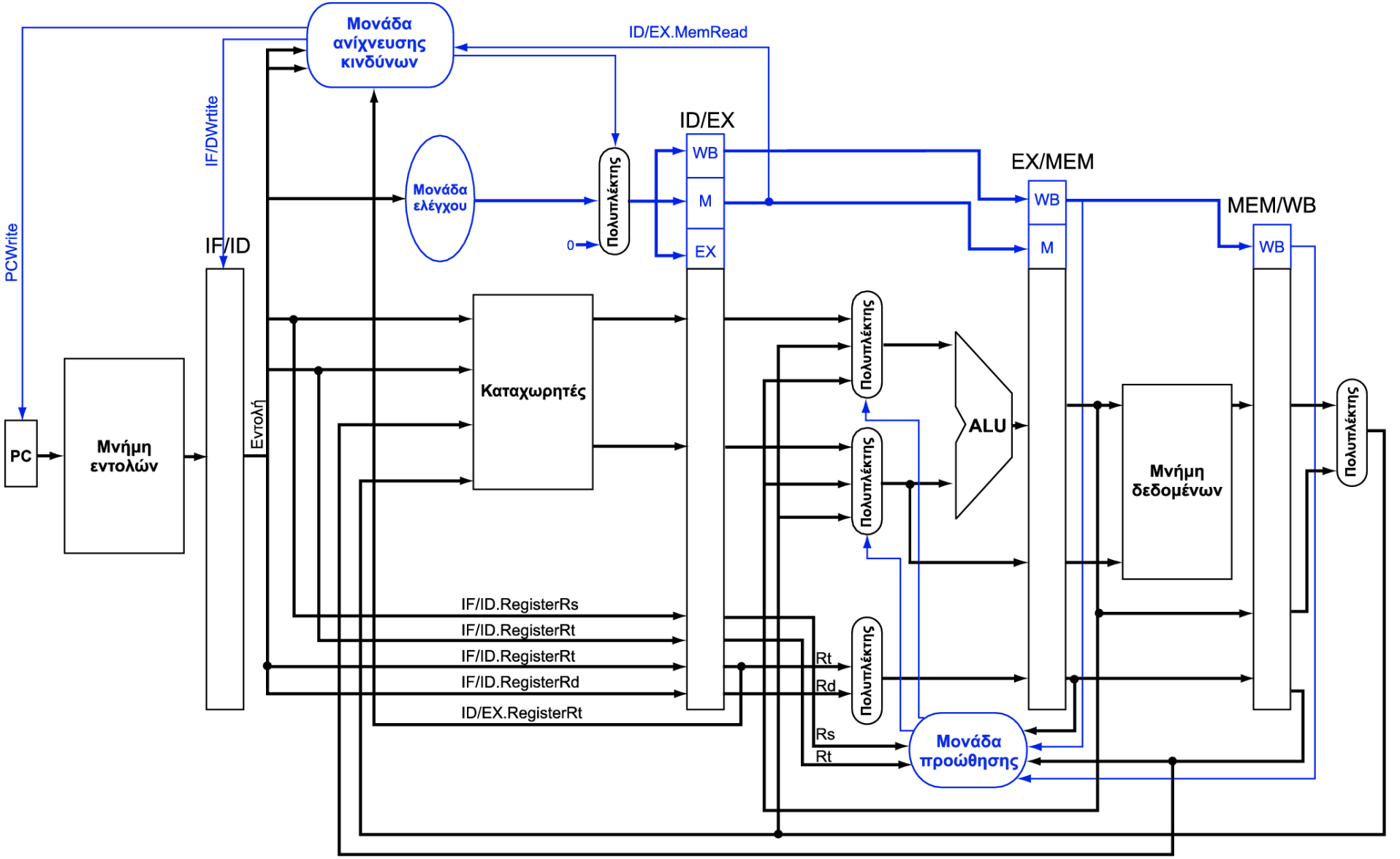
Καθυστέρηση/φυσαλίδα στη διοχέτευση



Καθυστέρηση/φυσαλίδα στη διοχέτευση



Διαδρομή δεδομένων με ανίχνευση κινδύνων



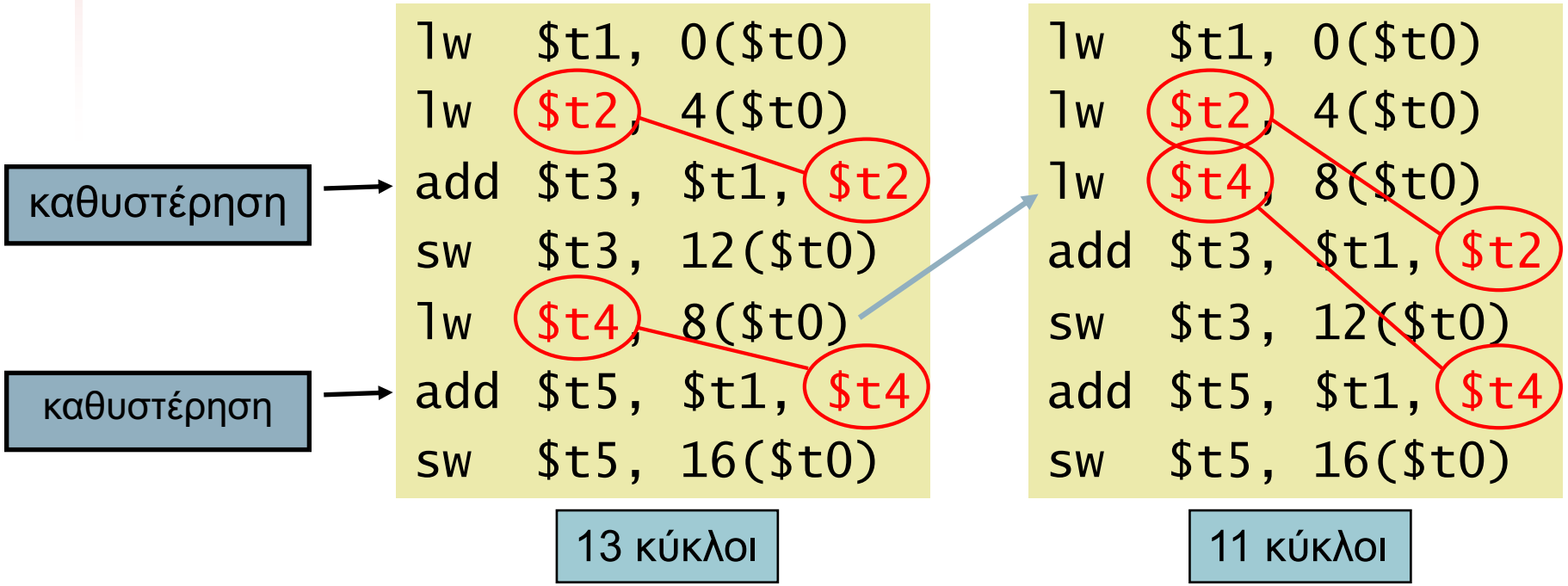
Καθυστερήσεις και απόδοση

ΓΕΝΙΚΗ εικόνα

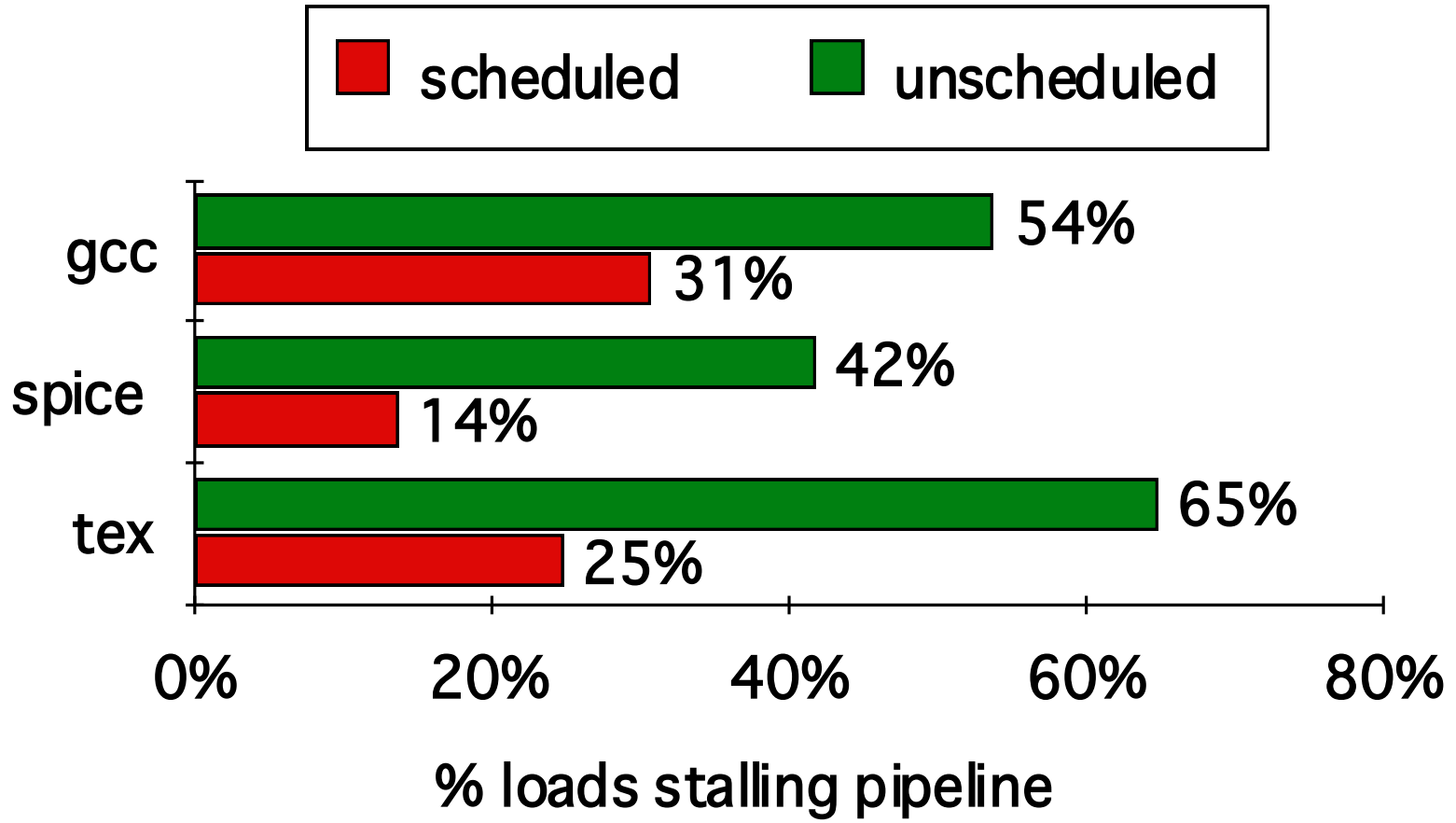
- Οι καθυστερήσεις μειώνουν την απόδοση
  - Αλλά είναι απαραίτητες για να πάρουμε σωστά αποτελέσματα
- Ο μεταγλωττιστής μπορεί να αναδιατάξει τον κώδικα για να αποφευχθούν οι κίνδυνοι και οι καθυστερήσεις
  - Απαιτεί γνώση της δομής της διοχέτευσης

Χρονοπρογρ/μός κώδικα για αποφυγή καθυστερήσεων

- Αναδιάταξη κώδικα για αποφυγή χρήσης του αποτελέσματος της load στην επόμενη εντολή
- Κώδικας C για το `A = B + E; C = B + F;`



Compiler Avoiding Load Stalls:

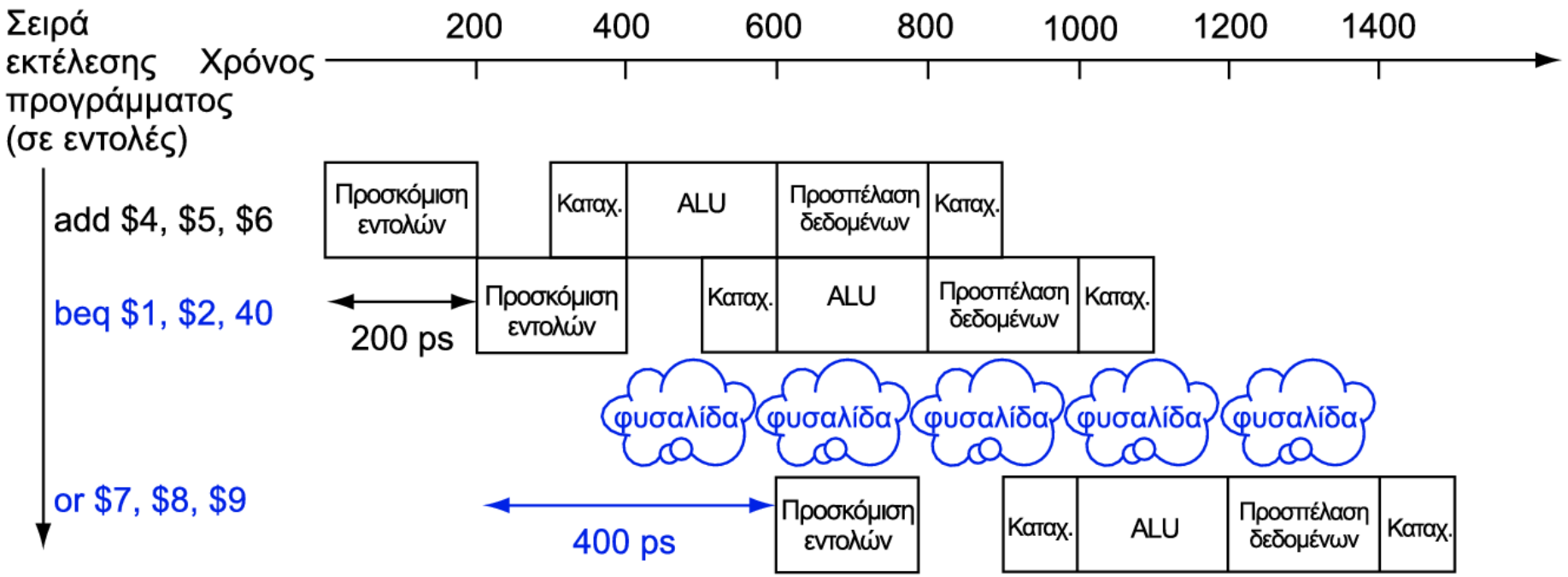


Κίνδυνοι ελέγχου

- Η διακλάδωση καθορίζει τη ροή του ελέγχου (flow of control)
  - Η προσκόμιση της επόμενης εντολής εξαρτάται από το αποτέλεσμα της διακλάδωσης
  - Η διοχέτευση δεν μπορεί να προσκομίσει πάντα τη σωστή εντολή
    - Ακόμη δουλεύει στο στάδιο ID της διακλάδωσης
- Στη διοχέτευση του MIPS
  - Πρέπει να συγκρίνει καταχωρητές και να υπολογίσει τη δ/νση προορισμού νωρίς στη διοχέτευση
  - Προσθήκη υλικού για να γίνουν στο στάδιο ID

Καθυστέρηση σε διακλάδωση

- Stall on branch
- Περίμενε μέχρι να καθοριστεί το αποτέλεσμα της διακλάδωσης πριν προσκομίσεις την επόμενη εντολή

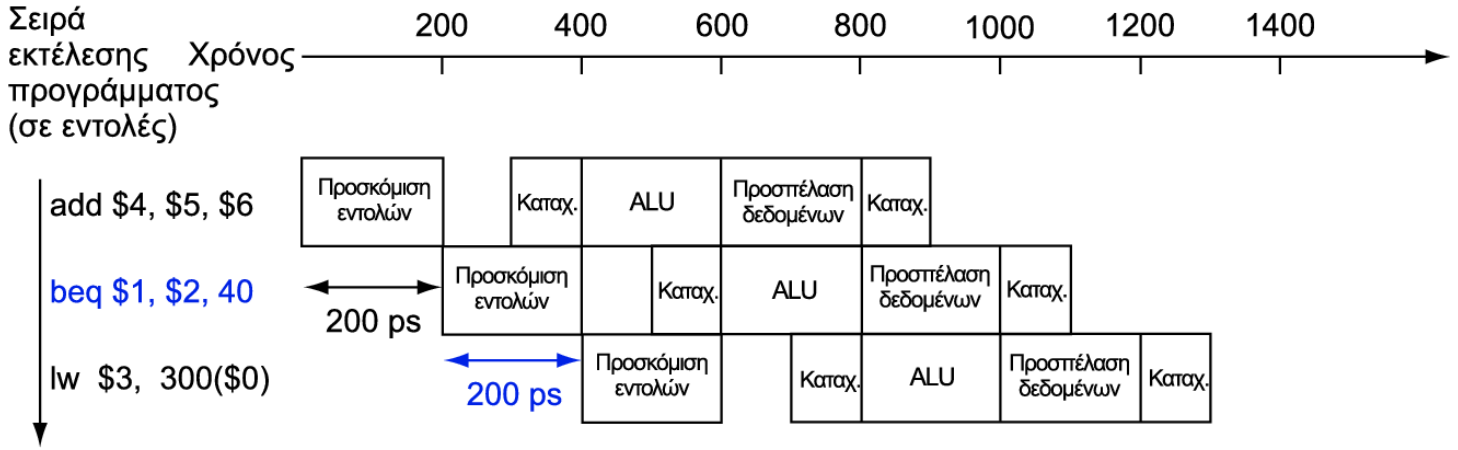


Πρόβλεψη διακλάδωσης

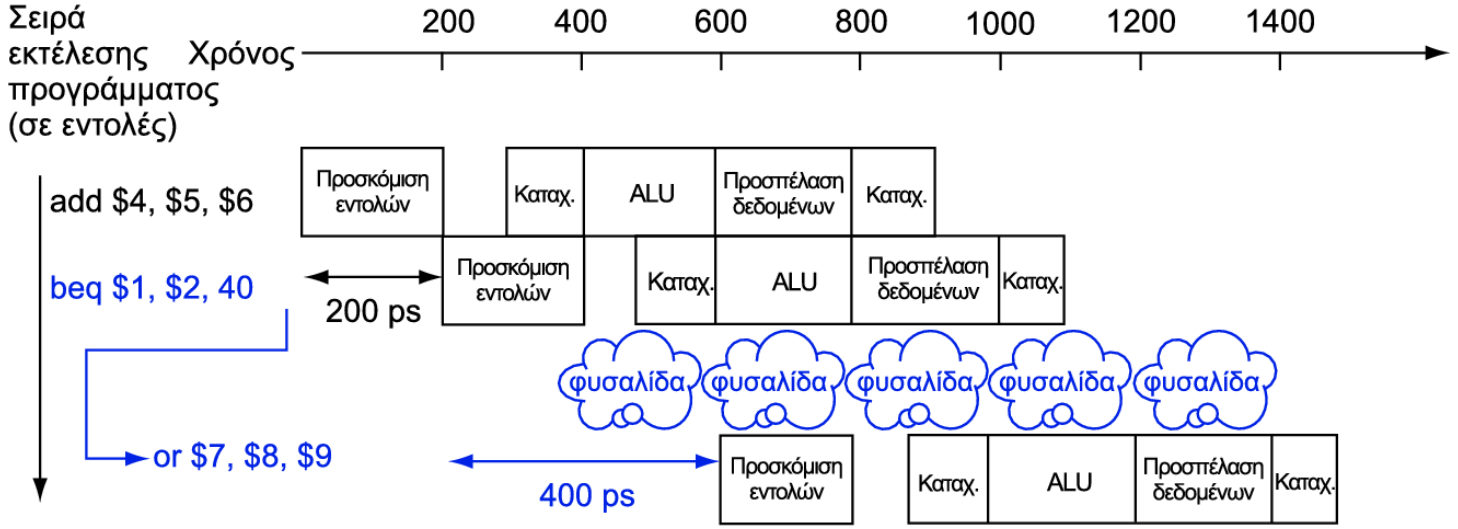
- Branch prediction
- Οι μεγαλύτερες διοχετεύσεις δεν μπορούν να καθορίσουν σύντομα το αποτέλεσμα της διακλάδωσης
  - Η ποινή καθυστέρησης (stall penalty) γίνεται υπερβολικά μεγάλη
- Πρόβλεψη (predict) του αποτελέσματος της διακλάδωσης
  - Καθυστέρηση μόνο αν η πρόβλεψη είναι λανθασμένη
- Στη διοχέτευση του MIPS
  - Μπορεί να γίνει πρόβλεψη μη λήψης της διακλάδωσης (predict not taken)
  - Προσκόμιση της εντολής μετά τη διακλάδωση, χωρίς καθόλου καθυστέρηση

MIPS με πρόβλεψη μη λήψης

Σωστή πρόβλεψη



Λανθασμένη πρόβλεψη



Πρόβλεψη διακλάδωσης

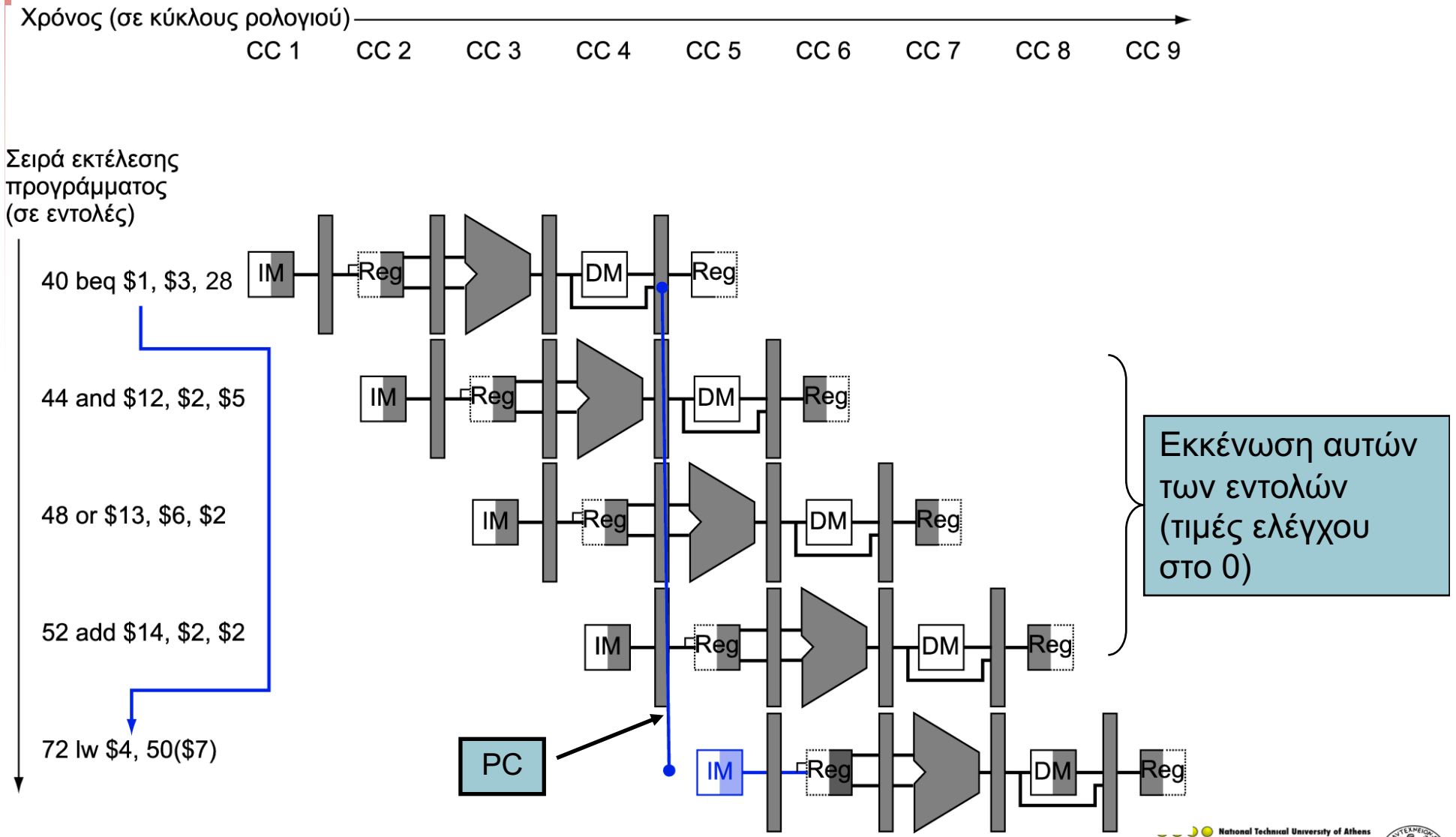
- Μπορεί να είναι λάθος
- Αν η πρόβλεψη είναι σωστή:
  - Κανένα πρόβλημα, εκτέλεση επόμενων εντολών
- Αν είναι λάθος:
  - Ξεκίνησε η ανάκληση και η εκτέλεση εντολών από το λάθος μονοπάτι
  - Οι εντολές αυτές πρέπει να «μην υπάρχουν» στην εκτέλεση
  - Τροποποίηση τους σε NOP (bubbles)
  - Υπάρχουν αλλά *δεν* αλλάζουν την κατάσταση του συστήματος
- Το πρόγραμμα μας λέει «τι» όχι «πως»

Πιο ρεαλιστική πρόβλεψη διακλάδωσης

- Στατική πρόβλεψη διακλάδωσης
  - Βασίζεται στην τυπική συμπεριφορά των διακλαδώσεων
  - Παράδειγμα: διακλαδώσεις σε βρόχους και εντολές if
    - Πρόβλεψη διακλαδώσεων προς τα πίσω (backward branches) ως λαμβανόμενες
    - Πρόβλεψη διακλαδώσεων προς τα εμπρός (forward branches) ως μη λαμβανόμενες
- Δυναμική πρόβλεψη διακλάδωσης
  - Το υλικό μετράει τη πραγματική συμπεριφορά διακλαδώσεων
    - π.χ., καταγράφει την πρόσφατη ιστορία κάθε διακλάδωσης
  - Υποθέτει ότι η μελλοντική συμπεριφορά θα συνεχίσει την τάση
    - Σε περίπτωση λάθους, γίνεται καθυστέρηση κατά την επαναπροσκόμιση, και ενημέρωση του ιστορικού

Κίνδυνοι διακλάδωσης

- Αν το αποτέλεσμα της διακλάδωσης καθορίζεται στο ΜΕΜ

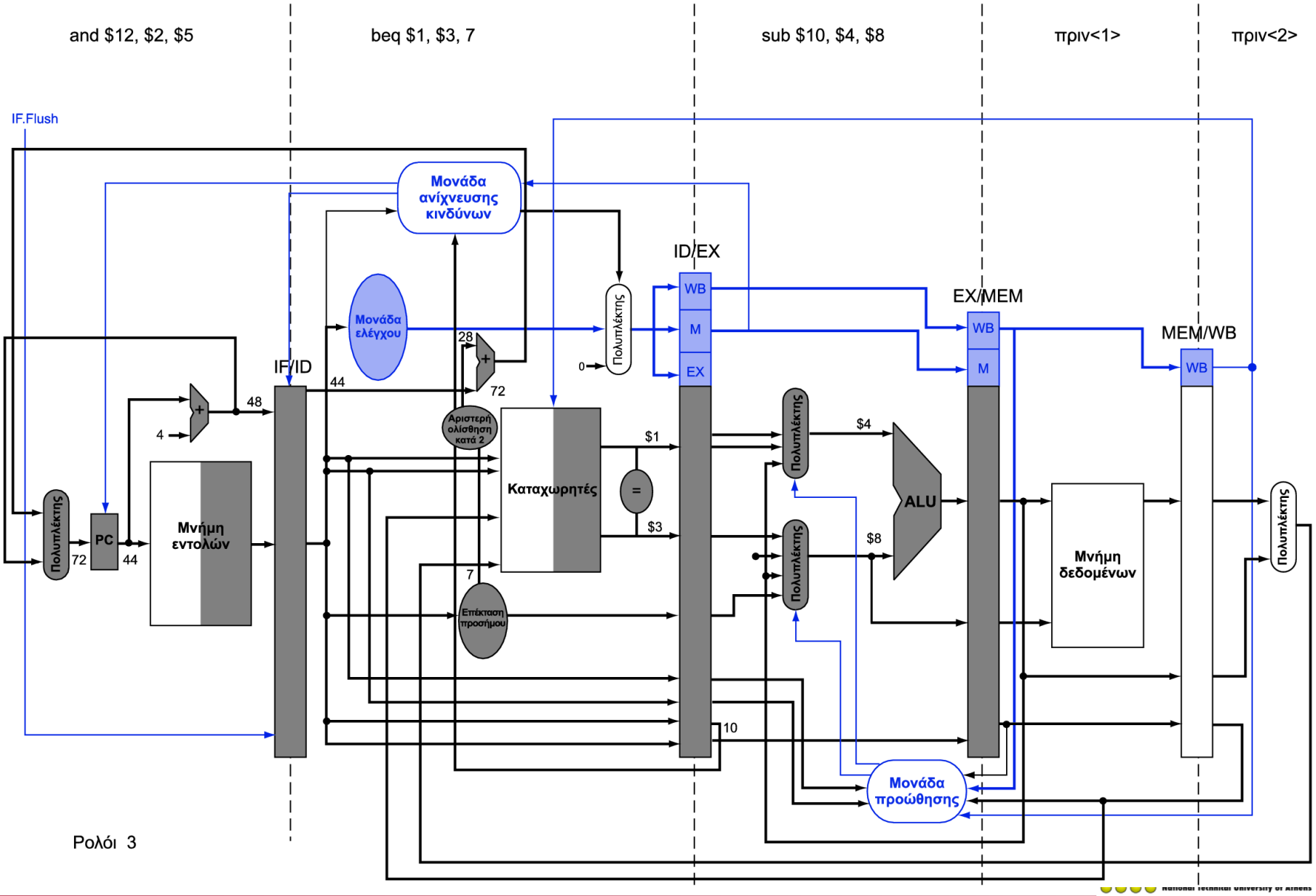


Μείωση καθυστέρησης διακλάδωσης

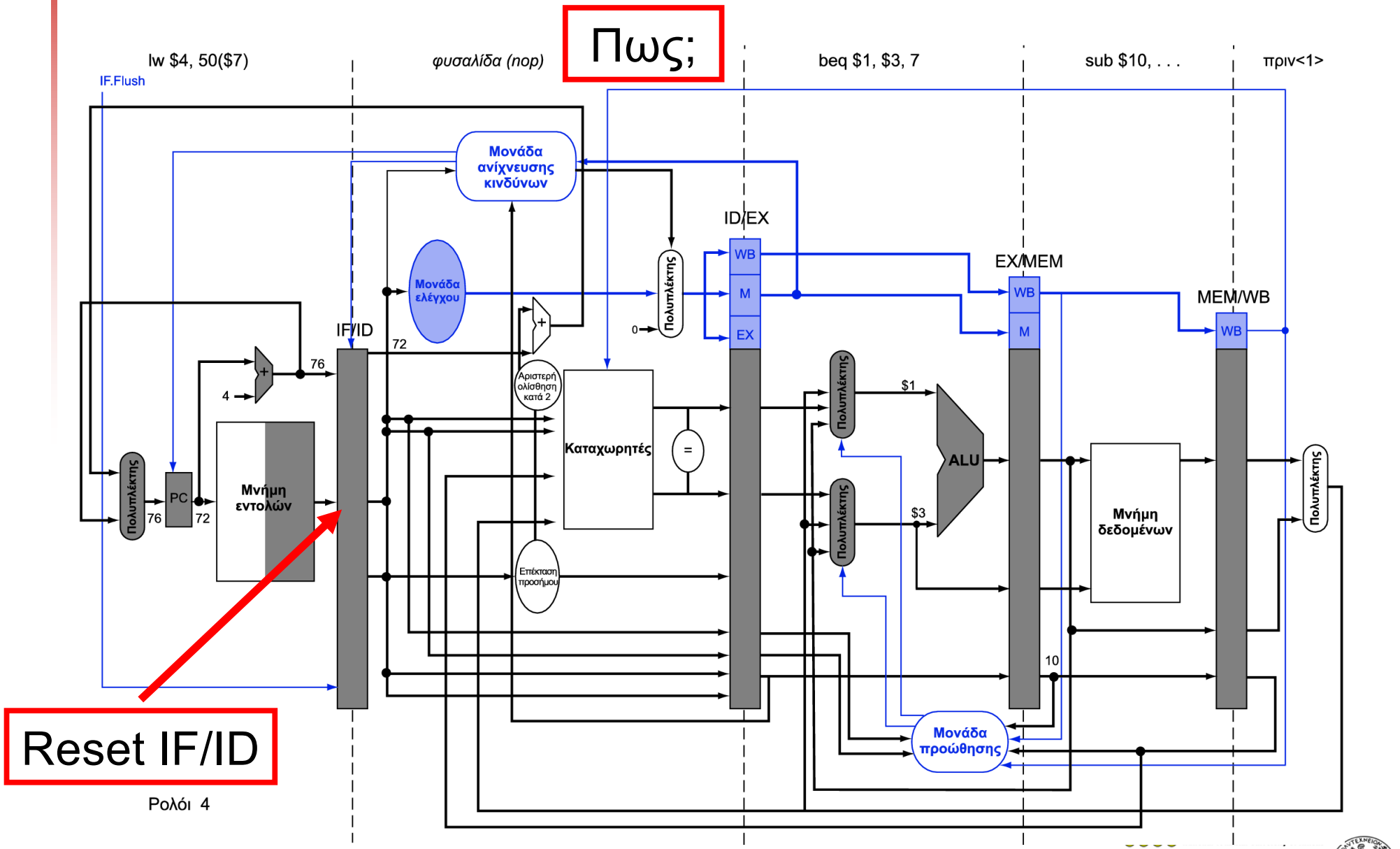
- Μεταφορά υλικού για προσδιορισμό αποτελέσματος στο στάδιο ID
  - Αθροιστής διεύθυνσης προορισμού
  - Συγκριτής καταχωρητών
- Παράδειγμα: λαμβανόμενη διακλάδωση

```
36:   sub   $10, $4, $8
40:   beq   $1,  $3, 7
44:   and   $12, $2, $5
48:   or    $13, $2, $6
52:   add   $14, $4, $2
56:   slt   $15, $6, $7
      ...
72:   lw    $4, 50($7)
```

Παράδειγμα: λαμβανόμενη διακλάδωση

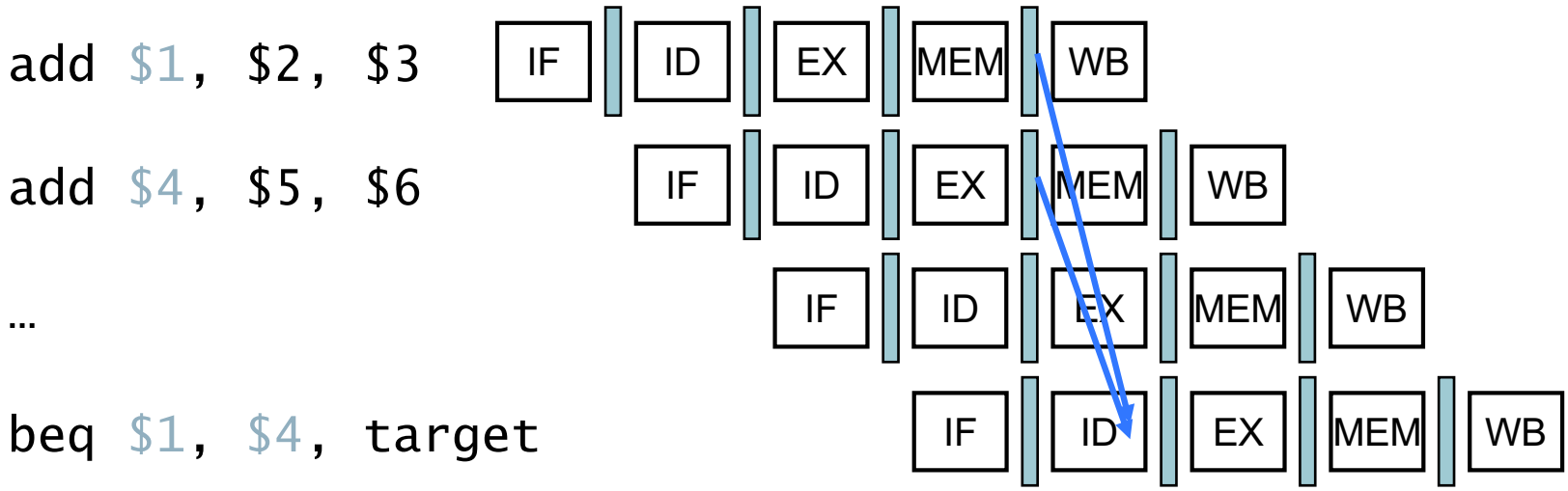


Παράδειγμα: λαμβανόμενη διακλάδωση



Κίνδυνοι δεδομένων για διακλαδώσεις

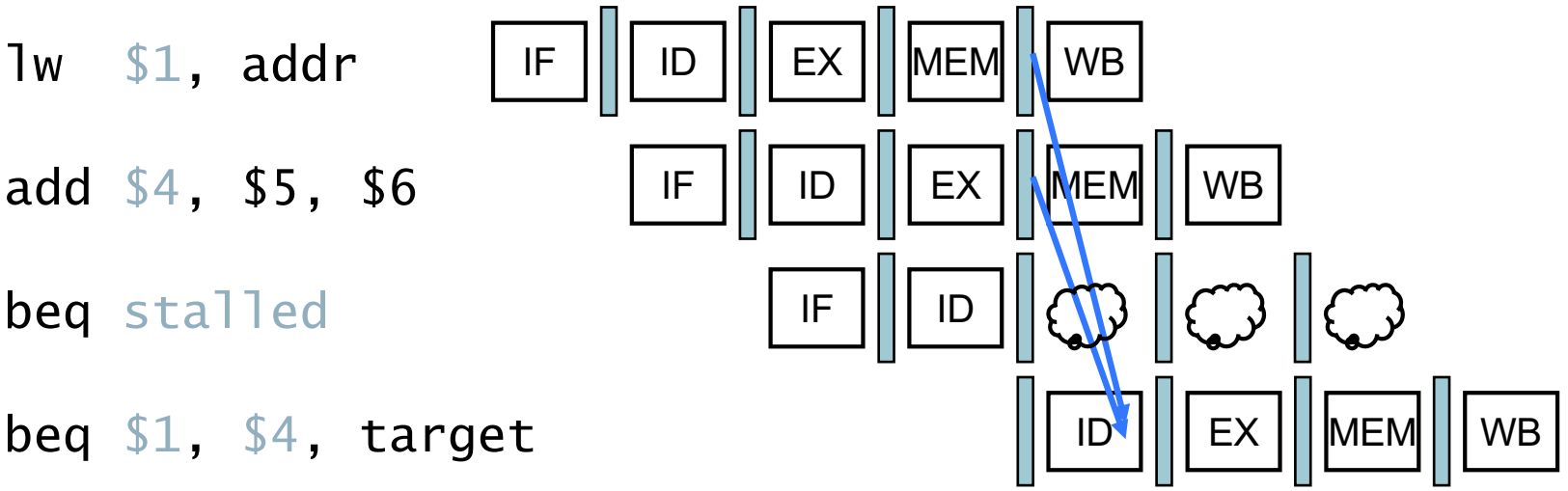
- Αν ένας καταχωρητής σύγκρισης είναι προορισμός εντολής ALU που προηγείται κατά 2 ή 3 θέσεις



- Μπορεί να λυθεί με προώθηση

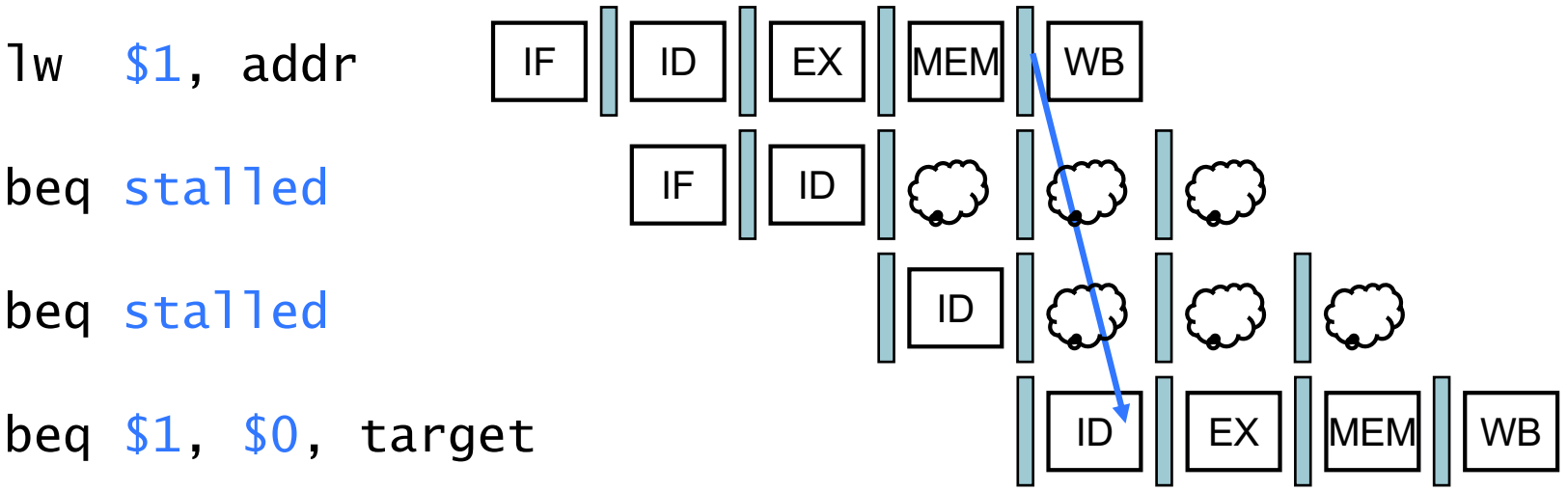
Κίνδυνοι δεδομένων για διακλαδώσεις

- Αν ένας καταχωρητής σύγκρισης είναι προορισμός εντολής ALU που προηγείται αμέσως ή εντολής φόρτωσης που προηγείται κατά 2 θέσεις
  - Χρειάζεται 1 κύκλος καθυστέρησης



Κίνδυνοι δεδομένων για διακλαδώσεις

- Αν ένας καταχωρητής σύγκρισης είναι προορισμός μιας εντολής φόρτωσης που προηγείται αμέσως
  - Χρειάζονται 2 κύκλοι καθυστέρησης

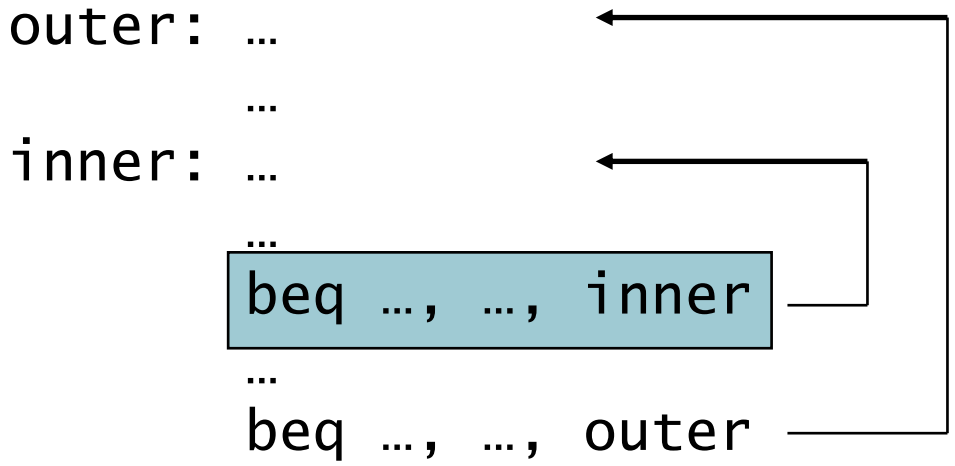


Δυναμική πρόβλεψη διακλάδωσης

- Σε πιο βαθιές και υπερβαθμωτές διοχετεύσεις, η ποινή της διακλάδωσης είναι πιο σημαντική
- Χρήση δυναμικής πρόβλεψης
  - Προσωρινή μνήμη πρόβλεψης διακλάδωσης (branch prediction buffer), που λέγεται και πίνακας ιστορικού διακλάδωσης (branch history table)
  - Δεικτοδοτείται από τις διευθύνσεις της πρόσφατης εντολής διακλάδωσης
  - Αποθηκεύει το αποτέλεσμα (λήψη/μη λήψη)
  - Για εκτέλεση μιας διακλάδωσης
    - Έλεγχος του πίνακα, υπόθεση του ίδιου αποτελέσματος
    - Εκκίνηση προσκόμισης από την επόμενη ή τον προορισμό
    - Αν λάθος, εκκένωση διοχέτευσης και αντιστροφή πρόβλεψης

Διάταξη πρόβλεψης 1 bit: μειονέκτημα

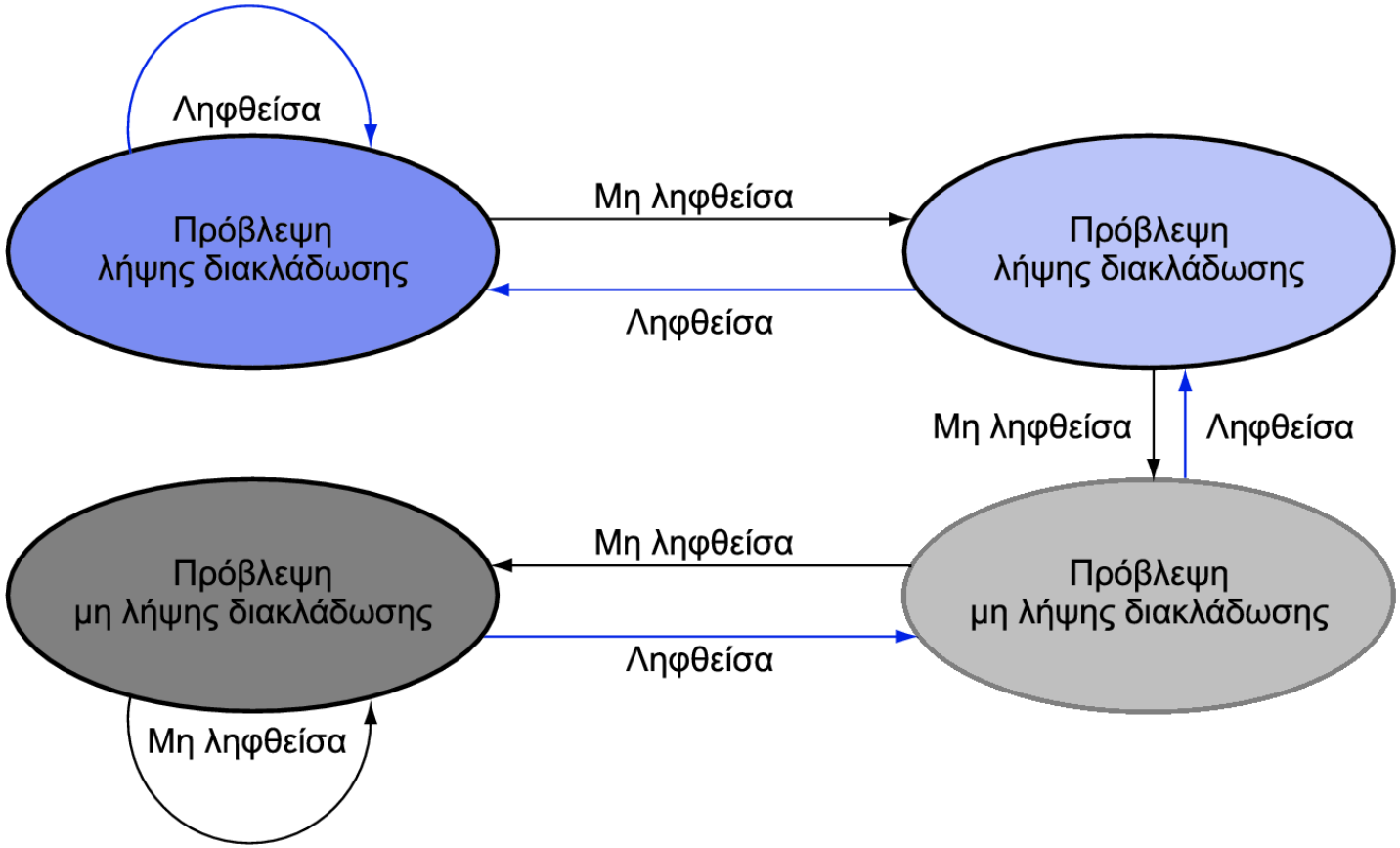
- Οι διακλαδώσεις του εσωτερικού βρόχου προβλέπονται λανθασμένα δύο φορές



  - Λανθασμένη πρόβλεψη λήψης στην τελευταία επανάληψη του εσωτερικού βρόχου
  - Μετά λανθασμένη πρόβλεψη μη λήψης στην πρώτη επανάληψη του εσωτερικού βρόχου την επόμενη φορά που θα εκτελεστεί

Διάταξη πρόβλεψης των 2 bit

- Αλλάζει η πρόβλεψη μόνο μετά από δύο διαδοχικές λανθασμένες προβλέψεις



Dynamic Branch Prediction

- Why does prediction work?
  - 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
- Is dynamic branch prediction better than static branch prediction?
  - Seems to be
  - There are a small number of important branches in programs which have dynamic behavior
  - Loops, loops, loops…

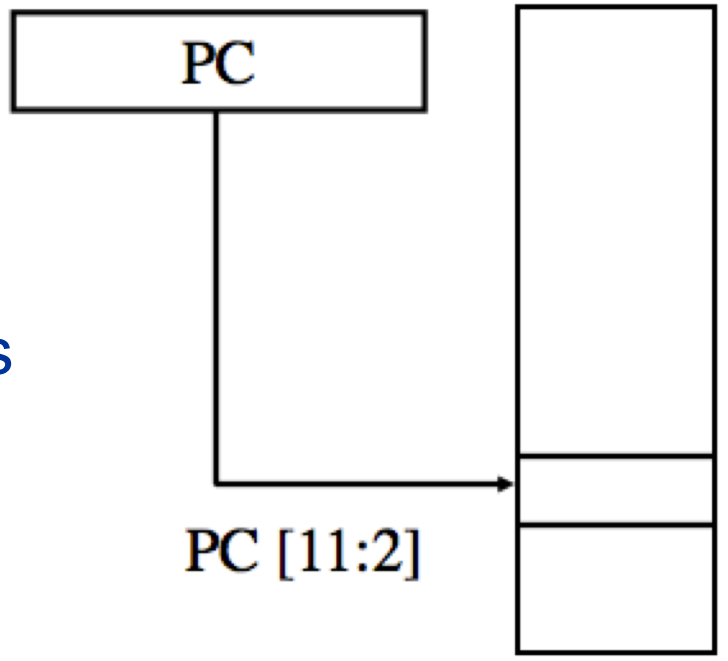
Dynamic Branch Prediction

- Performance = f(accuracy, cost of misprediction)
- Branch History Table (BHT): use lower bits of PC address to index a table of 1-bit values
  - Record whether or not branch taken last time
  - No address check (i.e. not a cache, why?)

Predicting Conditional Branch Outcome

Simplest dynamic branch prediction scheme uses a branch-prediction buffer or branch history table

- Small table indexed by the lower portion of the branch address
- Stores previous branch outcomes to predict next outcome
- Table is not tagged: entry may store the outcome of a different branch (Aliasing)
  - Smaller size, aliasing OK?
  - Correctness? Performance?



1K-entry prediction buffer

Dynamic Branch Prediction

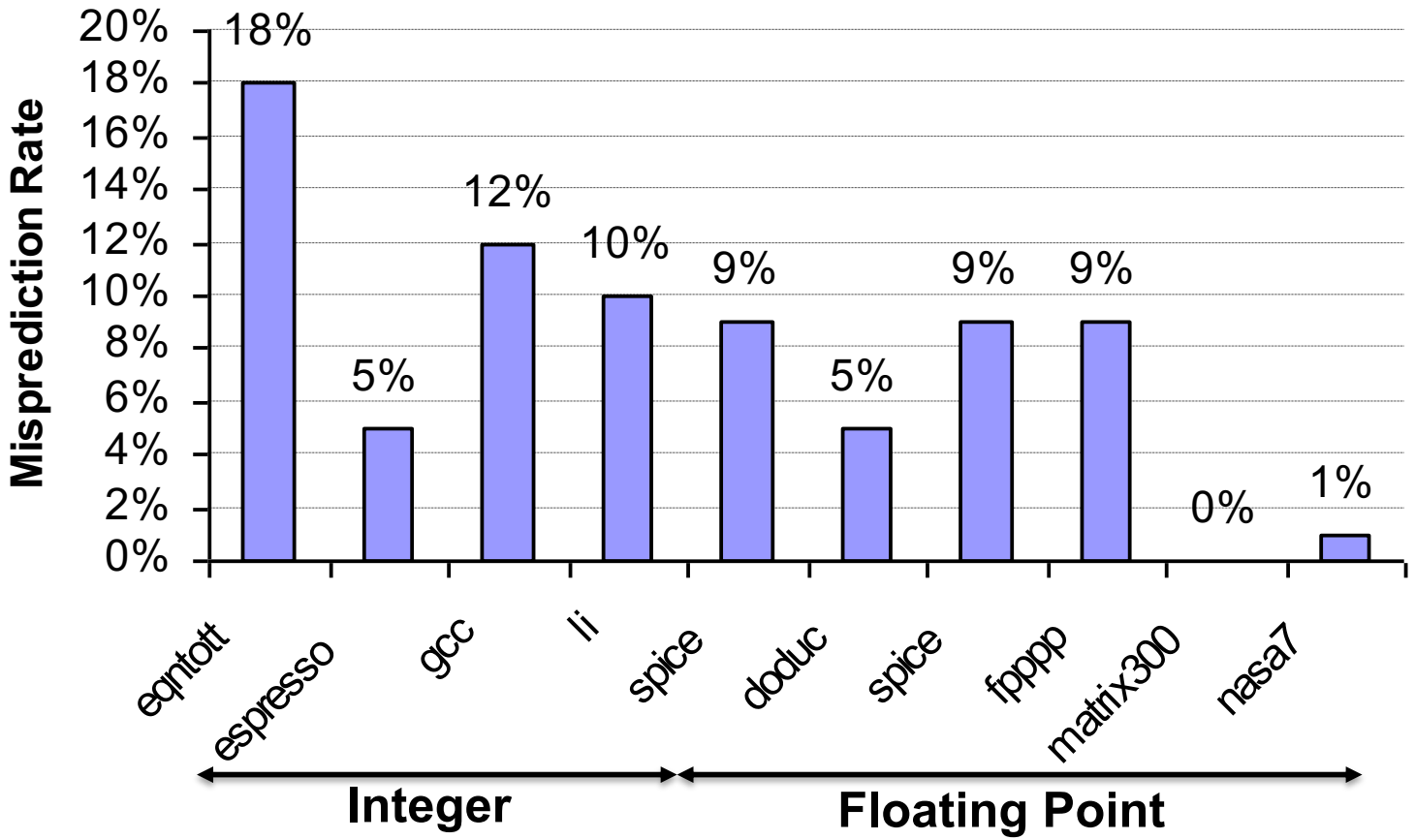
- Initialize table with zeros (not-taken)
- Loop 10 times:
  - Outcome: T T T T T T T T T N, T T …
  - Prediction: N T T T T T T T T T, N T …
  - Correct?: W C C C C C C C C W, W C …
- Problem: in a loop, 1-bit BHT will cause two mispredictions (avg is 9 iterations before exit):
  - End of loop case, when it exits instead of looping as before
  - First time through loop on next time through code, when it predicts exit instead of looping
  - What if I initialize the table to 1s (taken)?

Loop w/ 2-bit history

- Initialize table Weak-Taken
- Loop 10 times:
  - Outcome: T T T T T T T T T N, T T …
  - Prediction: T T T T T T T T T T, T T …
  - Correct?: C C C C C C C C C W, C C …
- 90% correct, 10% wrong, which one to use?
- Initialize table to Weak-Not-taken?
  - Outcome: T T T T T T T T T N, T T …
  - Prediction: N T T T T T T T T T, T T …
  - Correct?: W C C C C C C C C W, C C …
  - First one wrong, after that 90% correct

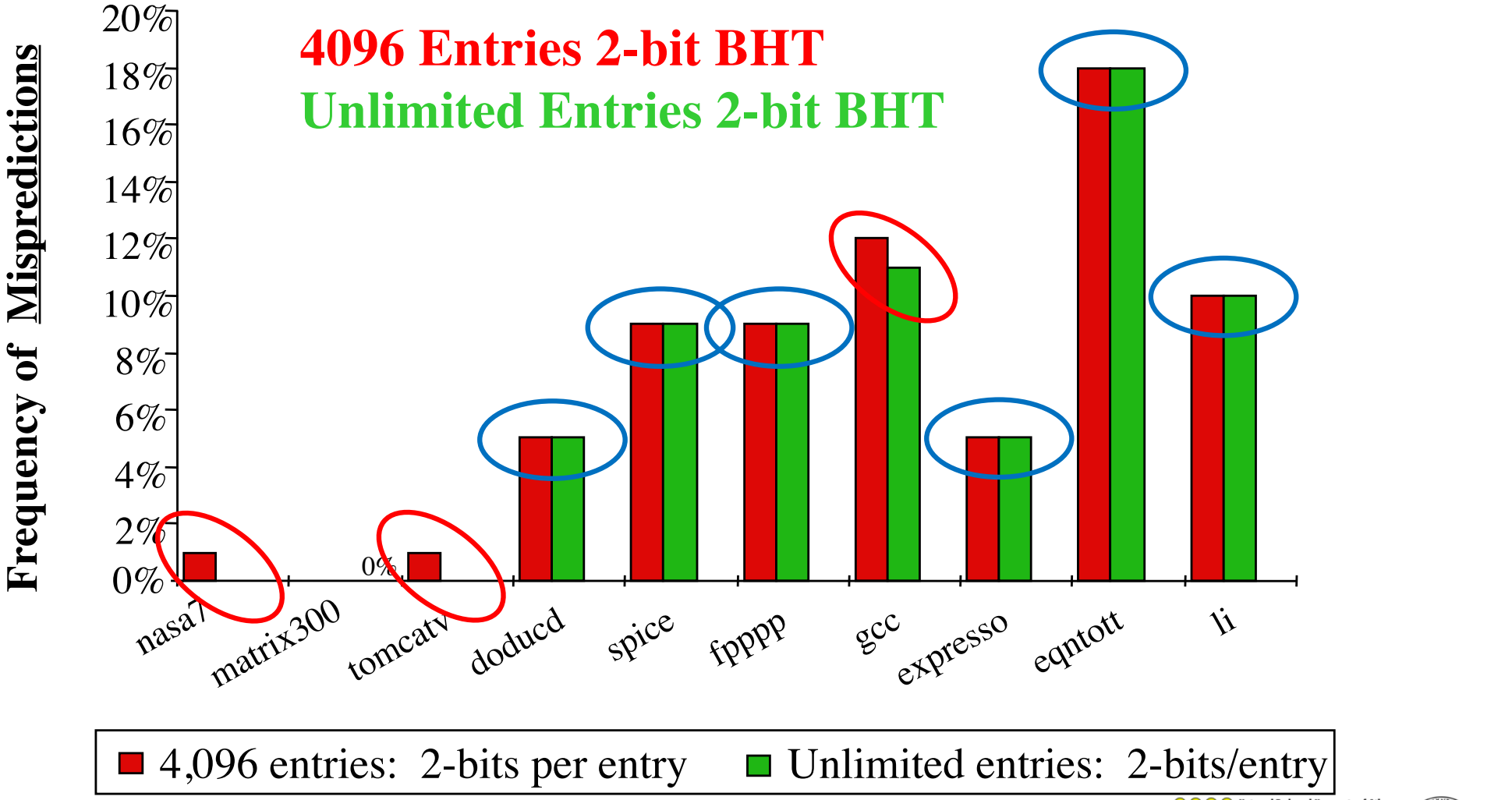
BHT Accuracy

- Mispredict because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch indexing the table
- 4Kentry table:



What if I had infinite size BHT?

4KEntry 2-bit BHT ~= Unlimited Entry 2-bit BHT



Υπολογισμός προορισμού διακλάδωσης

- Ακόμη και με πρόβλεψη, πρέπει ακόμη να υπολογιστεί η διεύθυνση διακλάδωσης
  - Ποινή 1 κύκλου για λαμβανόμενη διακλάδωση
- Προσωρινή μνήμη προορισμού διακλάδωσης (branch target buffer)
  - Κρυφή μνήμη για διευθύνσεις προορισμού
  - Δεικτοδοτείται από τον PC όταν προσκομίζεται η εντολή
    - Αν υπάρχει ευστοχία (hit) και η εντολή είναι διακλάδωση με πρόβλεψη λήψης, μπορεί να γίνει άμεση προσκόμιση του προορισμού

Εξαιρέσεις και διακοπές

- «Μη αναμενόμενα» συμβάνται που απαιτούν αλλαγή της ροής του ελέγχου
  - Διαφορετικές αρχιτεκτονικές συνόλου εντολών χρησιμοποιούν τους όρους διαφορετικά
- Εξαίρεση (exception)
  - Παρουσιάζεται μέσα στη CPU
    - π.χ., μη ορισμένος κωδικός λειτουργίας (undefined opcode), υπερχείλιση (overflow), κλήση συστήματος (syscall), …
- Διακοπή (interrupt)
  - Από έναν εξωτερικό ελεγκτή εισόδου/εξόδου
- Δύσκολος ο χειρισμός τους χωρίς να θυσιαστεί απόδοση

Χειρισμός εξαιρέσεων

- Στο MIPS, τις εξαιρέσεις διαχειρίζεται ένας Συνεπεξεργαστής Ελέγχου Συστήματος (System Control Coprocessor), ο CP0
- Αποθήκευση του PC της εντολής που διακόπτεται
  - Στο MIPS: Exception Program Counter (EPC)
- Αποθήκευση της ένδειξης του προβλήματος
  - Στον MIPS: καταχωρητής Cause (Αιτίου)
  - Υποθέτουμε 1 bit μόνο
    - 0 για μη ορισμένο κωδικό λειτουργίας (undefined opcode), και 1 για υπερχείλιση
- Άλμα στο χειριστή στη δ/νση 8000 00180

Παράδειγμα Διακοπής προγράμματος

```
…
…
add  $t0, $s0, 14

sub  $t1, $t0, $s3
ori   $t0, $t0, 13
lw    $s6, 104($t0)
…
…
```

Εξωτερική Διακοπή

```
# Interrupt Handler
mfc0 …# διάβασμα status reg
…
… # εξυπηρέτηση συσκευής
…
…
rfe # Return From Exception
```

Η εκτέλεση του Interrupt Handler παρεμβάλλεται μεταξύ της add και της sub, και μετά η εκτέλεση του αρχικού προγράμματος συνεχίζεται κανονικά

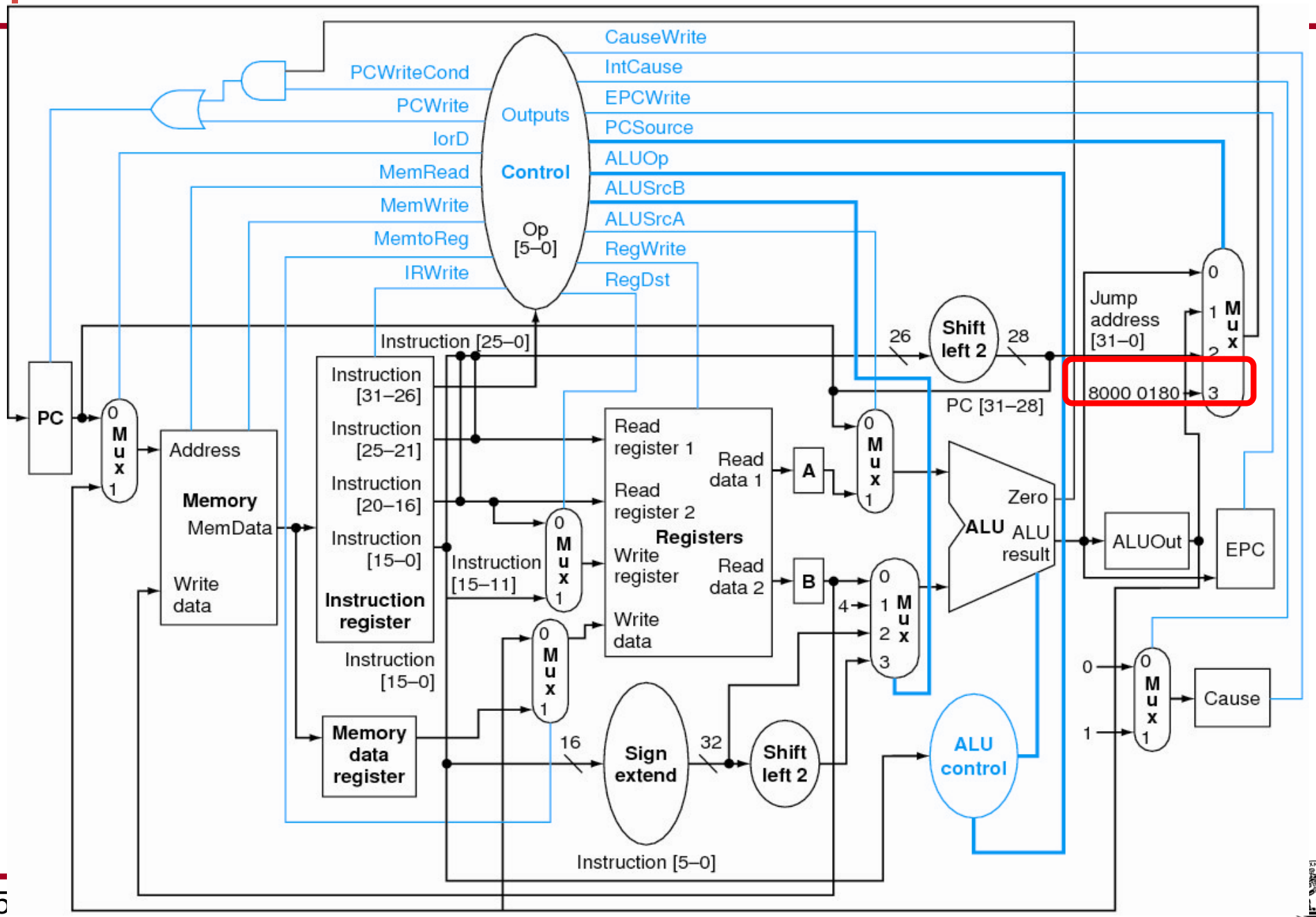
Εναλλακτικός μηχανισμός

- Διανυσματικές διακοπές (vectored interrupts)
  - Η δ/νση του χειριστή καθορίζεται από την αιτία
- Παράδειγμα:
  - Μη ορισμένος opcode: C000 0000
  - Υπερχείλιση: C000 0020
  - …: C000 0040
- Οι εντολές είτε
  - Ασχολούνται με τη διακοπή, είτε
  - Κάνουν άλμα στον πραγματικό χειριστή

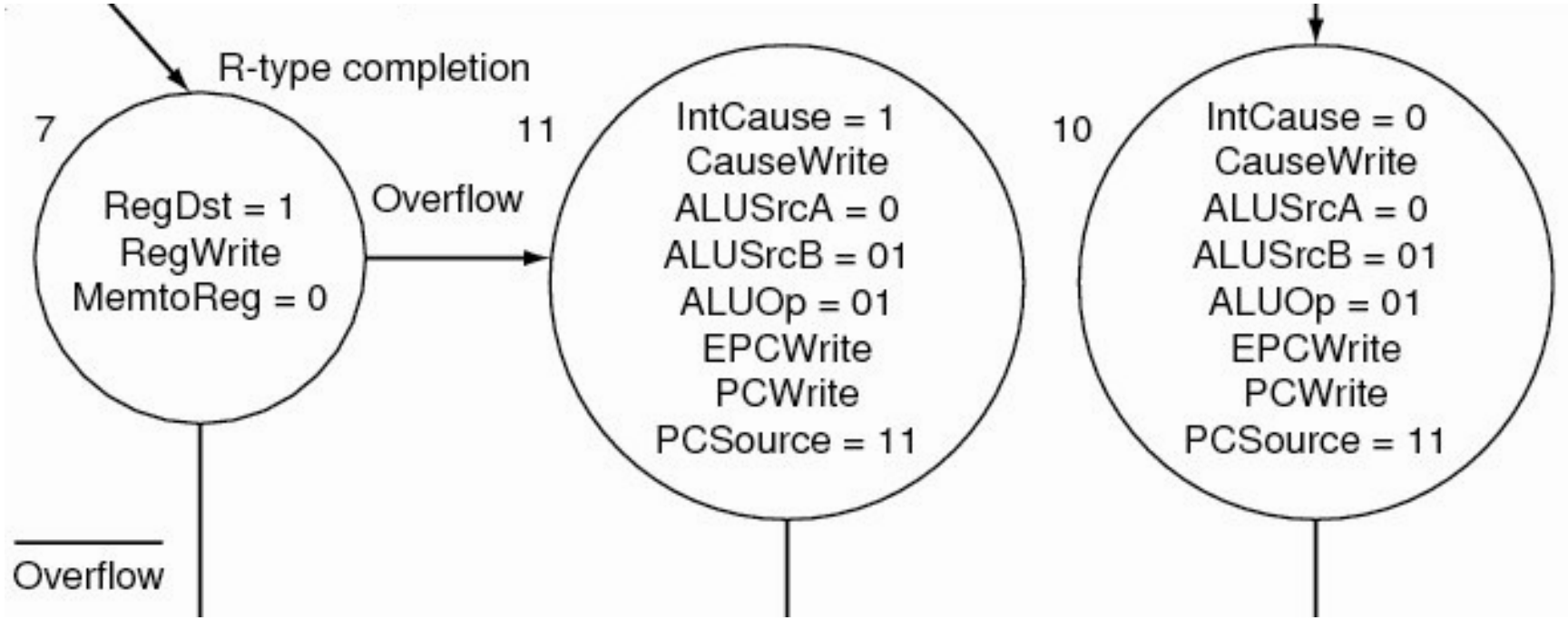
Ενέργειες του χειριστή

- Ανάγνωση αιτίου, και μετάβαση στο σχετικό χειριστή
- Καθορισμός απαιτούμενης ενέργειας
- Αν η εντολή είναι επανεκκινήσιμη (restartable)
  - Εκέλεσεση διορθωτικής ενέργειας
  - Χρήση του EPC για επιστροφή στο πρόγραμμα
- Αλλιώς
  - Τερματισμός προγράμματος
  - Αναφορά σφάλματος με χρήση του EPC, του αιτίου, …

Υποστήριξη Εξαιρέσεων – multi-cycle



Υποστήριξη Εξαιρέσεων – multi-cycle



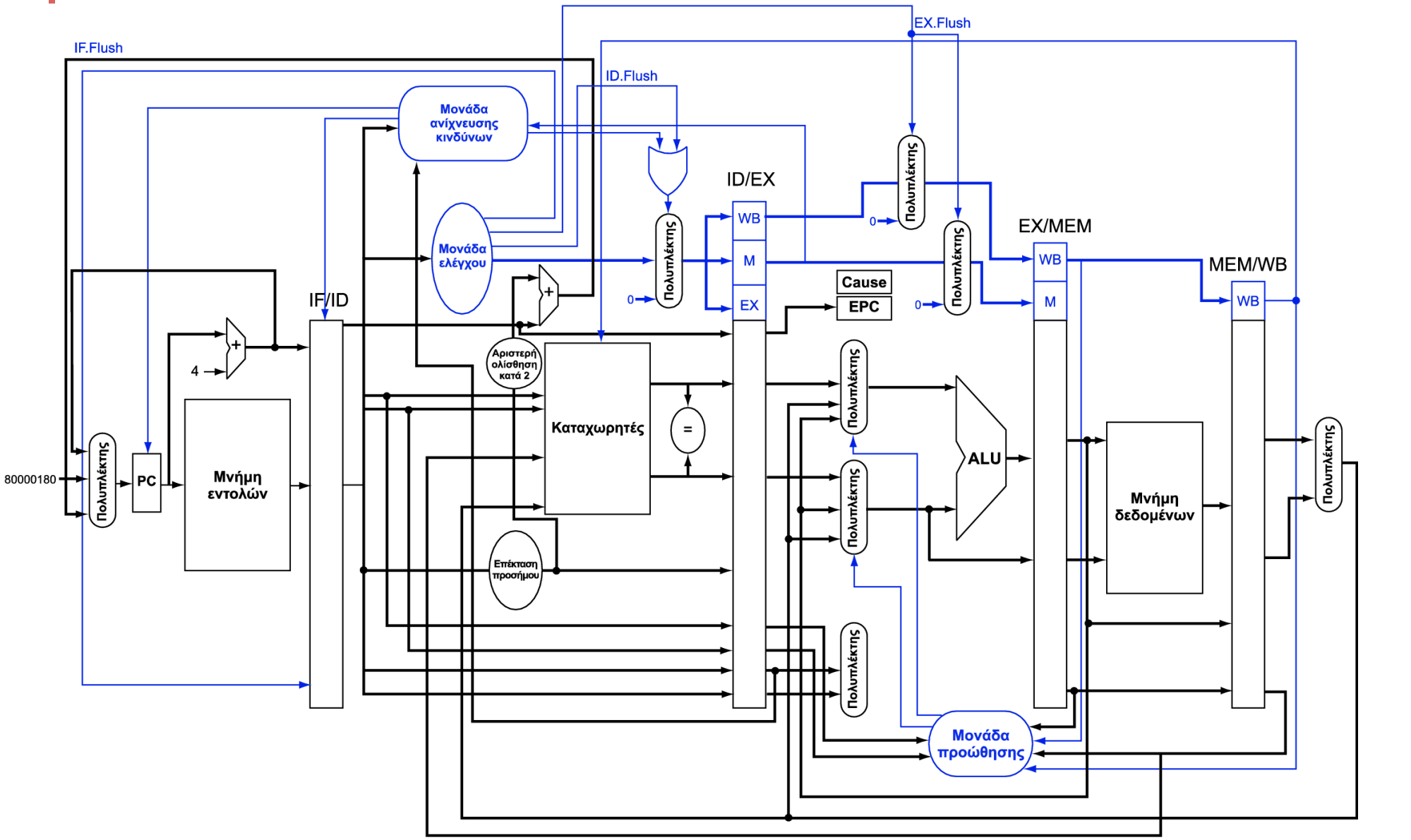
Εξαιρέσεις σε μια διοχέτευση

- Άλλη μορφή κινδύνου ελέγχου
- Θεωρήστε υπερχείλιση στην πρόσθεση στο στάδιο EX

  `add $1, $2, $1`

  - Αποφυγή ζημιάς (εγγραφής) στον $1
  - Ολοκλήρωση των προηγούμενων εντολών
  - Εκκένωση της add και των επόμενων εντολών
  - Ρύθμιση τιμών των καταχωρητών Cause και EPC
  - Μεταφορά ελέγχου στο χειριστή
- Όμοια με λανθασμένη πρόβλεψη διακλάδωσης
  - Χρήση μεγάλου μέρους του ίδιου υλικού

Διοχέτευση με εξαιρέσεις



Ιδιότητες των εξαιρέσεων

- Επανεκκινήσιμες εξαιρέσεις (restartable exceptions)
  - Η διοχέτευση μπορεί να εκκενώσει την εντολή
  - Ο χειριστής εκτελείται, μετά επιστρέφει στην εντολή
    - Επαναπροσκομίζεται και εκτελείται από την αρχή
- Ο PC αποθηκεύεται στον καταχωρητή EPC
  - Προσδιορίζει την εντολή που προκαλεί εξαίρεση
  - Στη πραγματικότητα αποθηκεύεται ο PC + 4
    - Ο χειριστής πρέπει να προσαρμοστεί

Παράδειγμα εξαίρεσης

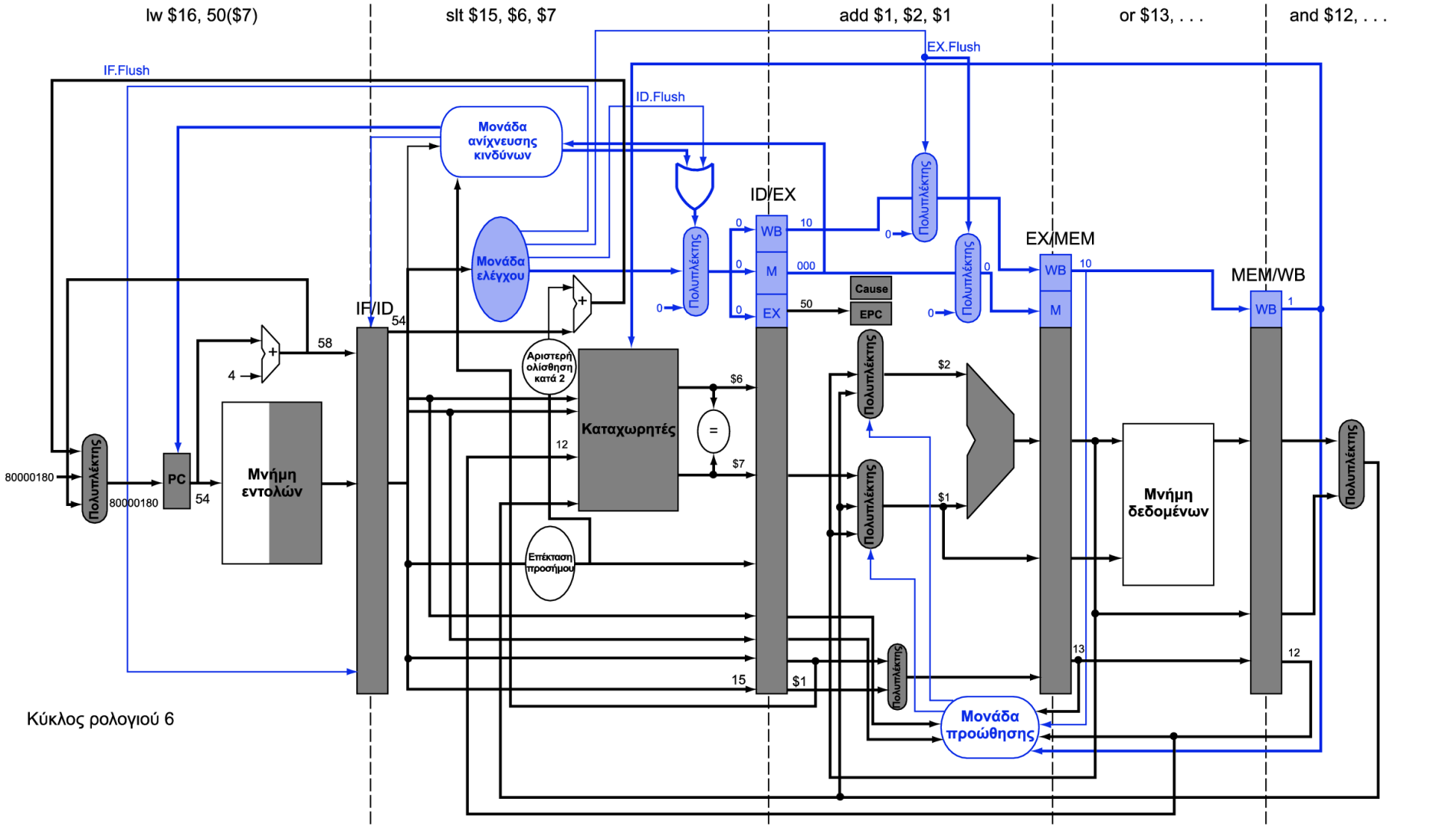
- Εξαίρεση στην add στον κώδικα

```
40      sub   $11, $2, $4
44      and   $12, $2, $5
48      or    $13, $2, $6
4c      add   $1,  $2, $1
50      slt   $15, $6, $7
54      lw    $16, 50($7)
…
```

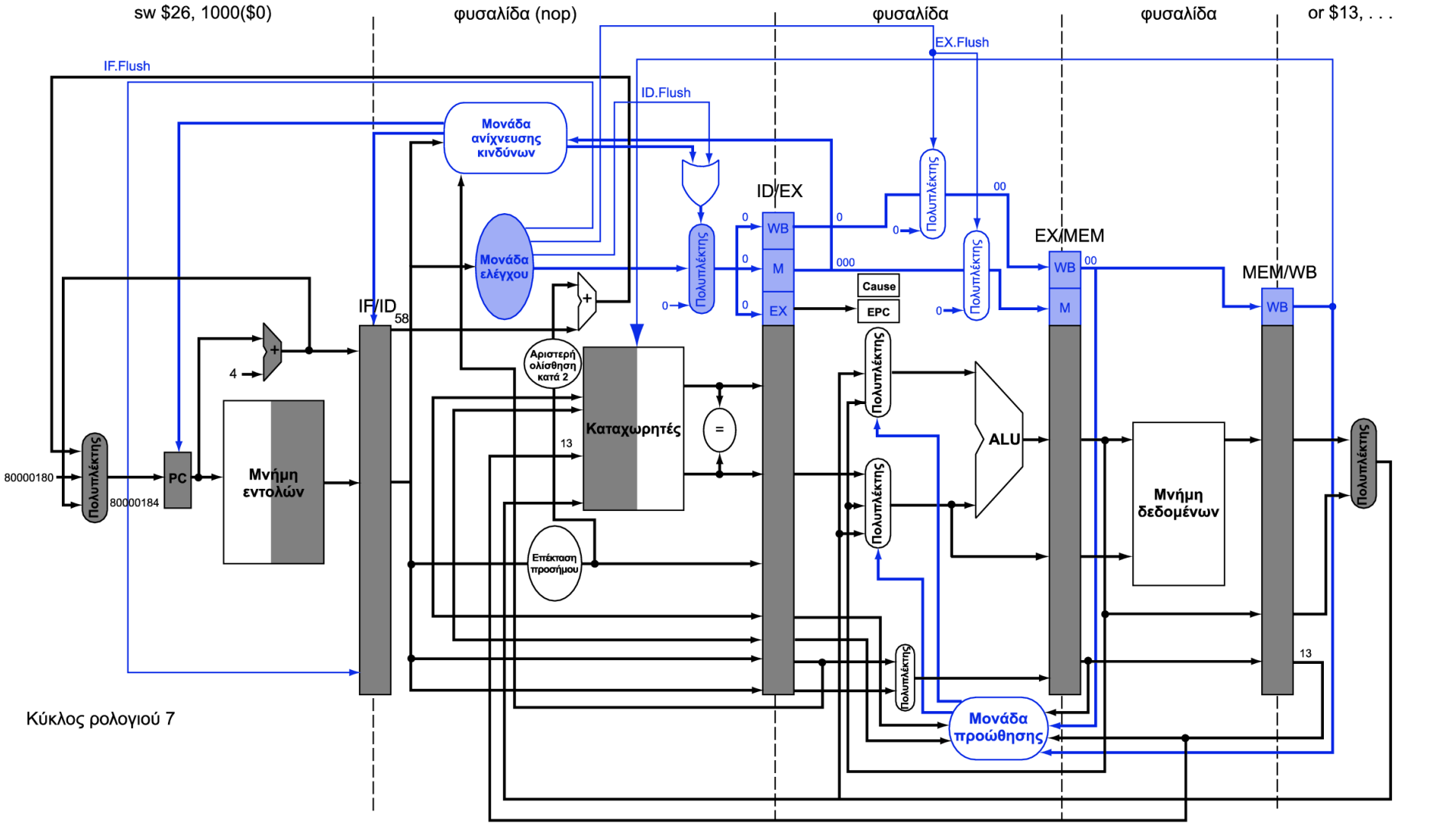
- Χειριστής

```
80000180    sw    $25, 1000($0)
80000184    sw    $26, 1004($0)
…
```

Παράδειγμα εξαίρεσης



Παράδειγμα εξαίρεσης



Πολλαπλές εξαιρέσεις

- Η διοχέτευση επικαλύπτει την εκτέλεση πολλών εντολών
  - Μπορούμε να έχουμε πολλές εξαιρέσεις ταυτόχρονα
- Απλή προσέγγιση: ασχολήσου με την εξαίρεση της νωρίτερης εντολής
  - Εκκένωση των επόμενων εντολών
  - «Ακριβείς» (“Precise”) εξαιρέσεις
- Σε σύνθετες διοχετεύσεις
  - Πολλές εντολές ξεκινούν ανά κύκλο
  - Ολοκλήρωση εκτός σειράς (out-of-order)
  - Η διατήρηση των ακριβών εξαιρέσεων είναι δύσκολη!

Μη ακριβείς (imprecise) εξαιρέσεις

- Απλώς σταμάτα τη διοχέτευση και αποθήκευσε την κατάσταση
  - Συμπεριλαμβάνεται και το αίτιο (ή τα αίτια) της εξαίρεσης
- Άφησε στο χειριστή να βρει
  - Ποιες εντολές είχαν εξαιρέσεις
  - Ποιες να ολοκληρώσει ή να εκκενώσει
    - Μπορεί να χρειαστεί μη αυτόματη ολοκλήρωση
- Απλοποιεί το υλικό, αλλά απαιτεί πιο σύνθετο λογισμικό του χειριστή
- Δεν είναι εφικτό για σύνθετες διοχετεύσεις με πολλαπλή εκκίνηση και εκτέλεση εκτός σειράς

Recall: Compute CPI?

- Start with Base CPI
- Add stalls

$$CPI = CPI_{base} + CPI_{stall}$$

$$CPI_{stall} = STALL_{type-1} \times freq_{type-1} + STALL_{type-2} \times freq_{type-2}$$

° **Suppose:**

- **CPI_{base}=1**
- **$Freq_{branch}$=20%, $freq_{load}$=30%**
- **Suppose branches always cause 1 cycle stall**
- **Loads cause a 100 cycle stall 1% of time**

° **Then: CPI = 1 + (1×0.20)+(100 × 0.30×0.01)=1.5**

° **Multicycle? Could treat as:**

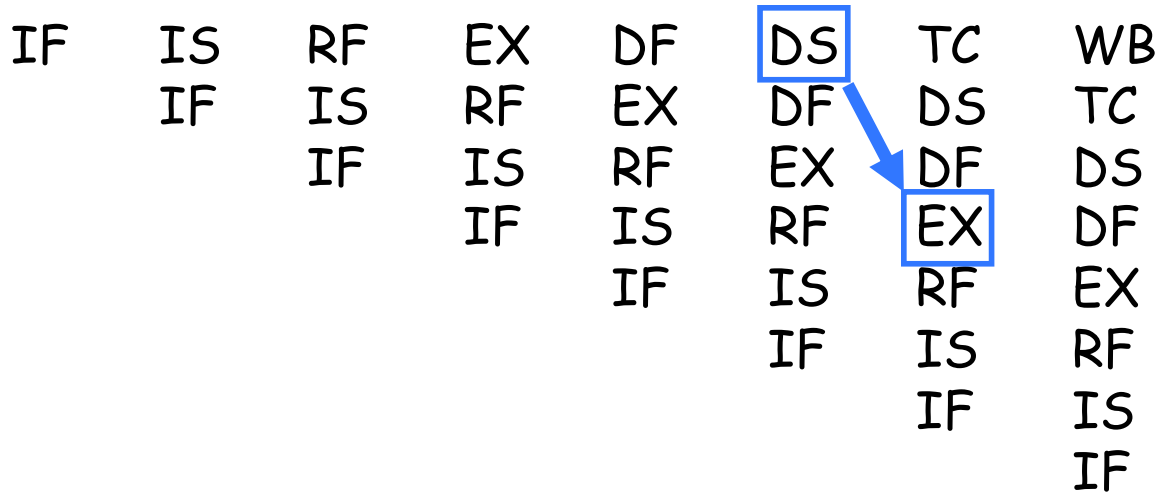
$$CPI_{stall} = (CYCLES - CPI_{base}) \times freq_{inst}$$

Case Study: MIPS R4000 (200 MHz @ 1991)

- 8 Stage Pipeline:
  - IF–first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access
  - IS–second half of access to instruction cache
  - RF–instruction decode and register fetch, hazard checking and also instruction cache hit detection
  - EX–execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation
  - DF–data fetch, first half of access to data cache
  - DS–second half of access to data cache
  - TC–tag check, determine whether the data cache access hit
  - WB–write back for loads and register-register operations
- 8 Stages: What is impact on Load delay? Branch delay? Why?

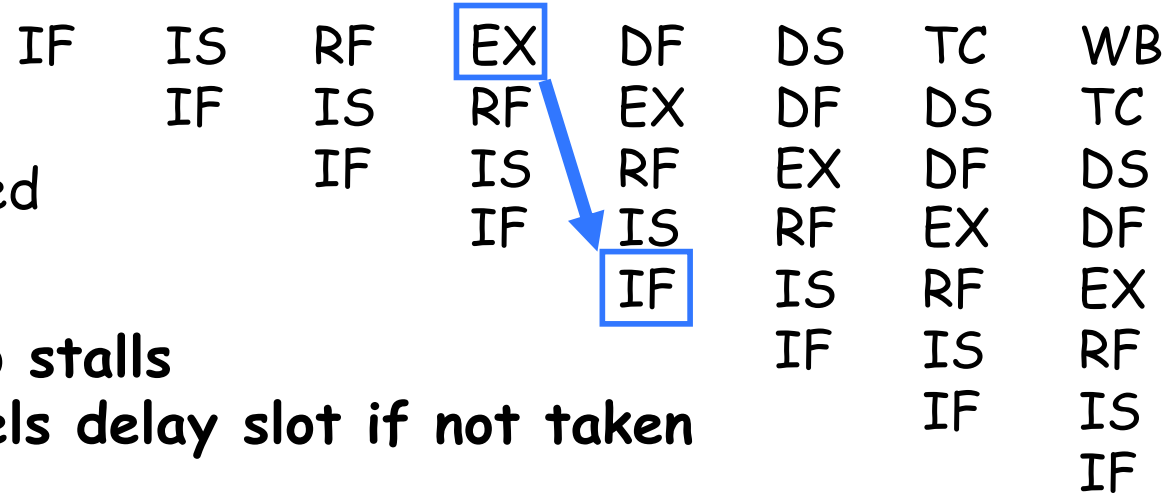
Case Study: MIPS R4000

TWO Cycle Load Latency



THREE Cycle Branch Latency

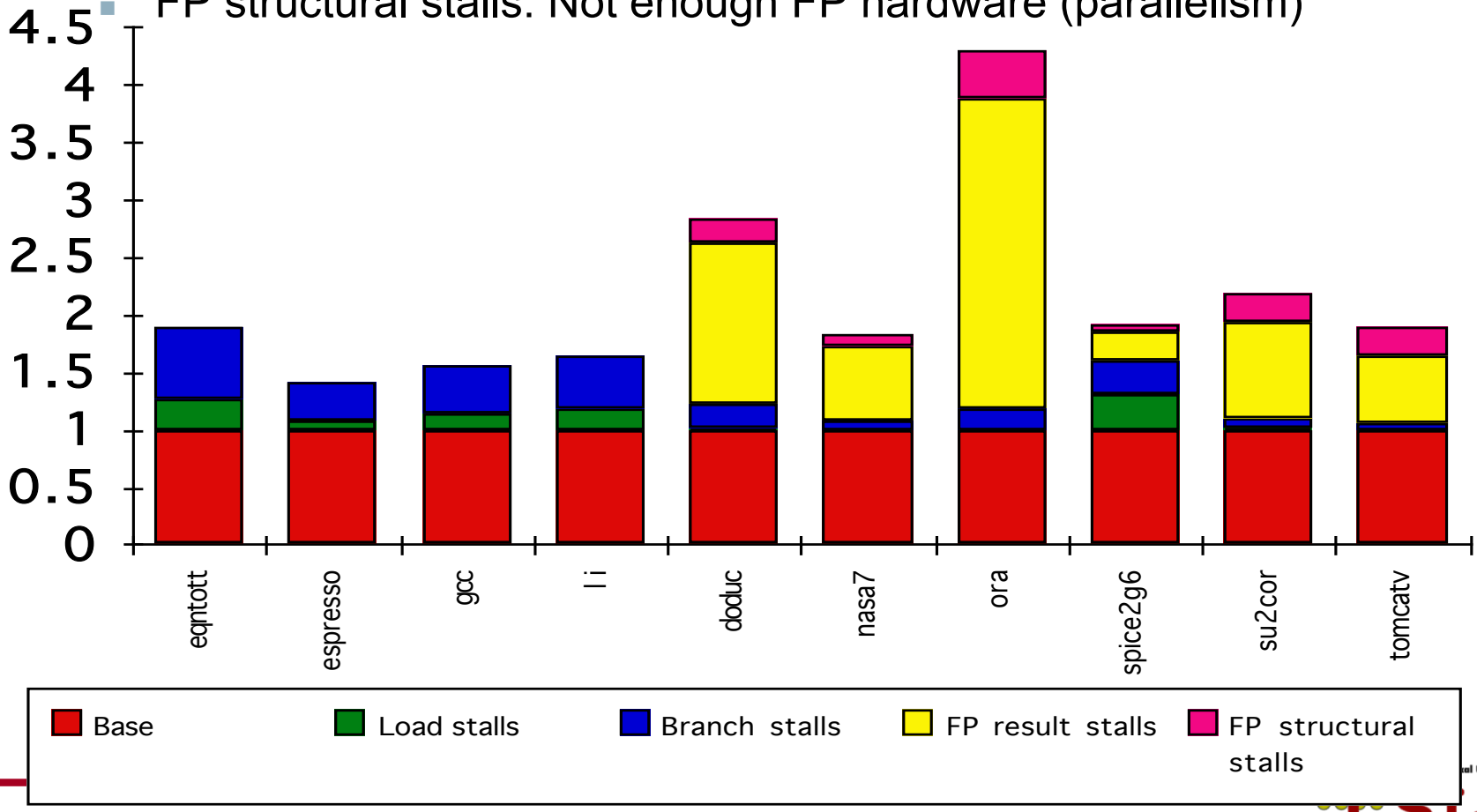
(conditions evaluated during EX phase)



Delay slot plus two stalls
Branch likely cancels delay slot if not taken

R4000 Performance

- Not ideal CPI of 1:
  - Load stalls (1 or 2 clock cycles)
  - Branch stalls (2 cycles + unfilled slots)
  - FP result stalls: RAW data hazard (latency)
  - FP structural stalls: Not enough FP hardware (parallelism)



Παραλληλία επιπέδου εντολής

- Instruction-Level Parallelism (ILP)
- Διοχέτευση: παράλληλη εκτέλεση πολλών εντολών
- Για αύξηση του ILP
  - Βαθύτερη διοχέτευση
    - Λιγότερη δουλειά ανά στάδιο ⇒ μικρότερος κύκλος ρολογιού
  - Πολλαπλή εκκίνηση (multiple issue)
    - Επανάληψη σταδίων διοχέτευσης ⇒ πολλές διοχετεύσεις
    - Εκκίνηση πολλών εντολών ανά κύκλο ρολογιού
    - CPI < 1, συνεπώς χρήση του Instructions Per Cycle (IPC)
    - Π.χ., 4GHz πολλαπλή εκκίνηση 4 δρόμων (4-way multiple-issue)
      - 16 BIPS, μέγιστο CPI = 0.25, μέγιστο IPC = 4
    - Αλλά οι εξαρτήσεις το μειώνουν στην πράξη

Πολλαπλή εκκίνηση

- Στατική πολλαπλή εκκίνηση
  - Ο μεταγλωττιστής ομαδοποιεί τις εντολές που θα ξεκινήσουν μαζί
  - Τις συσκευάζει σε «υποδοχές εκκίνησης» (“issue slots”)
  - Ο μεταγλωττιστής ανιχνεύει και αποφεύγει τους κινδύνους
- Δυναμική πολλαπλή εκκίνηση
  - Η CPU εξετάζει το ρεύμα των εντολών και επιλέγει εντολές για εκκίνηση σε κάθε κύκλο
  - Ο μεταγλωττιστής μπορεί να βοηθήσει με αναδιάταξη των εντολών
  - Η CPU επιλύει τους κινδύνους με προηγμένες τεχνικές κατά το χρόνο εκτέλεσης

Στατική πολλαπλή εκκίνηση

- Ο μεταγλωττιστής ομαδοποιεί τις εντολές σε «πακέτα εκκίνησης» (“issue packets”)
  - Ομάδα εντολών που μπορούν να ξεκινήσουν σε έναν κύκλο
  - Καθορίζεται από τους πόρους της διοχέτευσης που απαιτούνται
- Σκεφθείτε ένα πακέτο εκκίνησης ως μια πολύ μεγάλη εντολή
  - Προσδιορίζει πολλές ταυτόχρονες λειτουργίες
  - $\Rightarrow$ Πολύ μεγάλη λέξη εντολής (Very Long Instruction Word – VLIW)

Χρονοπρογ/σμός στατικής

- Ο μεταγλωττιστής πρέπει να αφαιρέσει κάποιους ή όλους τους κινδύνους
  - Αναδιάταξη εντολών σε πακέτα εκκίνησης
  - Όχι εξαρτήσεις μέσα σε ένα πακέτο
  - Πιθανόν κάποιες εξαρτήσεις μεταξύ πακέτων
    - Διαφορές μεταξύ αρχιτεκτονικών – ο μεταγλωττιστής πρέπει να το γνωρίζει!
  - Συμπλήρωση με εντολές nop αν χρειάζεται

Μια εύκολη λύση: {1 int + 1 fp instruction}

- Επιτρέπεται: 1 anything + 1 FP
  - I-Fetch 64-bits/cycle: {Int αριστερά, FP δεξιά}
  - Εκδίδει την δεύτερη εντολή μετά την έκδοση της πρώτης
  - >2 πόρτες στην FP RF λόγω {FP load, FP op}
  - Load delay ενός κύκλου => 3 εντολές!!!

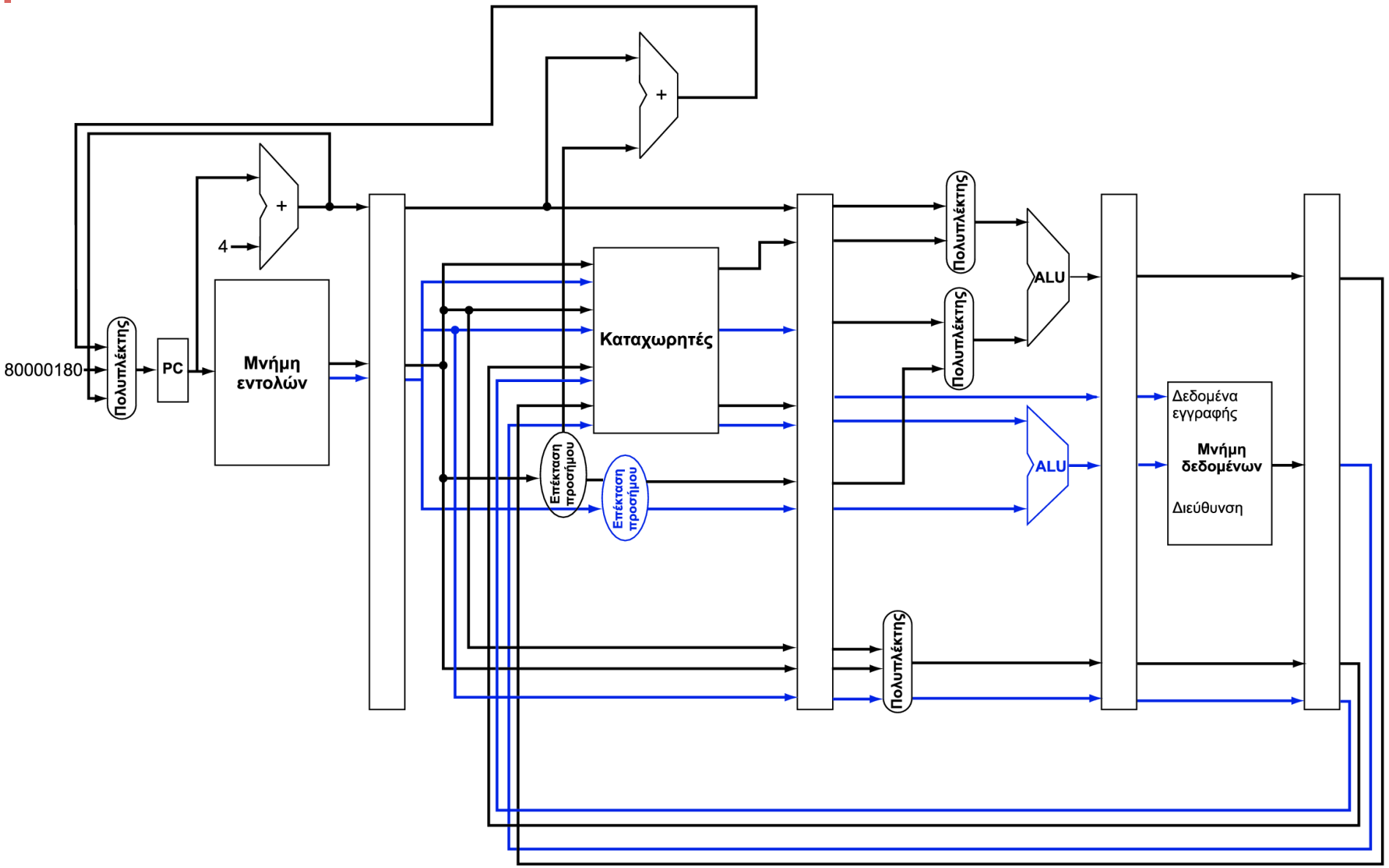
Type	*PipeStages*					
Int. instruction	IF	ID	EX	MEM	WB	
FP instruction	IF	ID	EX	MEM	WB	
Int. instruction		IF	ID	EX	MEM	WB
FP instruction		IF	ID	EX	MEM	WB
Int. instruction			IF	ID	EX	MEM WB
FP instruction			IF	ID	EX	MEM WB

MIPS με στατική διπλή εκκίνηση

- Πακέτα διπλής εκκίνησης
  - Μια εντολή ALU ή branch
  - Μια εντολή load ή store
  - Ευθυγραμμισμένες στα 64 bit
    - ALU/branch, μετά load/store
    - Συμπλήρωση μιας αχρησιμοποίητης εντολής με nop

Διεύθυνση	Τύπος εντολής	Στάδια διοχέτευσης						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

MIPS με στατική διπλή εκκίνηση



Κίνδυνοι στο MIPS με διπλή εκκίνηση

- Περισσότερες εντολές εκτελούνται παράλληλα
- Κίνδυνος δεδομένων EX
  - Η προώθηση απέφυγε τις καθυστερήσεις στην απλή εκκίνηση
  - Τώρα δεν μπορούμε να χρησιμοποιήσουμε το αποτέλεσμα της ALU σε μια load/store στο ίδιο πακέτο
    - ```
      add  $t0, $s0, $s1
      load $s2, 0($t0)
      ```
    - Χωρισμός σε δύο πακέτα, ουσιαστικά μία καθυστέρηση (stall)
- Κίνδυνος φόρτωσης-χρήσης
  - Και πάλι ένας κύκλος λανθάνοντος χρόνου χρήσης, αλλά τώρα δύο εντολές
- Ανάγκη πιο επιθετικού χρονοπρογραμματισμού
      ```

Παράδειγμα χρονοπρ/σμού

- Κώδικας: `while (p) {*p+=x; p--}`
- Χρονοπρογραμματισμός σε MIPS διπλής εκκίνησης:

```
Loop: lw   $t0, 0($s1)       # $t0=array element
      addu $t0, $t0, $s2     # add scalar in $s2
      sw   $t0, 0($s1)       # store result
      addi $s1, $s1,–4       # decrement pointer
      bne  $s1, $zero, Loop  # branch $s1!=0
```

	ALU/branch	Load/store	κύκλος
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1,–4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- IPC = 5/4 = 1.25 (σύγκριση με μέγιστο IPC = 2)

Ξετύλιγμα βρόχου

- Loop unrolling
- Επανάληψη του σώματος του βρόχου για να εμφανιστεί μεγαλύτερη παραλληλία
  - Μειώνει την επιβάρυνση ελέγχου του βρόχου
- Χρήση διαφορετικών καταχωρητών ανά επανάληψη
  - Ονομάζεται «μετονομασία καταχωρητών (“register renaming”)
  - Αποφυγή «αντεξαρτήσεων» (“anti-dependencies”) που μεταφέρονται στο βρόχο (loop-carried)
    - Store που ακολουθείται από load στον ίδιο καταχωρητή
    - Ονομάζεται επίσης και «εξάρτηση ονόματος» (“name dependence”)
      - Επαναχρησιμοποίηση ενός ονόματος καταχωρητή

Παράδειγμα ξετυλίγματος βρόχου

- Κώδικας «ξετυλιγμένος» 4 φορές (θεωρούμε ότι #στοιχείων είναι πολλαπλάσιο του 4):

```
While (p) {
   *p+=x;          // t0=*p; t0+=x; *p=t0;
   *(p+1) += x; // t1=*p; t1+=x; *p=t1;
   *(p+2) += x; // t2=*p; t2+=x; *p=t2;
   *(p+3) += x; // t3=*p; t3+=x; *p=t3;
   p = p-4; // μείωσε τον δείκτη κατά 4
}
```

Παράδειγμα ξετυλίγματος βρόχου

	ALU/branch	Load/store	Κύκλος
Loop:	addi $s1, $s1,-16	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t4, $s2	sw $t1, 12($s1)	6
	nop	sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

- IPC = 14/8 = 1.75
  - Πλησιέστερο στο 2, αλλά με κόστος σε καταχωρητές και μέγεθος κώδικα

Εικασία (speculation)

- «Μαντεψιά» – τι να κάνουμε με μια εντολή
  - Εκκίνηση λειτουργίας το συντομότερο
  - Έλεγχος αν η εικασία ήταν σωστή
    - Αν ναι, ολοκλήρωση της λειτουργίας
    - Αν όχι, επιστροφή (roll-back) και εκτέλεση του σωστού
- Συνηθισμένη στη στατική και τη δυναμική πολλαπλή εκκίνηση
- Παραδείγματα
  - Εικασία στο αποτέλεσμα διακλάδωσης
    - Επιστροφή αν η ληφθείσα διαδρομή είναι διαφορετική
  - Εικασία σε φόρτωση (load)
    - Επιστροφή αν η θέση μνήμης έχει αλλάξει τιμή

Εικασία μεταγλωττιστή/υλικού

- Ο μεταγλωττιστής μπορεί να αναδιατάξει τις εντολές
  - π.χ., μετακίνηση μιας φόρτωσης πριν από μια διακλάδωση
  - Μπορεί να περιλάβει εντολές «διόρθωσης» για να ανακάμψει από λάθος εικασίες
- Το υλικό μπορεί να δει πιο πέρα για εντολές προς εκτέλεση
  - Αποθηκεύει προσωρινά τα αποτελέσματα μέχρι να προσδιορίσει ότι χρειάζονται πραγματικά
  - Εκκενώνει τις προσωρινές μνήμες αν γίνει λάθος εικασία

Εικασία και εξαιρέσεις

- Τι γίνεται αν συμβεί εξαίρεση σε μια εντολή που εκτελείται με εικασία;
  - π.χ., φόρτωση με εικασία πριν τον έλεγχο κενού δείκτη (null-pointer check)
- Στατική εικασία
  - Μπορεί να προσθέσει υποστήριξη από την αρχιτεκτονική συνόλου εντολών για εξαιρέσεις που έχουν μετατεθεί (deferring exceptions)
- Δυναμική εικασία
  - Μπορεί να αποθηκεύει προσωρινά τις εξαιρέσεις μέχρι την ολοκλήρωση της εντολής (που μπορεί να μη συμβεί)

Δυναμική πολλαπλή εκκίνηση

- Υπερβαθμωτοί (“superscalar”) επεξεργαστές
- Η CPU αποφασίζει αν θα ξεκινήσουν 0, 1, 2, … εντολές σε κάθε κύκλο
  - Αποφυγή κινδύνων δομής και δεδομένων
- Αποφεύγει την ανάγκη για χρονοπρογραμματισμό από το μεταγλωττιστή
  - Παρόλο που αυτός μπορεί ακόμη να βοηθήσει
  - Η CPU εγγυάται τη σημασιολογία του κώδικα

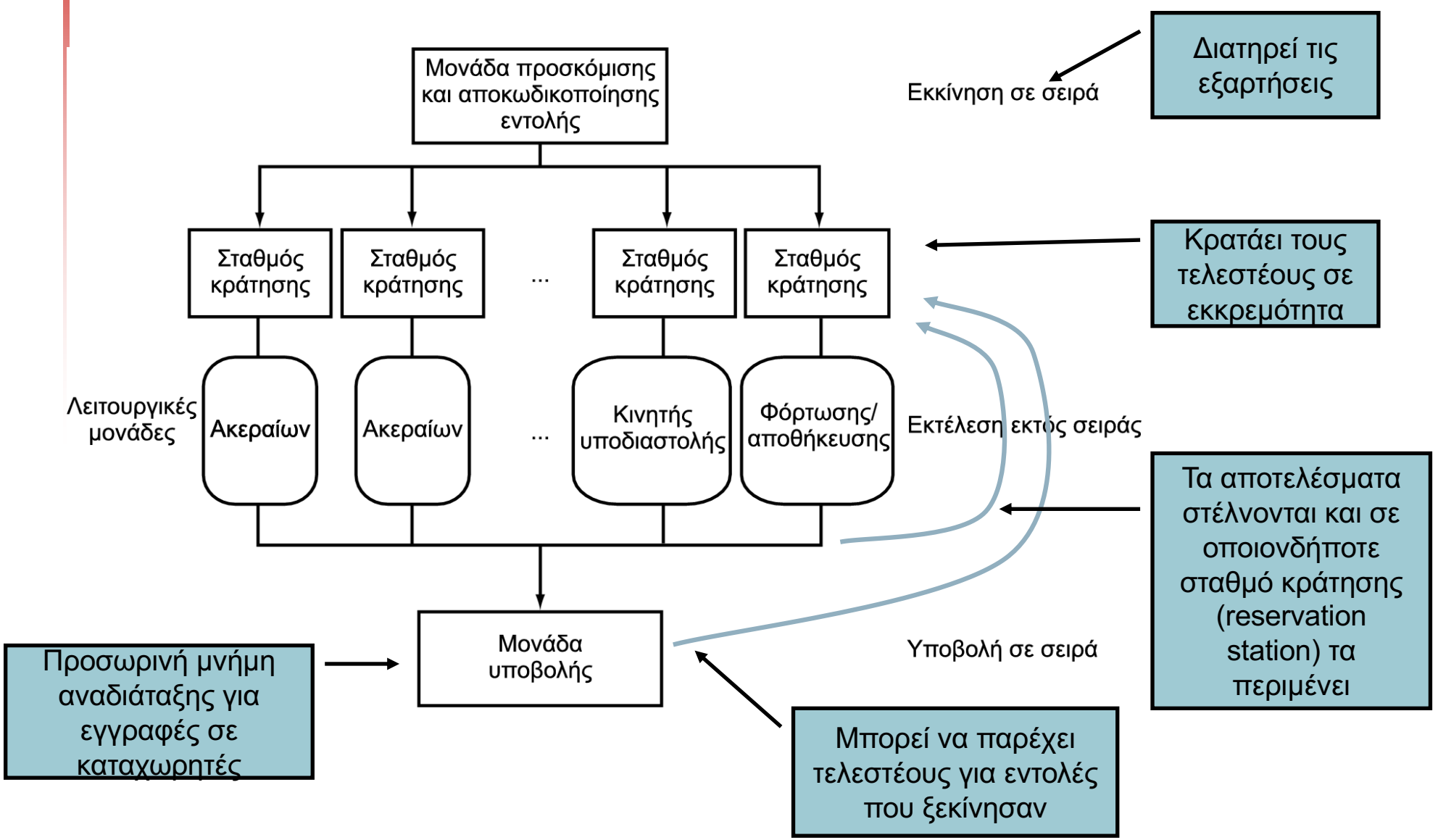
Χρονοπρογ/σμός δυναμικής

- Η CPU μπορεί να εκτελεί εντολές εκτός σειράς (out of order) για αποφυγή καθυστερήσεων
  - Αλλά δέσμευση (commit) του αποτελέσματος σε καταχωρητές σε σειρά
- Παράδειγμα

```
lw    $t0, 20($s2)
addu  $t1, $t0, $t2
sub   $s4, $s4, $t3
slti  $t5, $s4, 20
```

  - Μπορεί να ξεκινήσει η sub ενώ η addu περιμένει τη lw

CPU με δυναμικό χρον/σμό



Μετονομασία καταχωρητών

- Οι σταθμοί κράτησης (reservation stations) και η προσωρινή μνήμη αναδιάταξης (reorder buffer) στην πράξη παρέχουν μετονομασία καταχωρητών
- Κατά την εκκίνηση εντολής σε σταθμό κράτησης
  - Αν ο τελεστέος είναι διαθέσιμος στο αρχείο καταχωρητών ή την προσωρινή μνήμη αναδιάταξης
    - Αντιγράφεται στο σταθμό κράτησης
    - Δε χρειάζεται άλλο στον καταχωρητή, και μπορεί να αντικατασταθεί
  - Αν ο τελεστέος δεν είναι ακόμα διαθέσιμος
    - Θα παρασχεθεί στο σταθμό κράτησης από μια λειτουργική μονάδα
    - Μπορεί να μη χρειαστεί ενημέρωση καταχωρητή

Εικασία (speculation)

- Πρόβλεψη διακλάδωσης και συνέχιση της εκκίνησης
  - Δε γίνεται δέσμευση (commit) μέχρι να καθοριστεί το αποτέλεσμα της διακλάδωσης
- Εικασία φόρτωσης (load speculation)
  - Αποφυγή της καθυστέρησης της φόρτωσης και της αστοχίας κρυφής μνήμης (cache miss)
    - Πρόβλεψη της πραγματικής διεύθυνσης
    - Πρόβλεψη της τιμής φόρτωσης
    - Φόρτωση πριν την ολοκλήρωση εκκρεμουσών αποθηκεύσεων
    - Παράκαμψη/προώθηση τιμών αποθήκευσης προς τη μονάδα φόρτωσης
  - Δε γίνεται δέσμευση (commit) της φόρτωσης μέχρι να ξεκαθαριστεί η εικασία

Γιατί δυναμικός χρονοπ/σμός;

- Γιατί να μην αφήσουμε απλώς το μεταγλωττιστή να χρονοπρογραμματίσει τον κώδικα;
- Δεν είναι όλες οι καθυστερήσεις προβλέψιμες
  - π.χ., αστοχίες κρυφής μνήμης (cache misses)
- Δεν μπορεί να γίνει πάντα χρονοπρ/σμός γύρω από διακλαδώσεις
  - Το αποτέλεσμα της διακλάδωσης καθορίζεται δυναμικά
- Διαφορετικές υλοποιήσεις μιας αρχιτεκτονικής έχουν διαφορετικούς λανθάνοντες χρόνους και κινδύνους

Δουλεύει η πολλαπλή εκκίνηση;

ΓΕΝΙΚΗ εικόνα

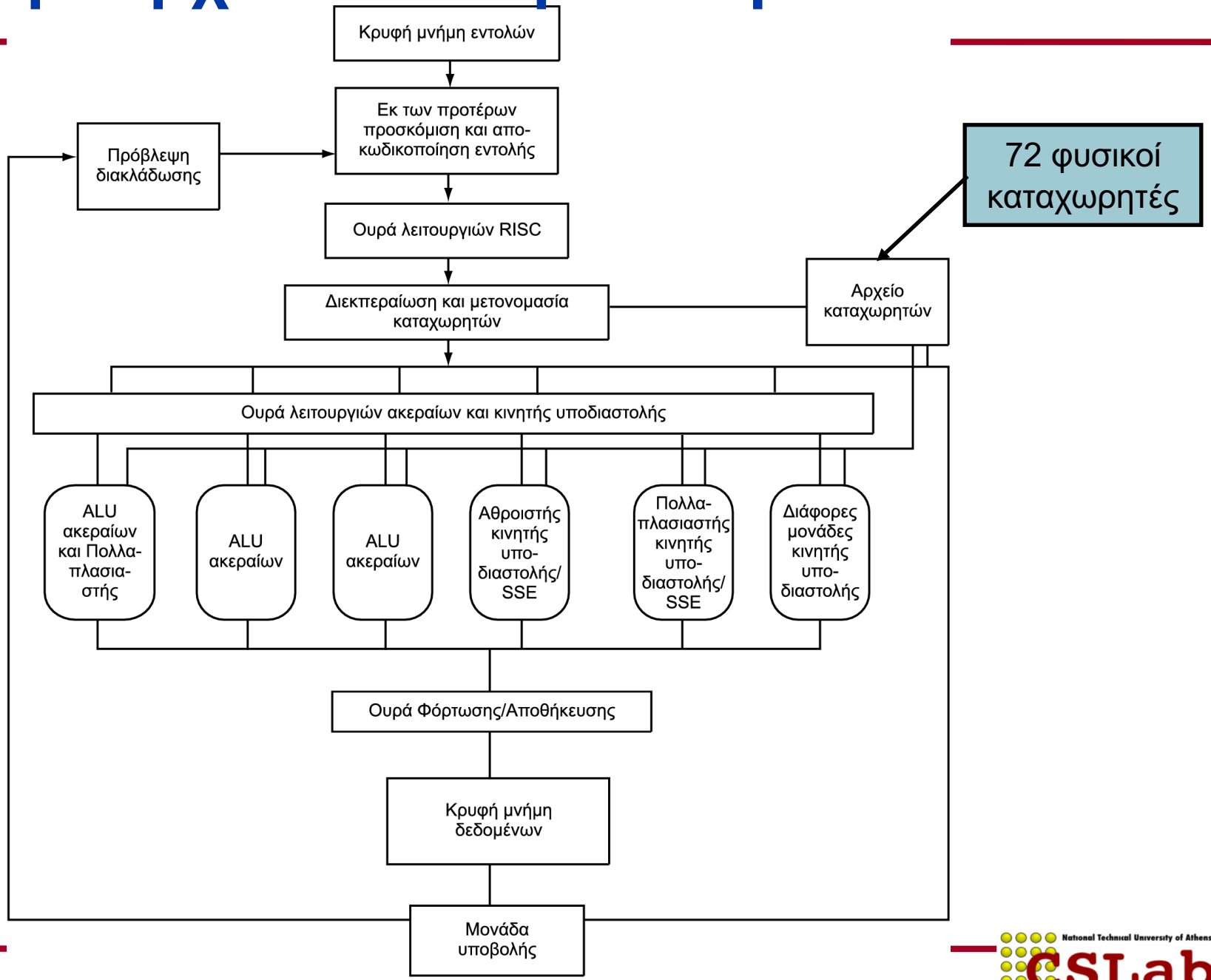
- Ναι, αλλά όχι όσο θα θέλαμε
- Τα προγράμματα έχουν πραγματικές εξαρτήσεις που περιορίζουν το ILP
- Μερικές εξαρτήσεις είναι δύσκολο να εξαλειφθούν
  - π.χ., ψευδωνυμία δείκτη (pointer aliasing)
- Κάποια παραλληλία είναι δύσκολο να εμφανιστεί
  - Περιορισμένο μέγεθος παραθύρου κατά την εκκίνηση εντολών
- Καθυστερήσεις μνήμης και περιορισμένο εύρος ζώνης
  - Δύσκολο να διατηρηθούν γεμάτες οι διοχετεύσεις
- Η εικασία μπορεί να βοηθήσει αν γίνει καλά

Αποδοτικότητα ισχύος

- Η πολυπλοκότητα του δυναμικού χρονοπρογραμματισμού και της εικασίας απαιτεί ηλεκτρική ισχύ
- Οι πολλοί απλούστεροι πυρήνες μπορεί να είναι καλύτεροι

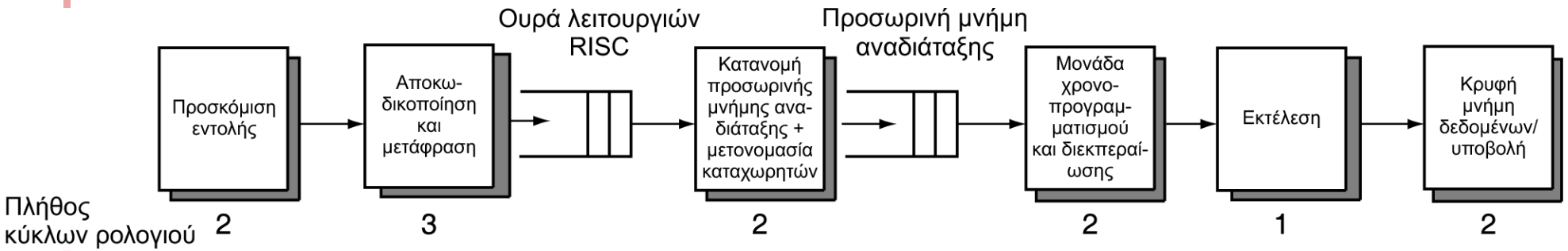
Μικροεπεξεργαστής	Έτος	Ρυθμός ρολογιού	Στάδια διοχέτευσης	Πλάτος εκκίνησης	Εκτός-σειράς/ Εικασία	Πυρήνες	Ισχύς
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

Μικροαρχιτεκτονική του Opteron X4



Διοχέτευση του Opteron X4

- Για ακέραιες λειτουργίες



  - Για κινητή υποδιαστολή είναι 5 στάδια μεγαλύτερη
  - Μέχρι 106 λειτουργίες RISC σε εξέλιξη
- Σημεία συμφόρησης
  - Πολύπλοκες εντολές με μεγάλες εξαρτήσεις
  - Λάθος προβλέψεις διακλάδωσης
  - Καθυστερήσεις προσπέλασης μνήμης

Πλάνες

- Η διοχέτευση είναι εύκολη (!)
  - Η βασική ιδέα είναι εύκολη
  - Ο διάβολος κρύβεται στις λεπτομέρειες
    - π.χ., ανίχνευση κινδύνων δεδομένων
- Η διοχέτευση είναι ανεξάρτητη από την τεχνολογία
  - Τότε γιατί δεν κάναμε πάντα διοχέτευση;
  - Τα περισσότερα τρανζίστορ κάνουν εφικτές τις πιο προηγμένες τεχνικές
  - Η σχεδίαση αρχιτεκτονικών συνόλου εντολών που σχετίζεται με τη διοχέτευση πρέπει να λαμβάνει υπόψη της τις τεχνολογικές τάσεις
    - π.χ., εντολές με κατηγορήματα

Παγίδες

- Η φτωχή σχεδίαση της αρχιτεκτονικής συνόλου εντολών μπορεί να κάνει δυσκολότερη τη διοχέτευση
  - π.χ., πολύπλοκα σύνολα εντολών (VAX, IA-32)
    - Σημαντική επιβάρυνση για να δουλέψει η διοχέτευση
    - Προσέγγιση του IA-32 με μικρολειτουργίες (micro-ops)
  - π.χ., πολύπλοκοι τρόποι διευθυνσιοδότησης
    - Παρενέργειες ενημέρωσης καταχωρητών, εμμεσότητα μνήμης
  - π.χ., καθυστερημένες διακλαδώσεις
    - Οι προηγμένες διοχετεύσεις έχουν μεγάλες υποδοχές καθυστέρησης

Συμπερασματικές παρατηρήσεις

- Η αρχιτεκτονική συνόλου εντολών επηρεάζει τη σχεδίαση της διαδρομής δεδομένων και της μονάδας ελέγχου
- Η διαδρομή δεδομένων και η μονάδα ελέγχου επηρεάζουν τη σχεδίαση της αρχιτεκτονικής συνόλου εντολών
- Η διοχέτευση βελτιώνει τη διεκπεραιωτική ικανότητα εντολών με τη χρήση παραλληλίας
  - Περισσότερες εντολές ολοκληρώνονται ανά δευτερόλεπτο
  - Ο λανθάνων χρόνος κάθε εντολής δε μειώνεται
- Κίνδυνοι: δομής, δεδομένων, ελέγχου
- Πολλαπλή εκκίνηση και δυναμικός χρονοπρογραμματισμός (παραλληλία επιπέδου εντολής – ILP)
  - Οι εξαρτήσεις περιορίζουν την επιτεύξιμη παραλληλία
  - Η πολυπλοκότητα οδηγεί στο τείχος της ισχύος (power wall)