

# ΗΥ 232

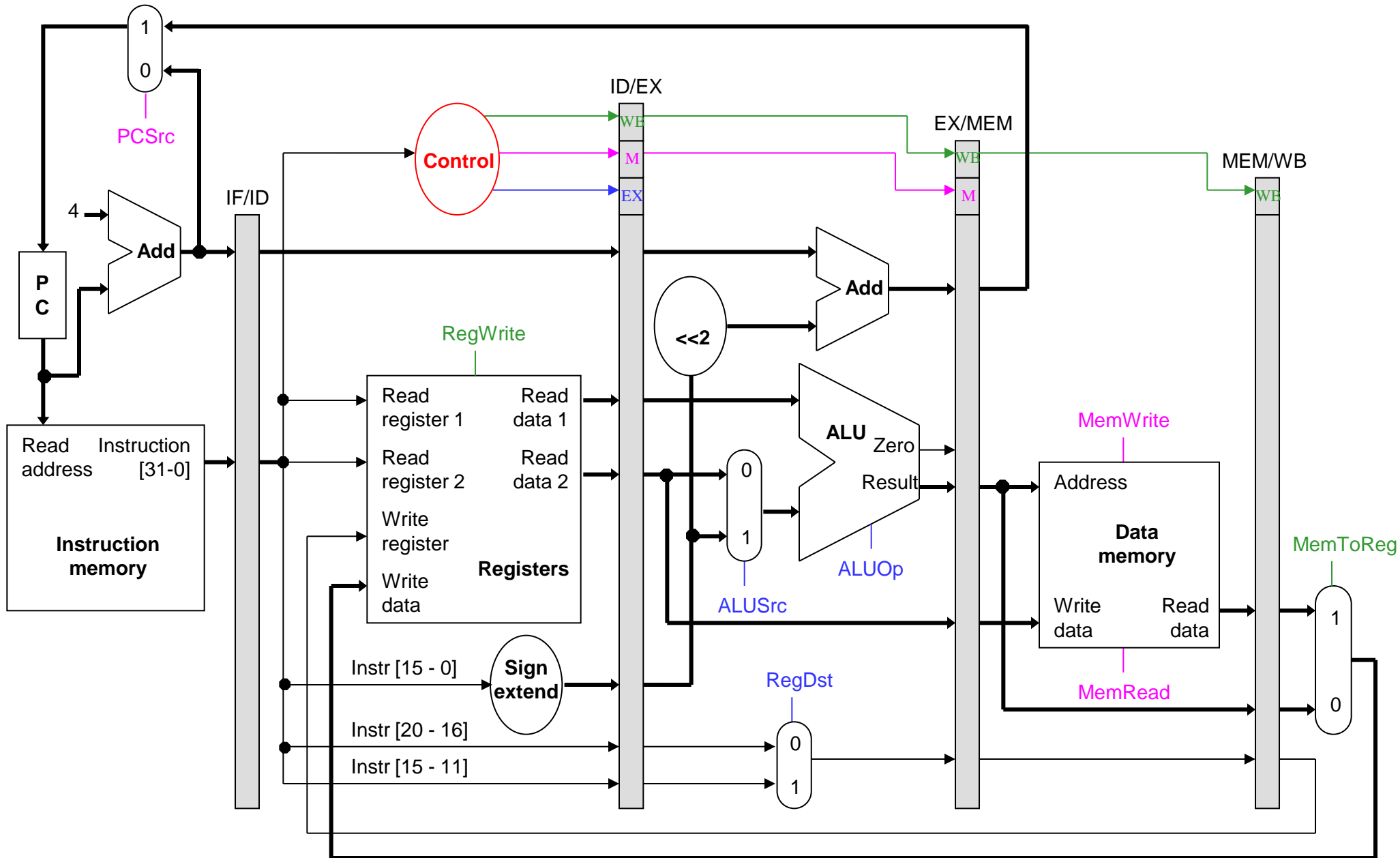
## Οργάνωση και Σχεδίαση Υπολογιστών

### Διάλεξη 11

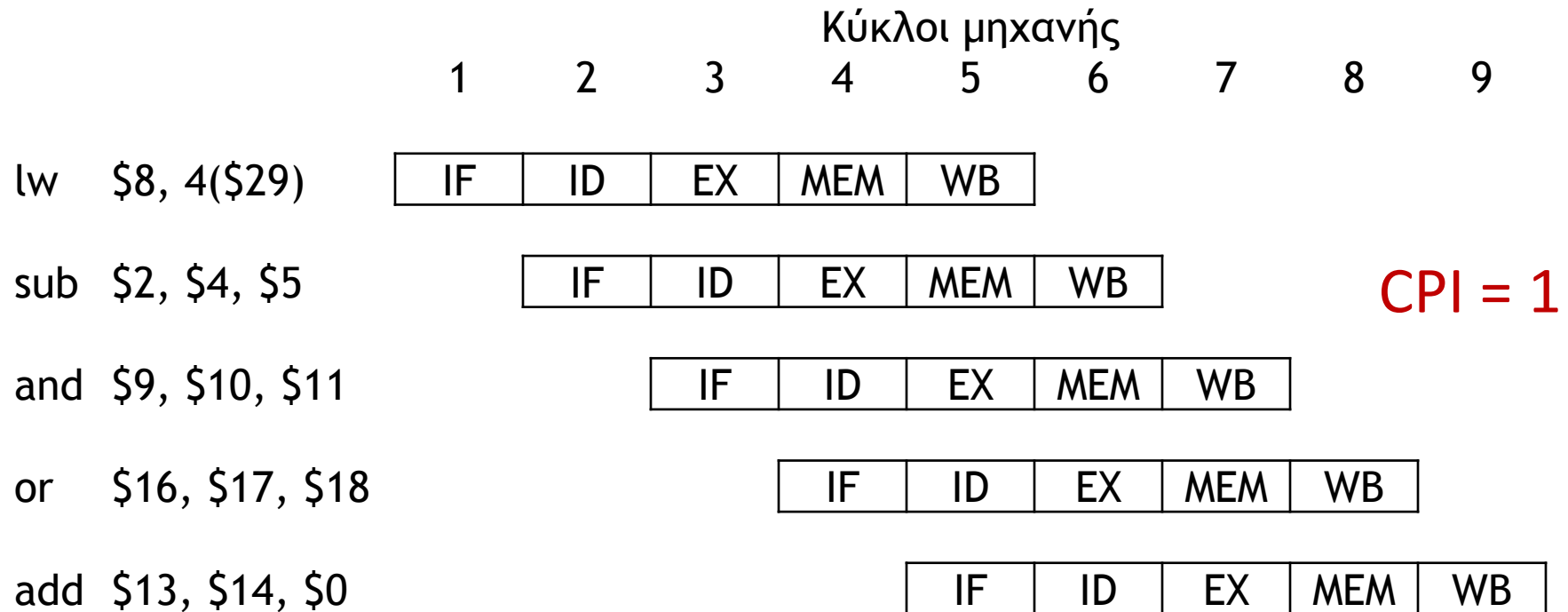
### Προώθηση (Forwarding)

Νίκος Μπέλλας  
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

# Η μέχρι τώρα μικρο-αρχιτεκτονική του MIPS



# Διάγραμμα Διοχέτευσης



- Η μικρο-αρχιτεκτονική που έχουμε αναλύσει μέχρι τώρα δείχνει μια απλουστευμένη κατάσταση.
  - Κάθε εντολή χρειάζεται 5 κύκλους εκτέλεσης.
  - Μία εντολή ξεκινάει σε κάθε κύκλο μηχανής
  - Μία εντολή τερματίζει σε κάθε κύκλο μηχανής (CPI=1)

# Η σειρά των εντολών είναι πολύ απλή

```
1000: lw    $8, 4($29)
1004: sub   $2, $4, $5
1008: and   $9, $10, $11
1012: or    $16, $17, $18
1016: add   $13, $14, $0
```

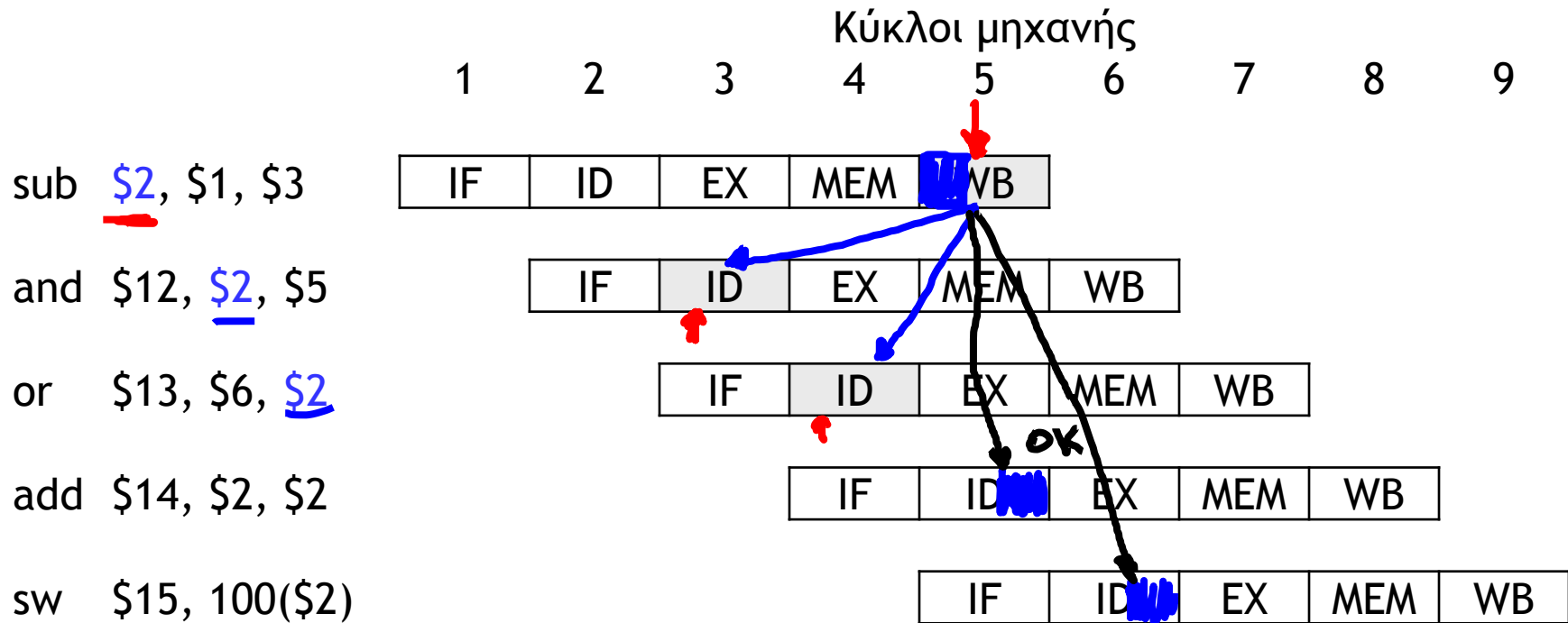
- Έχουμε κάνει κάποιες παραδοχές για την ακολουθία των εντολών που δυστυχώς δεν ισχύουν πάντα
  - Δεν υπάρχει εξάρτηση μεταξύ των δεδομένων που γράφει μια εντολή και των δεδομένων που μια επόμενη εντολή διαβάζει.
  - Αυτό είναι ιδανική περίπτωση και δεν συμβαίνει συχνά
  - Πολλές ακολουθίες αποτελούνται από εξαρτημένες εντολές

# Παράδειγμα με εξαρτημένες εντολές

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

- Η πρώτη εντολή (**sub**) γράφει στον καταχωρητή **\$2**
- Ο καταχωρητής **\$2** χρησιμοποιείται από εκεί και πέρα ως είσοδος στις επόμενες εντολές
- Αυτό δεν είναι πρόβλημα στην υλοποίηση ενός κύκλου.
  - Κάθε εντολή εκτελείται σε ένα κύκλο και η επόμενη παίρνει τα δεδομένα κανονικά από τον καταχωρητή **\$2**
- Αλλά τι ακριβώς συμβαίνει στην μικρο-αρχιτεκτονική διοχέτευσης;

# Κίνδυνος Δεδομένων (Data Hazards)



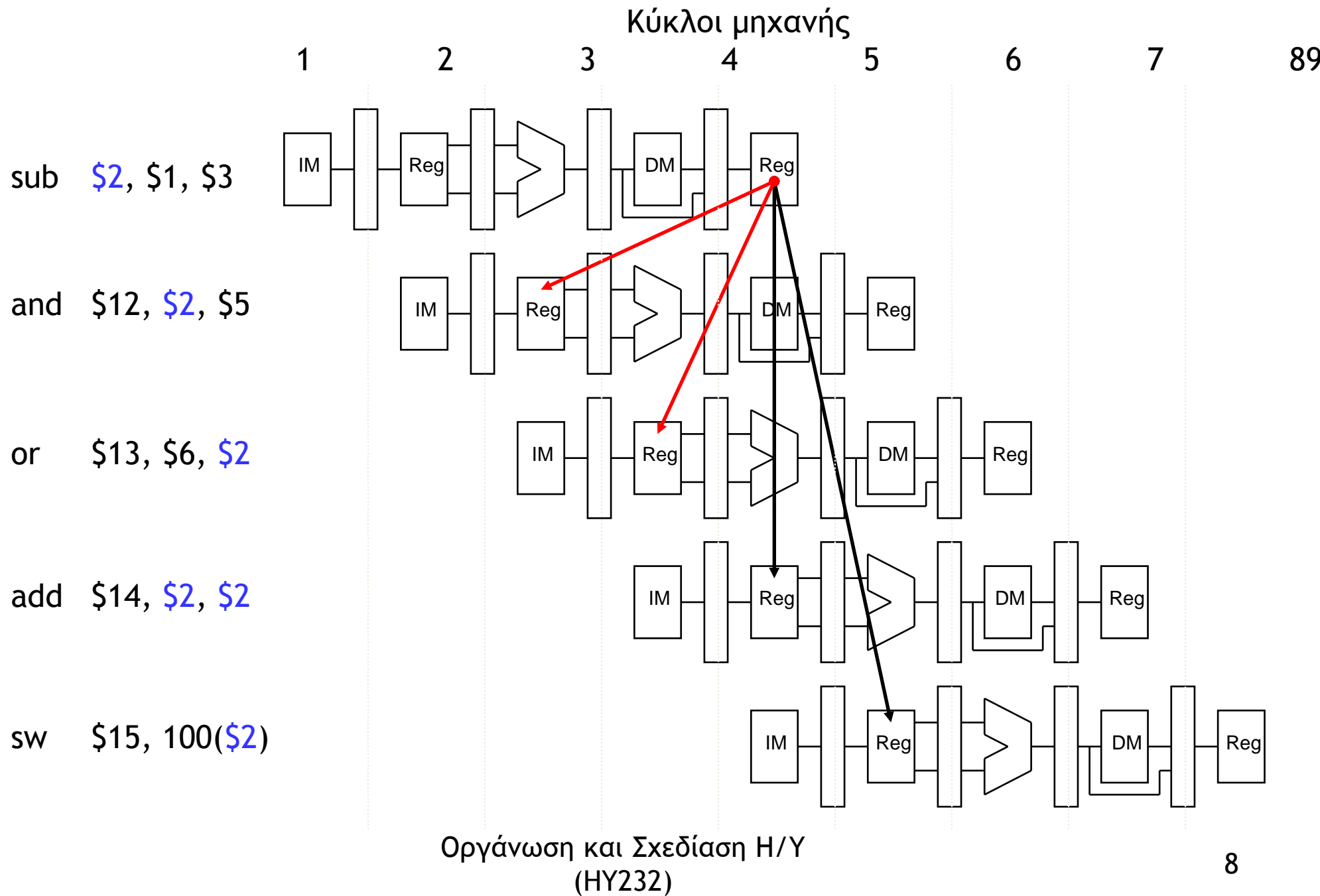
- Η εντολή **sub** γράφει στον καταχωρητή **\$2** στον κύκλο 5. Αυτό δημιουργεί data hazards στην μικρο-αρχιτεκτονική διοχέτευσης.
  - Η εντολή **and** διαβάζει τον καταχωρητή **\$2** στον κύκλο 3. Αφού η εντολή **sub** δεν έχει ακόμα γράψει τον **\$2**, η **and** θα διαβάσει τα παλιά δεδομένα του καταχωρητή **\$2**
  - Η εντολή **or** διαβάζει τον καταχωρητή **\$2** στον κύκλο 4, παίρνοντας επίσης την παλιά του τιμή.
  - Από εκεί και πέρα τα πράγματα δουλεύουν εντάξει. Οι επόμενες δύο εντολές εκτελούνται όταν η **sub** τελειώσει

# Κίνδυνος Δεδομένων (Data Hazards)



- Τα βέλη δείχνουν την ροή των δεδομένων μεταξύ των εντολών
  - Η αρχή του βέλους δείχνει πότε γράφεται ο καταχωρητής \$2
  - Το τέλος του βέλους δείχνει πότε διαβάζεται ο καταχωρητής \$2
- Όταν ένα βέλος δείχνει πίσω στον χρόνο αυτό σημαίνει ότι υπάρχει κίνδυνος δεδομένων (data hazard).
  - Κόκκινα βέλη στο σχήμα

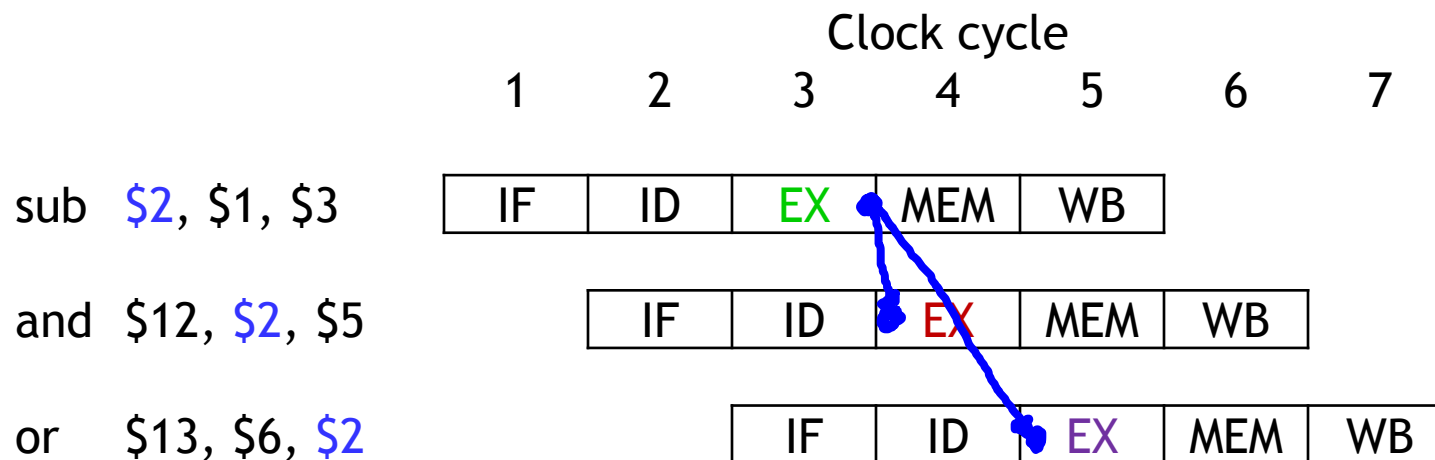
# Ακόμα ένα διάγραμμα





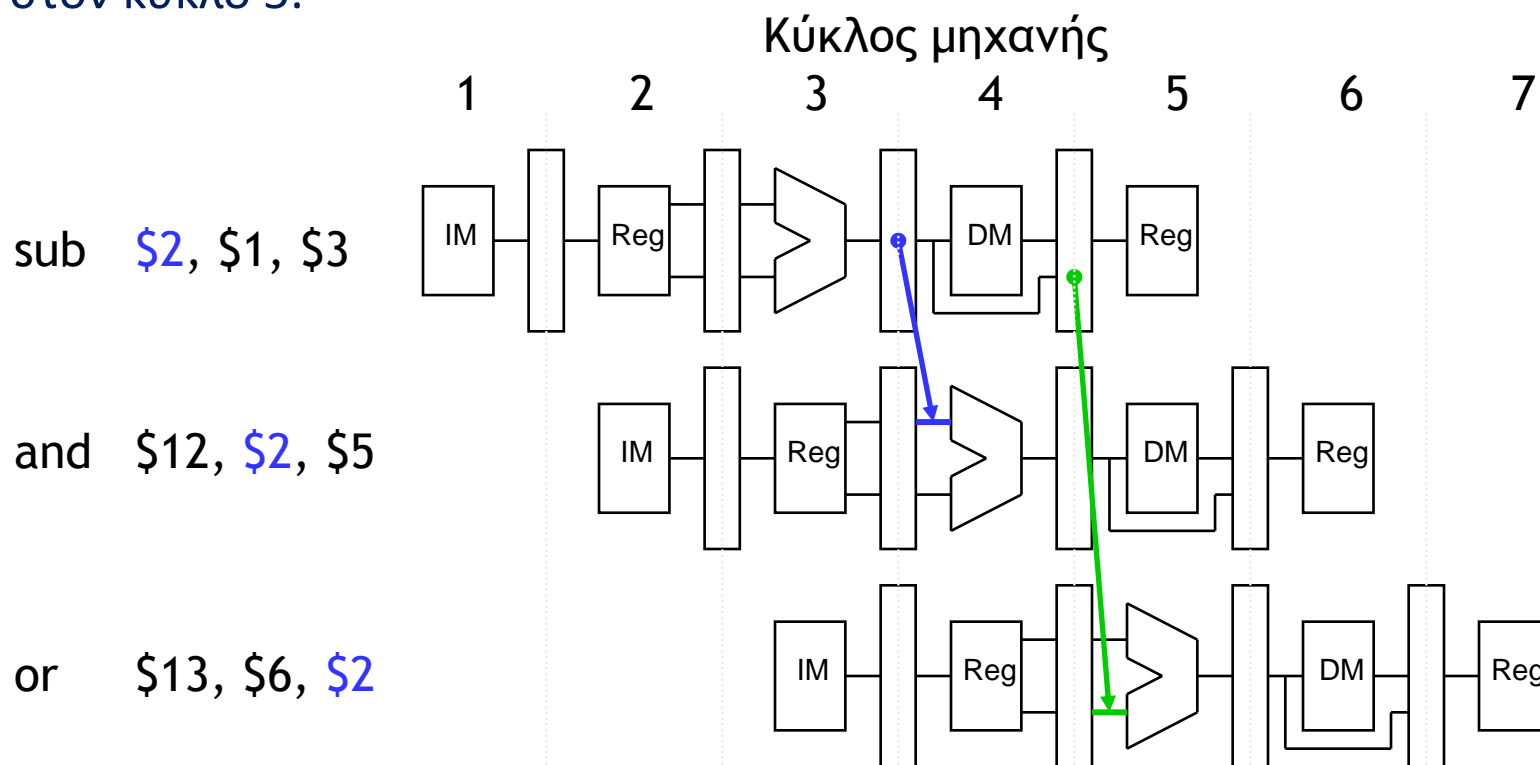
# Κίνδυνοι δεδομένων

- Είναι αναγκαίο να απαλειφθούν οι κίνδυνοι δεδομένων ώστε οι εντολές **and** και **or** να χρησιμοποιήσουν την πραγματική τιμή του καταχωρητή \$2. Αλλιώς η μικρο-αρχιτεκτονική μας είναι λάθος
- Πότε ακριβώς δημιουργείται η νέα τιμή του καταχωρητή \$2 από την εντολή **sub** ;
  - Στο τέλος του σταδίου EX στον κύκλο 3
- Πότε ακριβώς χρειάζεται η νέα τιμή του καταχωρητή \$2 από την εντολής **and** και **or** ;
  - Στο στάδιο EX στους κύκλους 4 και 5



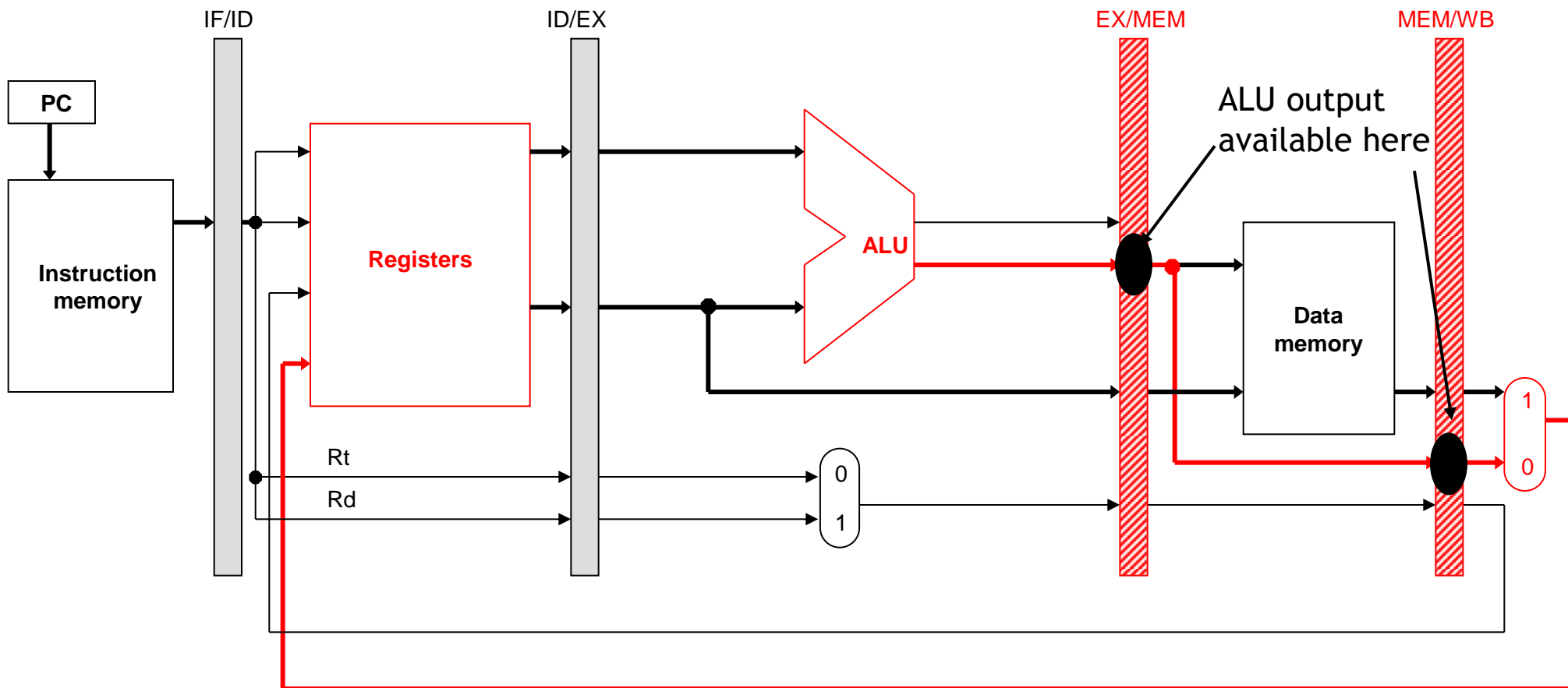
# Πρώθηση δεδομένων (Forwarding)

- Το αποτέλεσμα **\$1-\$3** υπολογίζεται στο τέλος του σταδίου EX στον κύκλο 3 πριν χρειαστεί στους κύκλους 4 και 5
- Προωθούμε αυτό το αποτέλεσμα στις επόμενες εντολές με την χρήση των καταχωρητών διοχέτευσης:
  - Η **and** λαμβάνει την τιμή **\$1-\$3** από τον καταχωρητή διοχέτευσης *EX/MEM* στον κύκλο 4.
  - Η **or** λαμβάνει την τιμή **\$1-\$3** από τον καταχωρητή διοχέτευσης *MEM/WB* στον κύκλο 5.



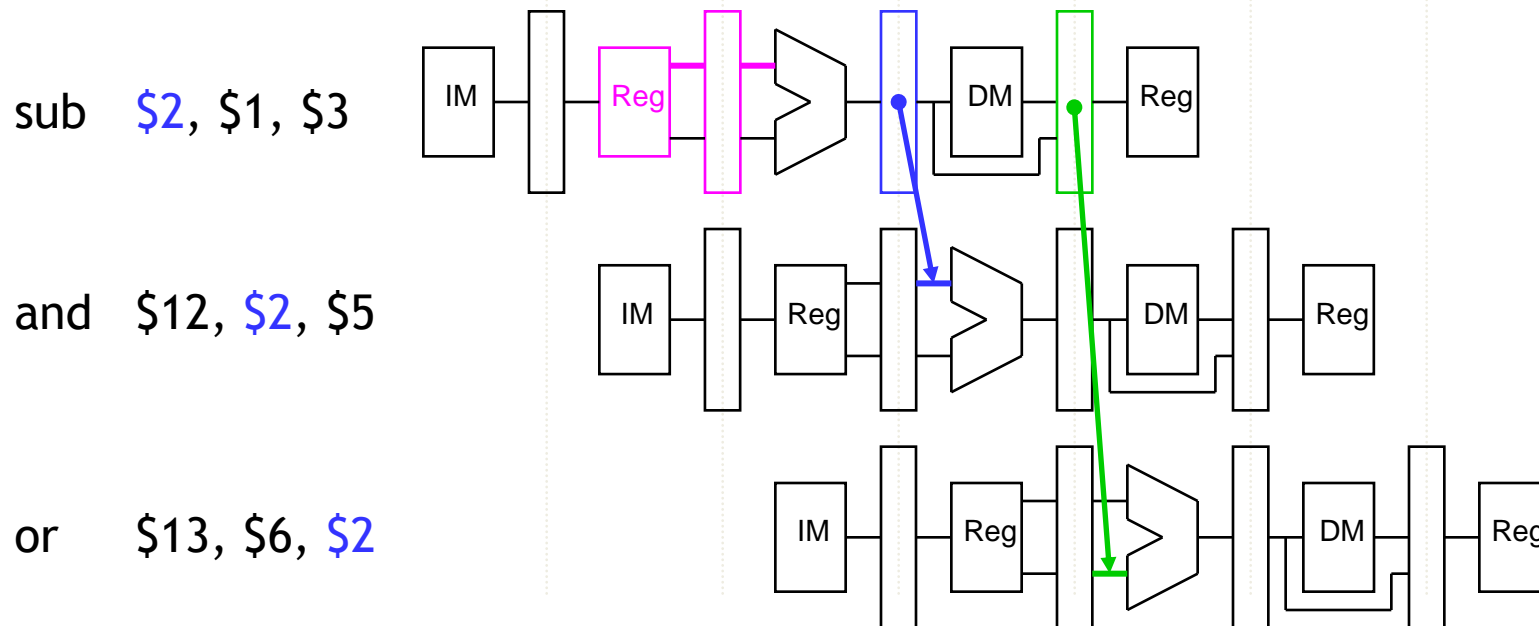
# Πρώθηση δεδομένων (Forwarding)

- Οι καταχωρητές διοχέτευσης IF/ID ID/EX EX/MEM MEM/WB χρησιμοποιούν:
  - για να αποθηκεύουν ενδιάμεσα αποτελέσματα στο τέλος κάθε σταδίου, ΚΑΙ
  - για να προωθούν δεδομένα στις αμέσως επόμενες εντολές πριν αυτά τα δεδομένα γραφούν σε καταχωρητές

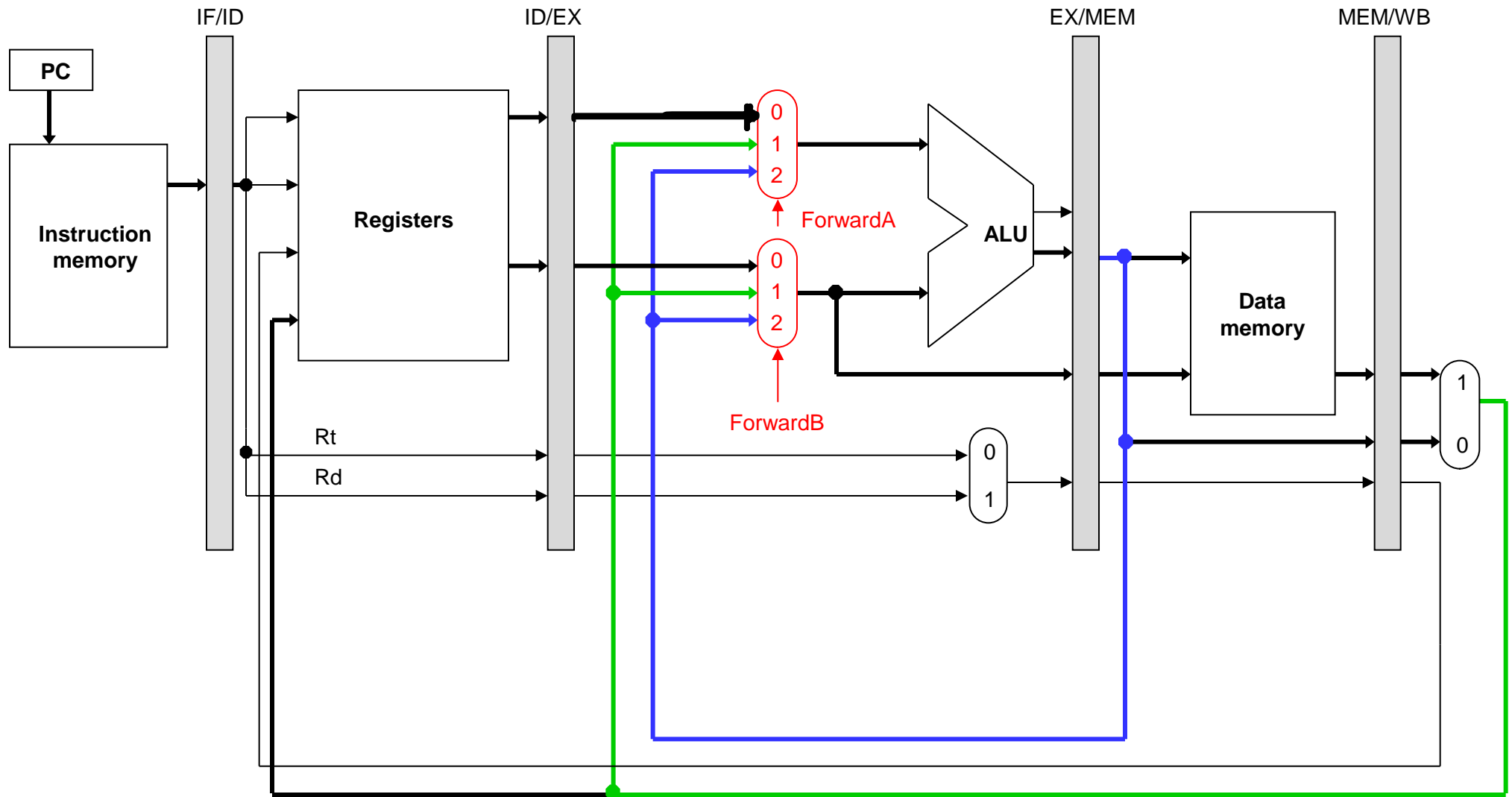


# Μονάδα Προώθησης (Forwarding Unit)

- Η **μονάδα προώθησης** επιλέγει σε κάθε κύκλο μηχανής την σωστή είσοδο της ALU στο στάδιο EX:
  - Εάν δεν υπάρχει κίνδυνος δεδομένων, οι είσοδοι της ALU προέρχονται από τους καταχωρητές, όπως έχουμε δει
  - Εάν υπάρχει κίνδυνος δεδομένων, ένας ή δύο είσοδοι προέρχονται από τους καταχωρητές διοχέτευσης EX/MEM ή MEM/WB
- Δύο νέοι πολυπλέκτες χρησιμοποιούνται (*ForwardA* και *ForwardB*) για να επιλέξουν τις εισόδους της ALU



# Datapath με προώθηση δεδομένων

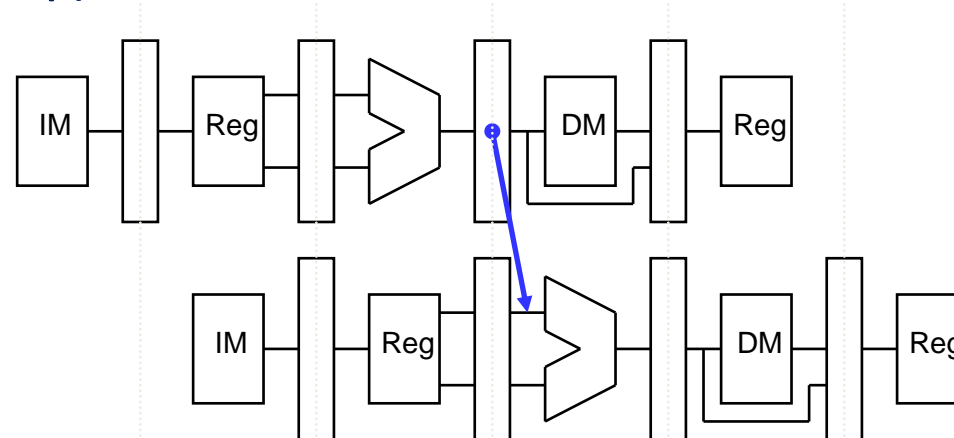


# Καθορισμός κινδύνων δεδομένων EX/MEM

- Πως καθορίζουμε ότι υπάρχει κίνδυνος δεδομένων;
- Ποιος είναι ο μηχανισμός που χρησιμοποιεί το hardware για να το ανακαλύψει;
- Κίνδυνος δεδομένων **EX/MEM** υφίσταται μεταξύ μιας εντολής στο στάδιο EX (and) και της αμέσως προηγούμενης εντολής (sub), εφόσον:
  1. Η αμέσως προηγούμενη εντολή (sub) γράφει σε κάποιον καταχωρητή, και
  2. αυτός ο καταχωρητής είναι είσοδος στην ALU της εντολής (and) στο στάδιο EX.
- Ο συμβολισμός *ID/EX.RegisterRt* σημαίνει το πεδίο *rt* στον καταχωρητή διοχέτευσης *ID/EX*.

sub \$2, \$1, \$3

and \$12, \$2, \$5



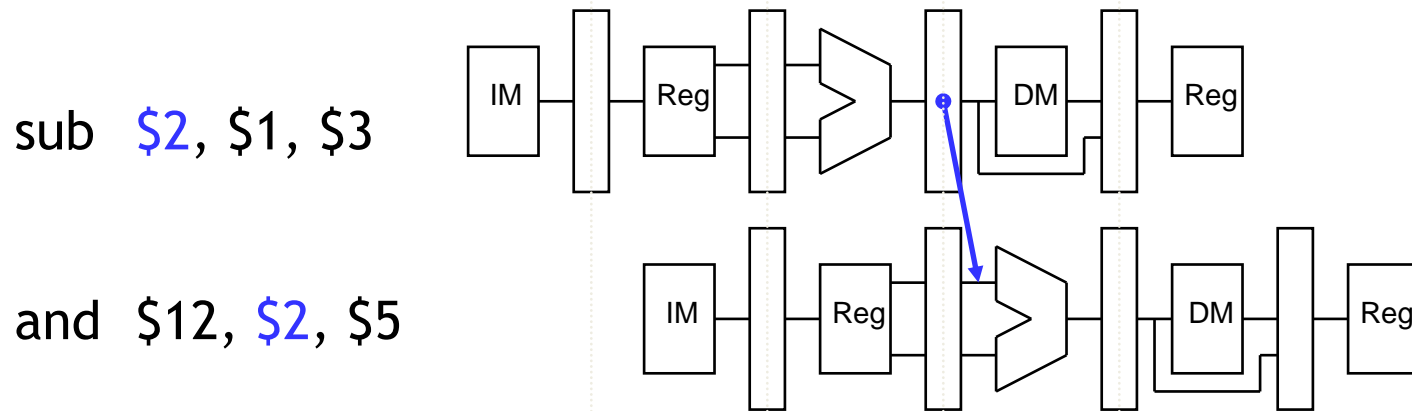
# Εξισώσεις EX/MEM

- Εξίσωση για την πρώτη είσοδο της ALU

```
if (EX/MEM.RegWrite = 1 and  
    EX/MEM.RegisterRd != 0 and  
    EX/MEM.RegisterRd = ID/EX.RegisterRs)  
    then ForwardA = 2
```

- Εξίσωση για την δεύτερη είσοδο της ALU

```
if (EX/MEM.RegWrite = 1 and  
    EX/MEM.RegisterRd != 0 and  
    EX/MEM.RegisterRd = ID/EX.RegisterRt)  
    then ForwardB = 2
```

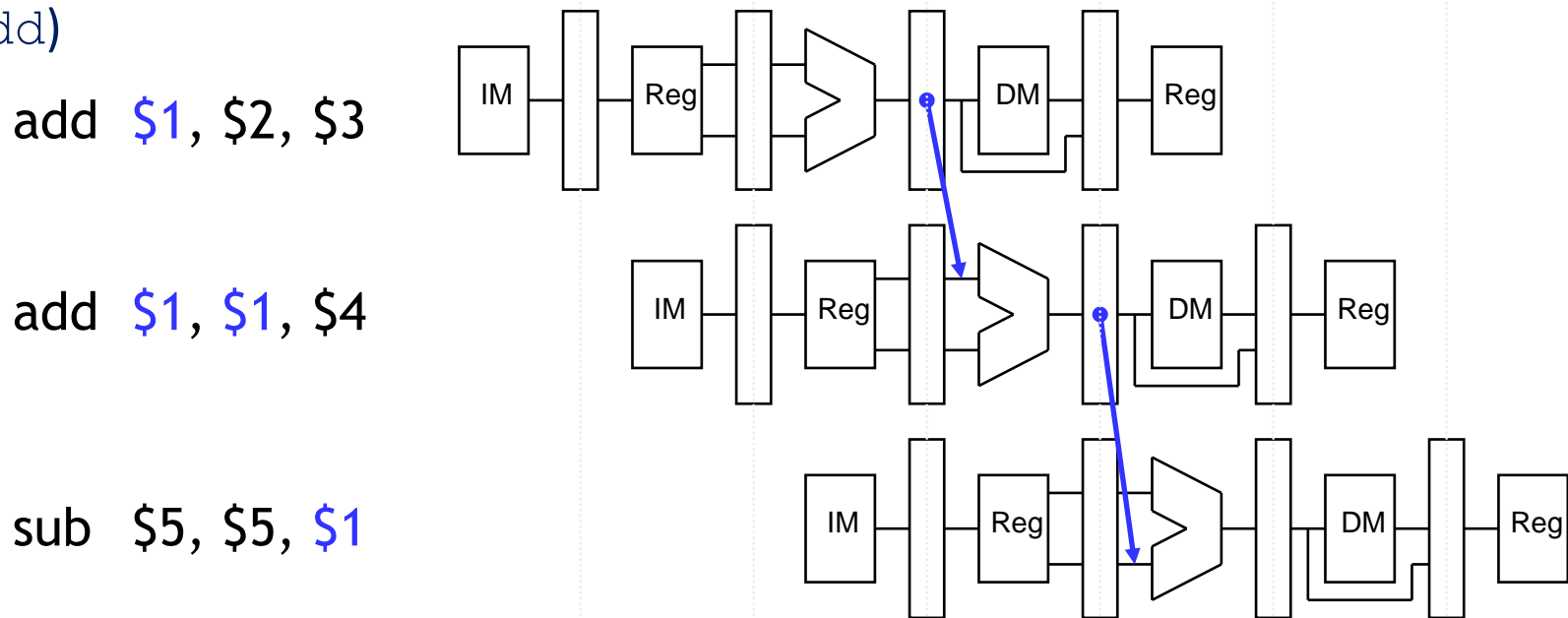


# Καθορισμός κινδύνων δεδομένων MEM/WB

- Κίνδυνος δεδομένων **MEM/WB** μπορεί να υφίσταται μεταξύ μιας εντολής στο στάδιο EX (*sub*) και μιας εντολής δύο κύκλους πριν (*add*).
- Ένα ενδιαφέρον πρόβλημα είναι να γράφουμε έναν καταχωρητή σε διαδοχικές εντολές

```
add  $1, $2, $3
add  $1, $1, $4
sub  $5, $5, $1
```

- Ο καταχωρητής **\$1** γράφεται από δύο διαδοχικές εντολές. Η τρίτη εντολή (*sub*) πρέπει να χρησιμοποιήσει τα δεδομένα της εντολής αμέσως από πάνω (δεύτερη *add*)





# Εξισώσεις MEM/WB

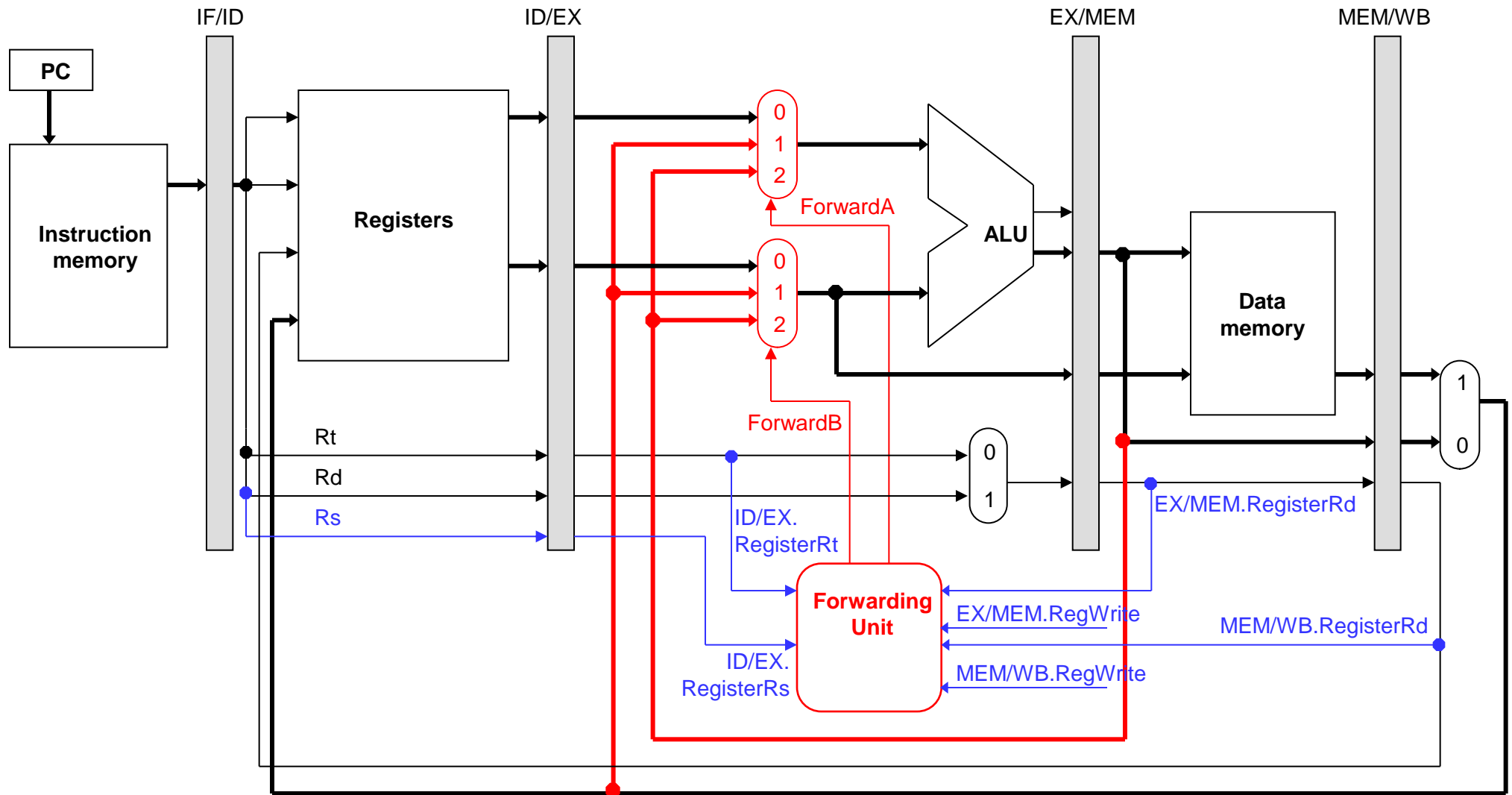
- Εξίσωση MEM/WB για την πρώτη είσοδο της ALU

```
if (MEM/WB.RegWrite = 1 and  
    MEM/WB.RegisterRd != 0 and  
    MEM/WB.RegisterRd = ID/EX.RegisterRs and  
    (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs or EX/MEM.RegWrite = 0)  
then ForwardA = 1
```

- Εξίσωση MEM/WB για την δεύτερη είσοδο της ALU

```
if (MEM/WB.RegWrite = 1 and  
    MEM/WB.RegisterRd != 0 and  
    MEM/WB.RegisterRd = ID/EX.RegisterRt and  
    (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt or EX/MEM.RegWrite = 0)  
then ForwardB = 1
```

# Απλοποιημένο datapath με προώθηση

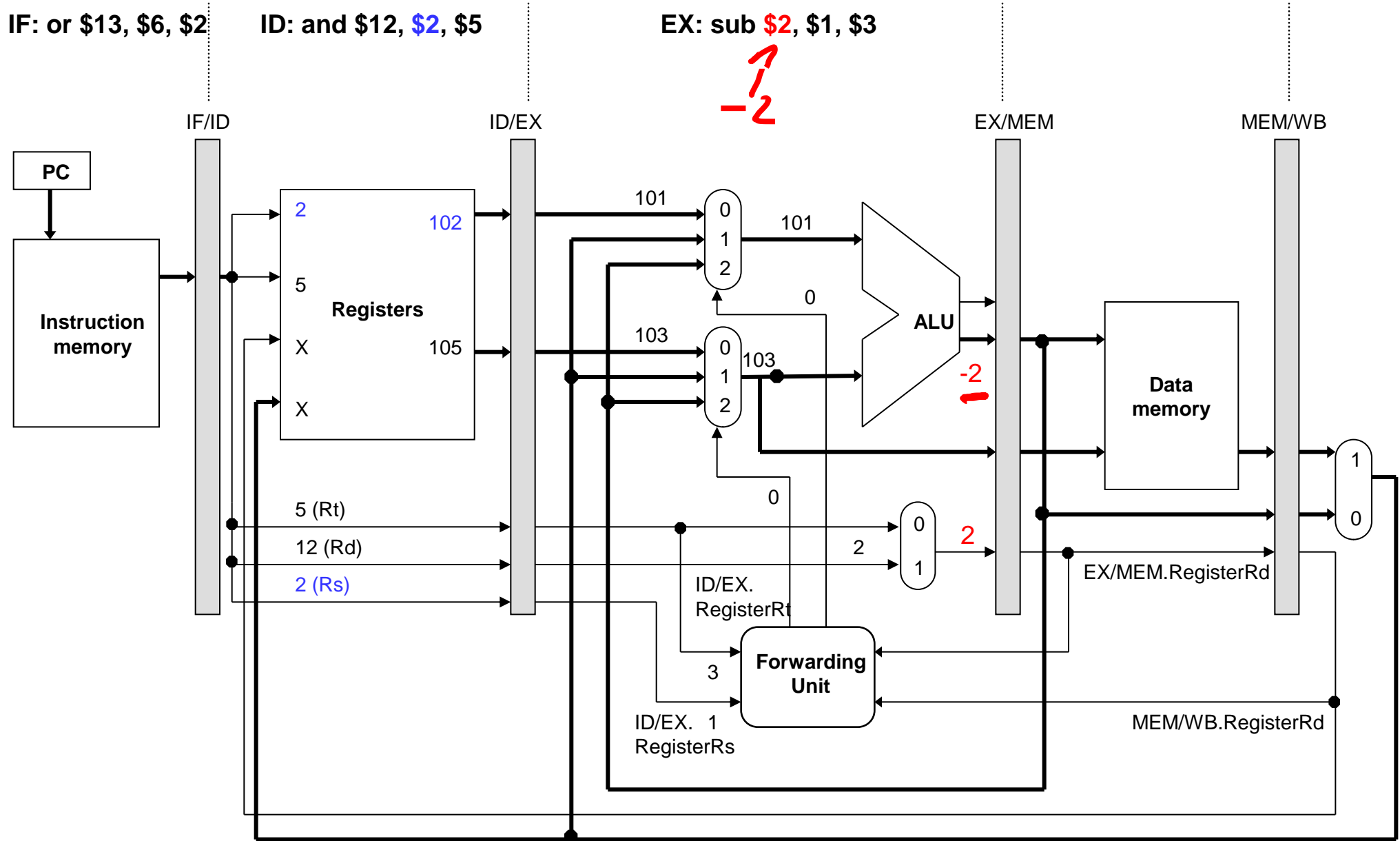


# Παράδειγμα εκτέλεσης στην νέα μικρο-αρχιτεκτονική

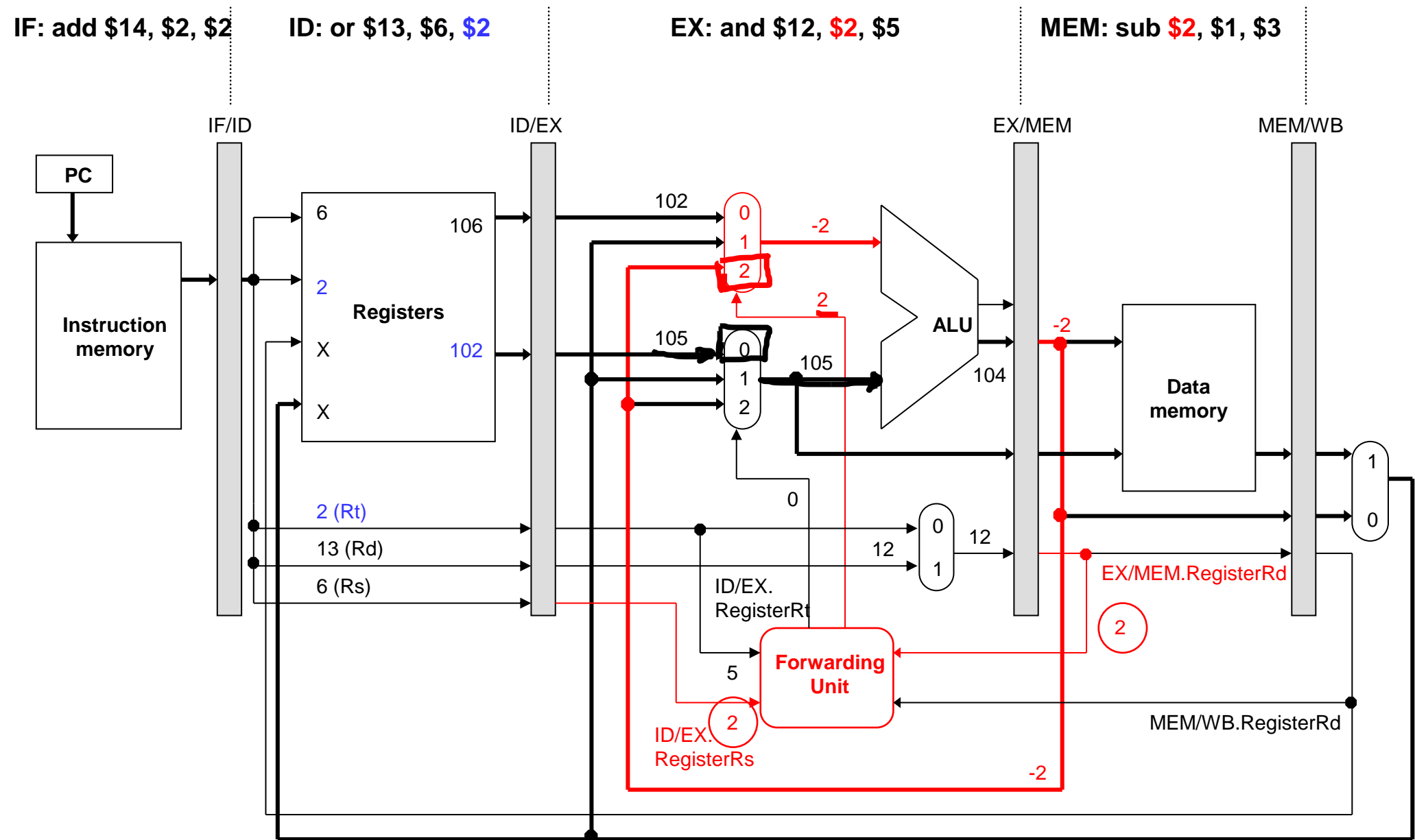
```
sub $2, $1, $3  
and $12, $2, $5  
or $13, $6, $2  
add $14, $2, $2  
sw $15, 100($2)
```

- Θεωρούμε πάλι ότι η αρχική τιμή κάθε καταχωρητή \$N είναι  $N+100$ 
  - Μετά την πρώτη εντολή ο καταχωρητής \$2 ισούται με -2
  - Οι υπόλοιπες εντολές θα πρέπει να χρησιμοποιήσουν το -2 σαν μία από τις εισόδους τους

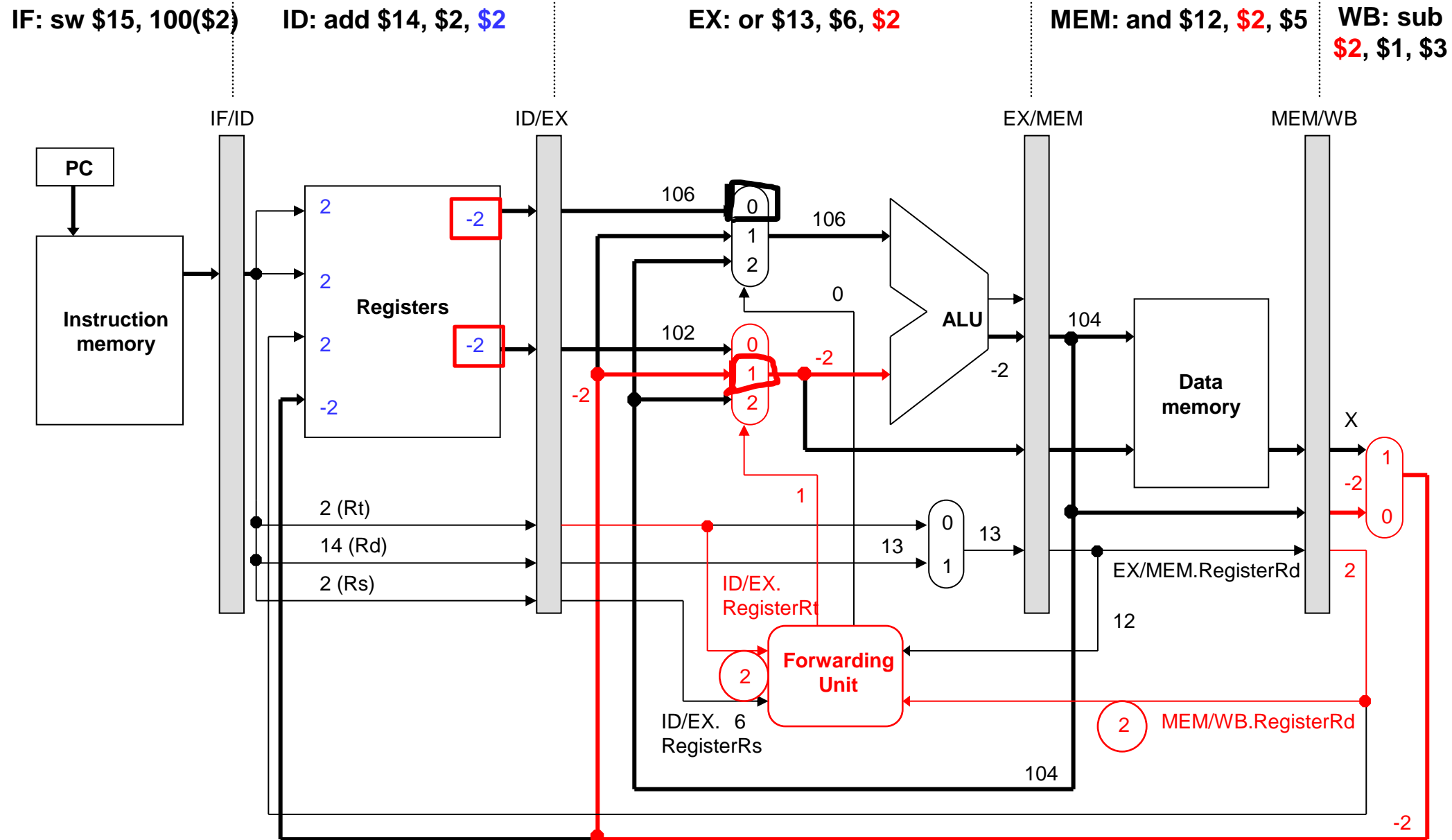
# Κύκλος 3



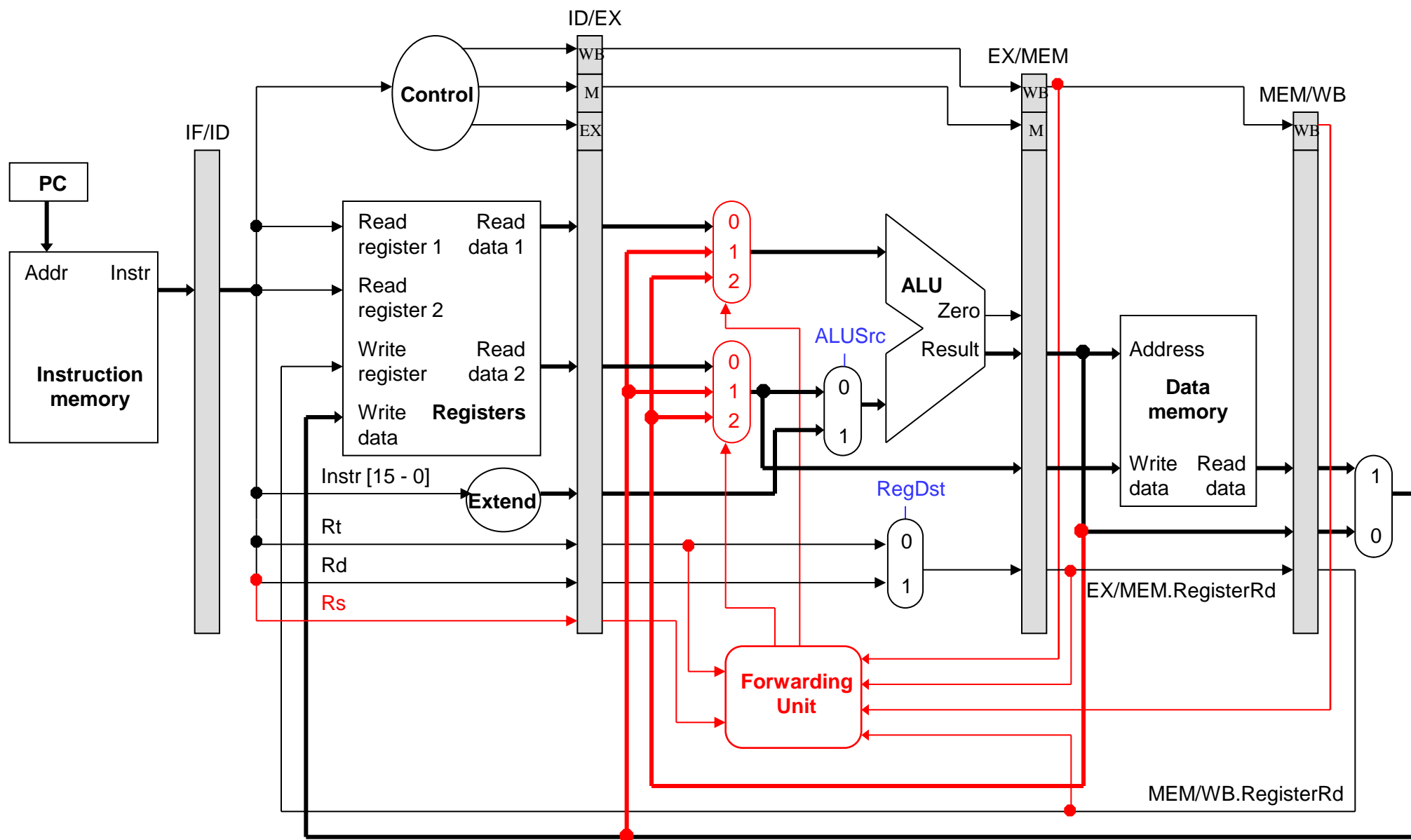
## Κύκλος 4. Προώθηση από καταχωρητή EX/MEM



# Κύκλος 5. Προώθηση από καταχωρητή MEM/WB

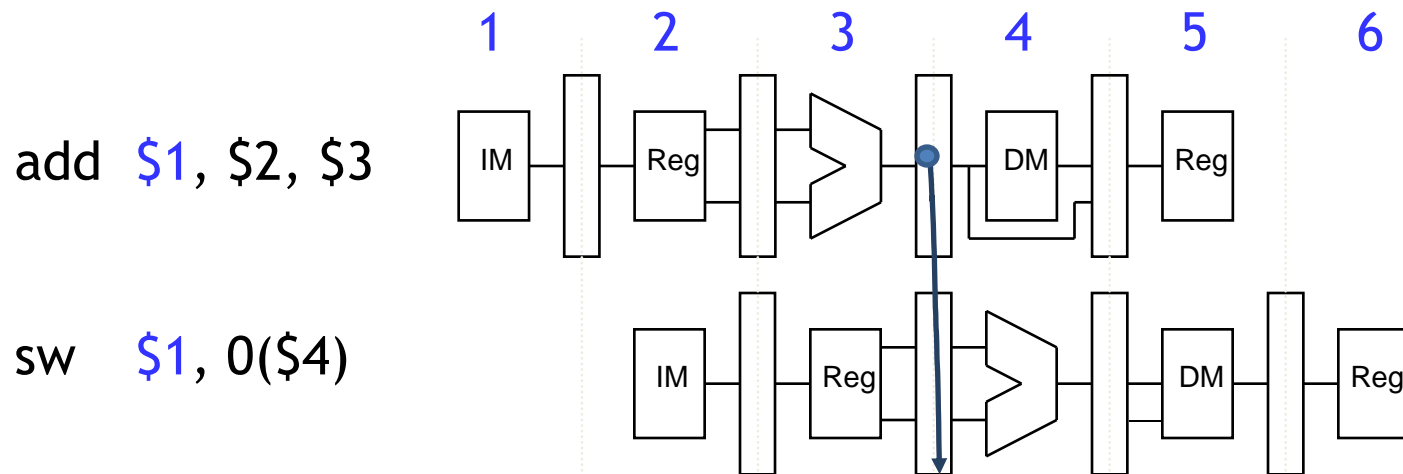
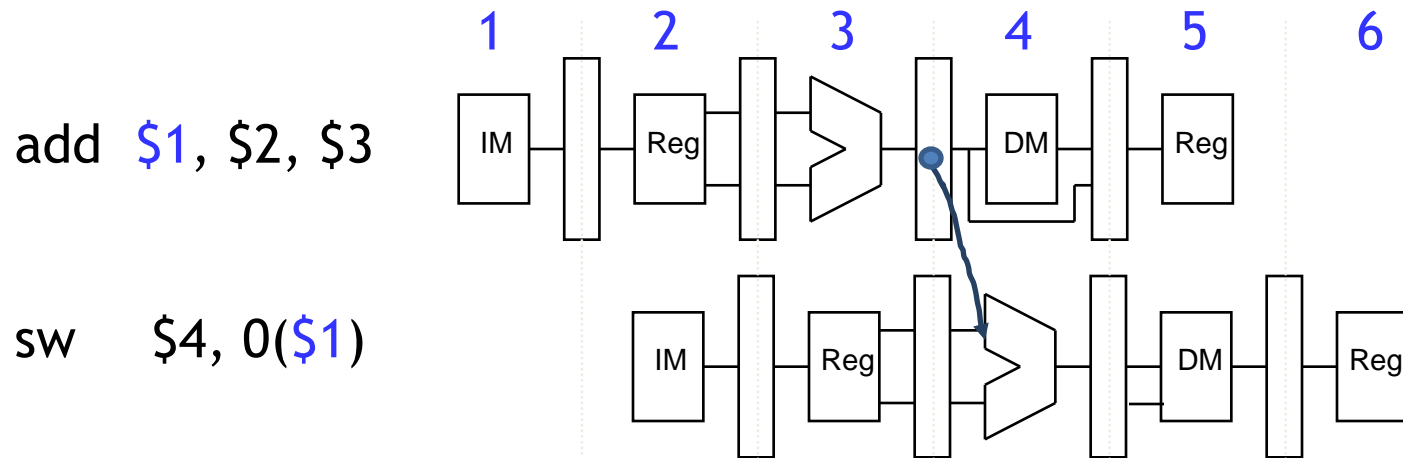


# Η μικρο-αρχιτεκτονική μας μέχρι τώρα



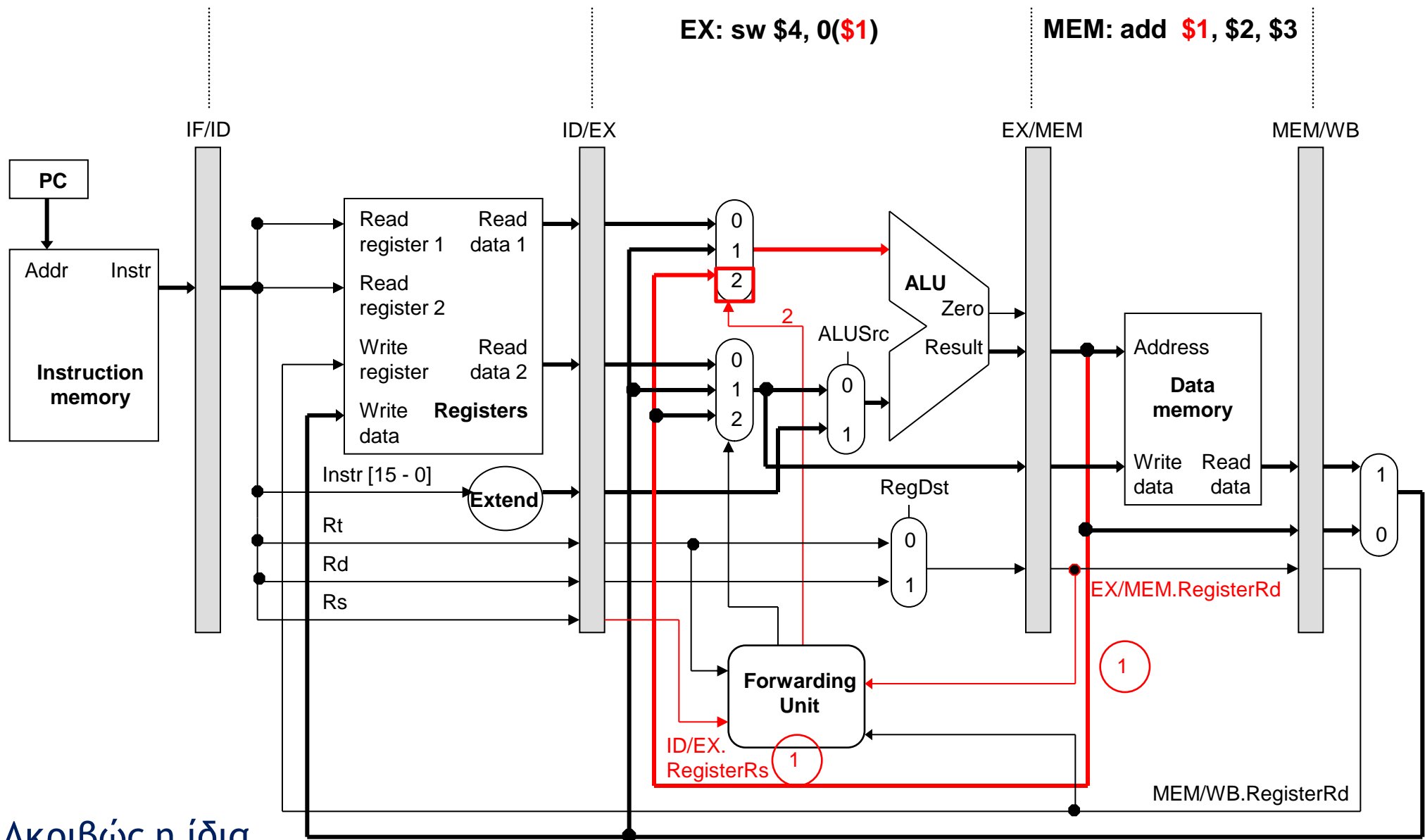
# Τι συμβαίνει με τις εντολές store

- Δύο απλές περιπτώσεις:



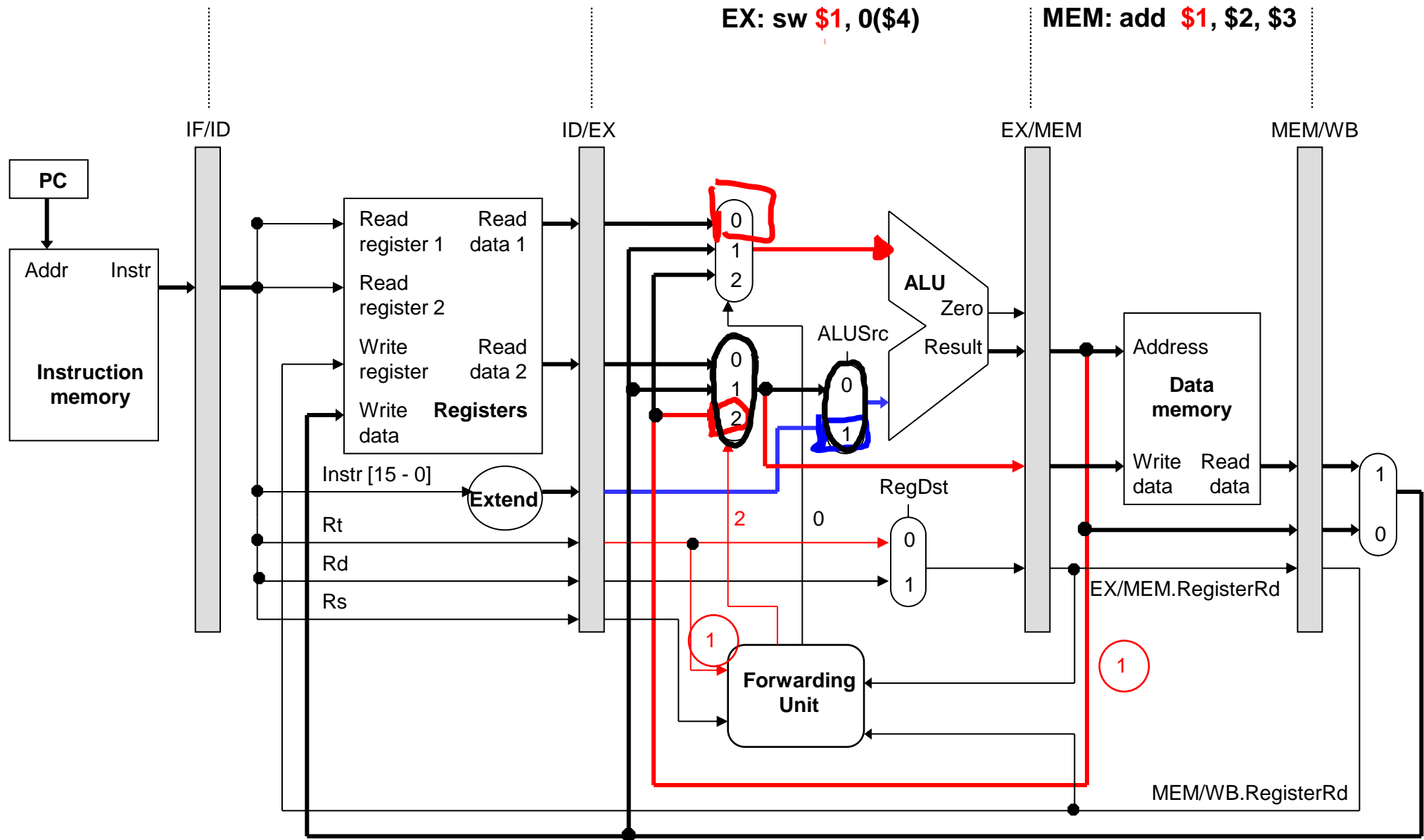


# Πρώθηση με την store (I)



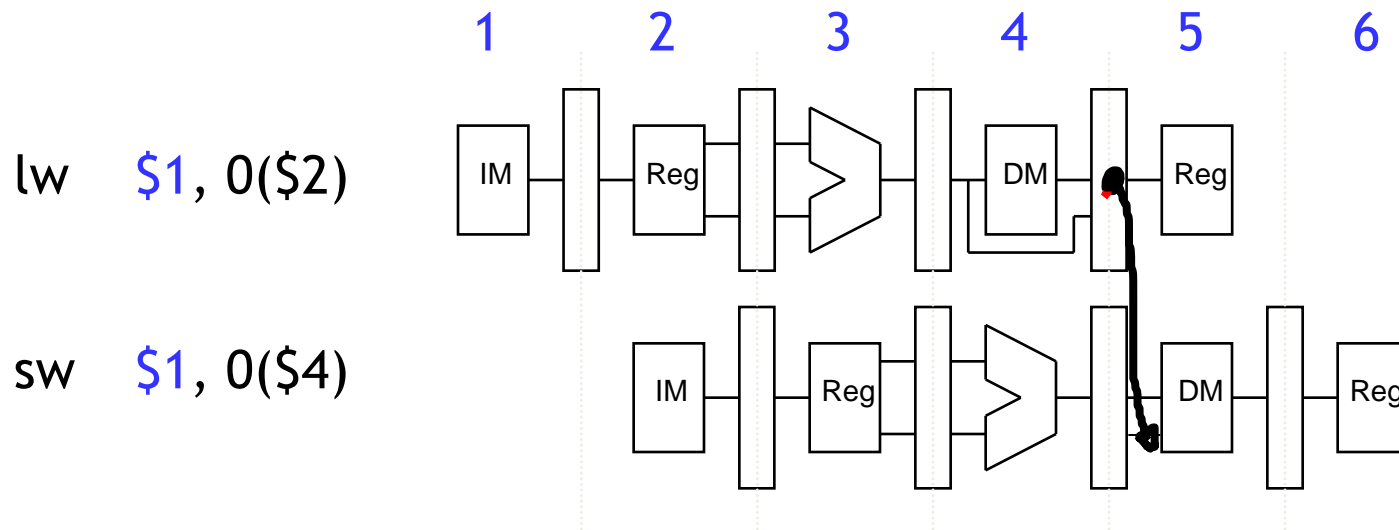
Ακριβώς η ίδια  
περίπτωση εάν έχουμε  
loads

## Πρώθηση με την store (II)



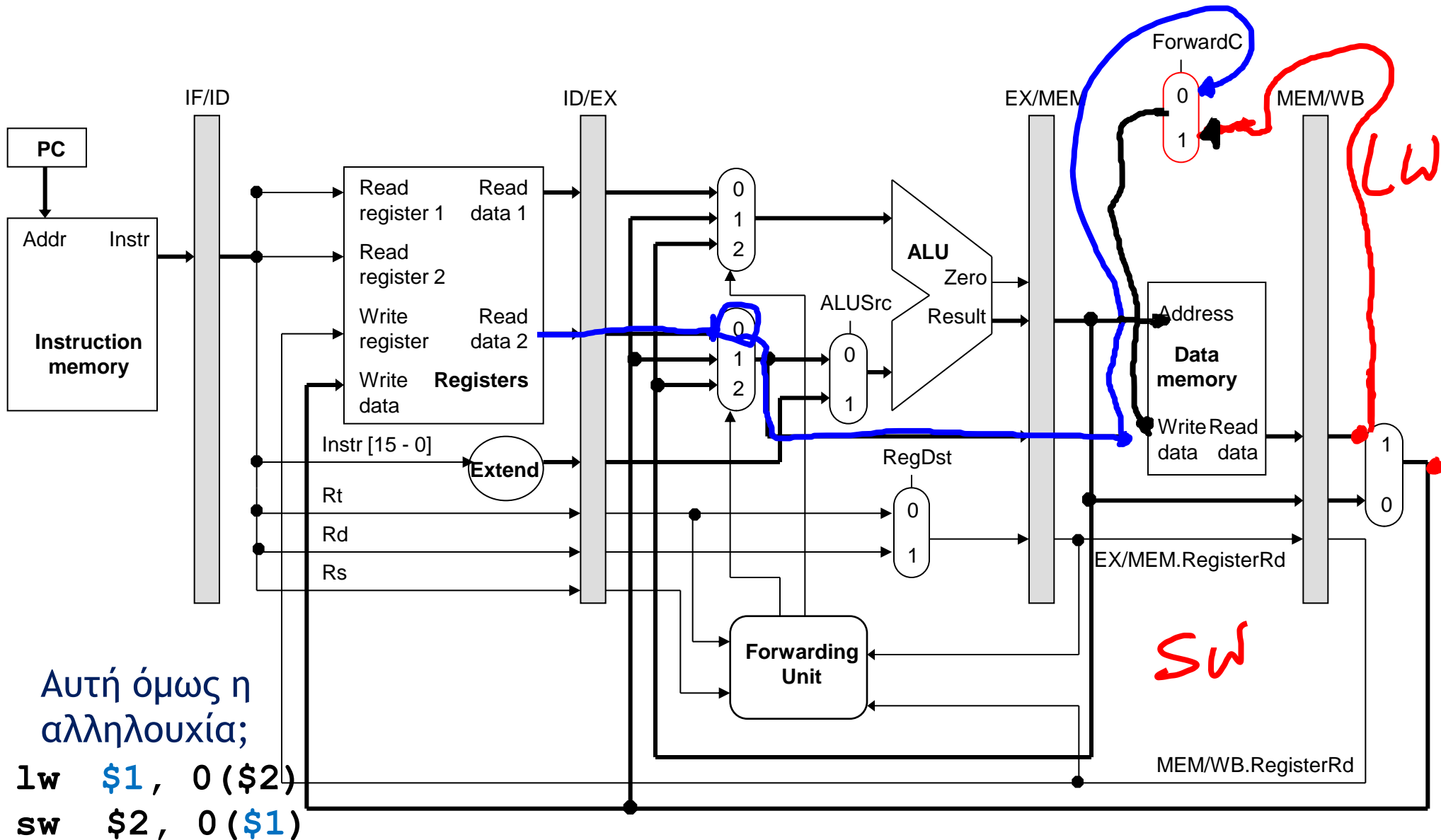
# Τι συμβαίνει με τις εντολές store (III)

- Πιο δύσκολη περίπτωση όταν η store ακολουθεί μια load εντολή:



- Η **lw** παράγει τα δεδομένα της στο τέλος του σταδίου 4.
- Η **sw** χρειάζεται τον \$1 στην αρχή του σταδίου 5.
- Η **lw** δημιουργεί συνήθως την ανάγκη οι επόμενες εντολές να καθυστερήσουν (βλέπε επόμενο μάθημα)
  - Σε αυτήν όμως την συγκεκριμένη περίπτωση μπορούμε να το αποφύγουμε αυτό. Χρειάζεται αλλαγή όμως στο datapath

# Αλληλουχία lw-sw



# Η συνολική εικόνα

- Η απλότητα της αρχιτεκτονικής MIPS κάνει το πρόβλημα της προώθησης πολύ εύκολο.
- Κάθε εντολή MIPS μπορεί να γράψει το πολύ σε έναν καταχωρητή
  - Αυτή η ιδιότητα απλουστεύει πολύ την σχεδίαση της μονάδας προώθησης μιας και υπάρχει μόνο ένας καταχωρητής προορισμού που πρέπει να προωθηθεί.
- Η προώθηση είναι πολύ σημαντική σε μικρο-αρχιτεκτονικές με πολλά στάδια διοχέτευσης (super-pipelining)
  - Η σύγχρονοι επεξεργαστές έχουν 20+ στάδια διοχέτευσης
  - Γιατί τόσα πολλά;