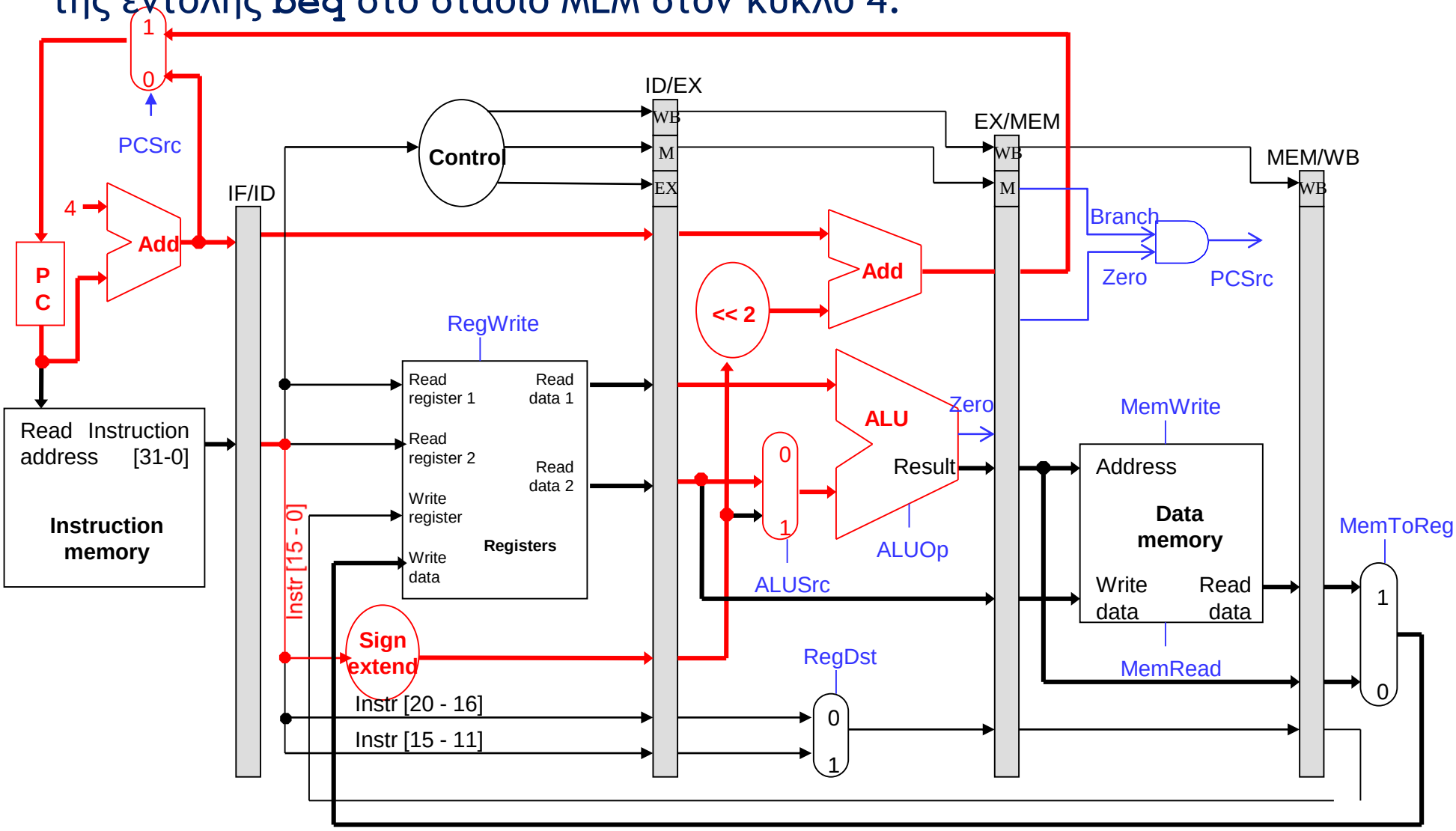
ΗΥ 232
Οργάνωση και Σχεδίαση Υπολογιστών

Διάλεξη 13
Διακλαδώσεις

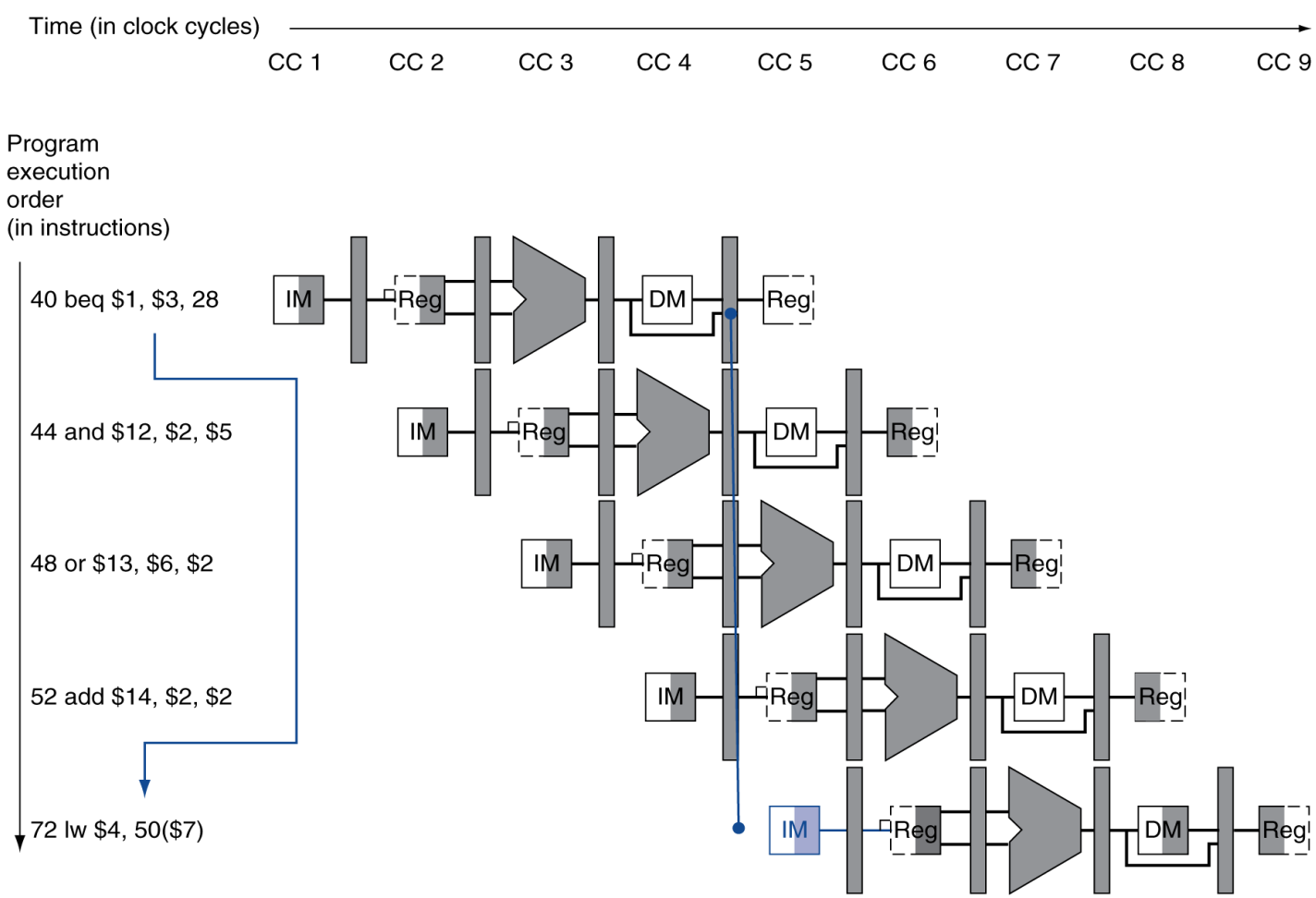
Νίκος Μπέλλας
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ

Η μέχρι τώρα μικρο-αρχιτεκτονική (Εντολές Διακλάδωσης)

Η μικρο-αρχιτεκτονική μας παίρνει απόφαση για την κατεύθυνση και τον προορισμό της εντολής **beq** στο στάδιο MEM στον κύκλο 4.



Κίνδυνοι Ελέγχου (Control Hazards)



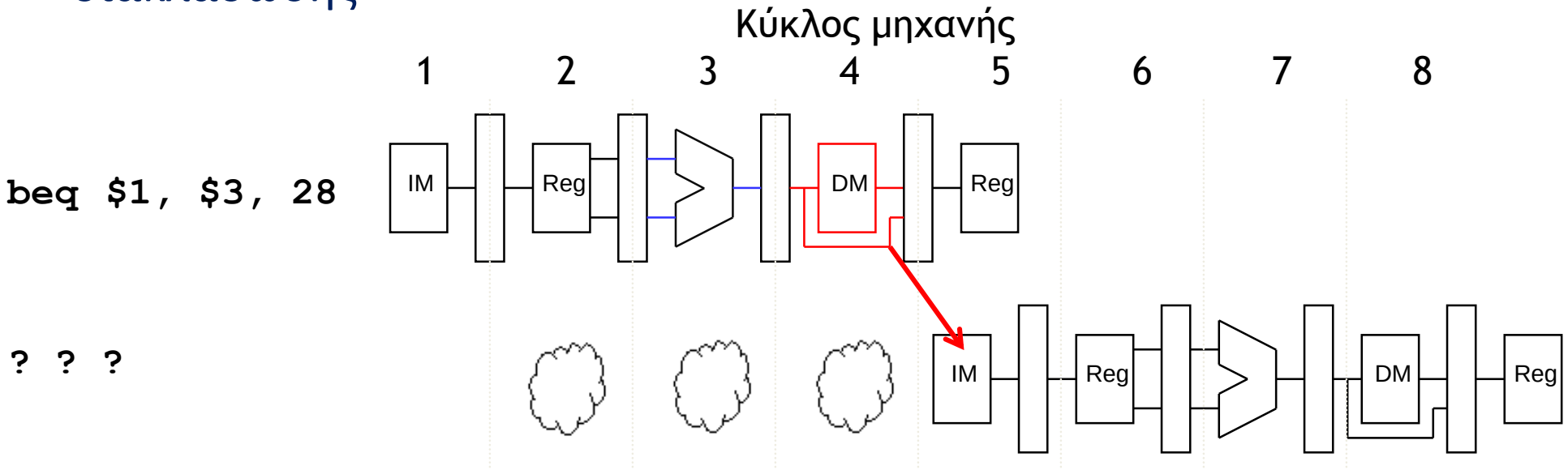
Η μικρο-αρχιτεκτονική μας παίρνει απόφαση για την κατεύθυνση της εντολής **`beq`** στο στάδιο MEM στον κύκλο 4.

Αλλά πρέπει να ξέρουμε αυτήν απόφαση πιο νωρίς για να φέρουμε την σωστή επόμενη εντολή.

Αλλιώς, αντί να φέρουμε την **`lw`** εάν η εντολή **`beq`** κάνει άλμα (TAKEN), θα φέρουμε την **`and`**

Καθυστέρηση (Stalling)

- Μία λύση είναι να καθυστερήσουμε το pipeline κατά 3 κύκλους μέχρι να πάρουμε την απόφαση για το αποτέλεσμα της εντολής διακλάδωσης



- Στην συγκεκριμένη περίπτωση, θα πρέπει να δημιουργήσουμε 3 εντολές **`nop`** ώστε στην συνέχεια να φέρουμε την σωστή εντολή από την μνήμη.
- Τρείς κύκλοι κάθε φορά που έχουμε εντολή διακλάδωσης είναι πολλοί. Μπορούμε να τους μειώσουμε; Είναι σημαντικό;

Ανάλυση Απόδοσης

- Όχι άλμα ⇒ Καμία επιβάρυνση
- Άλμα ⇒ 3 κύκλοι επιβάρυνσης
- Έστω ότι
  - 20% εντολές διακλάδωσης
  - 50% των εντολών διακλάδωσης κάνουν άλμα
  - Νέο CPI = [1 + (0.20*0.50) * 3] =

    = [1 + 0.1 * 3] = 1.3 (30% επιβάρυνση)

Πιθανότητα λανθασμένου μονοπατιού

Επιβάρυνση (penalty) λανθασμένου μονοπατιού

Πως μπορούμε να μειώσουμε τους δύο αυτούς όρους;

1η Μέθοδος μείωσης της επιβάρυνσης: Μείωση των κύκλων καθυστέρησης

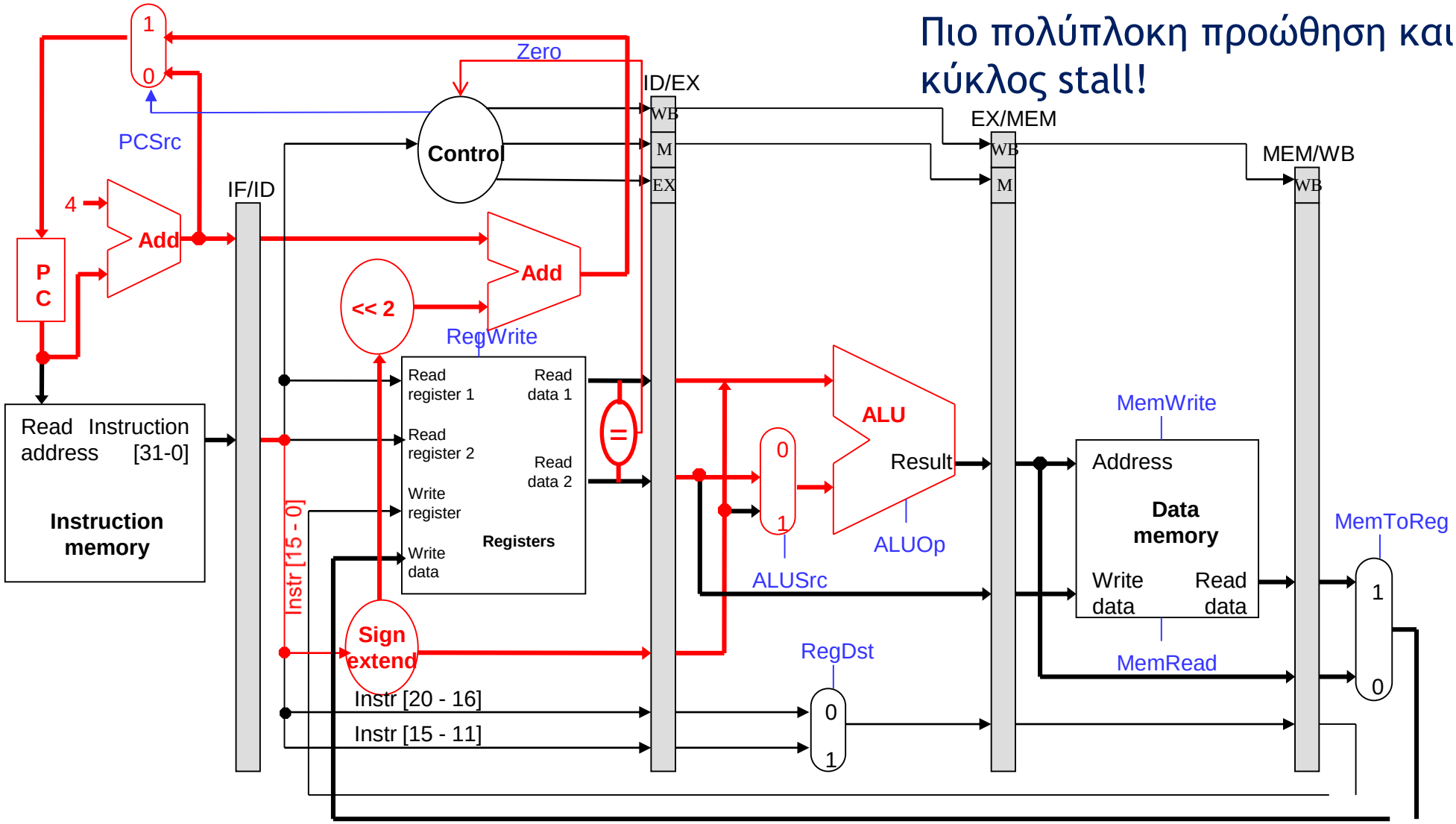
- Η εκτέλεση μιας εντολής διακλάδωσης (**`beq rs, rt, Label`**) απαιτεί δύο πράγματα:
  - Τον υπολογισμό της διεύθυνσης προορισμού: PC <= Label
  - Την απόφαση για το εάν η εντολή διακλάδωσης είναι TAKEN ή NOT TAKEN
- Και οι δύο αυτοί υπολογισμοί πρέπει να γίνουν νωρίτερα (δηλ. να μεταφερθούν πιο πριν στο pipeline) για να μπορέσουμε να μειώσουμε την ποινή της καθυστέρησης. Θα μεταφέρουμε και τους δύο αυτούς υπολογισμούς στο στάδιο ID.
  - Ο υπολογισμός της διεύθυνσης προορισμού μπορεί εύκολα να μεταφερθεί από το στάδιο EX στο στάδιο ID με το να μεταφέρουμε την μικρή ALU και το left shift by 2.
  - Η απόφαση για την διακλάδωση μπορεί να γίνει με επιπλέον hardware που απλά θα ελέγχει εάν οι 2 καταχωρητές **`rs`** και **`rt`** που μόλις διαβάστηκαν από το register file είναι ίσοι.
  - Αυτό γίνεται απλά με 32 πύλες XOR. Μπορεί όμως να αυξήσει την περίοδο του ρολογιού.

Νέο pipeline για μικρότερο penalty στις εντολές διακλάδωσης

Τι θα γίνει εάν έχουμε τις εντολές

```
sub   $2, $1, $3
beq   $4, $2, L
```

Πιο πολύπλοκη προώθηση και ένας κύκλος stall!



2η Μέθοδος μείωσης της επιβάρυνσης: Καθυστερημένες Εντολές Διακλάδωσης

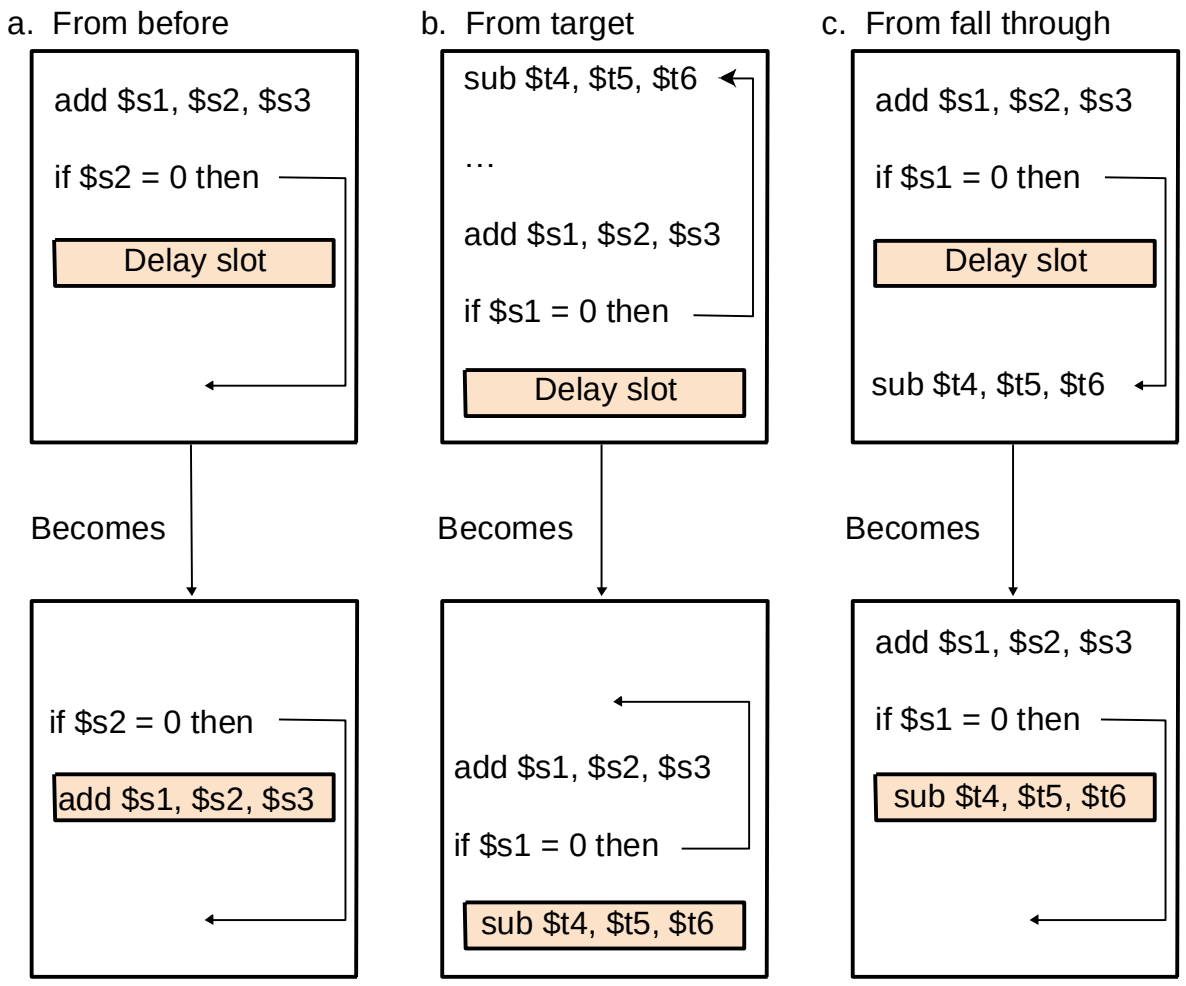
- Ιδέα: αφού έτσι και αλλιώς είμαστε υποχρεωμένοι να φέρουμε *N* extra εντολές στην CPU, ας προσπαθήσουμε να είναι χρήσιμες.
  - Στην αρχική αρχιτεκτονική N=3, ενώ στην βελτιωμένη N=1.
- Οι N επόμενες εντολές μετά την διακλάδωση εκτελούνται κανονικά.
  - Χωρίς pipeline flush
- *Delayed branch* : η διακλάδωση ουσιαστικά καθυστερεί N εντολές.
- ΄Η τιμή του N εξαρτάται από την αρχιτεκτονική

Delayed Branch

Ο compiler θα πρέπει να βρει Ν χρήσιμες εντολές και να τις τοποθετήσει μετά την διακλάδωση. Εδώ **N = 1.**

Όχι πάντα δυνατόν. Ίσως χρειαστούν extra εντολές από τον compiler.

Η μέθοδος χρησιμοποιείται μόνο για μικρά pipelines (πχ N=1).



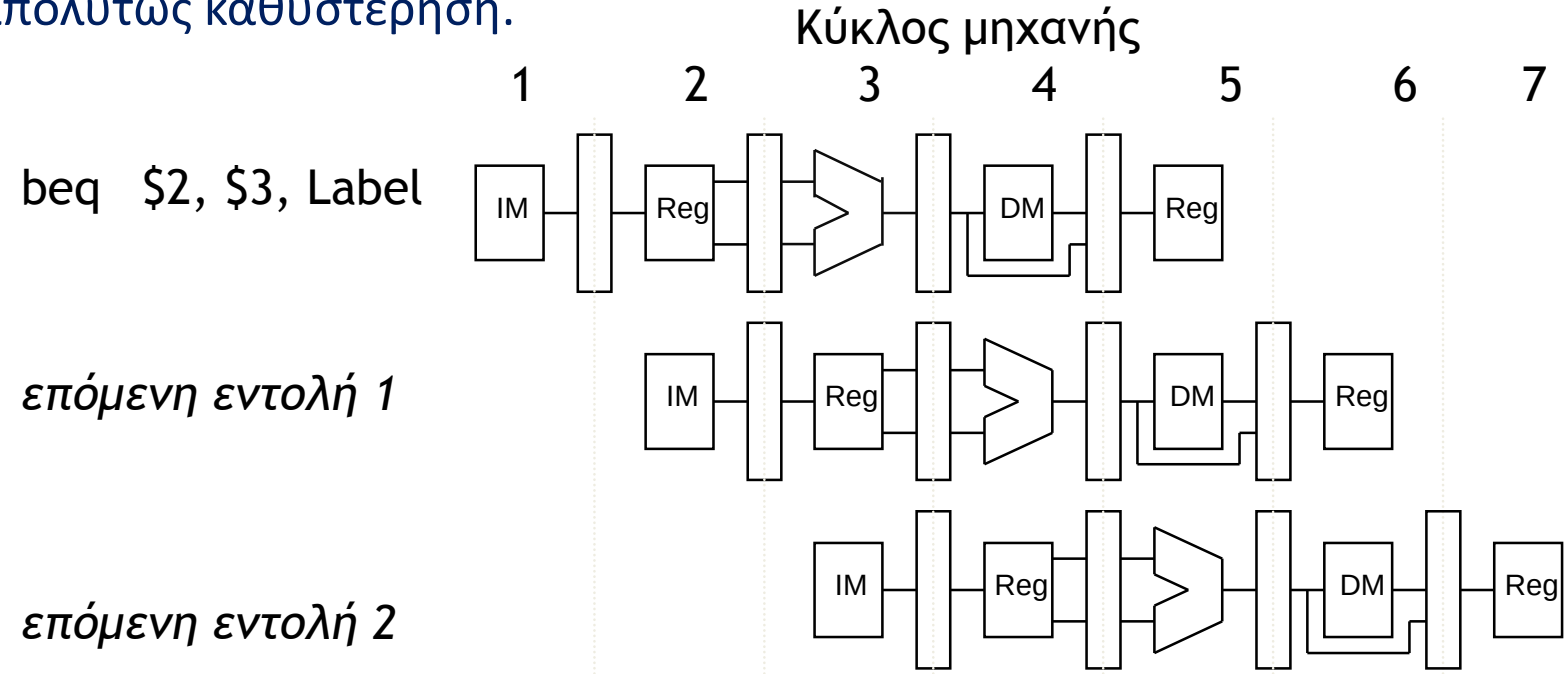
Εύκολο

Ίσως χρειάζονται αλλαγές στον κώδικα

Ίσως χρειάζονται αλλαγές στον κώδικα

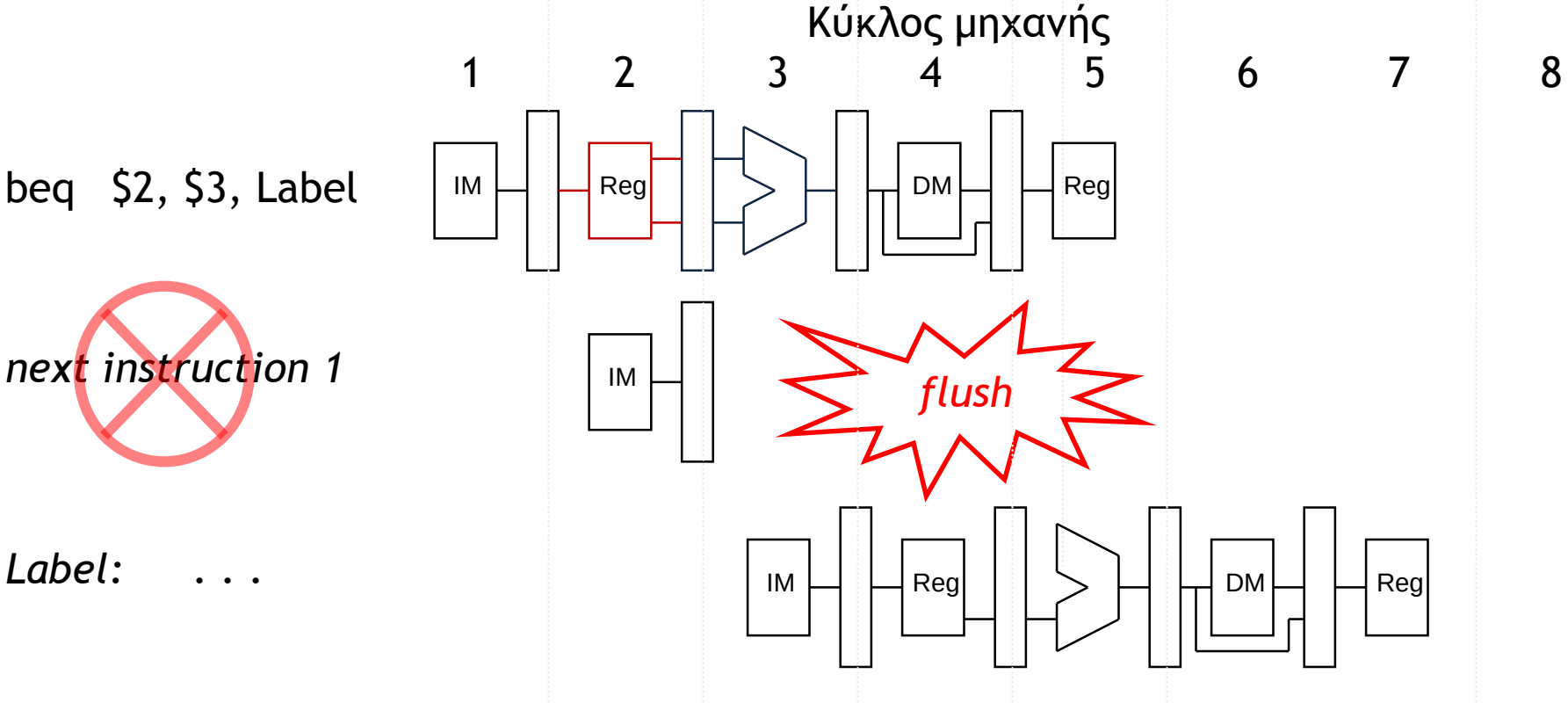
3η Μέθοδος μείωσης της επιβάρυνσης: Πρόβλεψη διακλάδωσης

- Αφού μειώσαμε τους κύκλους καθυστέρησης από 3 σε 1 και σε 0 ας δούμε πως μπορούμε να μειώσουμε ακόμα περαιτέρω τους κινδύνους ελέγχου
- Μπορούμε να “μαντέψουμε” την απόφαση της διακλάδωσης με διάφορους τρόπους
  - Η πιο απλή προσέγγιση είναι να θεωρήσουμε ότι η διακλάδωση είναι πάντα NOT TAKEN.
  - Απλά αύξησε τον PC<=PC+4 και φέρε την επόμενη εντολή ως σαν να μην είχαμε διακλάδωση
- Εάν αποδειχθούμε σωστοί, τότε το pipeline προχωράει full speed χωρίς καμία απολύτως καθυστέρηση.



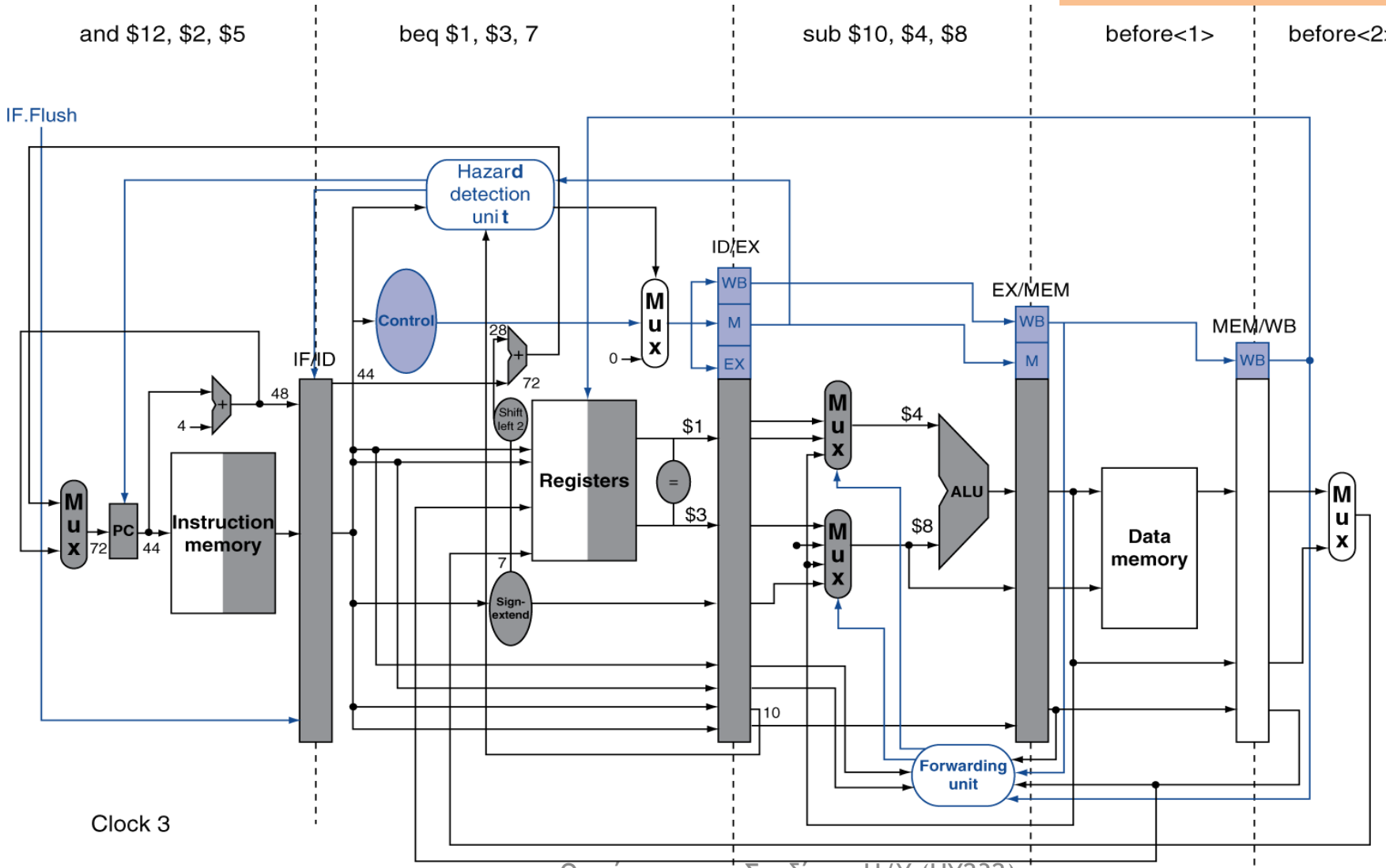
Πρόβλεψη διακλάδωσης

- Εάν η υπόθεσή μας είναι λάθος και η διακλάδωση είναι TAKEN, τότε θα έχουμε φέρει λάθος εντολή.
  1. Θα πρέπει να κάνουμε flush την εντολή αυτή και να βάλουμε στην θέση της μια εντολή **nop**. Το flush γίνεται στον καταχωρητή IF/ID
  2. Θα πρέπει επίσης να αλλάξουμε τον PC ώστε να αρχίσουμε αμέσως μετά να διαβάζουμε εντολές από την διεύθυνση μνήμης Label.
- Σε αυτήν την περίπτωση θα έχουμε έναν κύκλο καθυστέρηση (για N=1)

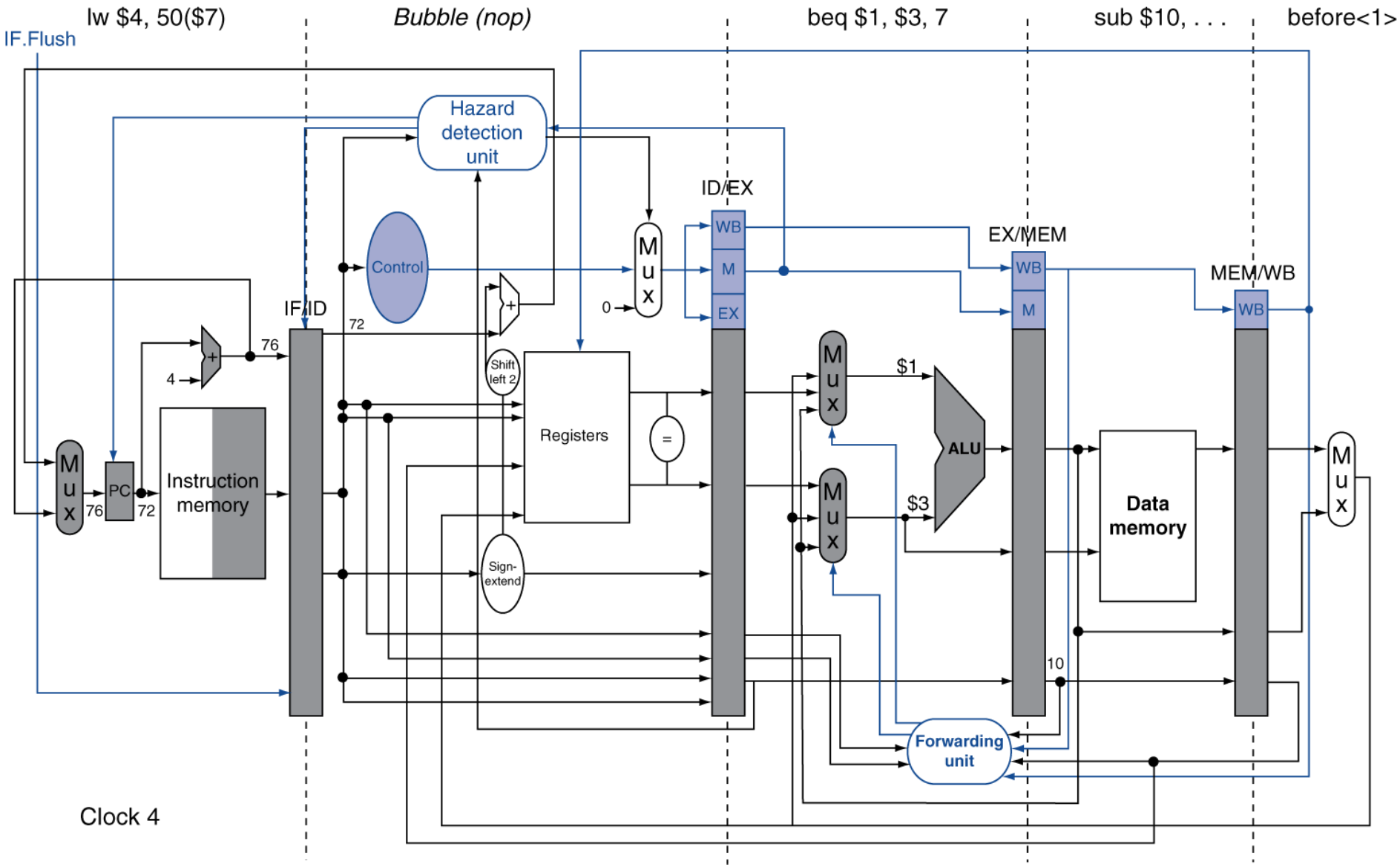


Παράδειγμα Λανθασμένης Πρόβλεψης

```
36:   sub   $10, $4, $8
40:   beq   $1,  $3, 7
44:   and   $12, $2, $5
 ...
72:   lw    $4, 50($7)
```



Παράδειγμα Λανθασμένης Πρόβλεψης



Στατική Πρόβλεψη Διακλάδωσης

- Στατική πρόβλεψη διακλάδωσης σημαίνει ότι η πρόβλεψη γίνεται από τον προγραμματιστή ή τον compiler πριν την εκτέλεση του προγράμματος.
- Στατικοί τρόποι πρόβλεψης:

1. Όλες οι διακλαδώσεις προβλέπονται ως NOT TAKEN
2. Backward taken, forward not taken (BTFN):
   - όλες οι διακλαδώσεις προς τα πίσω προβλέπονται ως TAKEN
   - όλες οι διακλαδώσεις προς τα μπροστά προβλέπονται ως NOT TAKEN
3. Profile-driven
   - Ο compiler χρησιμοποιεί αποτελέσματα από προηγούμενες εκτελέσεις του κώδικα για να καταλάβει την συμπεριφορά μιας διακλάδωσης
4. Ο ίδιος ο προγραμματιστής δίνει ενδείξεις για την κατεύθυνση μιας εντολής διακλάδωσης:
   - if (likely(x)) { ... }
   - if (unlikely(error)) { … }

Δυναμική Πρόβλεψη Διακλάδωσης

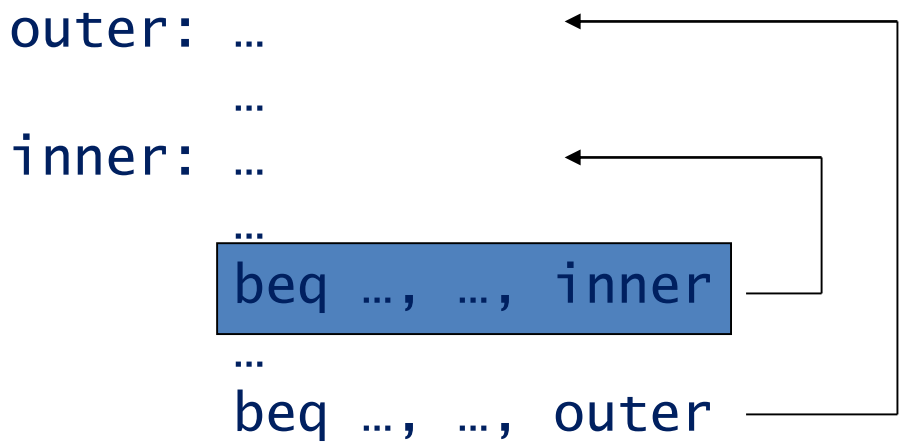
- Στους σύγχρονους επεξεργαστές με μεγάλο αριθμό σταδίων διοχέτευσης (~20), ο αριθμός των κύκλων μηχανής που πληρώνουμε για λάθος πρόβλεψης διακλάδωσης είναι πολύ μεγαλύτερος.
  - Σημαντικός παράγοντας για την χαμηλή απόδοση του συστήματος.
- Η δυναμική πρόβλεψη της κατεύθυνσης μιας διακλάδωσης προσπαθεί να "μάθει" από την συμπεριφορά προηγούμενων εκτελέσεων.

Δυναμική Πρόβλεψη Διακλάδωσης

- **Κανόνας 1:** Το αποτέλεσμα μιας εντολής διακλάδωσης Β (Taken / Not Taken) εξαρτάται από προηγούμενες εκτελέσεις της εντολής αυτής
  - Υπάρχει συσχέτιση σε διαδοχικές εκτελέσεις της Β

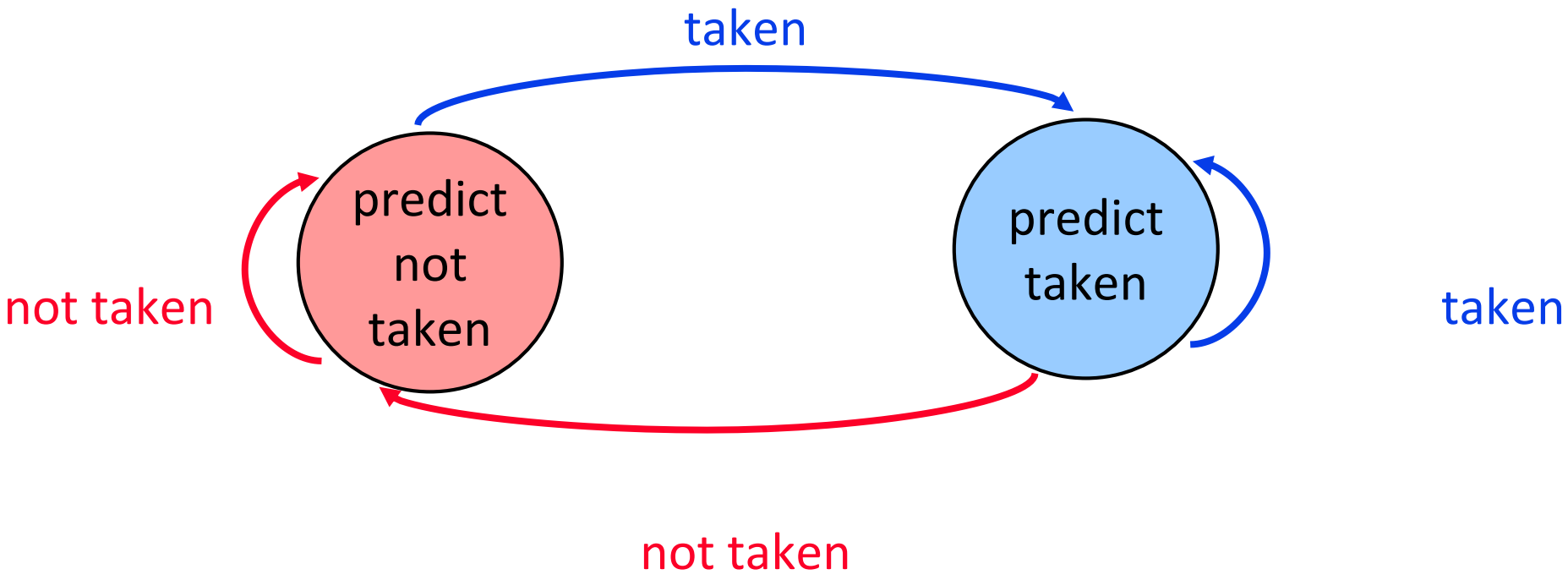
Δυναμική Πρόβλεψη Διακλάδωσης
Πρόβλεψη 1-bit

- Ο πιο απλός αλγόριθμος δυναμικής πρόβλεψης: προέβλεψε ότι η διακλάδωση θα συμπεριφερθεί όπως ακριβώς και στην αμέσως προηγούμενη εκτέλεση της.



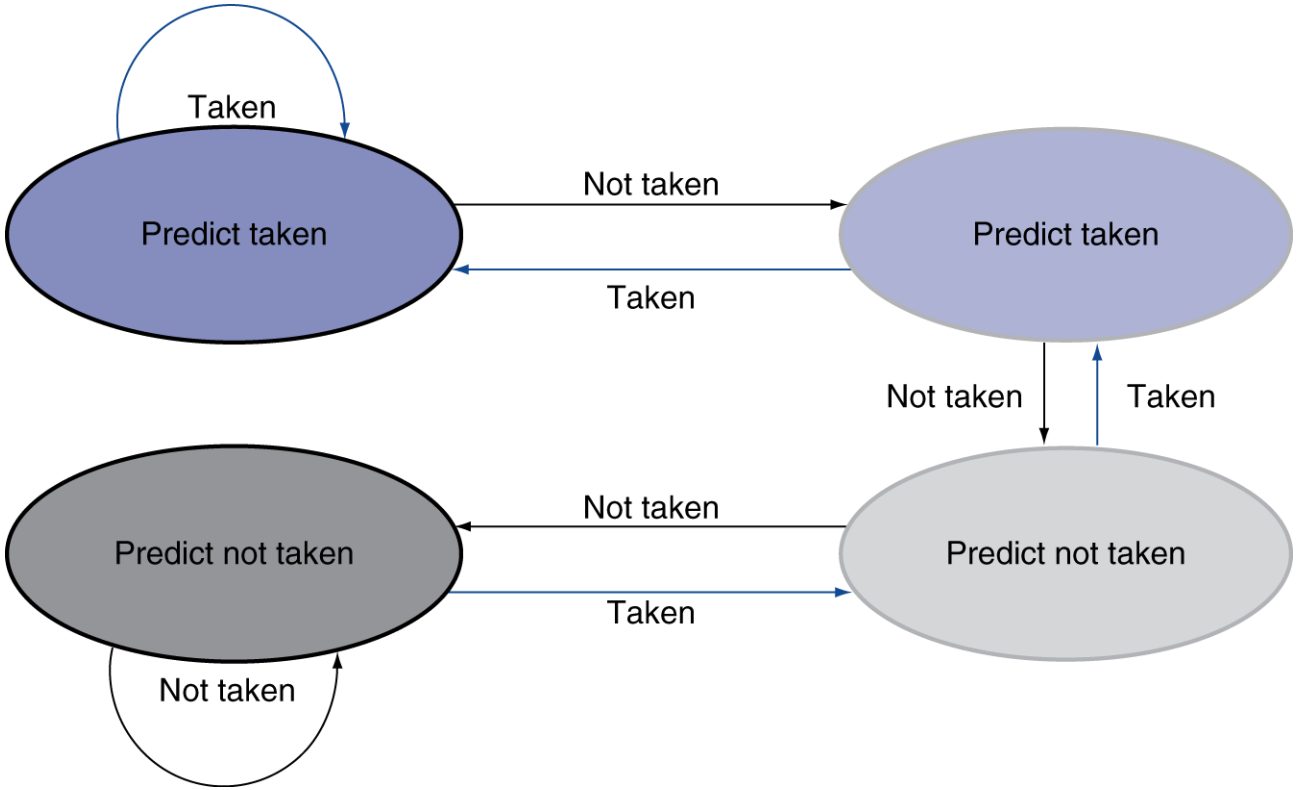
- Απλή αλλά ασταθής λύση.
- Στο παραπάνω κώδικα η εντολή **`beq`** του inner loop θα προβλεφθεί λάθος 2 φορές για κάθε εκτέλεση του εξωτερικού loop.

Πρόβλεψη 1-bit



Πρόβλεψη 2-bit

- Χρησιμοποιώντας μια μηχανή πεπερασμένων καταστάσεων μπορούμε να αποφύγουμε να αλλάζουμε απόφαση την πρώτη φορά που έχουμε λανθασμένη πρόβλεψη.
- Κάνουμε την πρόβλεψη πιο σταθερή



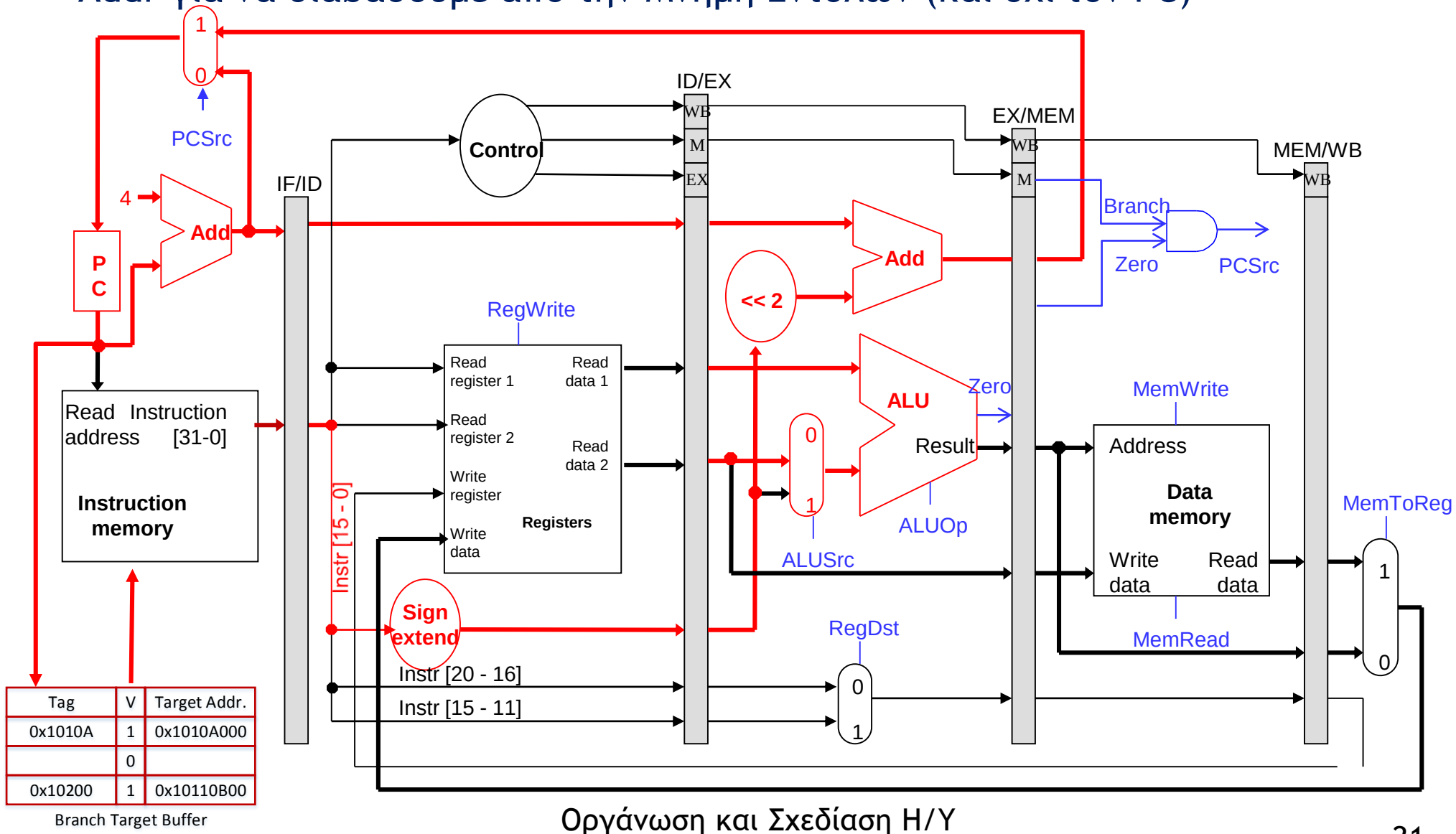
Υπολογισμός Διεύθυνσης Διακλάδωσης

- Εκτός από την πρόβλεψη για το εάν η διακλάδωση θα είναι TAKEN ή NOT TAKEN, πρέπει να υπολογίσουμε και την διεύθυνση προορισμού.
- Branch target buffer, BTB (προσωρινή μνήμη προορισμού διακλάδωσης)
  - Μνήμη cache (μνήμες caches στο επόμενο κεφ.) που αποθηκεύει μόνο εντολές διακλάδωσης , πληροφορίες σχετικά με την προηγούμενη συμπεριφορά της εντολής διακλάδωσης και την διεύθυνση προορισμού.
  - Κάθε φορά που είναι να φέρουμε μια καινούργια εντολή από την μνήμη, ελέγχουμε ταυτόχρονα και τον branch target buffer

    Εάν βρούμε την εντολή διακλάδωσης εκεί, και η διακλάδωση είναι TAKEN, τότε μπορούμε να φέρουμε την νέα εντολή από την διεύθυνση προορισμού που μας δείχνει η διακλάδωση

Branch Target Buffer (BTB)

Παράλληλη προσπέλαση της Μνήμης Εντολών και του BTB στο στάδιο IF.
Εάν βρούμε την διεύθυνση του PC μέσα στο BTB, τότε χρησιμοποιούμε την Target Addr για να διαβάσουμε από την Μνήμη Εντολών (και όχι τον PC)



Δυναμική Πρόβλεψη Διακλάδωσης

- **Κανόνας 1:** Το αποτέλεσμα μιας εντολής διακλάδωσης (Taken / Not Taken) εξαρτάται από το αποτέλεσμα προηγούμενων εκτελέσεων της εντολής αυτής
  - Υπάρχει συσχέτιση σε διαδοχικές εκτελέσεις της Β

- **Κανόνας 2:** Το αποτέλεσμα μιας εντολής διακλάδωσης Β (Taken / Not Taken) εξαρτάται και από το αποτέλεσμα και άλλων εντολών διακλάδωσης.
  - Συνήθως στην γειτονιά της Β

Δυναμική Πρόβλεψη Διακλάδωσης

Πρόσφατες εκτελέσεις μιας διακλάδωσης συσχετίζονται με μιά επόμενη εντολή διακλάδωσης

```
if (cond1)
...
if (cond1 AND cond2)
```

Εαν η πρώτη συνθήκη είναι False,
η δεύτερη θα είναι σίγουρα False

```
branch Y: if (cond1) a = 2;
...
branch X: if (a == 0)
```

Εαν η πρώτη συνθήκη είναι True,
η δεύτερη θα είναι σίγουρα False

Δυναμική Πρόβλεψη Διακλάδωσης

```
if (aa==2)                    ;; B1
      aa=0;
if (bb==2)                    ;; B2
      bb=0;
if (aa!=bb) {                 ;; B3
      ....
 }
```

if (B1 == True) && (B2 == True) then B3 = False

NT NT → T

Δυναμική Πρόβλεψη Διακλάδωσης

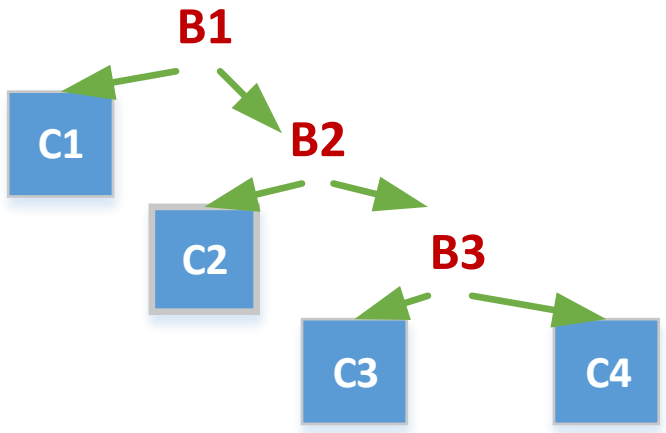
- Η ιδέα είναι να κάνουμε πρόβλεψη βασιζόμενοι στην εκτέλεση των αμέσως προηγούμενων εντολών διακλάδωσης.
  - Global Branch History Table (GBHT)

- Ακόμα καλύτερα:

Συνδιασμός του GBHT με την πρόβλεψη των 2 bits για τις προηγούμενες εκτελέσεις του ίδιου branch

Πρόβλεψη διακλάδωσης

```
if ((*ptr1==0) && ((*ptr2=*ptr3)==1) && (*ptr4>2))
    val5++;
else
    val6++;
```




```
            lw $t0, ptr1($0)
B1:         bnez $t0, L_val6
            lw $t1, ptr3($0)
            sw $t1, ptr2($0)
B2:         bne $t1, $t8, L_val6
            lw $t1, ptr4($0)
B3:         ble $t1, $t9, L_val6
            lw $s0, val5($0)
            addi $s0, $s0, 1
            sw $s0, val5($0)
            j Exit
L_val6:
            lw $s0, val6($0)
            addi $s0, $s0, 1
            sw $s0, val6($0)
Exit:
```

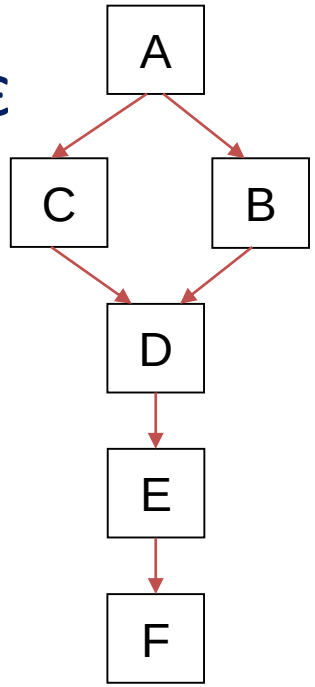
Πρόβλεψη διακλάδωσης

- Ο Pentium 4 (2003) έχει 20 στάδια διοχέτευσης
- Οι διακλαδώσεις εκτελούνται στο στάδιο 15
- Πολλαπλές εντολές branch (B1, B2, B3) ταυτόχρονα μέσα στην αρχιτεκτονική
- Ο BTB πρέπει να δώσει σωστές προβλέψεις για όλες
  - Αλλιώς, εκκένωση της CPU και μείωση της απόδοσης
- Speculative execution

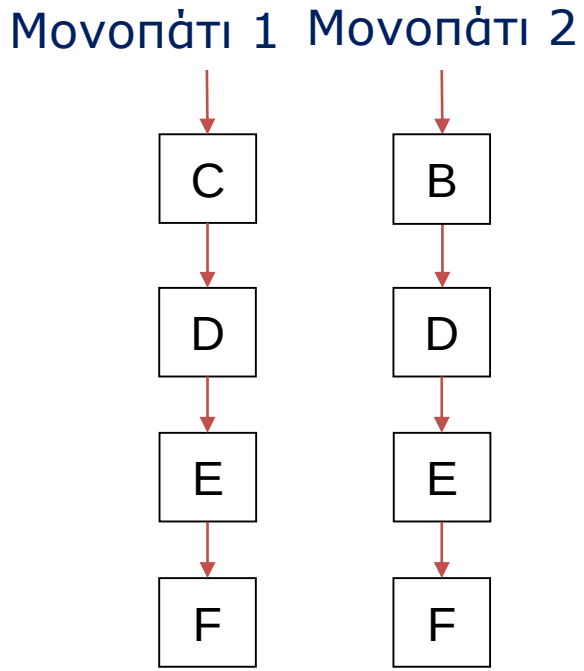
5η Μέθοδος μείωσης της επιβάρυνσης: ταυτόχρονη εκτέλεση πολλαπλών μονοπατιών

- Εάν δεν μπορούμε εύκολα να αποφασίσουμε την κατεύθυνση μιας διακλάδωσης, εκτελούμε και τις 2 κατευθύνσεις.
  - Χρήση σε επεξεργαστές υψηλής απόδοσης
  - Χρειαζόμαστε περισσότερο hardware
  - Μπορεί εύκολα να φύγει εκτός ελέγχου

Δύσκολη πρόβλεψη



Διπλό μονοπάτι



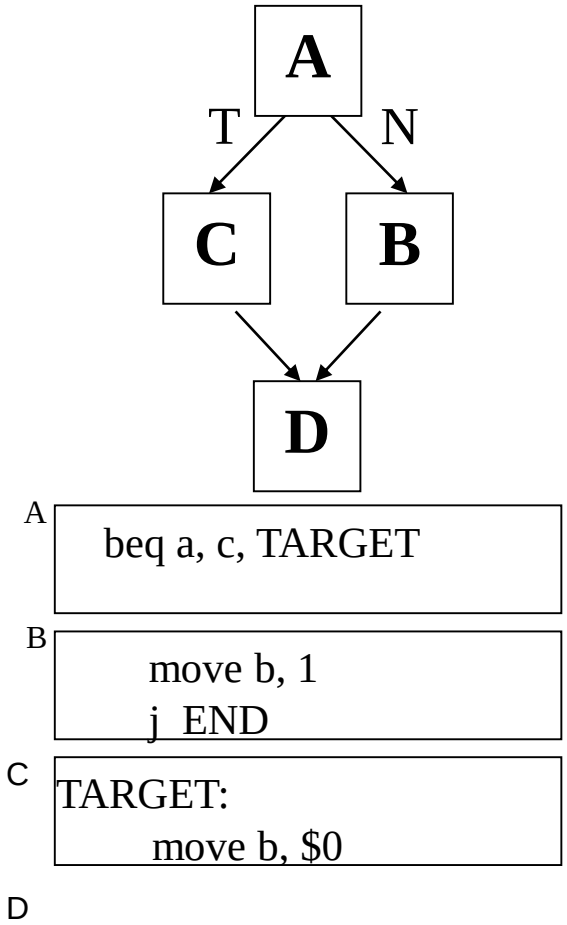
6η Μέθοδος μείωσης της επιβάρυνσης:
Εκτέλεση Υπό Συνθήκη: Predication

- Αφού μας ενοχλούν οι εντολές διακλάδωσης, ας τις ξεφορτωθούμε!
- Μετατροπή των κινδύνων ελέγχου (control hazards) σε κινδύνους δεδομένων (data hazards) ➔ Predication
- Πολλές σύγχρονες αρχιτεκτονικές το ακολουθούν
  - ARM, μερικώς και η x86

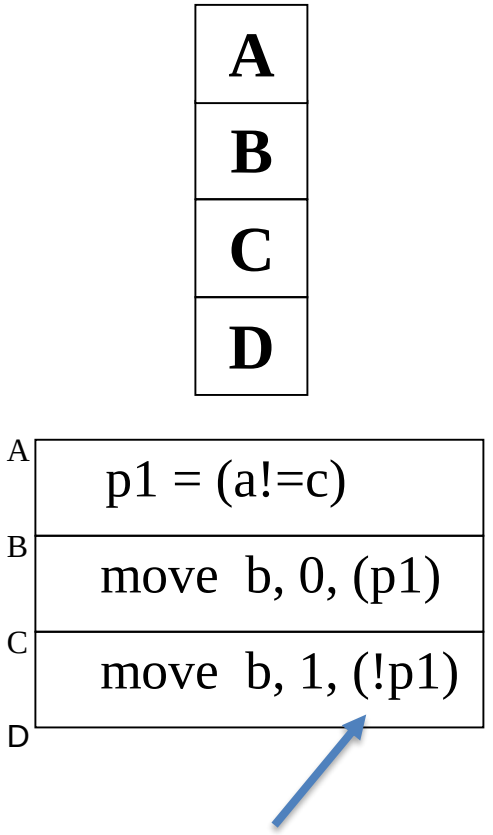
Παράδειγμα Predication

```
if (a!=c) {
    b = 0;
}
else {
   b = 1;
}
```

(branches)



(predicates)



Extra όρισμα εισόδου για κάθε εντολή.
Η εντολή εκτελείται μόνο εάν το όρισμα είναι TRUE
Εάν είναι FALSE, η εντολή γίνεται NOP

ARM processor is fully predicated

- Στον ARM, εάν βάλεις το κατάλληλο postfix, κάθε εντολή εκτελείται υπό συνθήκη (predicated)
  - add r0, r1, r2 ; Regular ADD: r0 = r1 + r2
  - addeq r0, r1, r2 ; r0 = r1 + r2 only if conditional flag ZERO is set
- Conditional move (CMOV), η πιό γνωστή predicated εντολή του x86.
- if (A==0) {S=T}

```
p1 = (R1 == 0)
CMOV R2, R3, <p1>
```

Predication

- Πλεονεκτήματα
  - Δεν έχουμε λανθασμένες προβλέψεις των διακλαδώσεων γιατί δεν έχουμε διακλαδώσεις!
  - Δεν υπάρχουν pipeline flushes
  - Ο compiler μπορεί να εφαρμόσει όλες τις βελτιστοποιήσεις του κώδικα που εμπόδιζαν οι εντολές διακλάδωσης
- Μειονεκτήματα
  - Κάνουμε περιττή δουλειά: φέρνουμε από την μνήμη και εκτελούμε (μέχρι ενός σημείου) και τα δύο μονοπάτια μιας διακλάδωσης
  - Πολλές εντολές διακλάδωσης είναι εύκολο να προβλεφθούν και δεν υποφέρουν από λάθος προβλέψεις
  - Χρειαζόμαστε υποστήριξη από το Instruction Set Architecture. Επιπλέον πολυπλοκότητα

Αποτίμηση της Πρόβλεψης Διακλάδωσης

- Οι εντολές διακλάδωσης είναι ο κυριότερος ένοχος μειωμένης απόδοσης του συστήματος (μετά τις εντολές προσπέλασης μνήμης) .
  - Σε επεξεργαστές υψηλής απόδοσης (high performance processors), η λανθασμένη πρόβλεψη του branch επιφέρει πολλούς κύκλους ποινής.
  - Πολλές εντολές θα πρέπει να γίνουν flush σε μια τέτοια περίπτωση.
- Όλοι οι μοντέρνοι επεξεργαστές χρησιμοποιούν δυναμική πρόβλεψη διακλαδώσεων.
  - Ακριβείς προβλέψεις (>95%) της κατεύθυνσης διακλάδωσης είναι πάρα πολύ σημαντικές για καλή απόδοση ενός επεξεργαστικού συστήματος υψηλής απόδοσης.
  - Για αυτό και όλες αυτές οι CPUs χρησιμοποιούν πολύπλοκα συστήματα πρόβλεψης διακλάδωσης που “μαθαίνουν” την συμπεριφορά της εντολής κατά την διάρκεια εκτέλεσης του προγράμματος.
  - Πολύ σημαντικό ερευνητικό θέμα τα τελευταία 20 χρόνια.