ΗΜΥ 314 – Εργαστήριο Αρχιτεκτονικής Υπολογιστών

Εργαστήρια 3 & 4

Pipelined MIPS

Μικροεπεξεργαστής MIPS: Υλοποίηση με Διασωλήνωση/Διοχέτευση

Μαρία Κ. Μιχαήλ
Αναπληρώτρια Καθηγήτρια ΗΜΜΥ

Εργαστηριακές Ασκήσεις 3 και 4

- Η περιγραφή των εργαστηριακών ασκήσεων γίνεται στο ακόλουθο PDF αρχείο, που είναι αναρτημένο στο Blackboard:

ECE314.Fall.2025.Lab.Exercises.3.and.4.pdf

- Αφού μελετήσετε **ενδελεχώς** το πιο πάνω αρχείο, προχωρήστε στα επόμενα slides…

Εργαστηριακές Ασκήσεις 3 και 4

Supported

MIPS32

Instructions

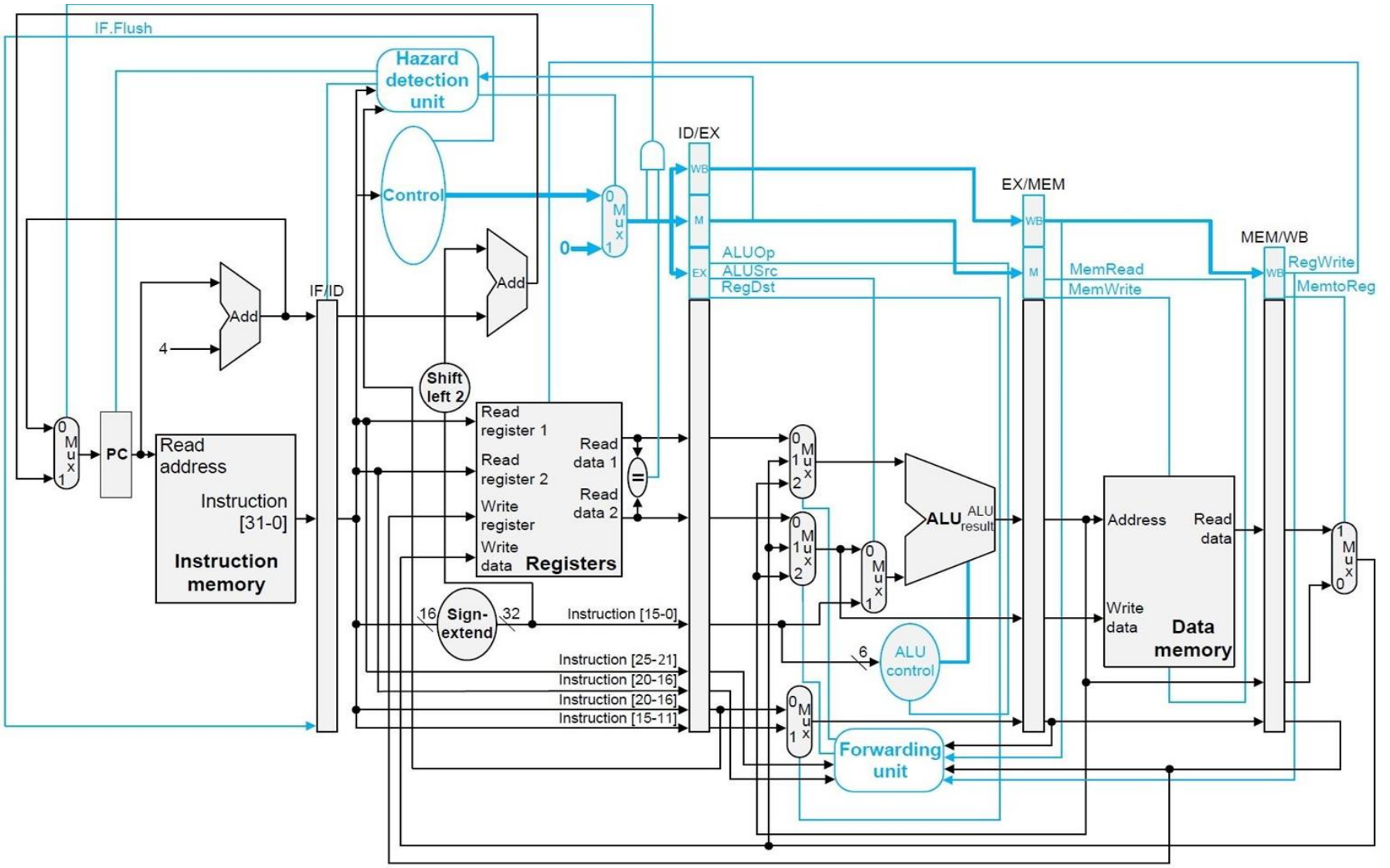
25 supported instructions

Singe-cycle implementation does **NOT** support:
jal jr lui

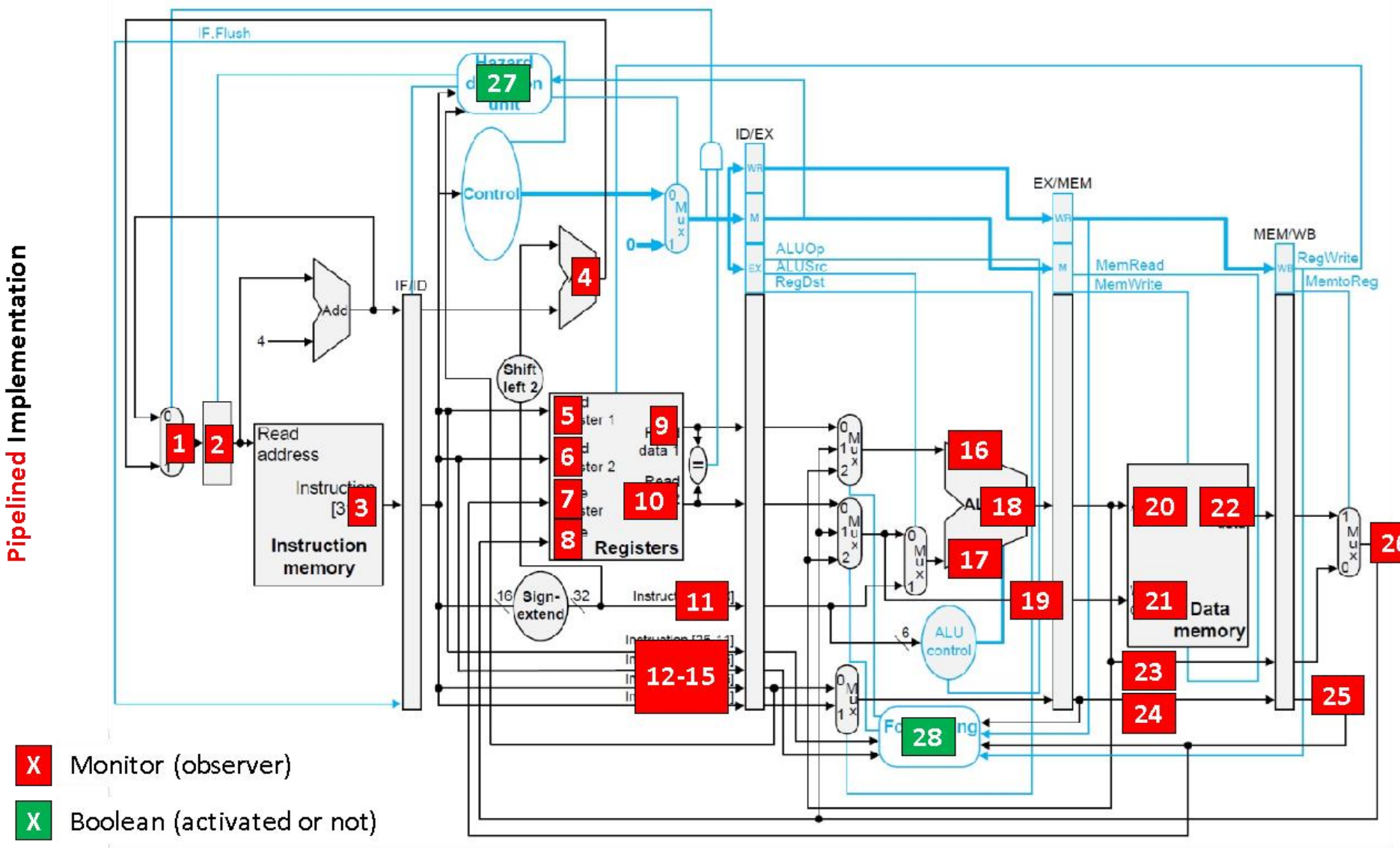
Pipelined implementation does **NOT** support:
j jal jr lui

Add	add
Add Immediate	addi
Add Imm. Unsigned	addiu
Add Unsigned	addu
And	and
And Immediate	andi
Branch On Equal	beq
Branch On Not Equal	bne
Jump	j
Jump And Link	jal
Jump Register	jr
Load Upper Imm.	lui
Load Word	lw
Nor	nor
Or	or
Or Immediate	ori
Set Less Than	slt
Set Less Than Imm.	slti
Set Less Than Imm. Unsigned	sltiu
Set Less Than Unsig.	sltu
Shift Left Logical	sll
Shift Right Logical	srl
Store Word	sw
Subtract	sub
Subtract Unsigned	subu

Εργαστηριακές Ασκήσεις 3 και 4



Εργαστηριακές Ασκήσεις 3 και 4



Επανάληψη σχετικής ύλης από ΗΜΥ 212

Σχεδιασμός επεξεργαστή MIPS με υλοποίηση **Διασωλήνωσης**

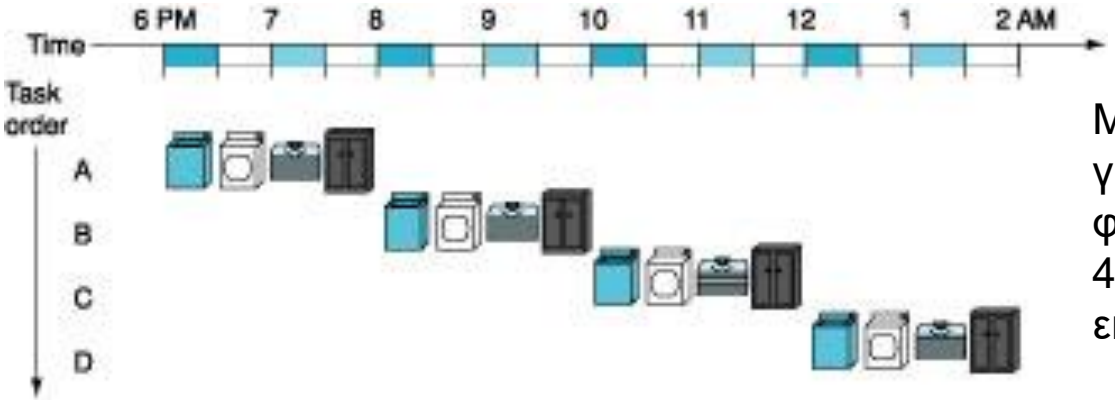
Επανάληψη σχετικής ύλης από ΗΜΥ 212

Ύλη που αφορά την Εργαστηριακή **Άσκηση 3**

Υλοποίηση **Διασωλήνωσης *χωρίς*** Μονάδες Ανίχνευσης Κινδύνων και Προώθησης

Διασωλήνωση [ή Διοχέτευση] (*Pipelining*)

Διασωληνώση: επικαλυπτόμενη εκτέλεση πολλαπλών εντολών σε έναν κύκλο του ρολογιού, όπως σε μια γραμμή συναρμολόγησης



Μέθοδος χωρίς διασωλήνωση για πλύσιμο ρούχων. Το κάθε φορτίο απαιτεί 2 ώρες, άρα για 4 φορτία ο συνολικός χρόνος εκτέλεσης είναι 4 x 2 = 8 ώρες

Διασωλήνωση 4ων σταδίων

Στάδιο διασωλήνωσης = 1 βήμα της εργασίας

Μέθοδος με διασωλήνωση. Το κάθε φορτίο εξακολουθεί να απαιτεί 2 ώρες, αλλά ο συνολικός χρόνος εκτέλεσης μειώνεται σε:
Συνολικός χρόνος = χρόνος για 1 φορτίο + (Αρ. φορτίων – 1) x χρόνο σταδίου =
2 + 3x0.5 = 3.5 ώρες

Διασωλήνωση (συν.)

- Μια εργασία (εντολή) χωρίζεται σε βήματα, τα οποία ονομάζονται **στάδια της διασωλήνωσης**
- Το κάθε στάδιο απαιτεί τον ίδιο χρόνο εκτέλεσης
- Τα διάφορα στάδια μπορούν να διασωληνωθούν εάν τα στάδια που εκτελούνται ταυτόχρονα δεν χρειάζονται τις ίδιες λειτουργικές μονάδες του μικροεπεξεργαστή
- **Η χρήση διασωλήνωσης βελτιώνει την ικανότητα διεκπεραίωσης (*throughput*), όχι τον χρόνο εκτέλεσης για μια εργασία**
- Όταν εκτελείται ένας μεγάλος αριθμός εργασιών, δημιουργείται **επιτάχυνση** (***speedup***) λόγω της διασωλήνωσης (η επιτάχυνση δεν μπορεί να είναι μεγαλύτερη από τα στάδια της διασωλήνωσης)
- Για το προηγούμενο παράδειγμα,
  speedup = (4x2) / (2+3x0.5) = 2.29
  Θεωρήστε ένα φορτίο με 100 εργασίες,
  speedup = (100x2) / (2+99x0.5) = 3.88

Διασωλήνωση για τον MIPS

- Ισχύουν οι ίδιοι γενικοί κανόνες. Υπάρχουν 5 βασικά βήματα:

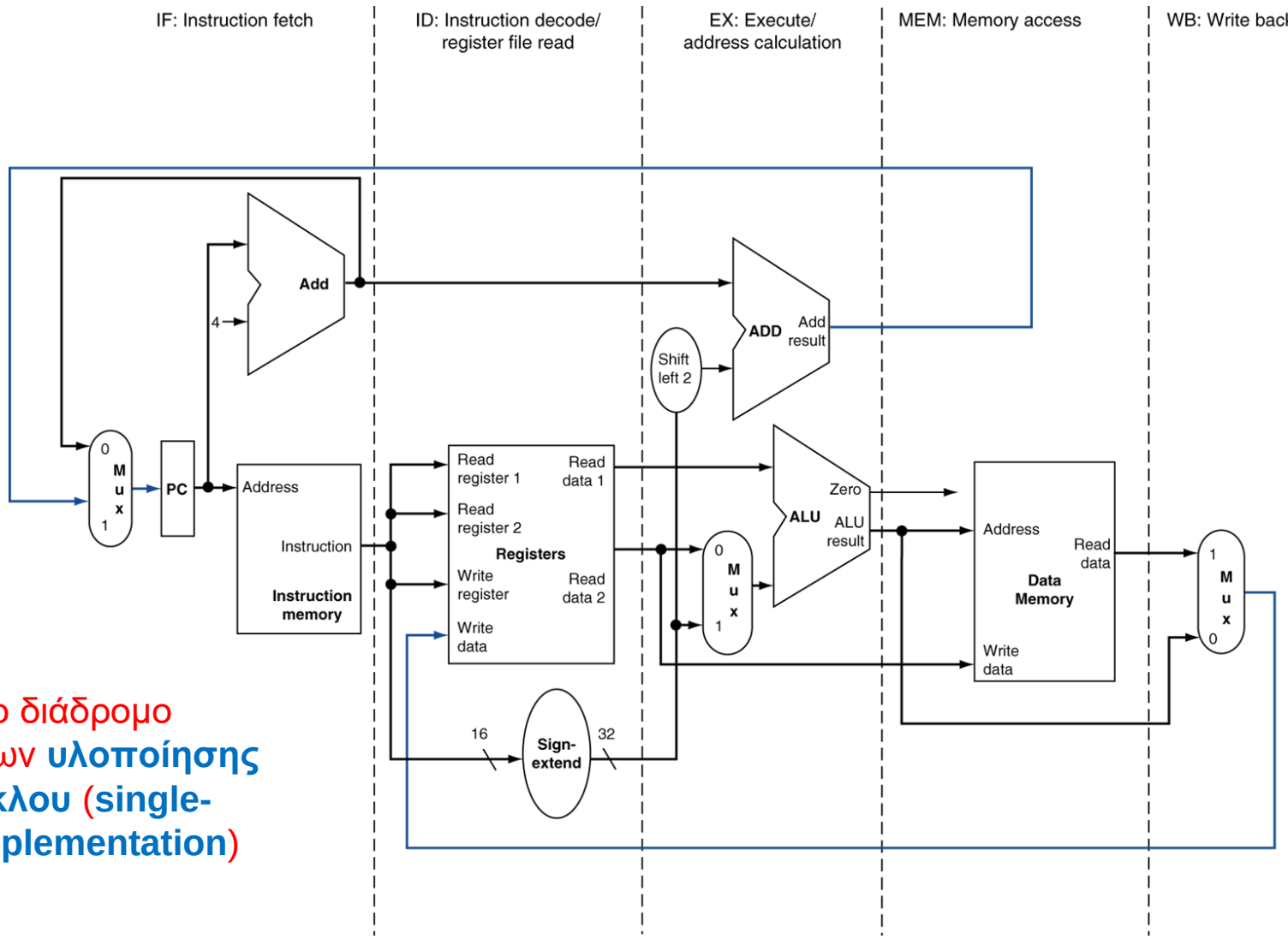
  1. Προσκόμιση Εντολής από Μνήμη (*Instruction Fetch*)
  2. Ανάγνωση Καταχωρητών και Αποκωδικοποίηση Εντολής (*Reg*)
  3. Εκτέλεση Αριθμητικής Εντολής ή Υπολογισμός Διεύθυνσης (*ALU*)
  4. Προσπέλαση Μνήμης για Δεδομένα (*Data access*)
  5. Ενημέρωση Καταχωρητή Γραφής (*Reg*)

- Επομένως, **η διασωλήνωση έχει 5 στάδια**
- Δεδομένου ενός μεγάλου φορτίου εργασίας (*workload*) ο επεξεργαστής με διασωλήνωση θα είναι περίπου 5 φορές γρηγορότερος από αυτόν χωρίς διασωλήνωση

Κίνδυνοι Διασωλήνωσης

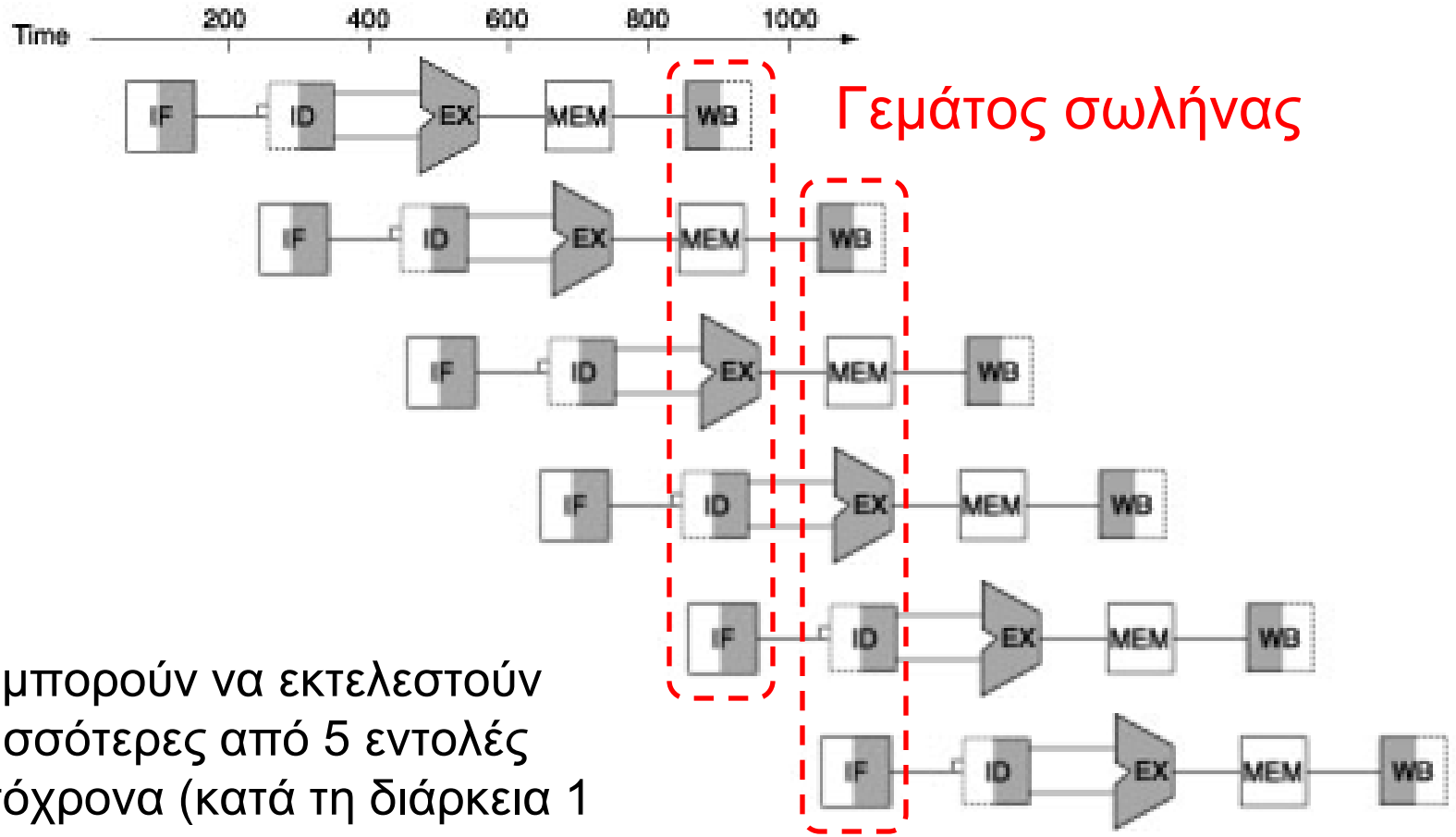
- Υπάρχουν 3 κατηγορίες κινδύνων:
  - **Δομικοί (*Structural*):** μια εντολή δεν μπορεί να εκτελεστεί στον προγραμματιζόμενο κύκλο διότι ο επεξεργαστής δεν υποστηρίζει το συνδυασμό των εντολών που πρέπει να εκτελεστούν στον **ίδιο κύκλο** (Τι θα συμβεί αν ο MIPS ενός κύκλου έχει μόνο 1 μονάδα μνήμης αντί για 2;)**.**
    Εύκολη η λύση για τον MIPS… [Two memories, see later]
  - **Δεδομένων (*Data*):** μια εντολή δεν μπορεί να εκτελεστεί στον προγραμματιζόμενο κύκλο διότι τα απαιτούμενα δεδομένα δεν είναι ακόμη διαθέσιμα
  - **Ελέγχου (*Control*):** μια εντολή δεν μπορεί να εκτελεστεί στον προγραμματιζόμενο κύκλο διότι η εντολή που προσκομίστηκε και μπήκε στο σωλήνα δεν είναι αυτή που θα έπρεπε να είχε προσκομιστεί (δηλ., η ροή των εντολών δεν είναι αυτή που αναμένεται)

Βασική Ιδέα Υλοποίησης Διασωλήνωσης



Πίσω στο διάδρομο δεδομένων **υλοποίησης ενός κύκλου (single-cycle implementation)**

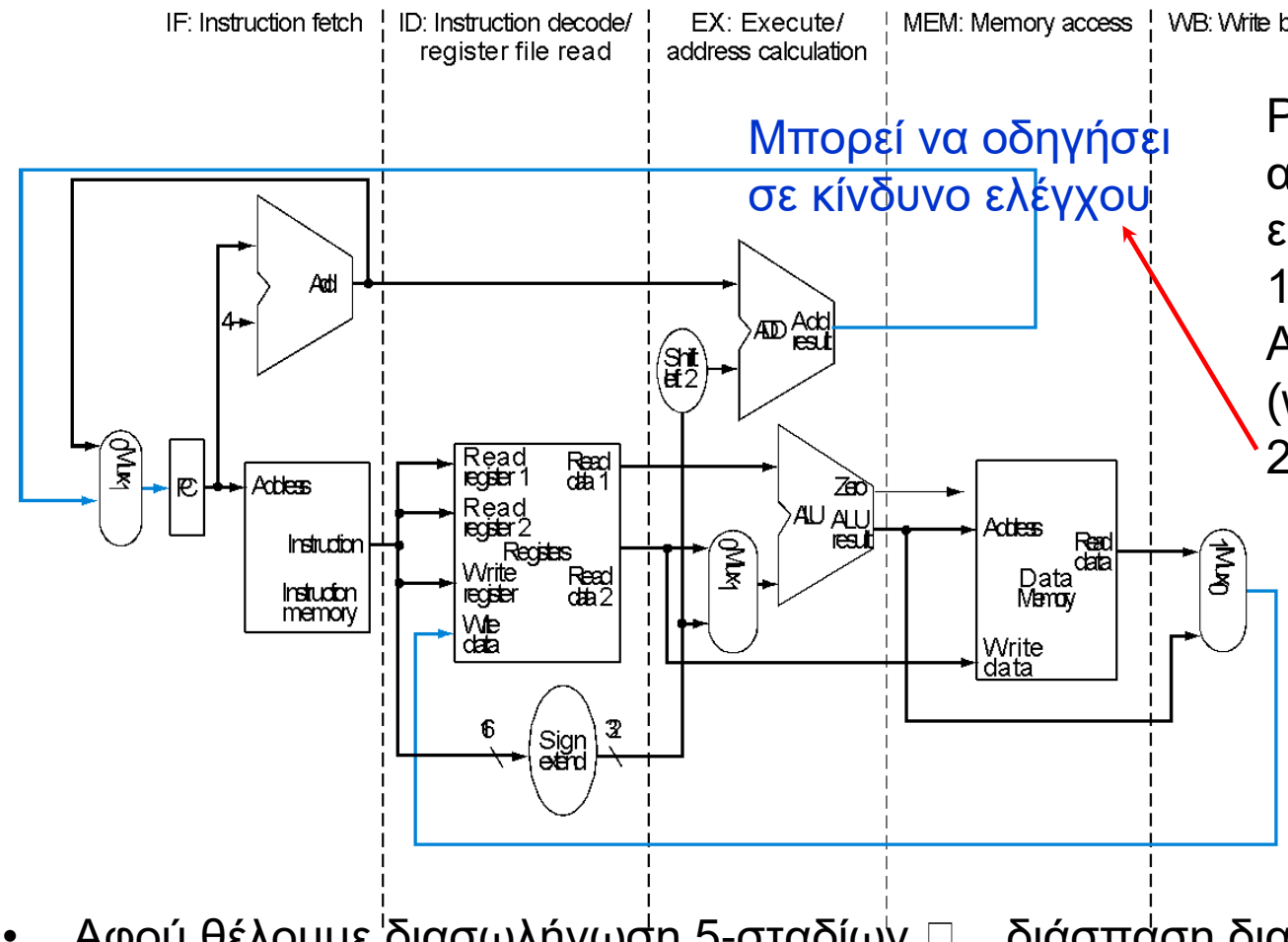
Πόσες εντολές μπορούν να εκτελούνται ταυτόχρονα;



Δεν μπορούν να εκτελεστούν περισσότερες από 5 εντολές ταυτόχρονα (κατά τη διάρκεια 1 κύκλου) με διασωλήνωση 5 σταδίων

Σκίαση δεξιά □
Ανάγνωση
Σκίαση αριστερά □
Γραφή

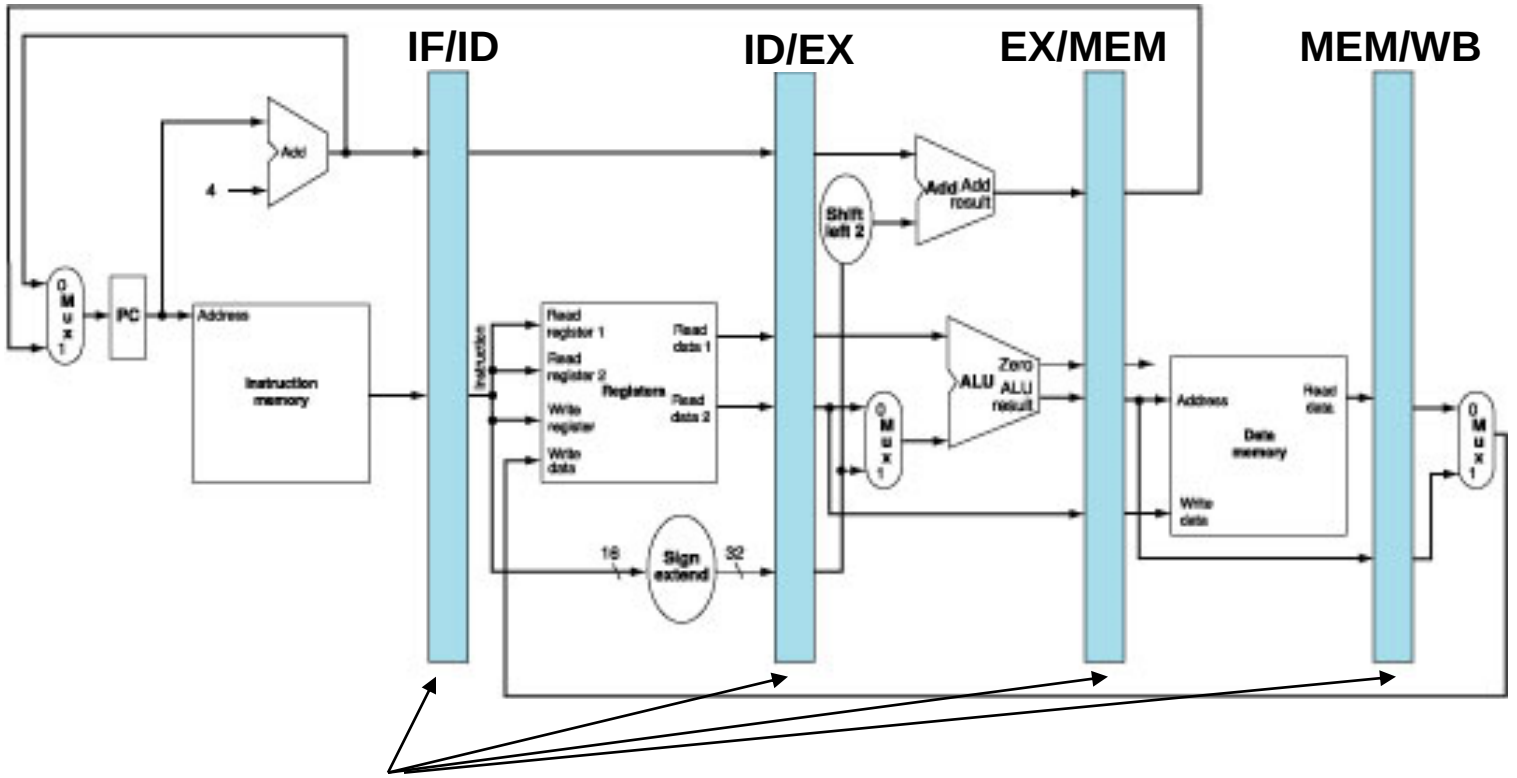
Βασική Ιδέα Υλοποίησης Διασωλήνωσης (συν.)



- Αφού θέλουμε διασωλήνωση 5-σταδίων διάσπαση διαδρόμου δεδομένων σε 5 στάδια/μέρη
- ***Τι πρέπει να προσθέσουμε για να διασπάσουμε το διάδρομο δεδομένων σε στάδια;***

Διαχωρισμός του Διαδρόμου Δεδομένων σε Στάδια

- Εισαγωγή καινούριων καταχωρητών (δεν είναι προσβάσιμοι από τον προγραμματιστή) για να κρατούν τα δεδομένα, έτσι ώστε τα κομμάτια του διαδρόμου δεδομένων να μπορούν να διαχωριστούν κατά τη διάρκεια της εκτέλεσης των εντολών



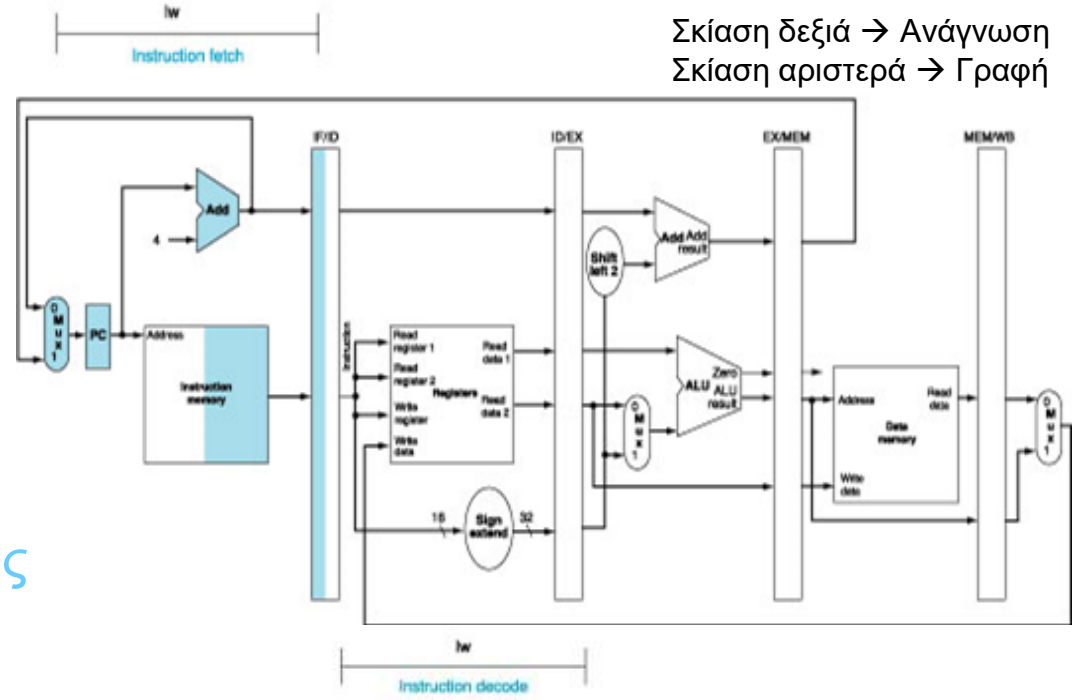
Καινούριοι καταχωρητές

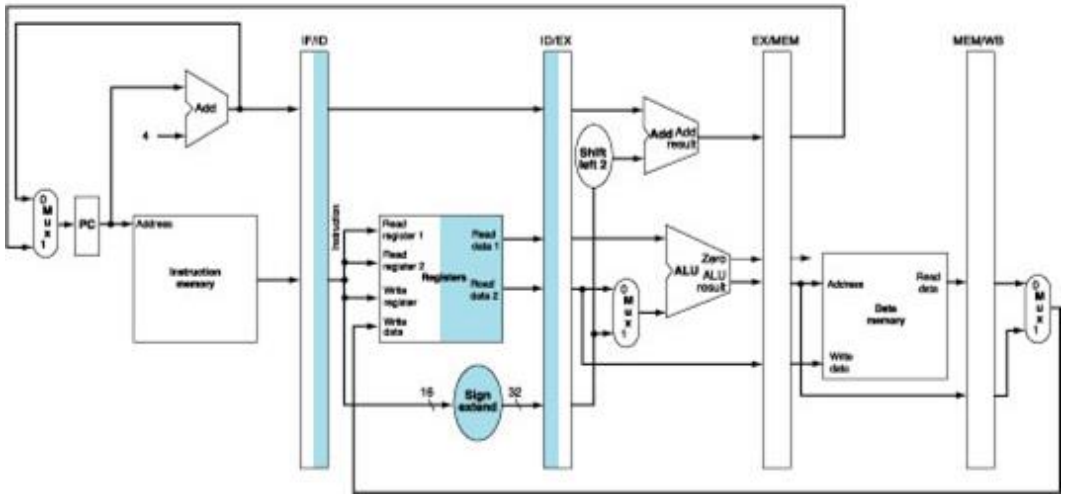
Παράδειγμα εντολής Load (`lw`)

1. Προσκόμιση Εντολής:
   Ανάγνωση εντολής από μνήμη με τη διεύθυνση του PC και αποθήκευσή της στον καταχωρητή IF/ID. Αύξηση του PC κατά 4 και αποθήκευση στον PC και στον IF/ID (για `beq`)

2. Αποκωδικοποίηση Εντολής και ανάγνωση αρχείου καταχωρητών (ΑΚ):
   Ανάγνωση του ΑΚ με τα κατάλληλα πεδία εντολής από τον IF/ID για Read register 1/2 και αποθήκευση του στον ID/EX. Sign-extend και αποθήκευση του offset στον ID/EX, μαζί με το PC+4

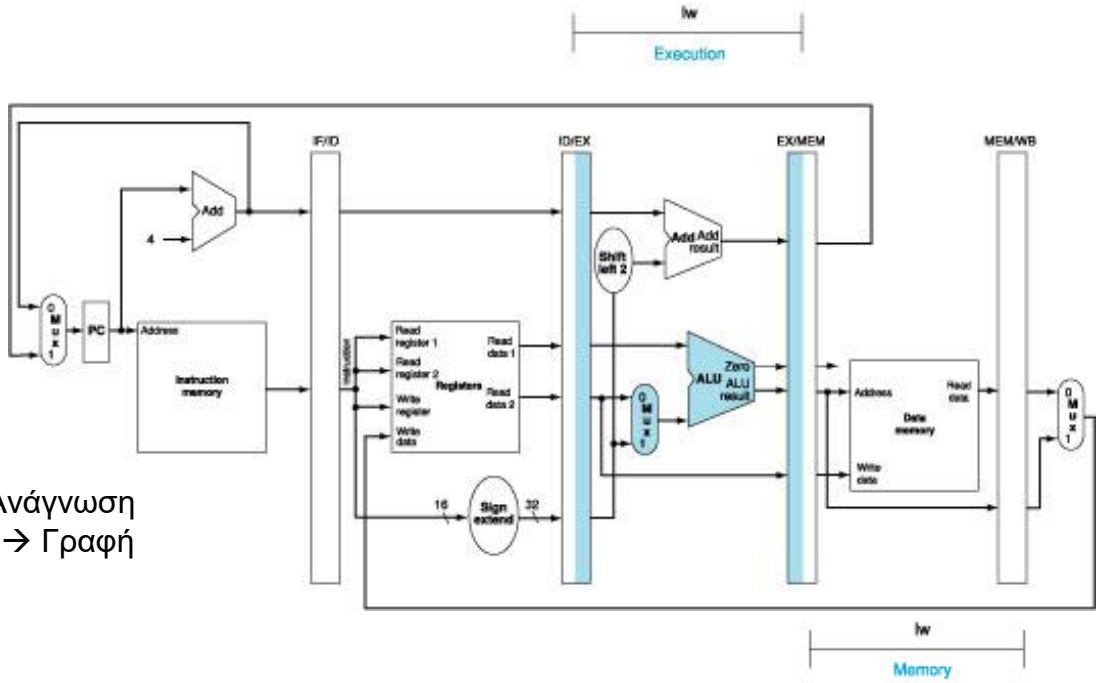
Σκίαση δεξιά → Ανάγνωση
Σκίαση αριστερά → Γραφή





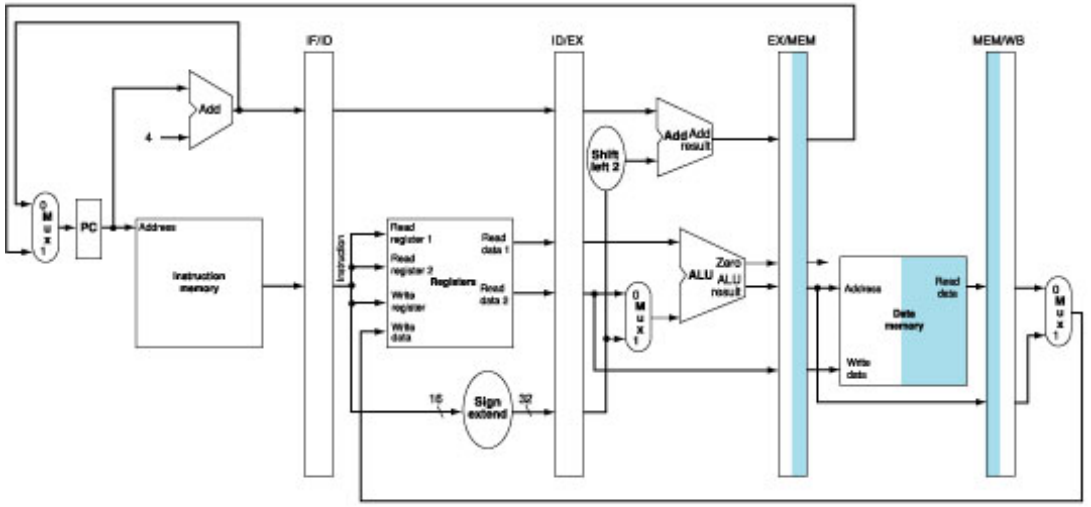
Παράδειγμα εντολής Load (`lw`) (συν.)

3. Υπολογισμός διεύθυνσης:
   Η ALU παίρνει τις απαραίτητες τιμές από τον ID/EX (Read register 1 και 32-bit sign-extended offset) και αποθηκεύει το αποτέλεσμα στον EX/MEM



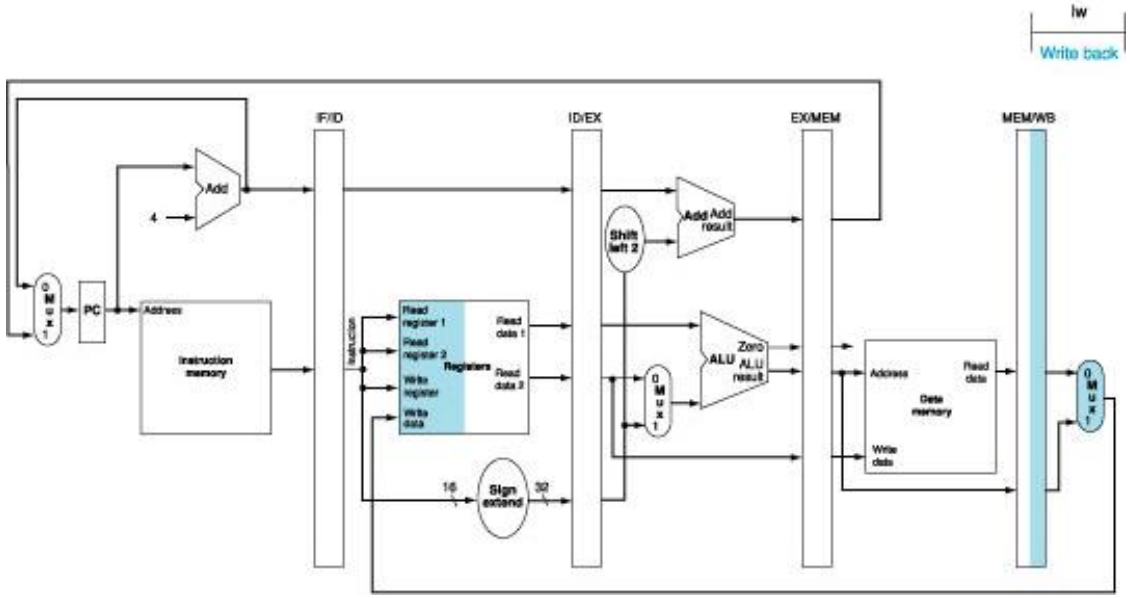
Σκίαση δεξιά → Ανάγνωση
Σκίαση αριστερά → Γραφή

4. Προσπέλαση Μνήμης:
   Ανάγνωση από την μνήμη με την διεύθυνση που είναι το αποτέλεσμα που υπολογίστηκε από την ALU στο προηγούμενο στάδιο, και αποθήκευση των δεδομένων στον MEM/WB



Παράδειγμα εντολής Load (`lw`) (συν.)

5. Γραφή στο ΑΚ:
   Τα δεδομένα από την μνήμη που αποθηκεύτηκαν στον MEM/WB στο προηγούμενο στάδιο γράφονται στο ΑΚ



Σκίαση δεξιά→ Ανάγνωση
Σκίαση αριστερά → Γραφή

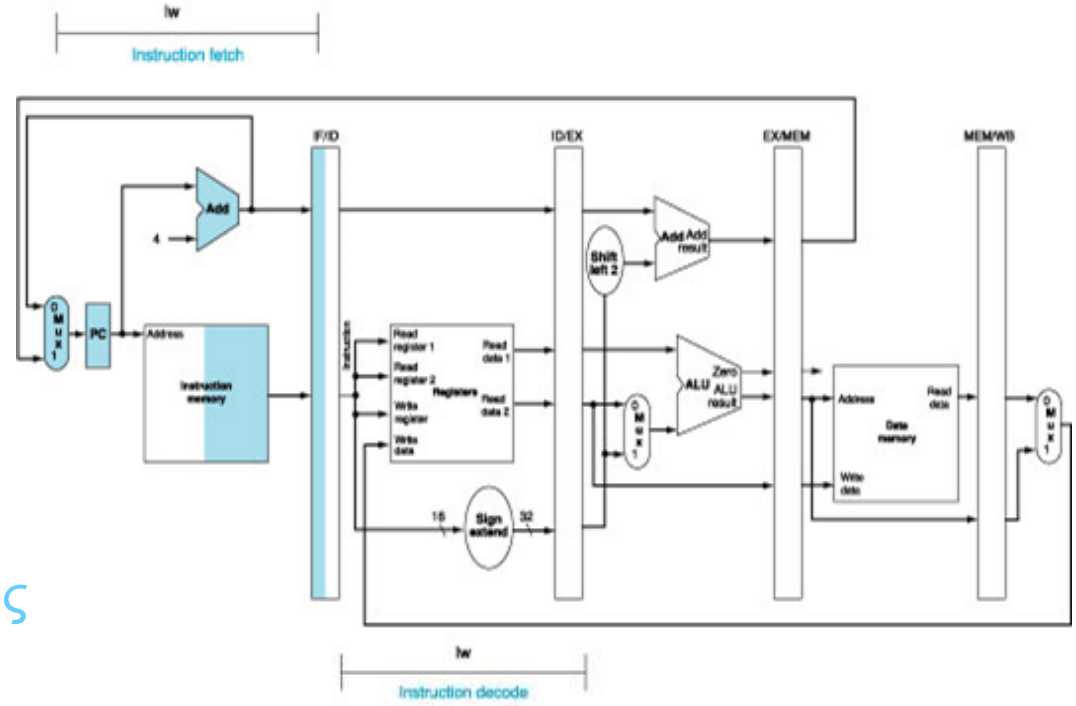
1η Βασική Ιδέα

Κάθε φορά που προχωρεί μια εντολή από ένα στάδιο του σωλήνα στο επόμενο, αποθηκεύεται στον αντίστοιχο εσωτερικό καταχωρητή *οποιαδήποτε* πληροφορία μπορεί να χρειαστεί από *οποιοδήποτε* μεταγενέστερο στάδιο

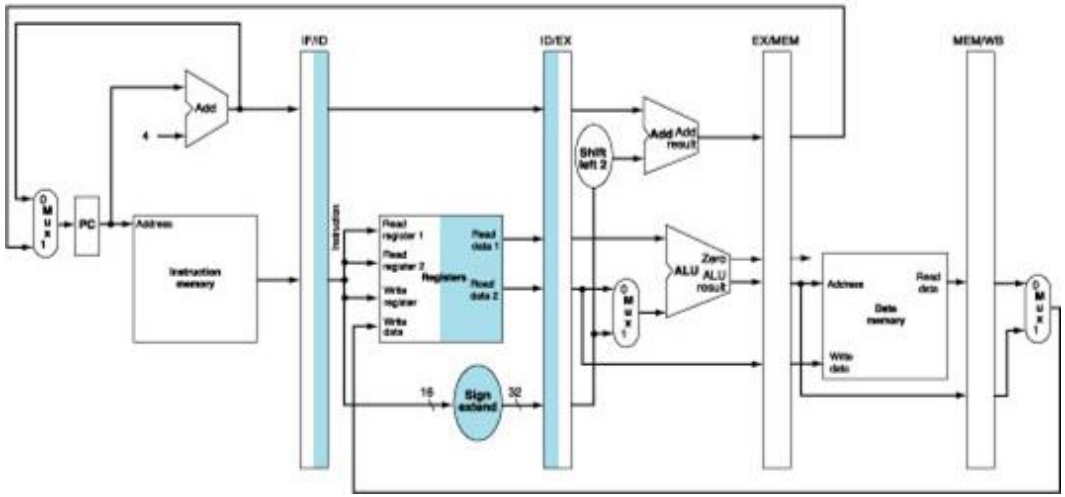
Παράδειγμα εντολής Store (`sw`)

Στάδια 1, 2 → ίδια με `lw`

1. Προσκόμιση Εντολής:
   Ανάγνωση εντολής από μνήμη με τη διεύθυνση του PC και αποθήκευσή της στον καταχωρητή IF/ID. Αύξηση του PC κατά 4 και αποθήκευση στον PC και στον IF/ID (για `beq`)



2. Αποκωδικοποίηση Εντολής και ανάγνωση αρχείου καταχωρητών (ΑΚ):
   Ανάγνωση του ΑΚ με τα κατάλληλα πεδία εντολής από τον IF/ID για Read register 1/2 και αποθήκευση του στον ID/EX. Sign-extend και αποθήκευση του offset στον ID/EX, μαζί με το PC+4

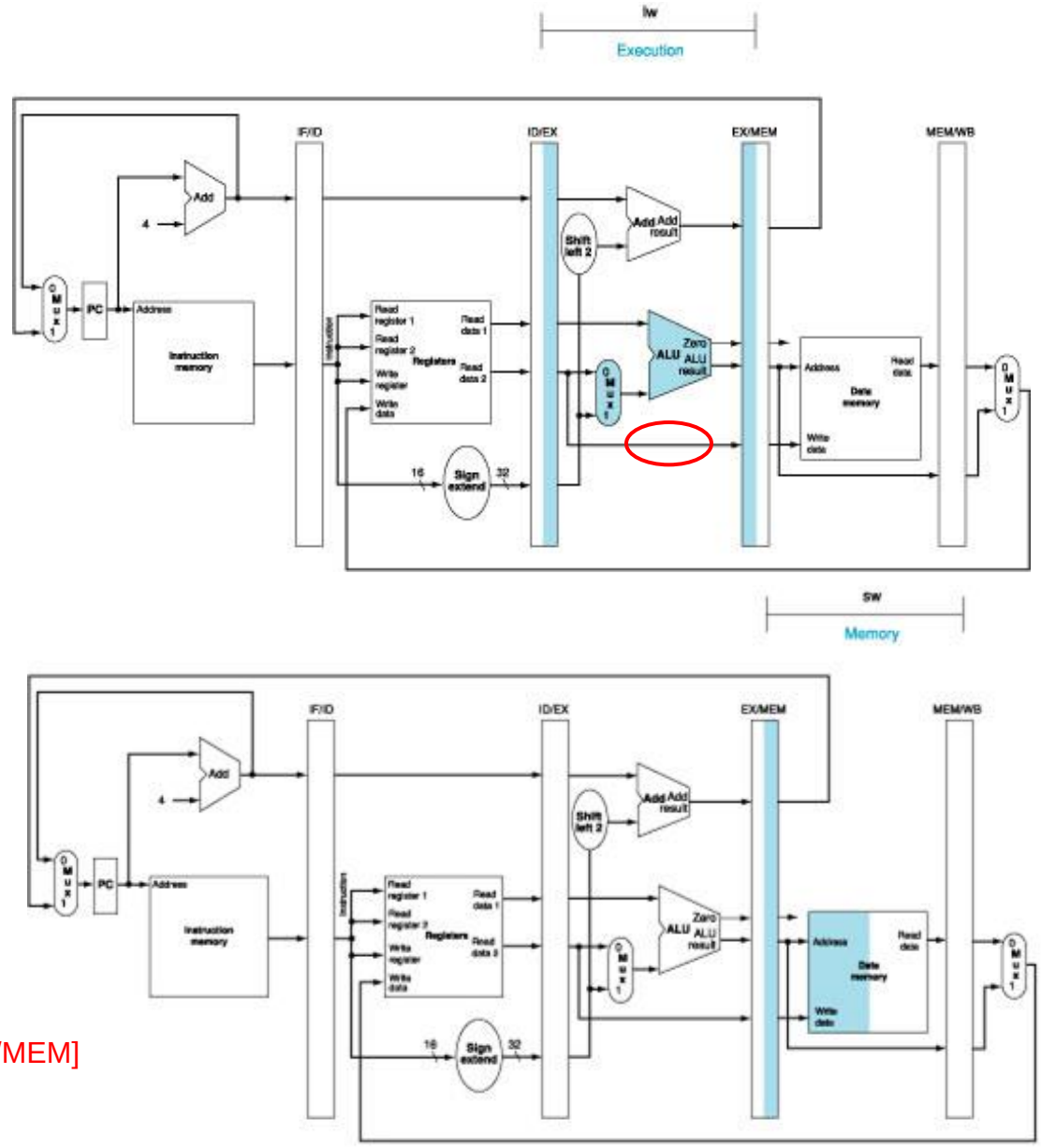


Παράδειγμα εντολής Store (`sw`) (συν.)

3. Υπολογισμός διεύθυνσης:
   Η ALU παίρνει τις απαραίτητες τιμές από τον ID/EX (Read register 1 και 32-bit sign-extended offset) και αποθηκεύει το αποτέλεσμα στον EX/MEM.
   Η τιμή του Read register 2 μεταφέρεται από τον ID/EX στον EX/MEM

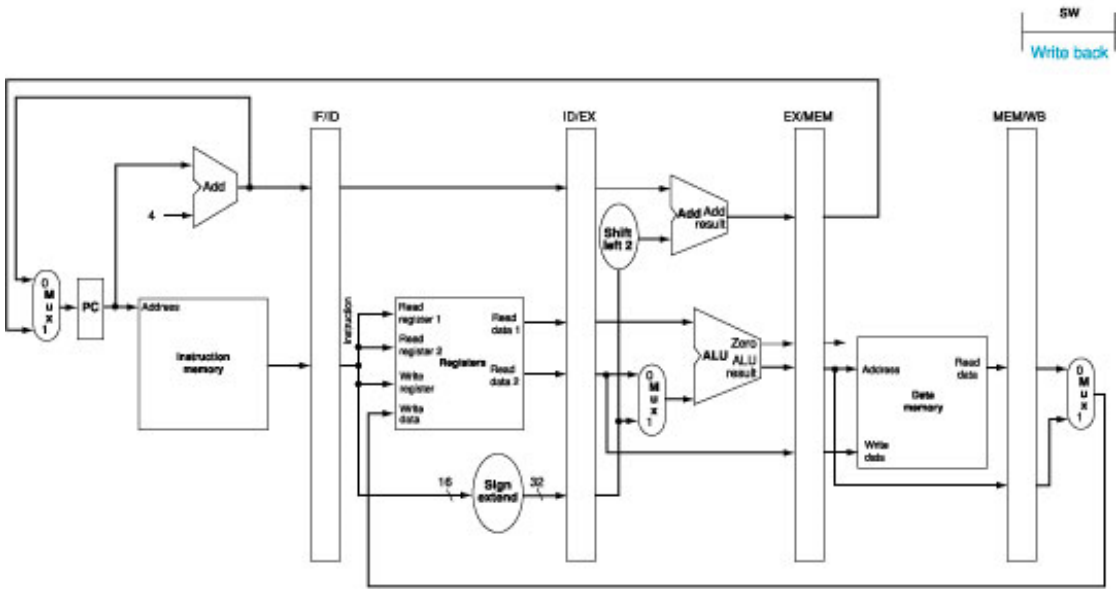
4. Προσπέλαση Μνήμης:
   Πρόσβαση στη μνήμη με την διεύθυνση που είναι το αποτέλεσμα που υπολογίστηκε από την ALU στο προηγούμενο στάδιο και αποθήκευση δεδομένων

   Που θα βρούμε τα δεδομένα που θα αποθηκευτούν στη μνήμη; [EX/MEM]



Παράδειγμα εντολής Store (`sw`) (συν.)

5. Γραφή στο ΑΚ:
   Καμία απαραίτητη ενέργεια (παρόλα αυτά η εκτέλεση του είναι απαραίτητη!)

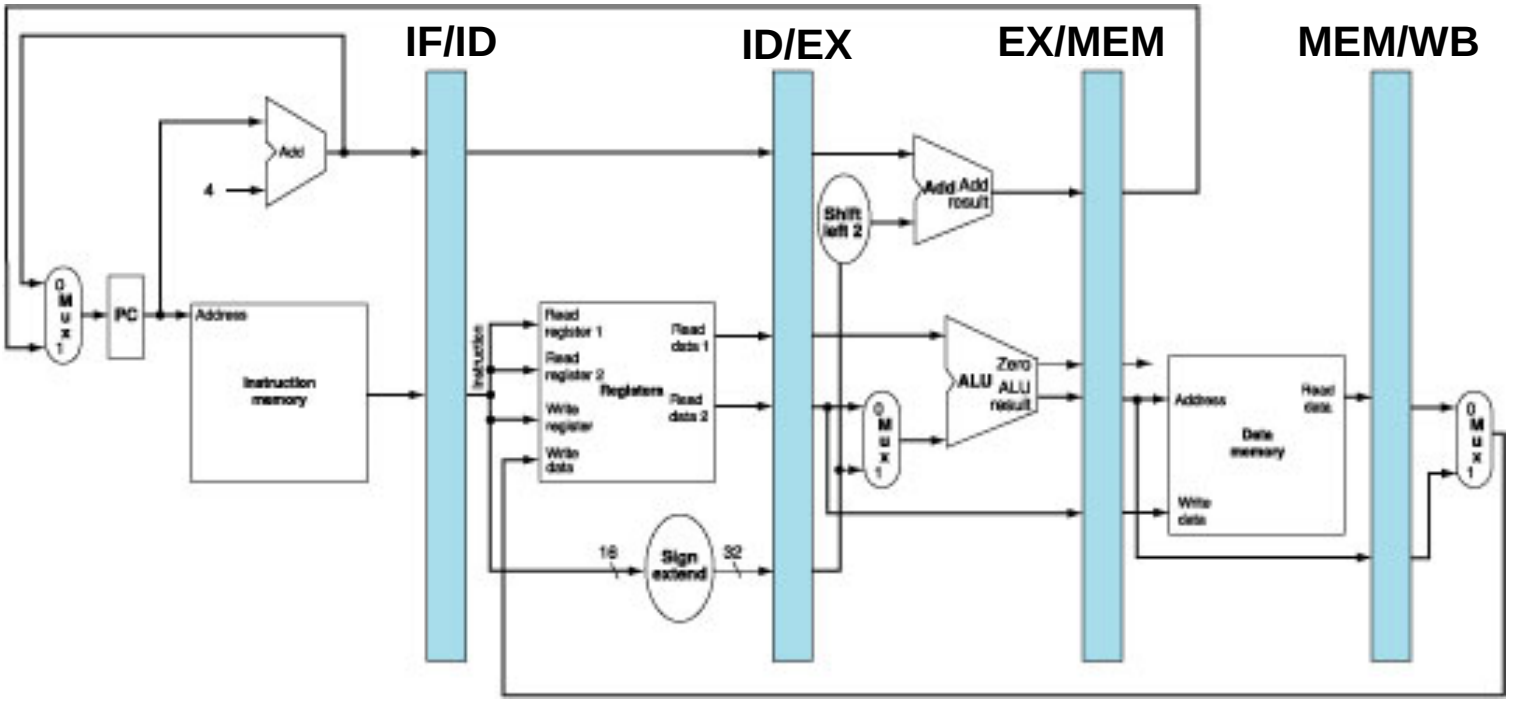


2η Βασική Ιδέα

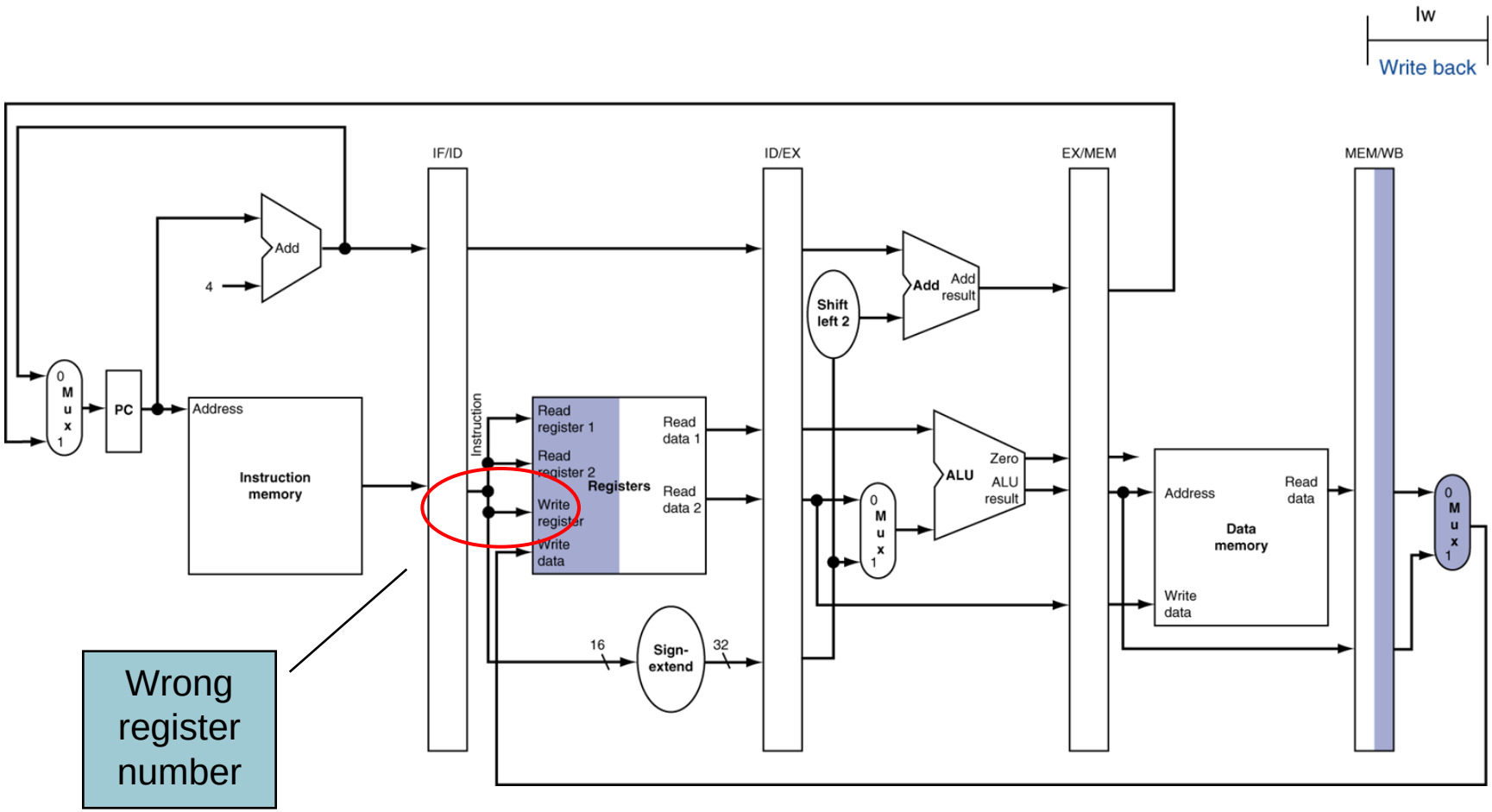
Κάθε λογικό στοιχείο (λειτουργική μονάδα) στον διάδρομο δεδομένων (Μνήμες Εντολών και Δεδομένων, ALU, θήρες ανάγνωσης ΑΚ και θήρες γραφής ΑΚ) μπορούν να χρησιμοποιηθούν *μόνο σε ένα στάδιο της διασωλήνωσης*. Διαφορετικά, μπορεί να δημιουργηθούν *δομικοί κίνδυνοι*.

Πρόβλημα

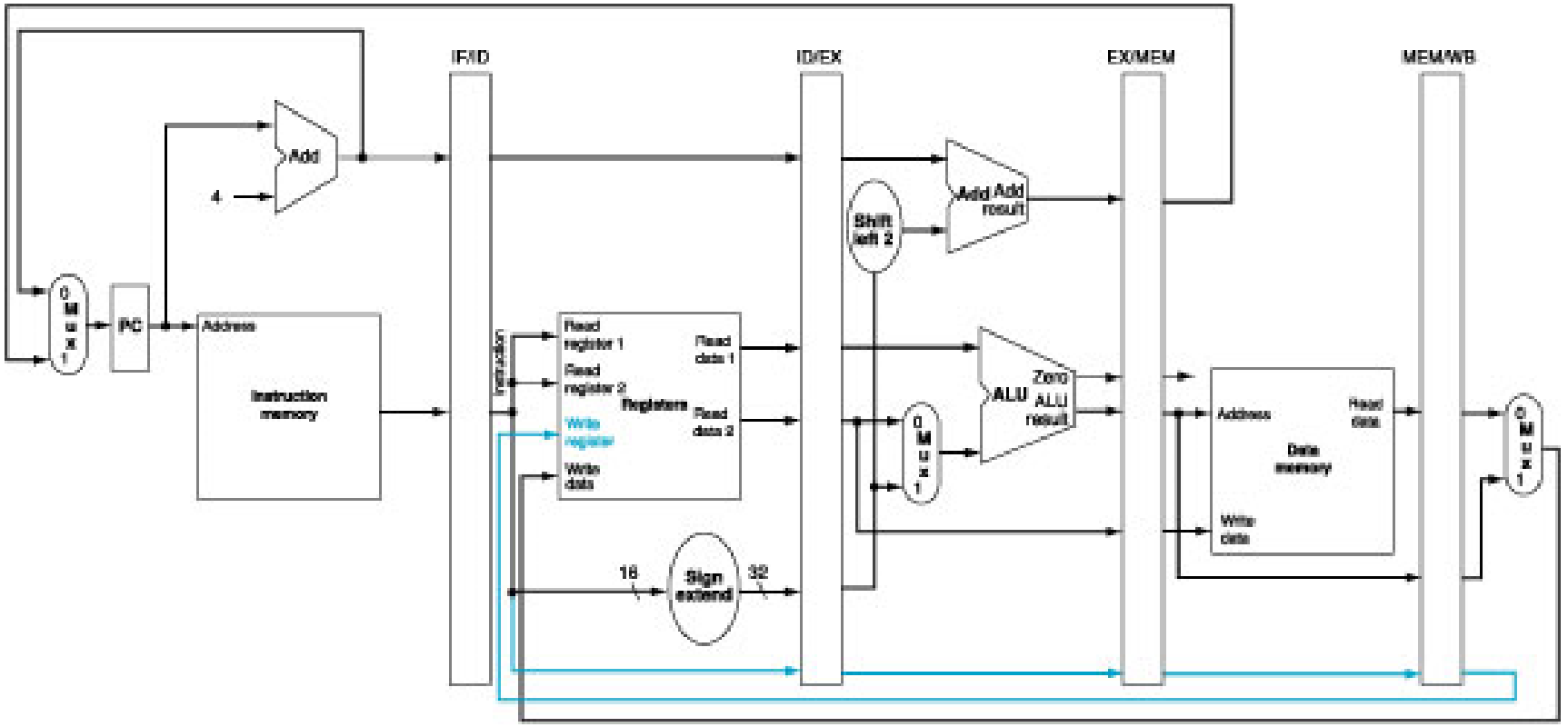
- ***Μπορείτε να διακρίνεται κάποιο πρόβλημα, ακόμα και στην απουσία εξαρτώμενων δεδομένων;***
- ***Ποιά εντολή μπορεί να δημιουργήσει το πρόβλημα;***



WB for Load


lw
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32
Wrong register number

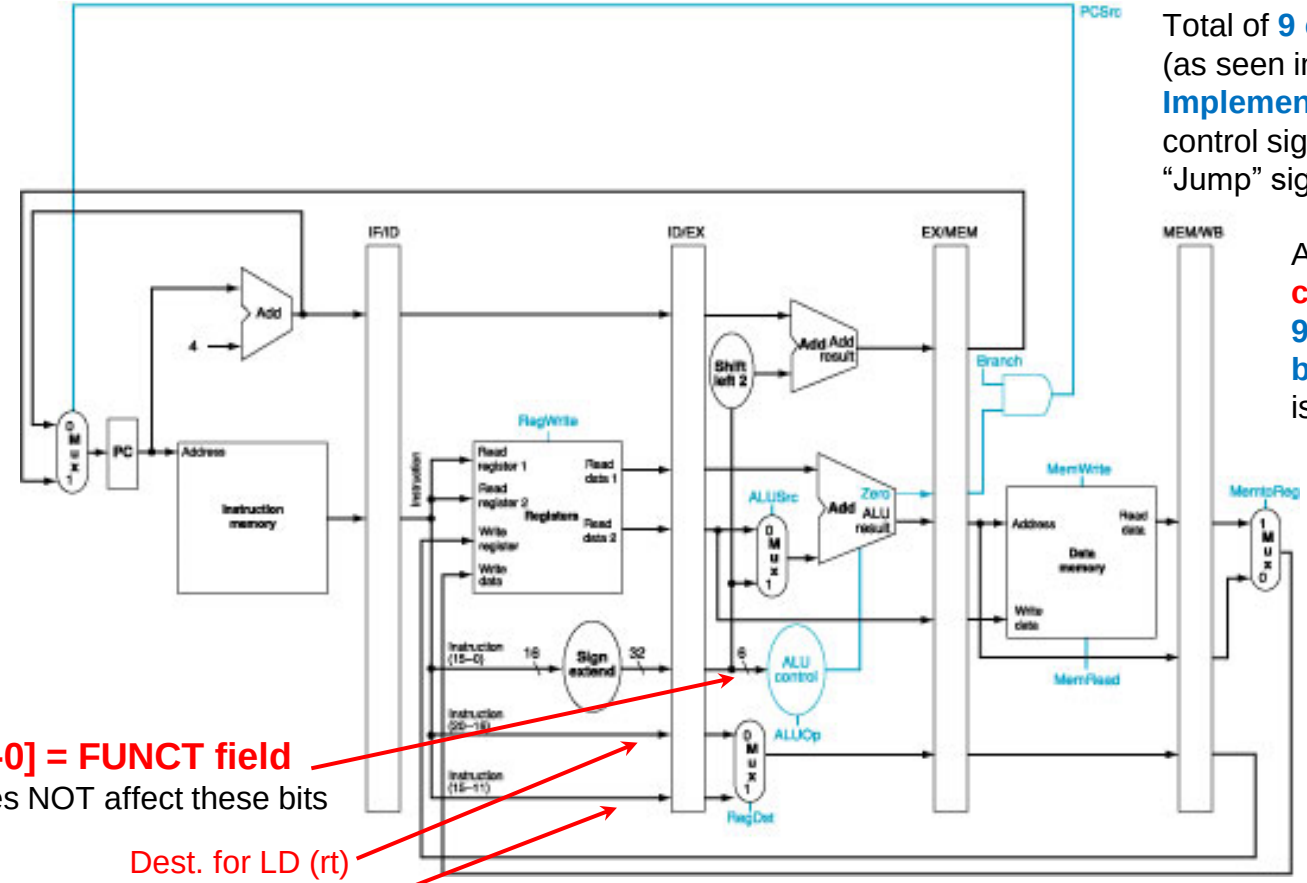
Διόρθωση Διαδρόμου Δεδομένων



Για την εντολή **lw**, πρέπει ο **αριθμός** του καταχωρητή γραφής να μεταφέρεται μέσω των καινούριων καταχωρητών στο τελευταίο στάδιο της εντολής (5ο)

Μονάδα Ελέγχου με Διασωλήνωση

- Πρώτα εξετάζουμε την απλή, «ιδανική», κατάσταση…
- Ξεχωρίζουμε τα σήματα ελέγχου στο διάδρομο δεδομένων (τα ίδια με αυτά της υλοποίησης χωρίς διασωλήνωση)
- PC, IF/ID, ID/EX, EX/MEM, MEM/WB πρέπει να μπορούν να ενημερωθούν σε κάθε κύκλο (στάδιο διασωλήνωσης), έτσι **δεν χρειάζονται σήματα ελέγχου για γραφή**



Total of **9 control signal bits** (as seen in the **Single-Cycle Implementation**). The only control signal missing is the "Jump" signal.

Actually, there are **8 control signals**, but **9 control signal bits**, since **ALUOp** is a 2-bit signal.

Instruction[5-0] = FUNCT field
Sign Extension does NOT affect these bits

Dest. for LD (rt)
Dest. For R-type (rd)

Μονάδα Ελέγχου με Διασωλήνωση (συν.)

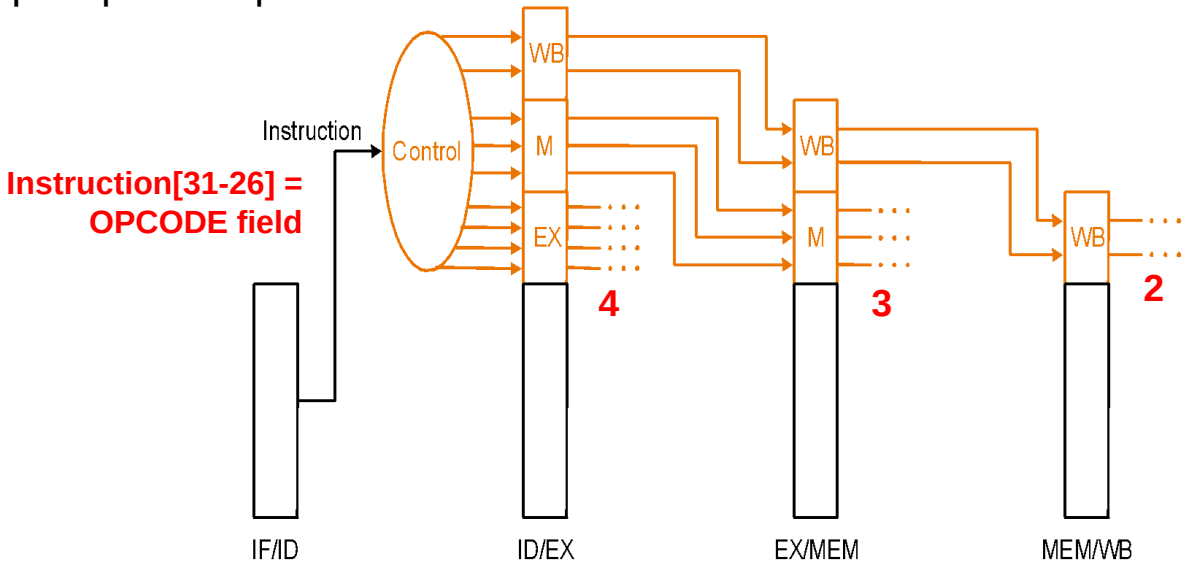
- Υπάρχουν 5 στάδια. Τι πρέπει να ελεγθεί σε κάθε στάδιο (βλέπε προηγούμενο slide);
  - Προσκόμιση Εντολής και Αύξηση PC (PCSrc)
  - Αποκωδικοποίηση Εντολής / Ανάγνωση Καταχωρητών
  - Εκτέλεση (**ALUOp**, ALUSrc, Zero, RegDst) [3 signals, **4 signal bits**]
  - Στάδιο Μνήμης (Branch, MemRead, MemWrite) [**3 signal bits**]
  - Γραφή στο ΑΚ (MemtoReg, RegWrite) [**2 signal bits**]

- Πώς θα χειριζόμασταν το θέμα του ελέγχου σε ένα εργοστάσιο κατασκευής αυτοκινήτων;
  - Με ένα πολυδάπανο κέντρο ελέγχου που θα λέει στον καθένα τι πρέπει να κάνει και πότε;
  - Ή ένα αυτοματοποιημένο σύστημα που επιτρέπει στο κάθε αυτοκίνητο να «κουβαλάει» μαζί του τα σήματα ελέγχου;

Μονάδα Ελέγχου με Διασωλήνωση (συν.)

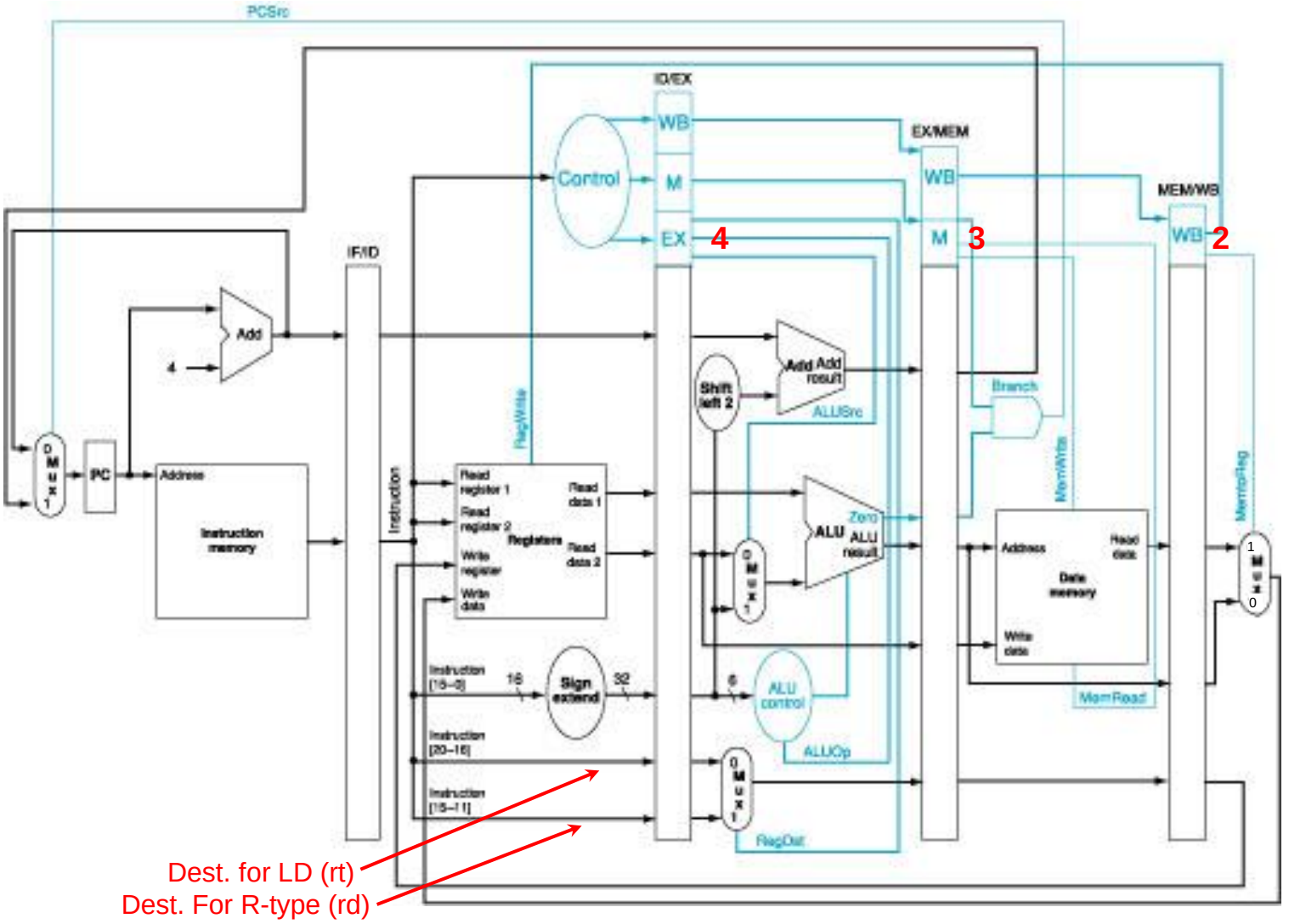
Κατηγορία Εντολής	Σήματα Ελέγχου για Εκτέλεση/Υπολογισμό Διεύθυνσης (Στάδιο 3)				Σήματα Ελέγχου για Προσπέλαση Μνήμης (Στάδιο 4)			Σήματα Ελέγχου για Ενημερωση ΑΚ (Στάδιο 5)	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
`lw`	0	0	0	1	0	1	0	1	1
`sw`	X	0	0	1	0	0	1	0	X
`beq`	X	0	1	0	1	0	0	0	X

- Τα σήματα ελέγχου μεταφέρονται από στάδιο σε στάδιο, όπως τα υπόλοιπα απαραίτητα δεδομένα

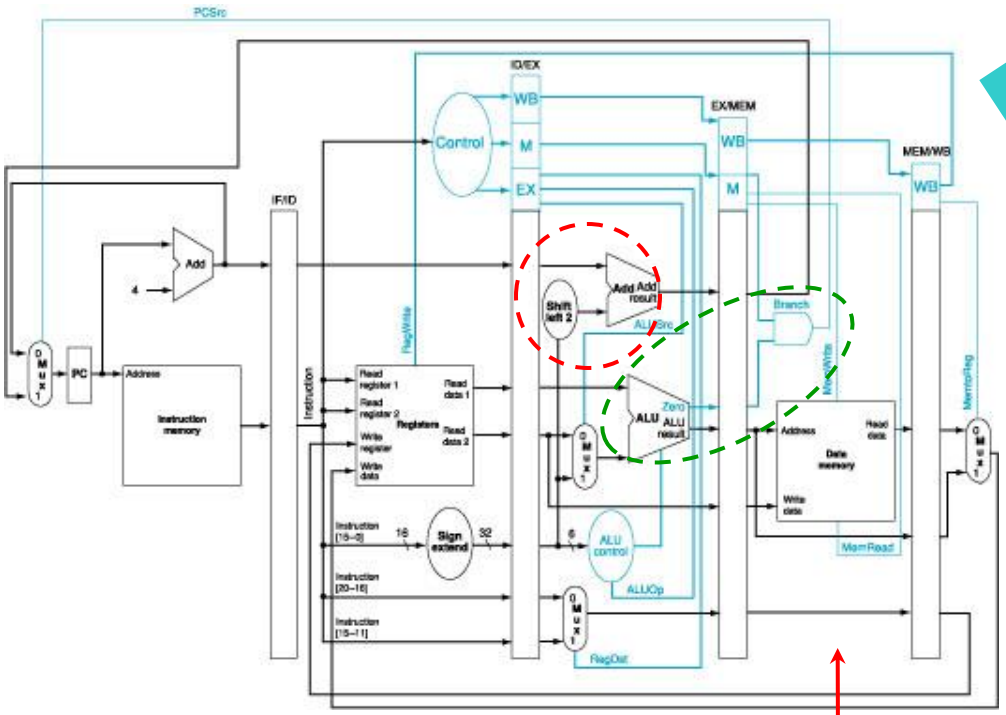


These are the same **9 control signal bits** we saw in the **Single-Cycle Implementation**! The only control signal missing is the “Jump” signal.

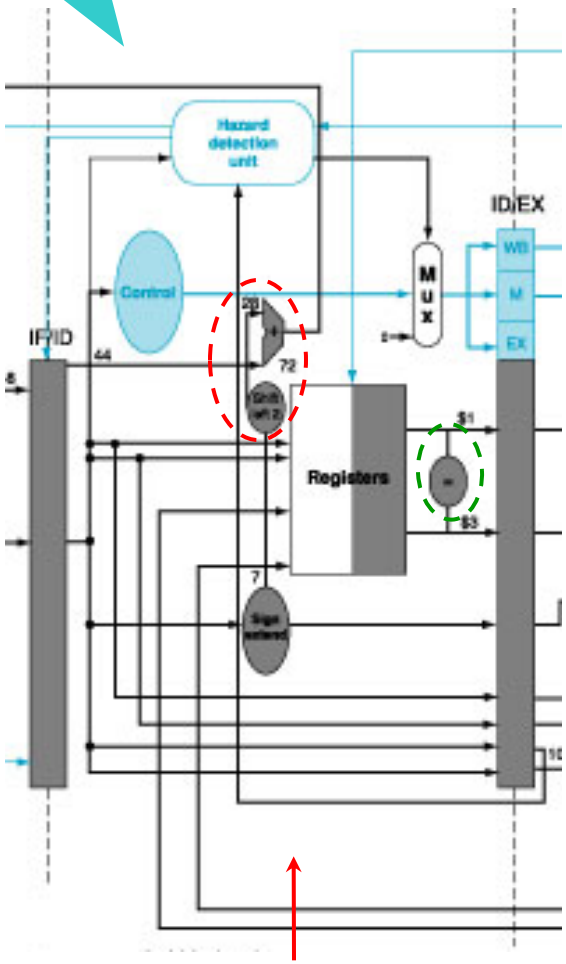
Διάδρομος Δεδομένων με Μονάδα Ελέγχου



Γρηγορότερη απόφαση διακλαδώσεως

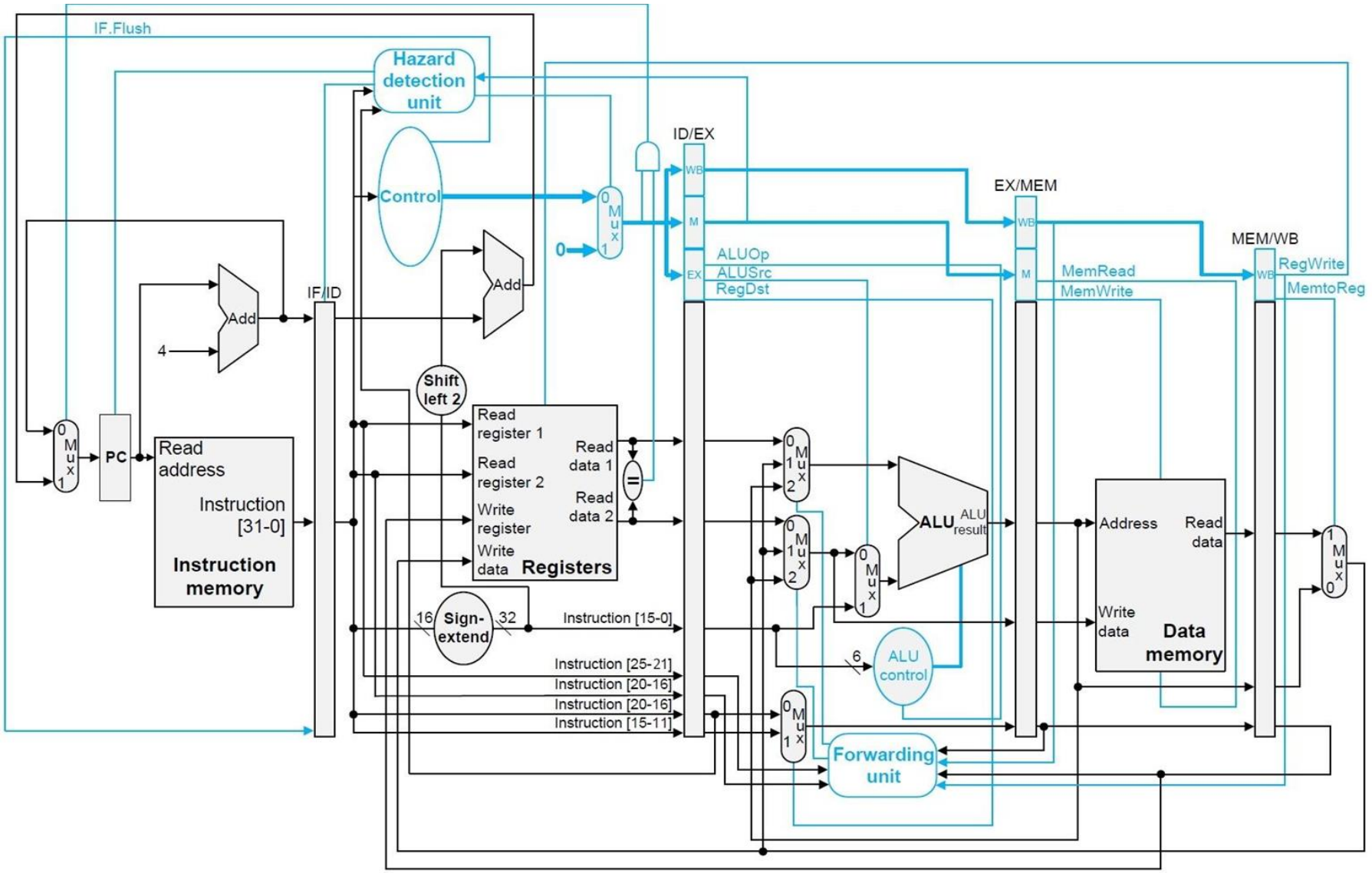


Branch outcome was *originally* resolved in this stage (MEM)



Branch outcome in *now* resolved in this stage (ID)

Σύνοψη: Εργαστηριακές Ασκήσεις 3 και 4



Επανάληψη σχετικής ύλης από ΗΜΥ 212

Ύλη που αφορά την Εργαστηριακή **Άσκηση 4**

Προσθήκη Μονάδων **Ανίχνευσης Κινδύνων** και **Προώθησης** στην υλοποίηση **Διασωλήνωσης**

Data Hazards in ALU Instructions

- Consider this sequence:

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

- We can resolve hazards with forwarding
  - How do we detect when to forward?

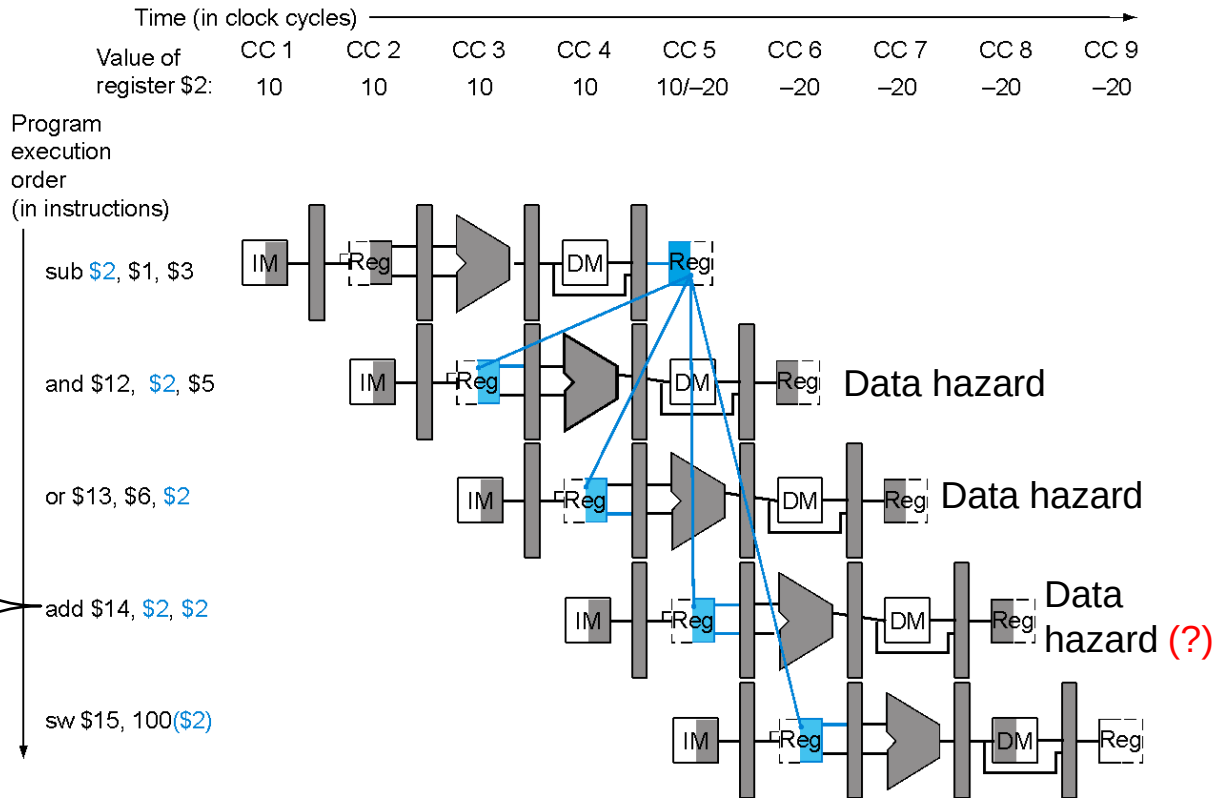
Εξάρτηση (*Dependency*) – Κίνδυνοι Δεδομένων

- Μπορεί να δημιουργηθεί πρόβλημα όταν μια εντολή ξεκινήσει να εκτελείται πριν την ολοκλήρωση κάποιας άλλης
  - Οι εξαρτήσεις που «προχωρούν πίσω στο χρόνο» είναι βασικά **κίνδυνοι δεδομένων**

- Οι 4 τελευταίες εντολές εξαρτώνται από το αποτέλεσμα του καταχωρητή `$2` από την 1η εντολή

- Θεωρήστε ότι ο `$2` είχε την τιμή 10 πριν την εντολή `sub` και -20 μετά την `sub`

- Η εξάρτηση μεταξύ των `sub` και `add` μπορεί να αφαιρεθεί αν το ΑΚ σχεδιαστεί ώστε να επιτρέπει ανάγνωση και γραφή σε 1 κύκλο



Λύση βάση Λογισμικού (*Software Solution*)

- Ο μεταγλωττιστής αποτρέπει τη δημιουργία κινδύνων
- Πού εισάγουμε “**nop**” (μια εντολή που δεν κάνει καμία πράξη που να αλλάζει την κατάσταση του επεξεργαστή);

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

NOP in MIPS:

```
sll $0,$0,0
```

(All 32 bits of this instruction are 0)

Λύση βάση Λογισμικού (*Software Solution*)

- Ο μεταγλωττιστής αποτρέπει τη δημιουργία κινδύνων
- Πού εισάγουμε "**nop**" (μια εντολή που δεν κάνει καμία πράξη που να αλλάζει την κατάσταση του επεξεργαστή);

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

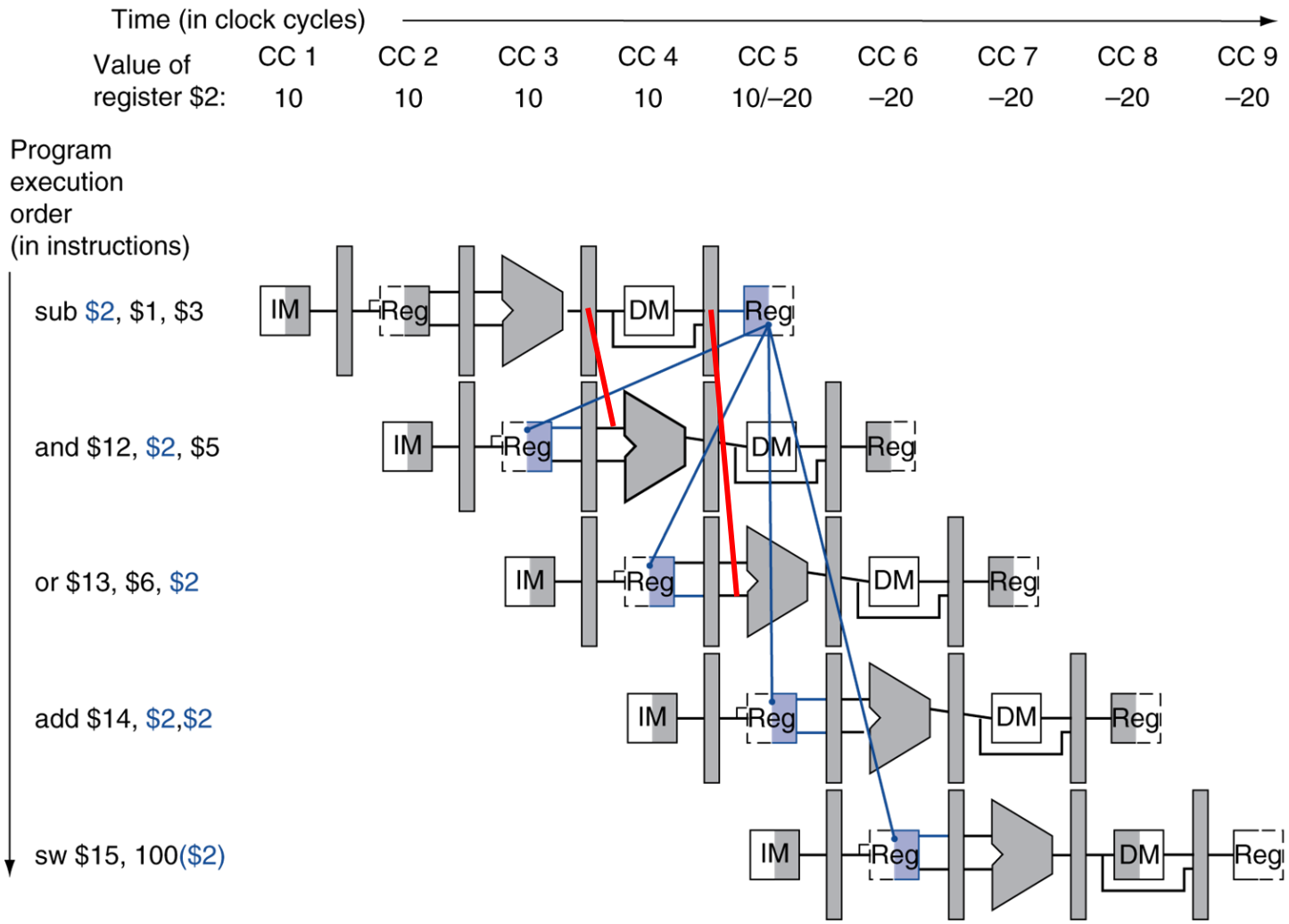
NOP in MIPS:

```
sll $0,$0,0
```

(All 32 bits of this instruction are 0)

- Πρόβλημα: Αυξάνει τον χρόνο εκτέλεσης κατά 2 κύκλους!

Dependencies & Forwarding

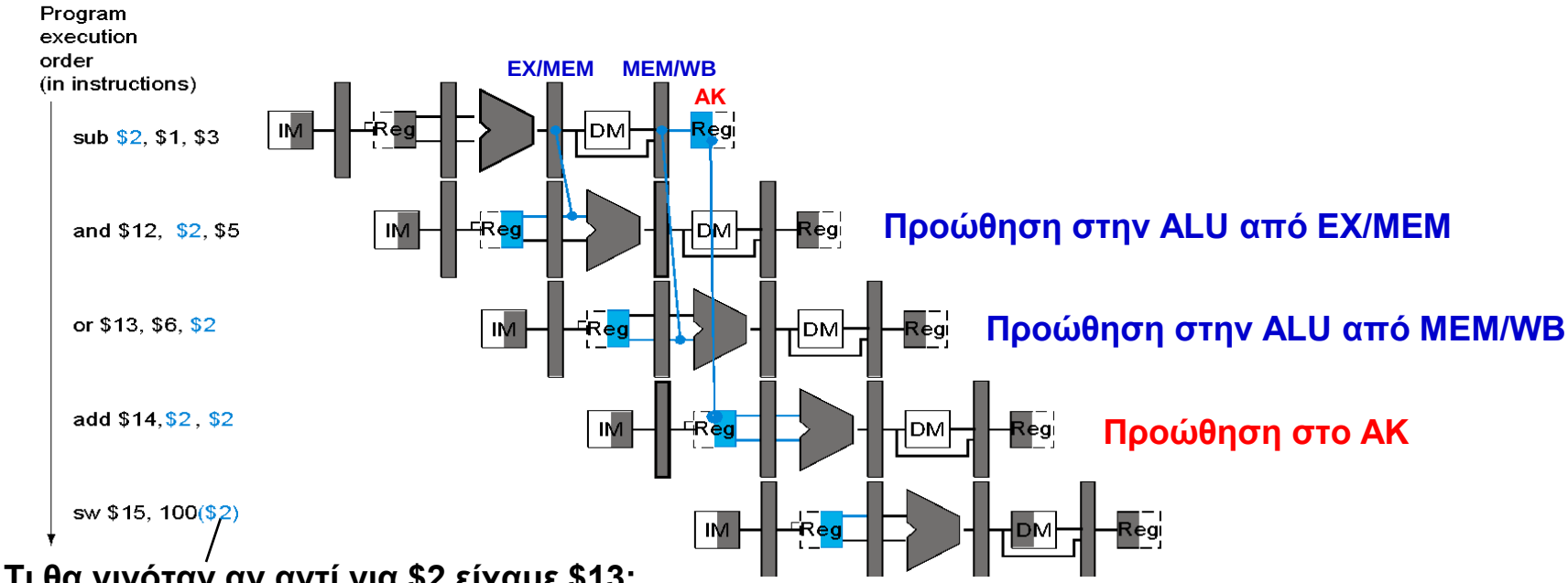


Προώθηση (*Forwarding*)

- Χρήση προσωρινών αποτελεσμάτων, χωρίς να περιμένει ο επεξεργαστής να γραφτούν στο αρχείο καταχωρητών (AK):
  - Προώθηση στην ALU (*ALU forwarding*)
  - Προώθηση στο AK για χειρισμό read/write στον ίδιο καταχωρητή (*Register File forwarding*)

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X

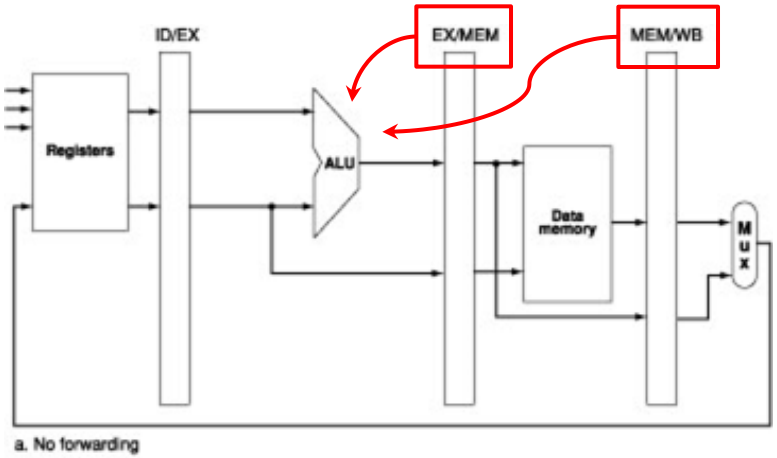


Τι θα γινόταν αν αντί για $2 είχαμε $13;

[Same as SUB-OR dependency]

Υλοποίηση Προώθησης

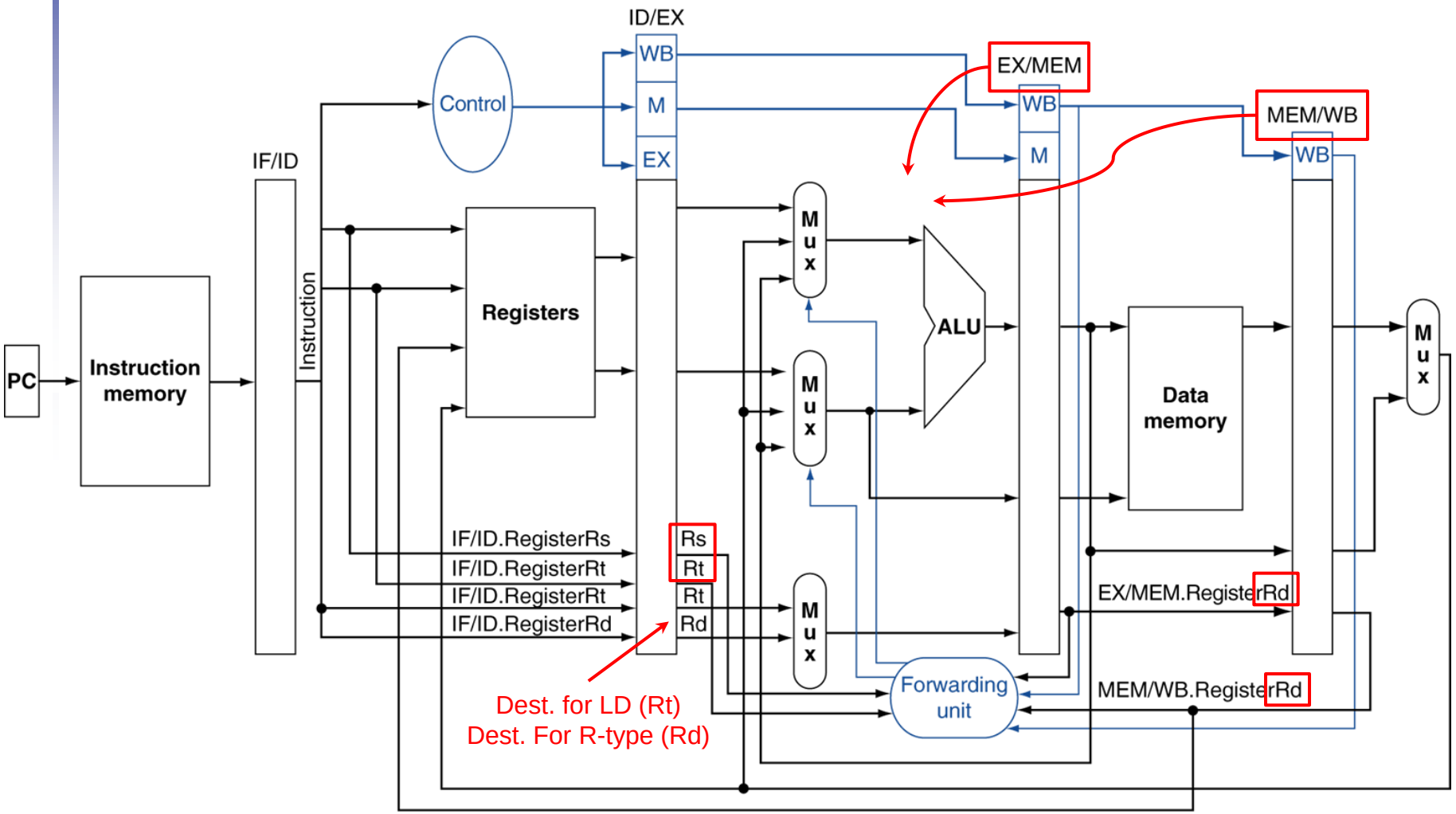
- **Pipeline_Register.Instruction_Register** υποδηλώνει την τιμή ενός καταχωρητή που ορίζεται σε μια εντολή (**`rs`**, **`rt`**, ή **`rd`**) και που είναι αποθηκευμένη σε έναν από τους καινούριους καταχωρητές (IF/ID, ID/EX, EX/MEM, MEM/WB). Π.χ.,
  - EX/MEM.RegisterRd (= **`rd`** στον EX/MEM)
  - ID/EX.RegisterRs (= **`rs`** στον ID/EX)
  - ID/EX.RegisterRt (= **`rt`** στον ID/EX)



a. No forwarding

- Για την υλοποίηση προώθησης στην ALU:
  1. EX/MEM.RegisterRd = ID/EX.RegisterRs / Rt υποδηλώνει έναν κίνδυνο δεδομένων στο στάδιο EX, όπου ο ***σωστός*** 1ος/2ος τελεστής της ALU βρίσκεται στον EX/MEM.RegisterRd, και όχι στον ID/EX.RegisterRs / Rt
  2. MEM/WB.RegisterRd = ID/EX.RegisterRs / Rt υποδηλώνει έναν κίνδυνο δεδομένων στο στάδιο MEM, όπου ο ***σωστός*** 1ος/2ος τελεστής βρίσκεται στον MEM/WB.RegisterRd, και όχι στον ID/EX.RegisterRs / Rt

Looking Ahead: Datapath with Forwarding



Υλοποίηση Προώθησης (συν.)

- Είναι πάντα απαραίτητη η προώθηση;
  - Χρειάζεται μόνο όταν το σήμα RegWrite είναι ενεργοποιημένο. Γιατί; [Some instructions do NOT write to registers]
  - Επίσης, για τον MIPS, όταν ένας από τους τελεστές είναι ο καταχωρητής **$0** η αντίστοιχη τιμή που χρησιμοποιείται είναι πάντα το μηδέν (δεν είναι ανάγκη να προωθηθεί ο **$0**) [If Rd of forwarding instruction is register $0, no need to forward]

- Οι είσοδοι της ALU πρέπει να έρθουν από πολυπλέκτες οι οποίοι επιλέγουν να προωθήσουν όταν είναι απαραίτητο

- **Συνθήκη Ανίχνευσης Κινδύνων στο στάδιο EX** και σήματα ελέγχου που πρέπει να ενεργοποιηθούν:

  Ανίχνευση Κινδύνων EX και Προώθηση

  if (EX/MEM.RegWrite)
  and (EX/MEM.RegisterRd != 0)
  and (EX/MEM.RegisterRd == ID/EX.Register**Rs**)) ForwardA = 10

  if (EX/MEM.RegWrite)
  and (EX/MEM.RegisterRd != 0)
  and (EX/MEM.RegisterRd == ID/EX.Register**Rt**)) ForwardB = 10

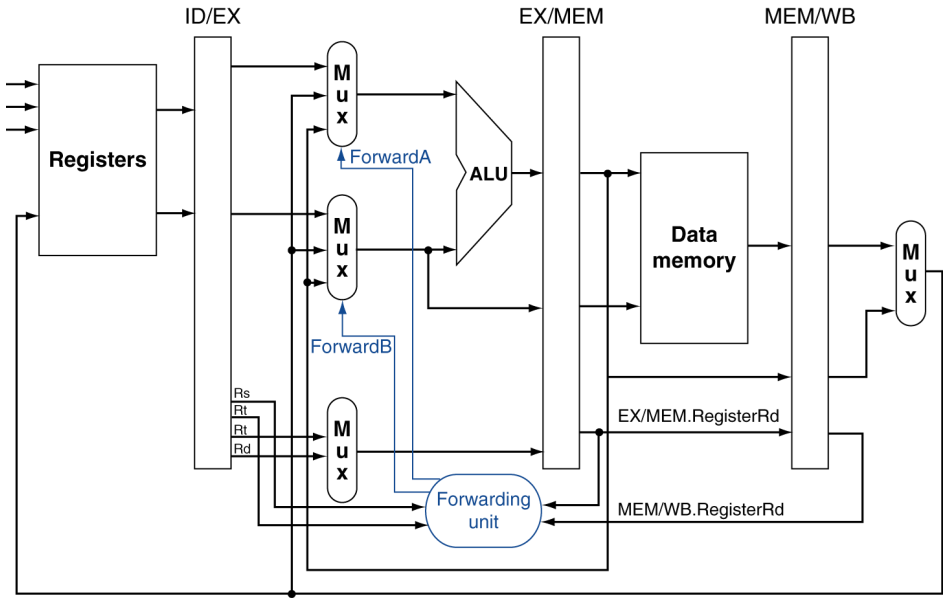
Υλοποίηση Προώθησης (συν.)

- **Συνθήκη Ανίχνευσης Κινδύνων στο στάδιο MEM** και σήματα ελέγχου που πρέπει να ενεργοποιηθούν:

Ανίχνευση Κινδύνων MEM και Προώθηση

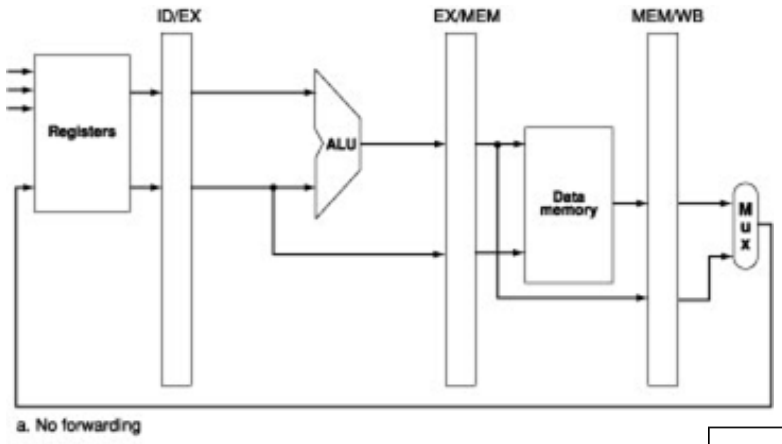
if (MEM/WB.RegWrite)
and (MEM/WB.RegisterRd != 0)
and (MEM/WB.RegisterRd == ID/EX.Register**Rs**)) ForwardA = 01

if (MEM/WB.RegWrite)
and (MEM/WB.RegisterRd != 0)
and (MEM/WB.RegisterRd == ID/EX.Register**Rt**)) ForwardB = 01



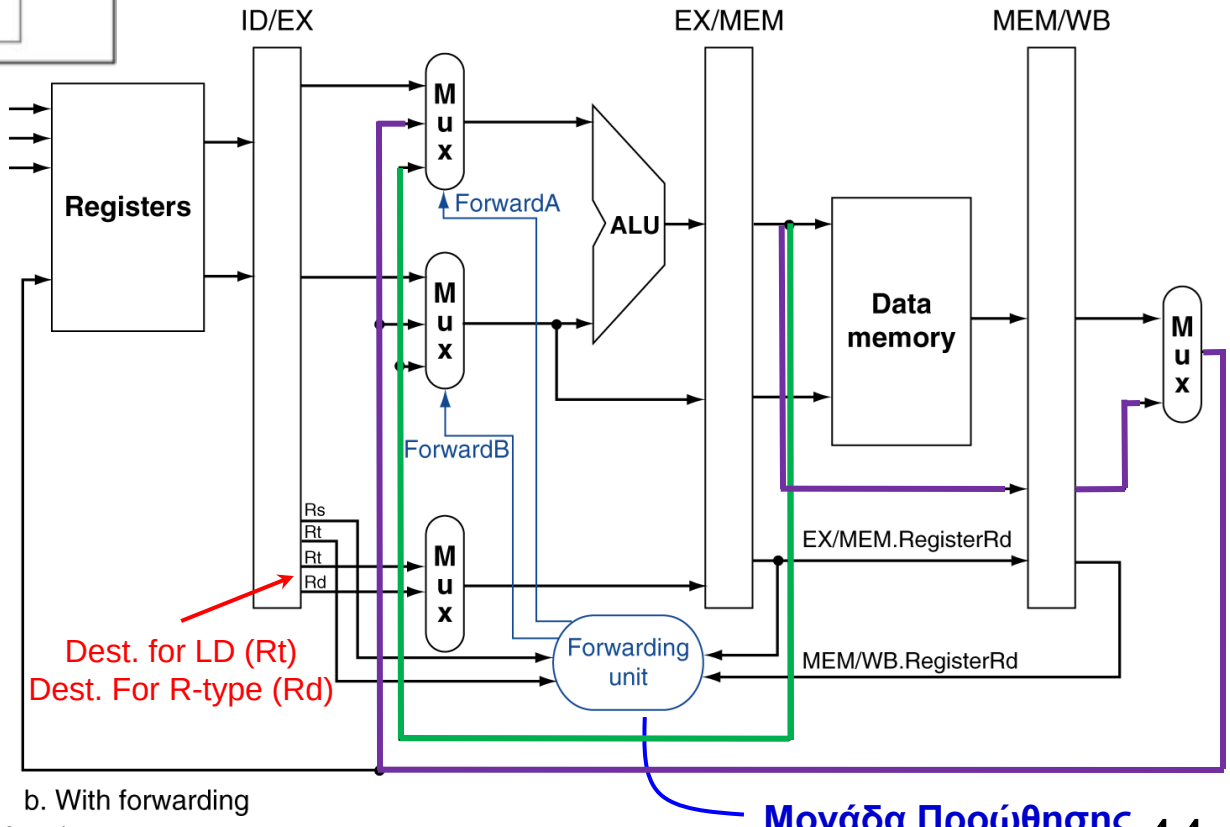
b. With forwarding

Υλοποίηση Προώθησης (συν.)

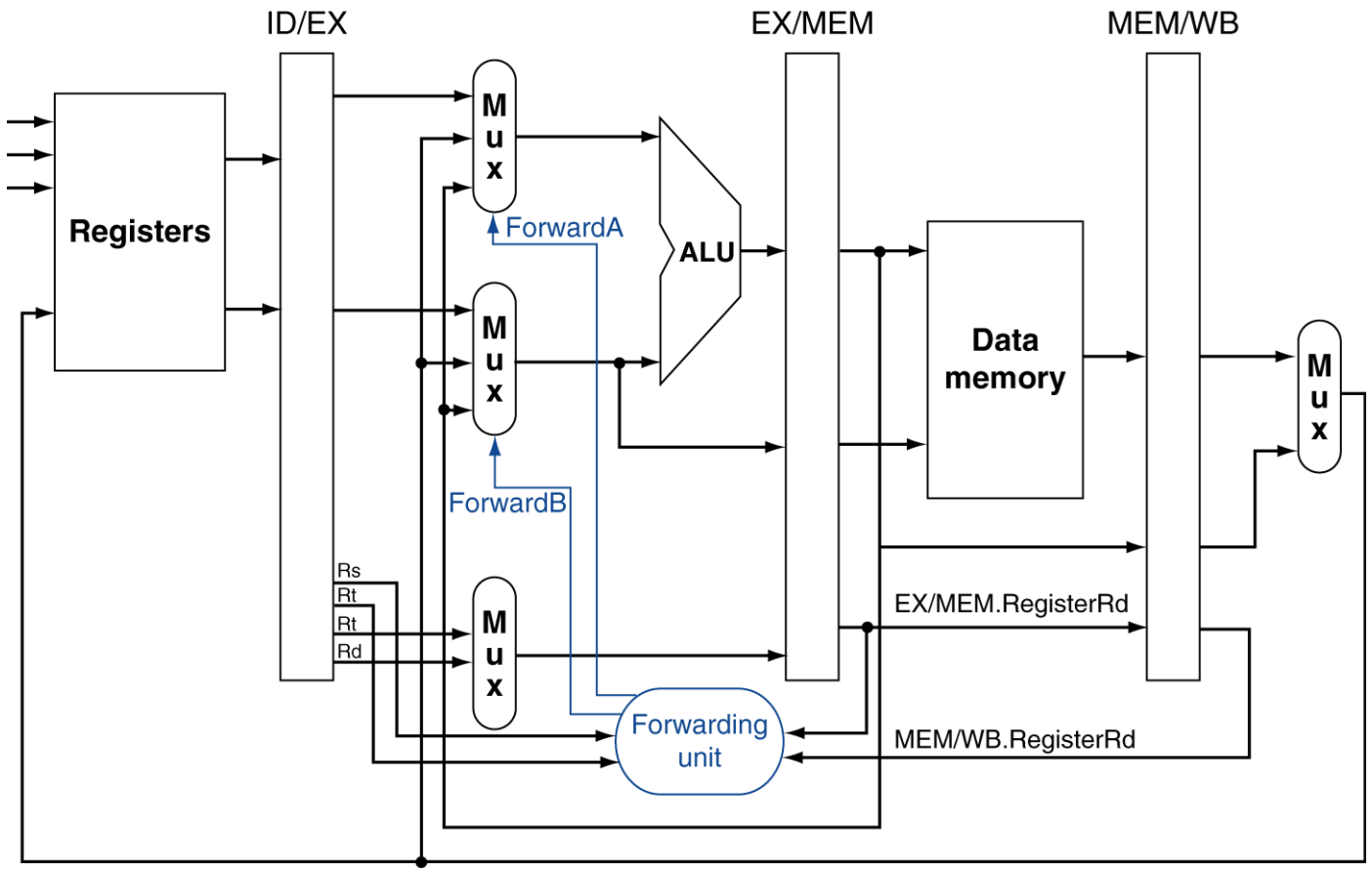


Θεωρήστε μόνο τις εντολές **`add`**, **`sub`**, **`or`**, **`and`** για προώθηση (R-type)

Σήμα Επιλογής MUX	Πηγή
ForwardA=00	ID/EX
ForwardA=10	**EX/MEM**
ForwardA=01	**MEM/WB**
ForwardB=00	ID/EX
ForwardB=10	**EX/MEM**
ForwardB=01	**MEM/WB**



Forwarding Paths



b. With forwarding

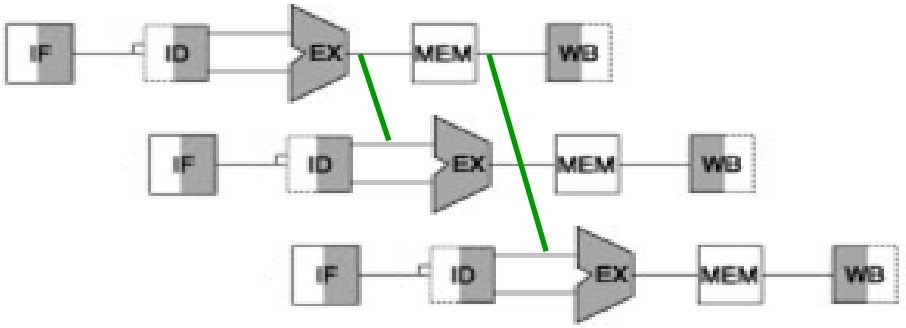
Υλοποίηση Προώθησης (συν.)

- Τι θα συμβεί αν έχουμε την πιο κάτω ακολουθία εντολών;

```
add $s1, $s1, $s2
add $s1, $s1, $s3
add $s1, $s1, $s4  # $s1=$s1+$s2+$s3+$s4
```

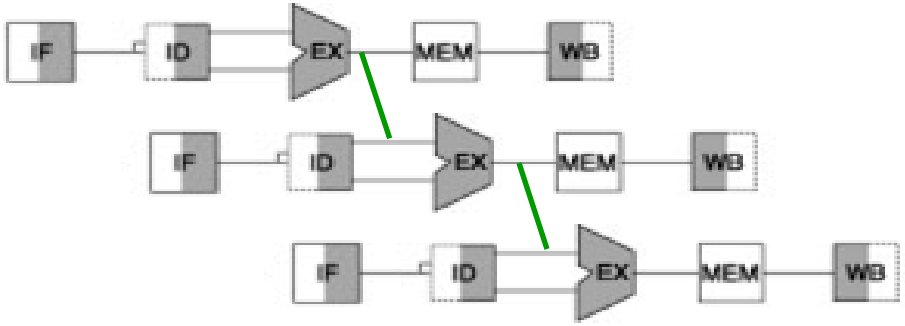
Θα δουλέψει αυτό;

[Όχι]



Αυτό;

[Ναι]



Double Data Hazard

- Consider the sequence:

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

- Both hazards occur
  - Want to use the most recent
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn't true

Υλοποίηση Προώθησης (συν.)

- Η συνθήκη ανίχνευσης κινδύνων στο στάδιο MEM πρέπει να περιλαμβάνει και την πιο κάτω προϋπόθεση:

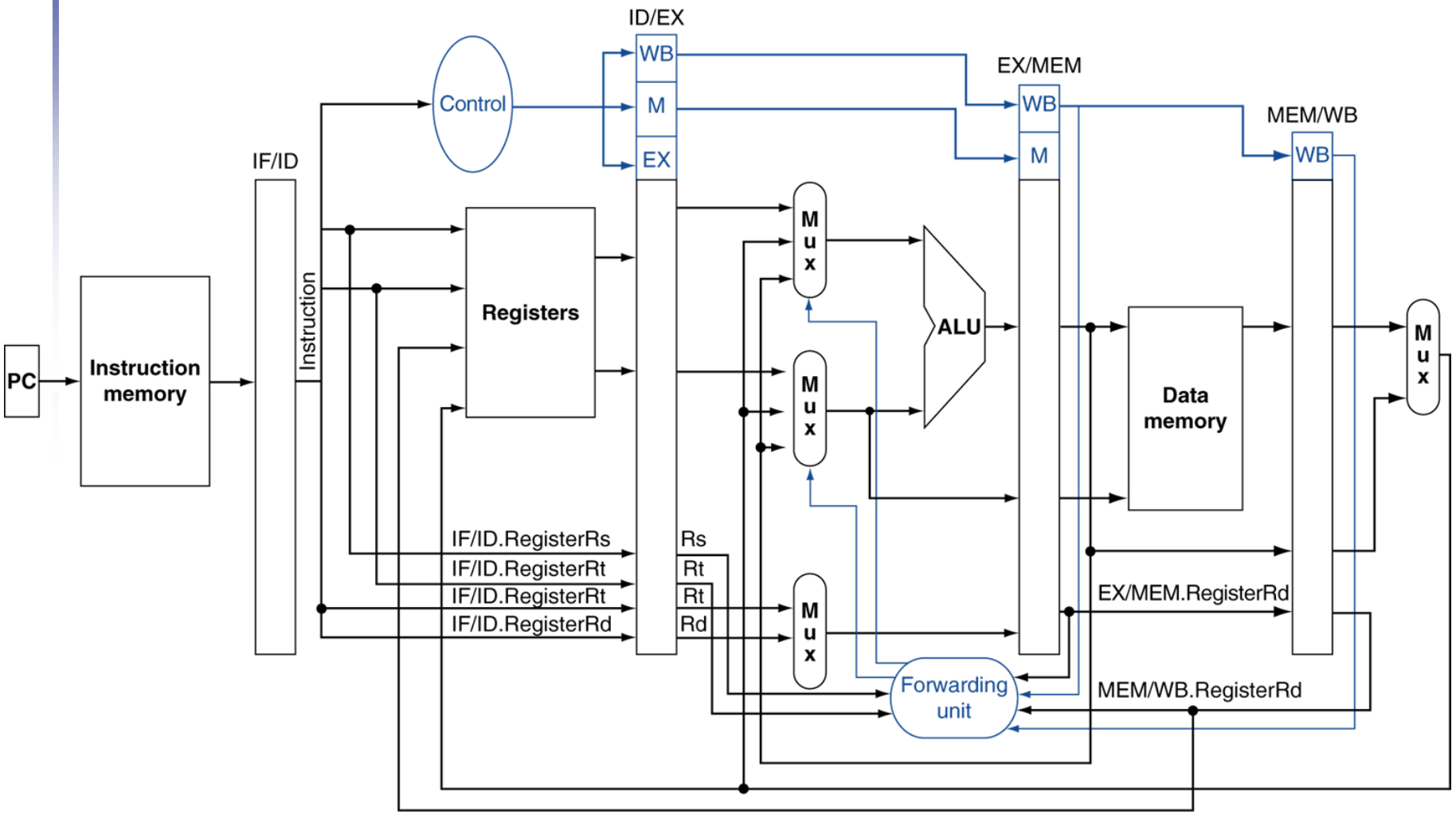
Ανίχνευση Κινδύνων MEM και Προώθηση

if (MEM/WB.RegWrite)
and (MEM/WB.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 (EX/MEM.RegisterRd != ID/EX.RegisterRt)
and (MEM/WB.RegisterRd == ID/EX.RegisterRt)) ForwardB = 01

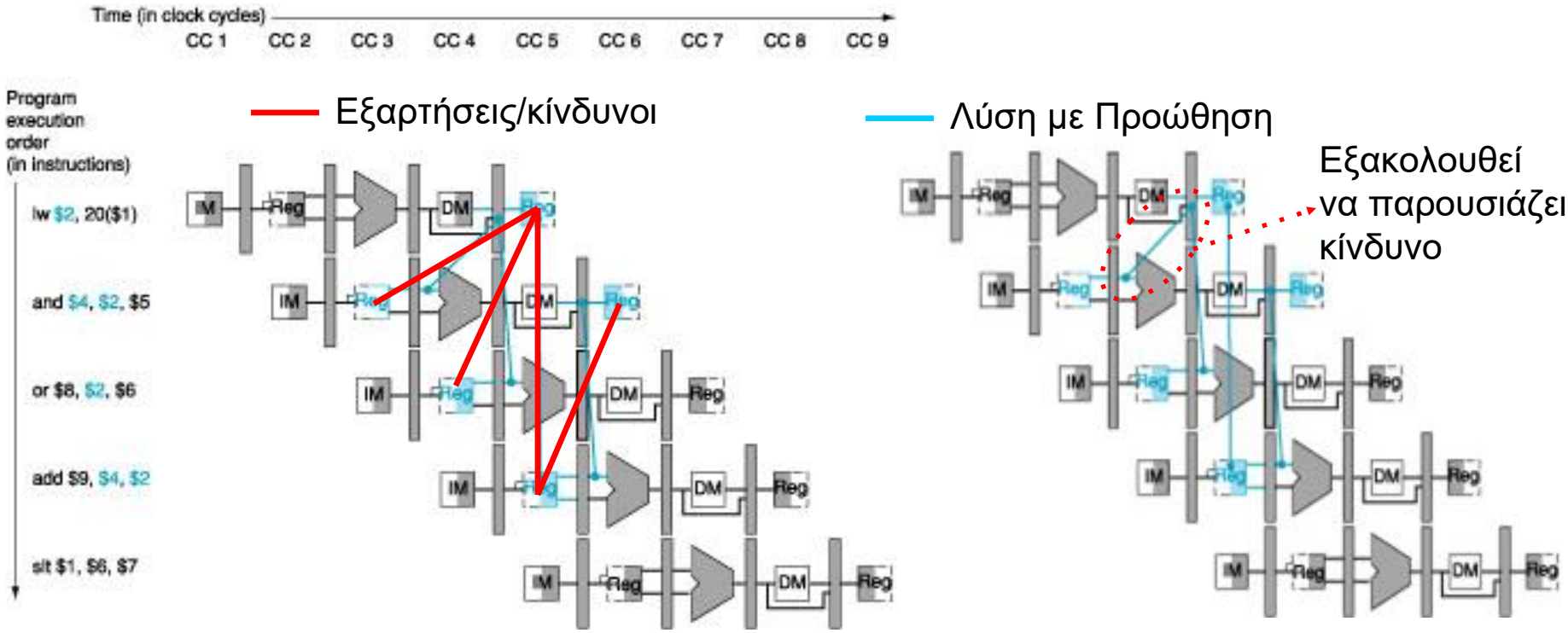
In other words: If we have BOTH EX and MEM hazards, treat the situation as an EX hazard to get the most recent result.

Datapath with Forwarding



Η προώθηση δεν είναι πάντα δυνατή

- Η εντολή **`lw`** μπορεί ακόμη να προκαλέσει κίνδυνο (Load-Use Hazard):
  - Μια εντολή προσπαθεί να διαβάσει έναν καταχωρητή. Στο προηγούμενο στάδιο υπάρχει μια εντολή **`lw`** που γράφει στον ίδιο καταχωρητή.



- Επομένως, χρειάζεται μια μονάδα ανίχνευσης κινδύνων που να καθυστερεί (με “stall”) την εντολή που ακολουθεί (αμέσως μετά) την εντολή **`lw`**

Υλοποίηση για Stall (για Load-Use Hazards)

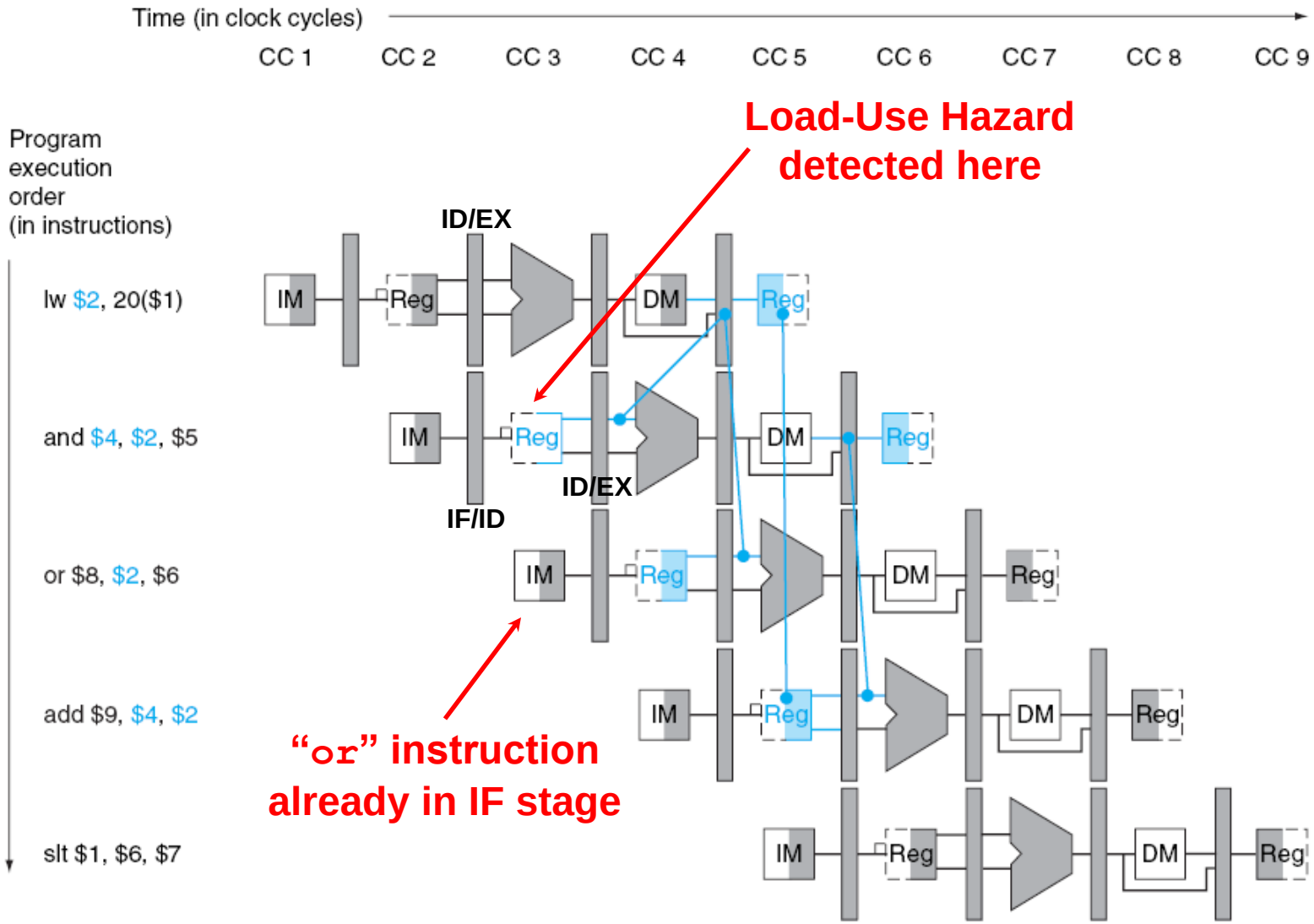
- Απαιτεί μια **μονάδα ανίχνευσης κινδύνων** (***hazard detection unit***) (επιπρόσθετα της μονάδας προώθησης) που λειτουργεί κατά τη διάρκεια του σταδίου ID (2ο στάδιο) **της εντολής που ακολουθεί την εντολή load** και εισάγει **stall** όπου χρειάζεται

- **Ανίχνευση Κινδύνων και Stalling**

  if (ID/EX.MemRead and [Is the instruction in the EX stage a Load?] (i.e., the next stage)
  ((ID/EX.RegisterRt == IF/ID.RegisterRs) or
  (ID/EX.RegisterRt == IF/ID.RegisterRt))) [Is the destination register of the Load in the EX stage a source of the instruction in the ID stage?] (i.e., the current stage)
  stall the pipeline

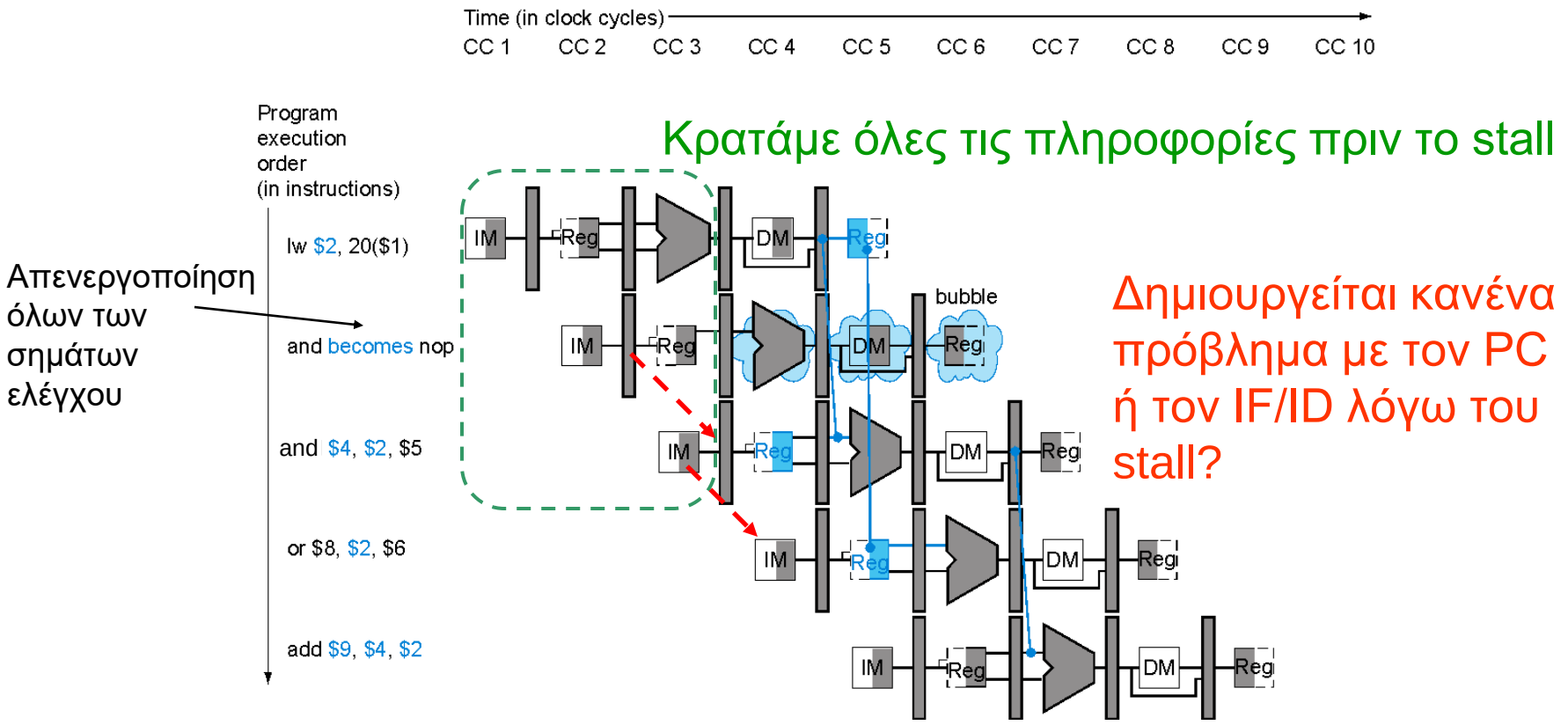
- **Γνωστό σαν Load-Use Hazard**

Ανίχνευση κινδύνου Load-Use



Παράδειγμα με Stall

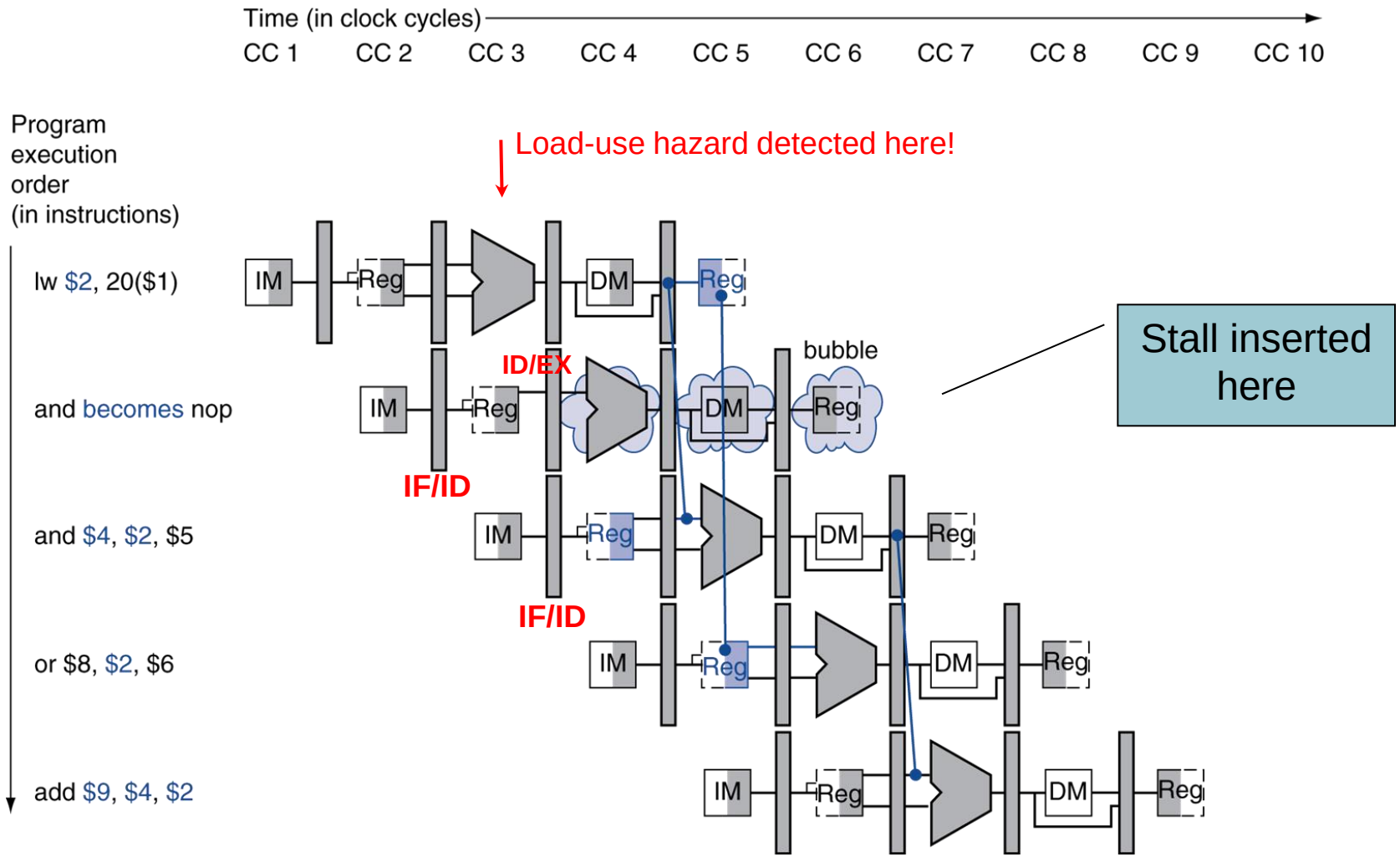
- Μπορούμε να εισάγουμε καθυστέρηση (stall) με το να «κρατήσουμε» μια εντολή στο ίδιο στάδιο



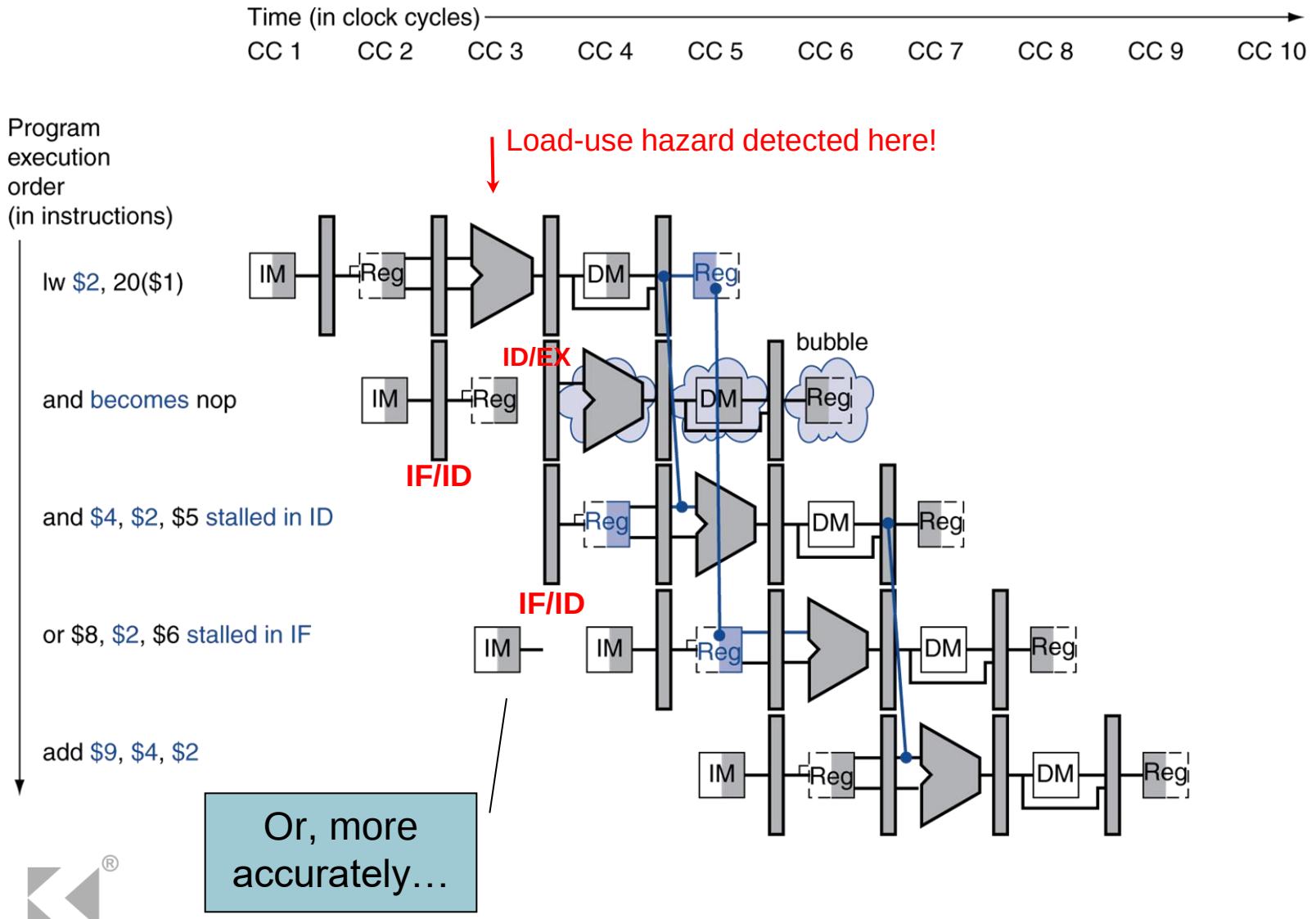
How to Stall the Pipeline

- Force **control values in ID/EX register to 0**
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction ("**`and`**") is decoded again
  - Following instruction ("**`or`**") is fetched again
  - 1-cycle stall allows MEM to read data for lw
    - Can subsequently forward to EX stage

Stall/Bubble in the Pipeline

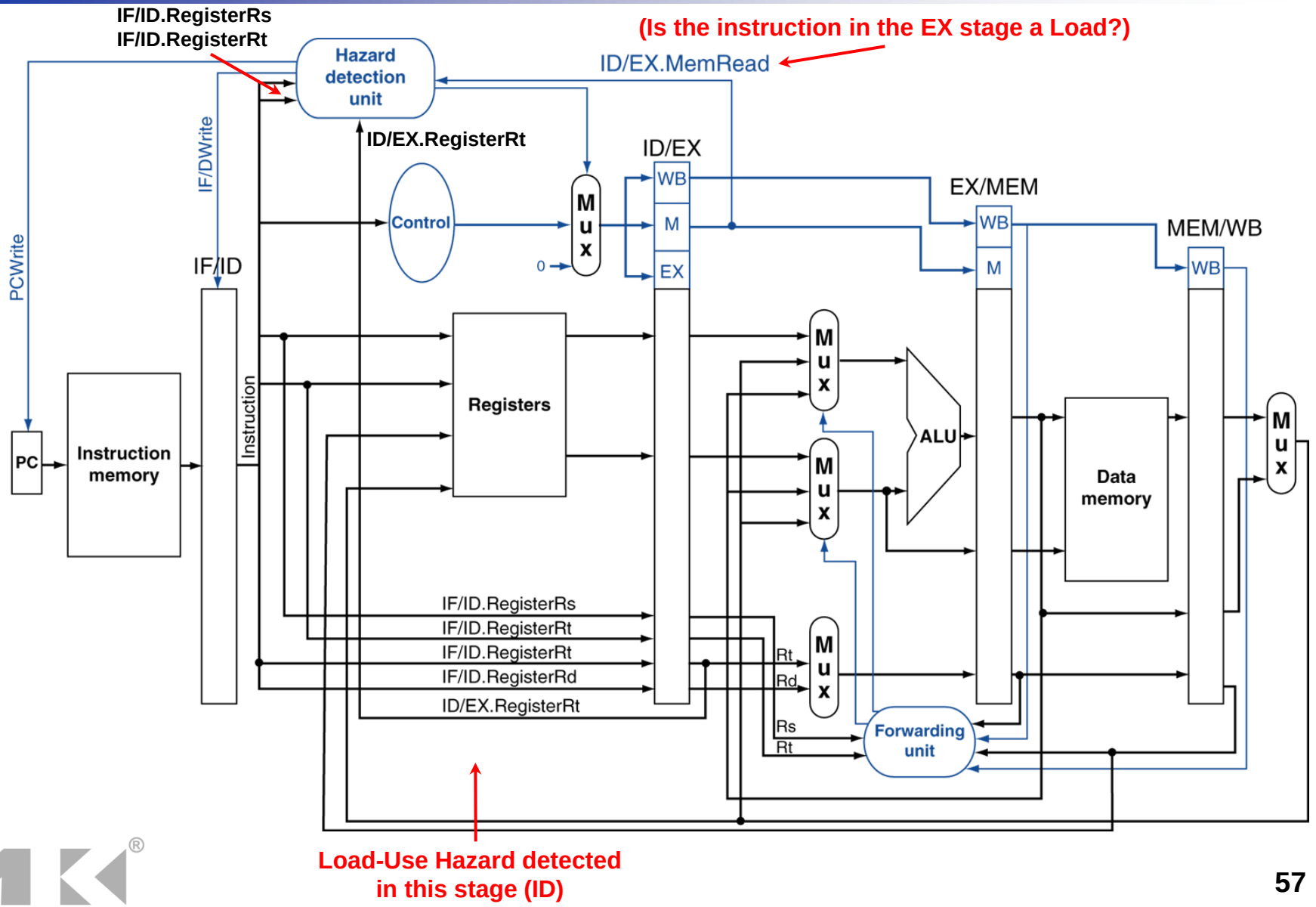


Stall/Bubble in the Pipeline



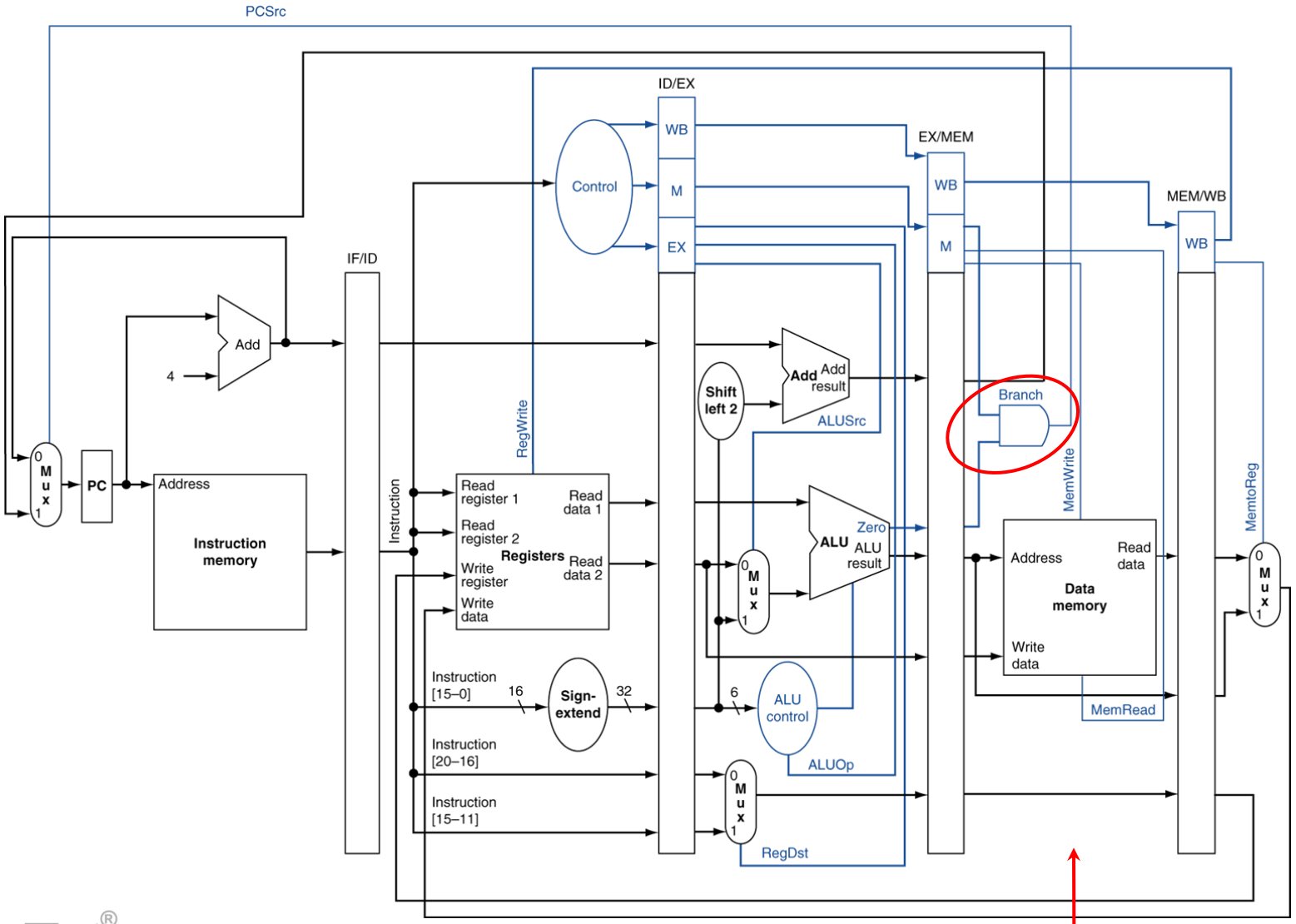
Datapath with Hazard Detection

[For Load-Use Hazards]



Κίνδυνοι Διακλαδώσεως ή Ελέγχου (Branch/Control)

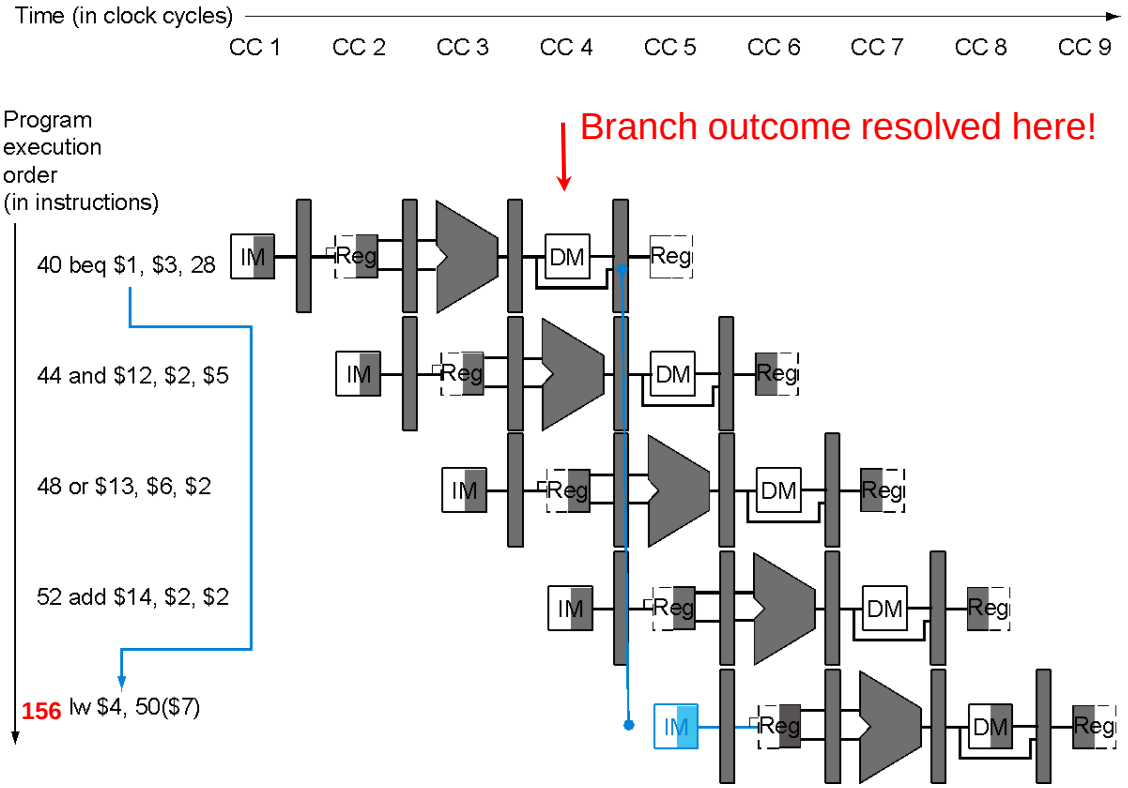
Reminder: Pipelined Control



Branch outcome resolved in this stage (MEM)

Κίνδυνοι Διακλαδώσεως ή Ελέγχου (*Branch or Control Hazards*)

- Μέχρι να υπολογιστεί η συνθήκη της διακλάδωσης, κάποιες εντολές έχουν ήδη αρχίσει να εκτελούνται! **Η συνθήκη της διακλάδωσης δεν υπολογίζεται μέχρι το στάδιο MEM** → οι επόμενες 3 εντολές φορτώνονται ανεξάρτητα της συνθήκης!



[PC-relative addressing: (40 + 4) + (28 x 4) = 156]

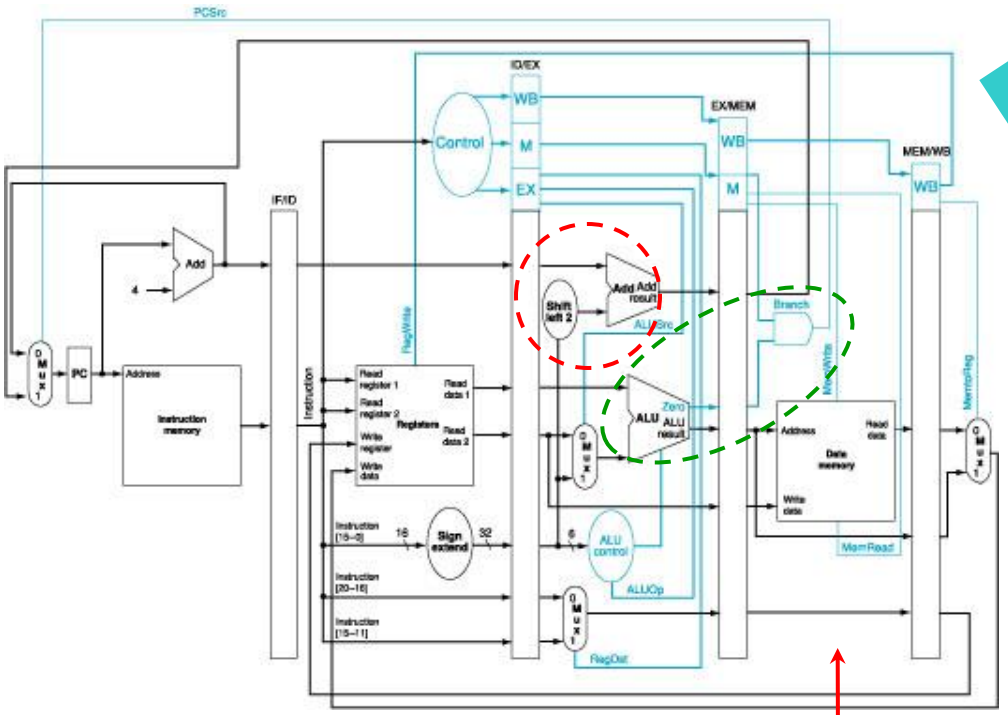
- **Προβλέπουμε ότι η συνθήκη της διακλάδωσης δεν ισχύει** (*"branch not taken"*)
  - Πρέπει να προσθέσουμε υλικό για να ξεφορτωθούμε (*flush*) 3 εντολές, αν η πρόβλεψη είναι λανθασμένη

Κίνδυνοι Διακλαδώσεως (συν.)

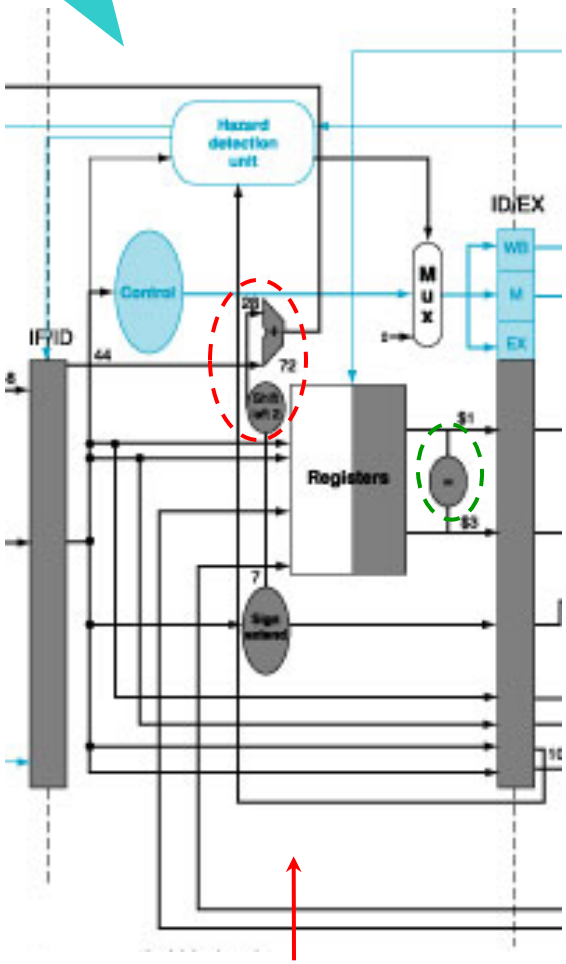
- Μπορούμε να μειώσουμε τον αριθμό των εντολών που πρέπει να ξεφορτωθούμε λόγω λανθασμένης πρόβλεψης
- Η εκτέλεση λειτουργιών για branch (υπολογισμός διεύθυνσης και συνθήκης διακλάδωσης) μετακινείται σε προηγούμενα στάδια (πριν το στάδιο MEM)
- Μπορούμε να μετακινήσουμε τον υπολογισμό της συνθήκης διακλάδωσης στο στάδιο ID;
  - **ΝΑΙ**. Με αυτόν τον τρόπο θα χρειάζεται να κάνουμε flush **μόνο μια εντολή** (στο IF στάδιο) αν η πρόβλεψη διακλάδωσης (branch not taken) είναι λανθασμένη
- Ο υπολογισμός της συνθήκης διακλάδωσης κατά τη διάρκεια του σταδίου ID μπορεί να δημιουργήσει κινδύνους:
  - Απαιτείται προώθηση σε καινούρια λογική που θα υπολογίζει τη συνθήκη (μέχρι στιγμής, η προώθηση γινόταν στην ALU)
  - Πιθανόν να συνεχίσει να χρειάζεται stall (π.χ., ALU εντολή αμέσως πριν την εντολή διακλάδωσης)

A stall may still be needed. For example, **if an ALU instruction immediately preceding a branch produces one of the operands for the comparison in the branch**, a stall will be required, since the EX stage for the ALU instruction will occur AFTER the ID cycle of the branch.

Γρηγορότερη απόφαση διακλαδώσεως

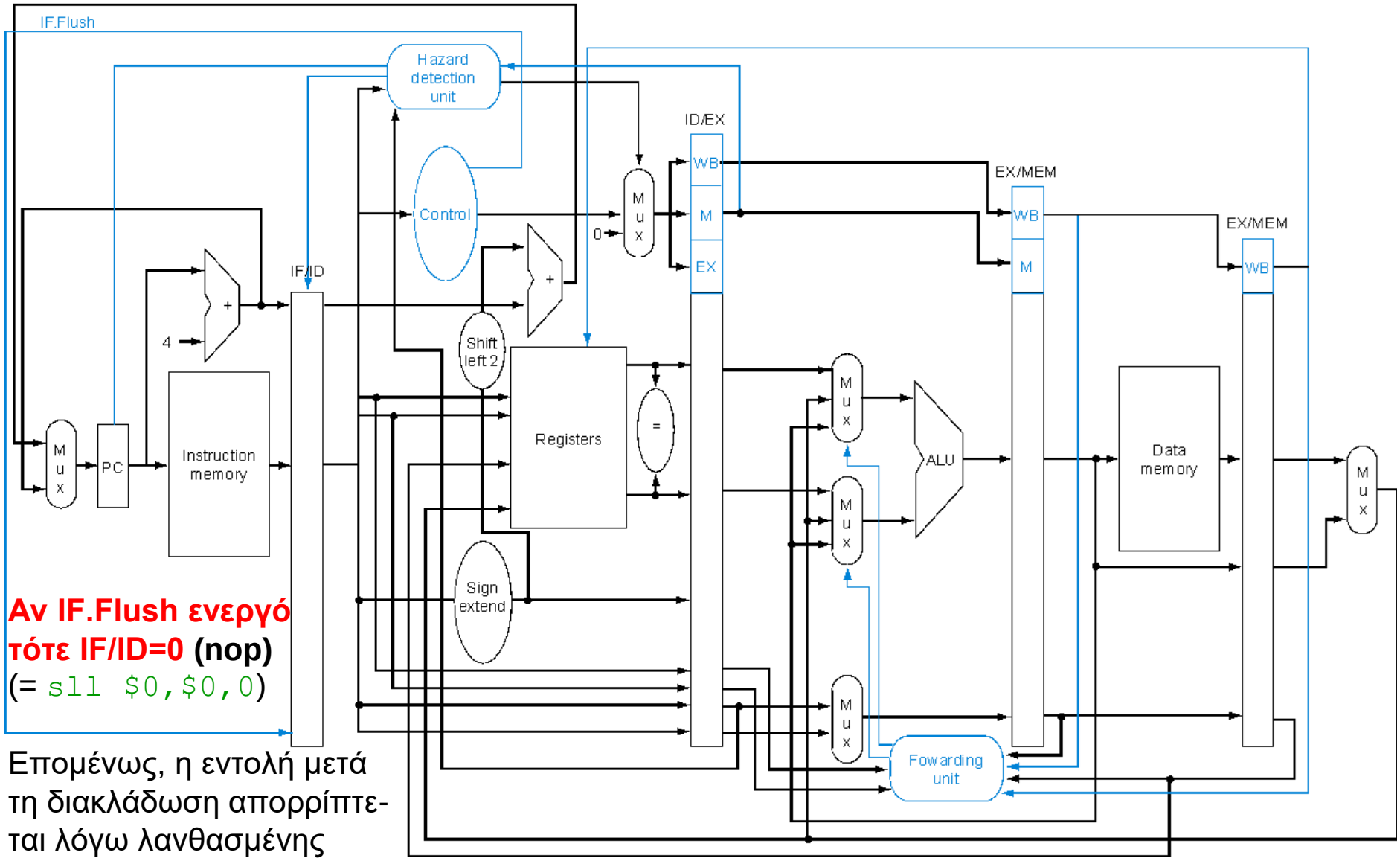


Branch outcome was *originally* resolved in this stage (MEM)



Branch outcome in *now* resolved in this stage (ID)

Απόρριψη Εντολών (*Flushing*)



Αν IF.Flush ενεργό τότε IF/ID=0 (nop)
(= `sll $0,$0,0`)

Επομένως, η εντολή μετά τη διακλάδωση απορρίπτεται λόγω λανθασμένης πρόβλεψης

Reducing Branch Delay

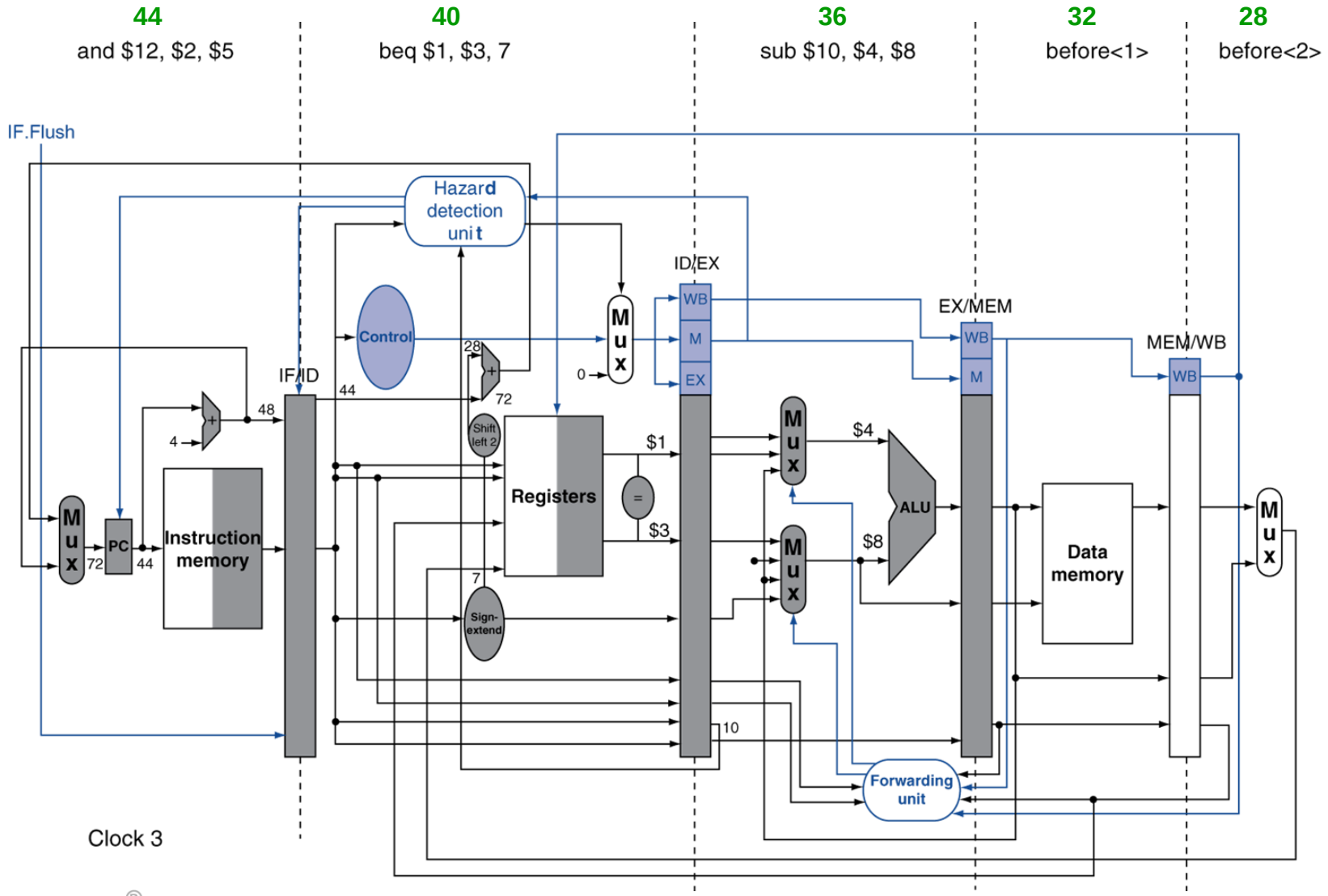
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

Pipeline view on next slide

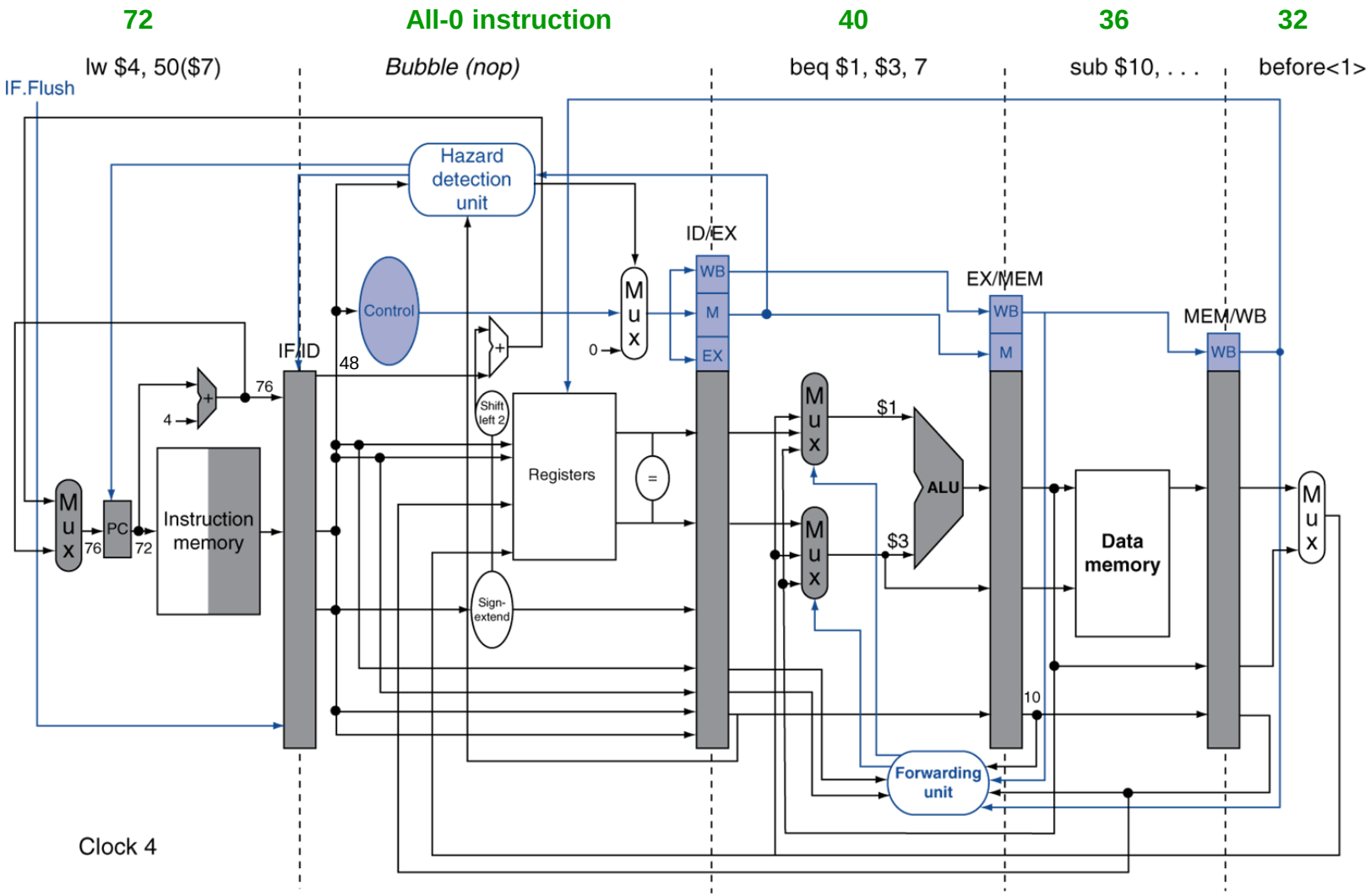
```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
     ...
72:  lw   $4, 50($7)
```

[PC-relative addressing: (40 + 4) + (7 x 4) = 72]

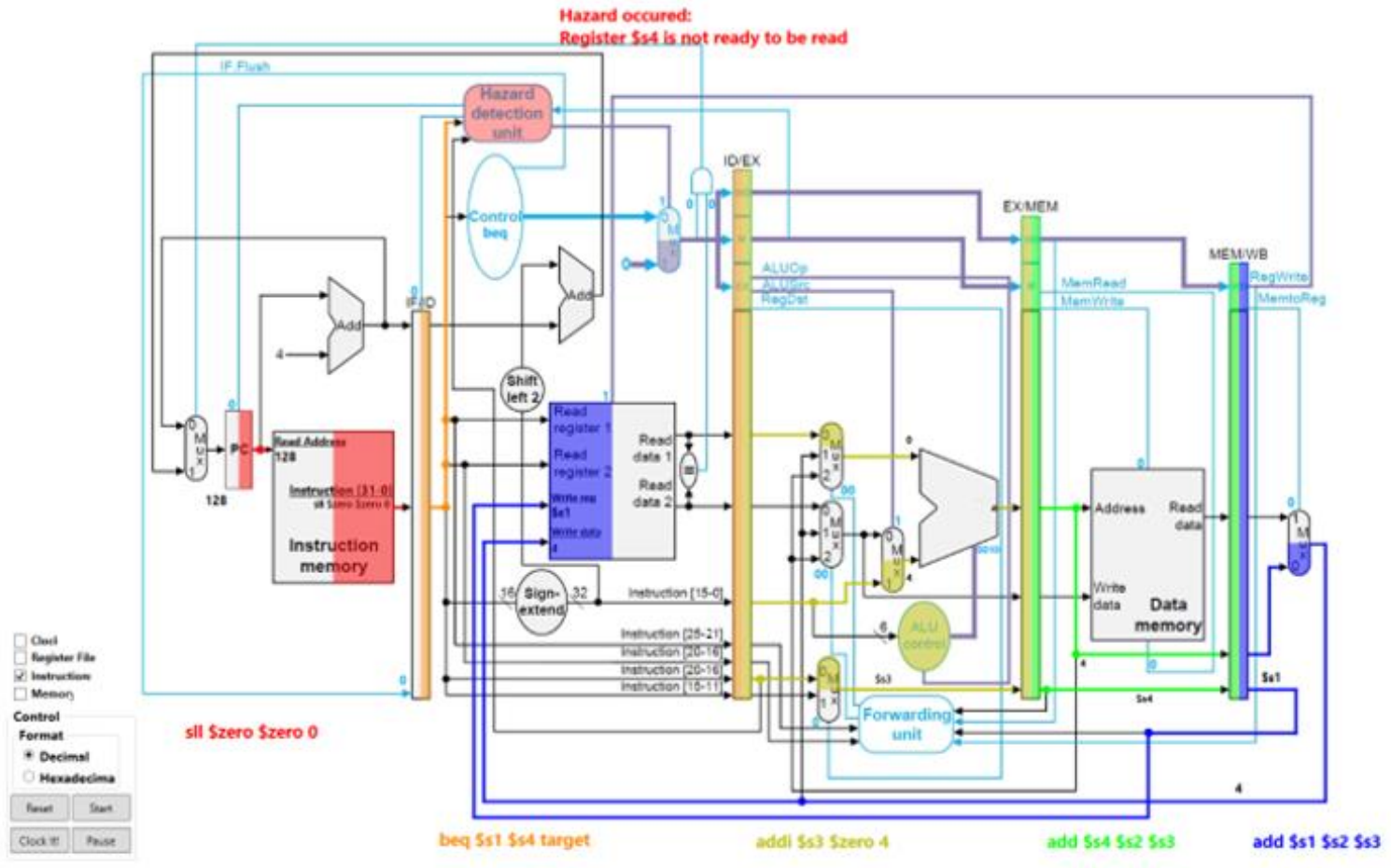
Example: Branch Taken



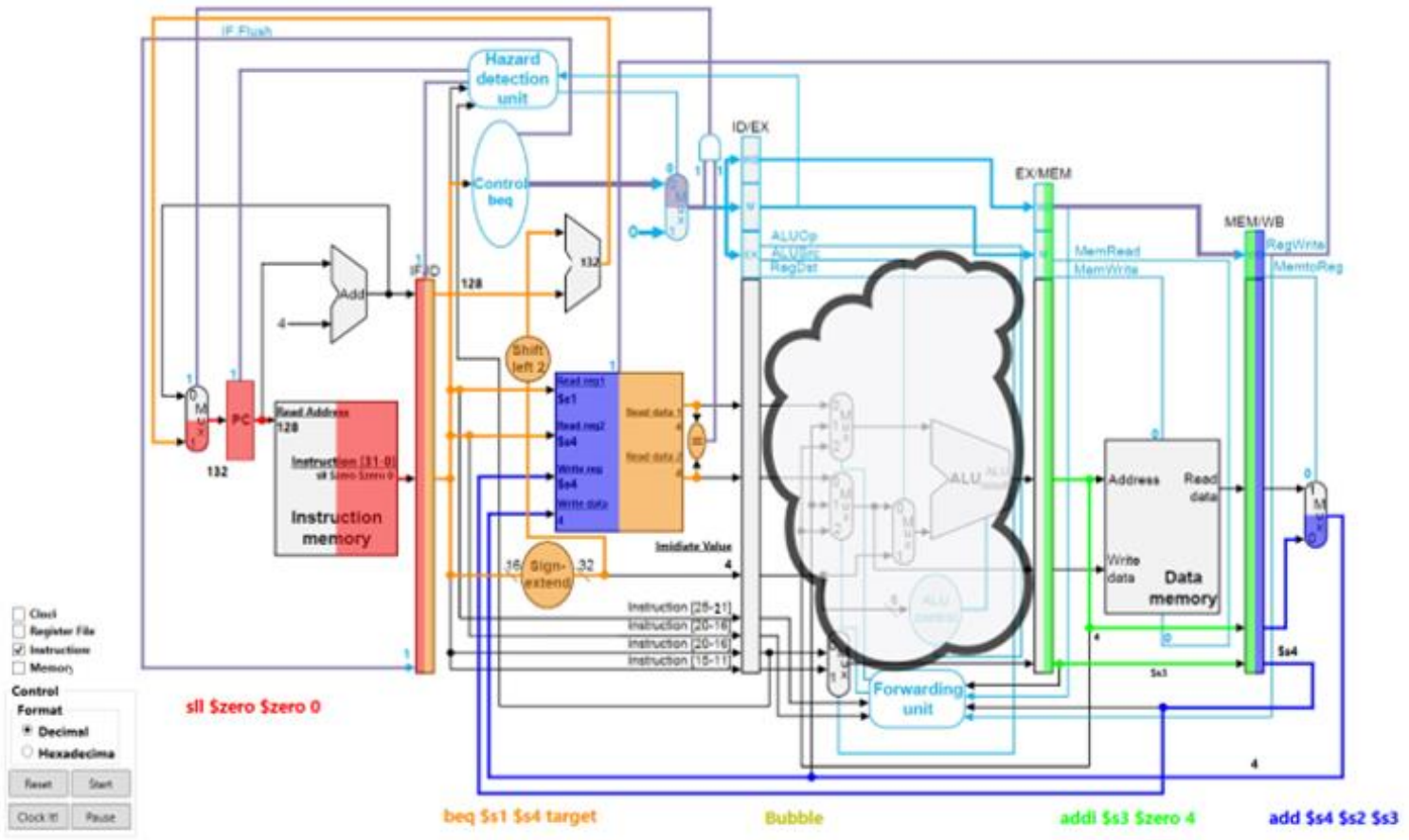
Example: Branch Taken



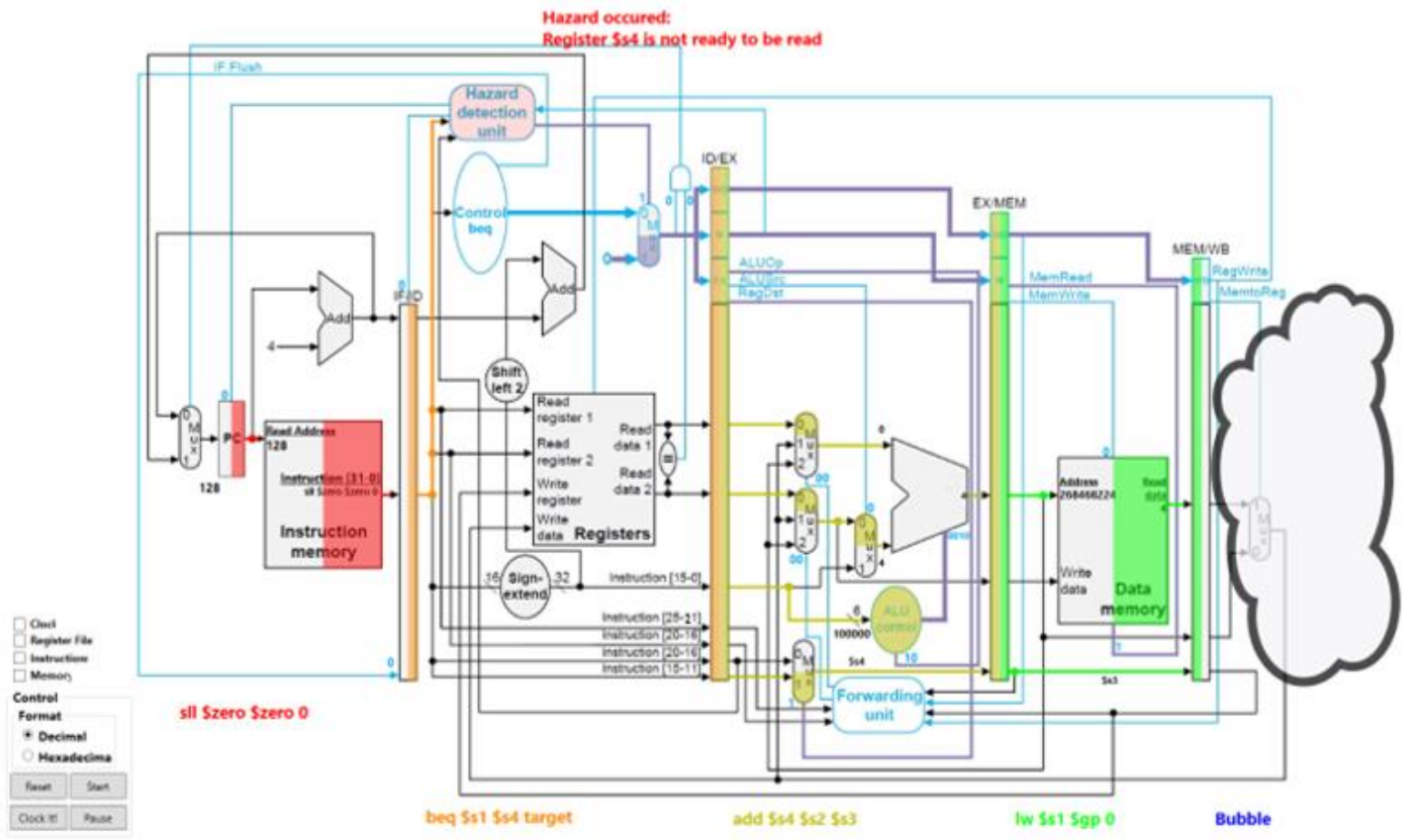
Data Hazards for Branches (Case 1)



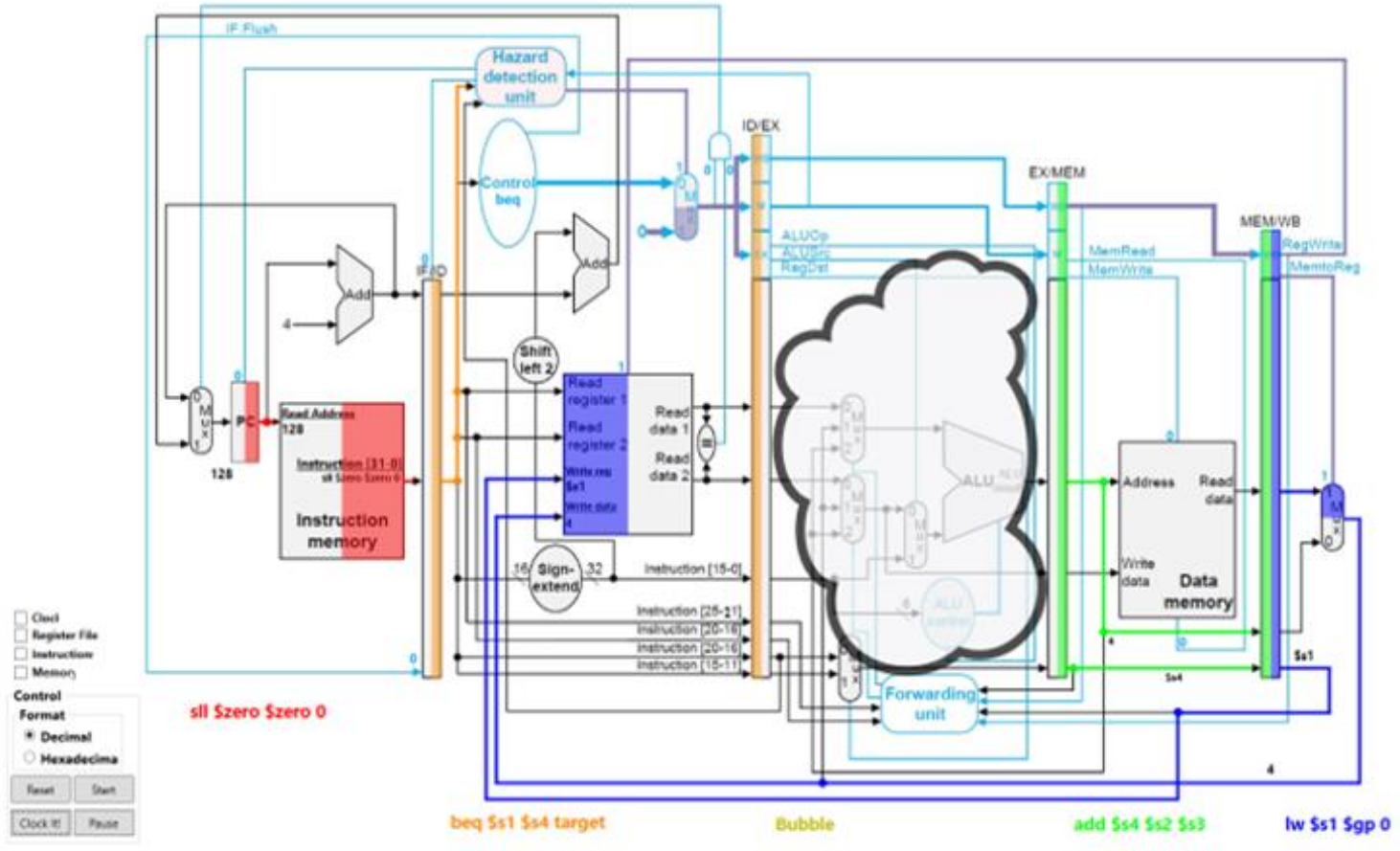
Data Hazards for Branches (Case 1)



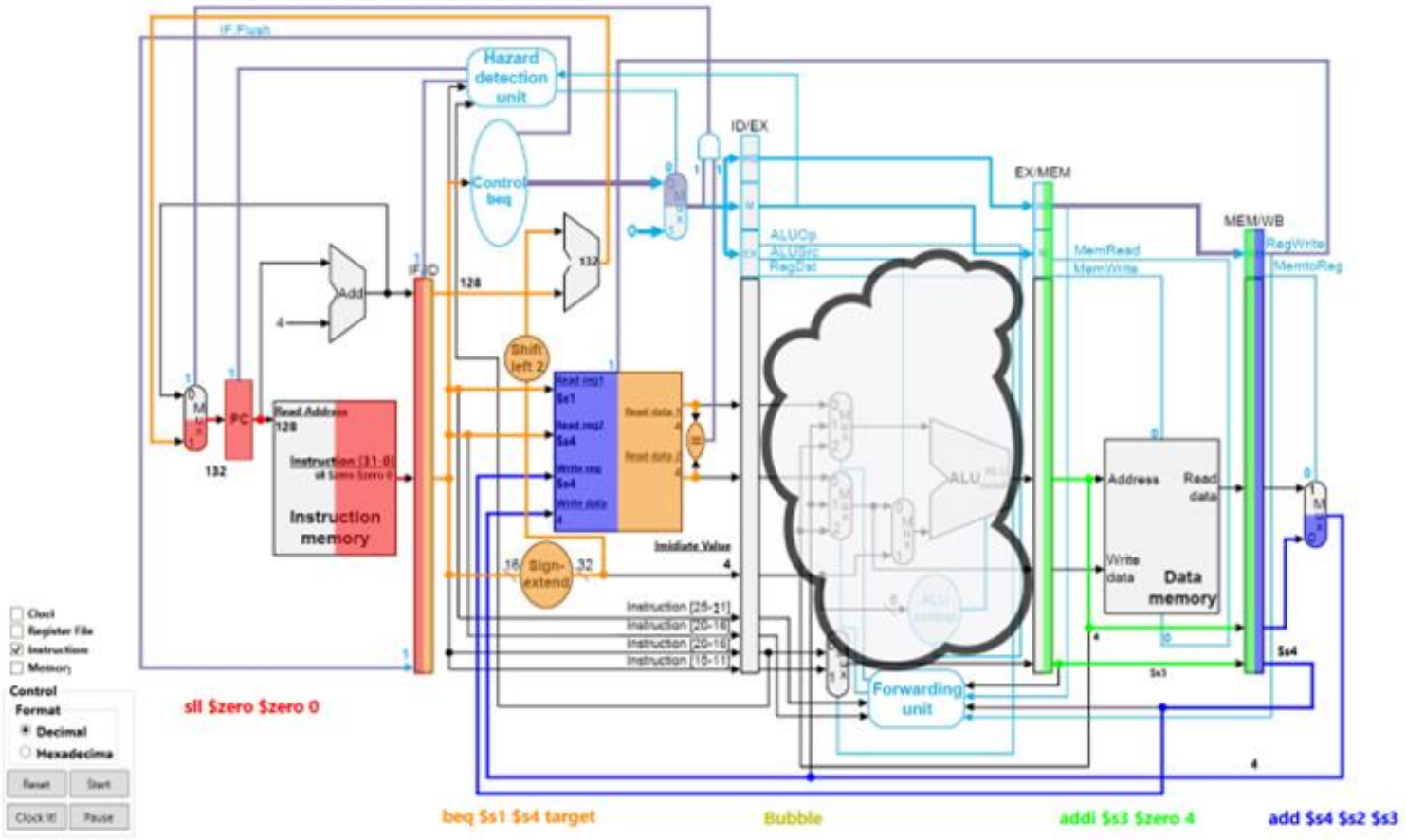
Data Hazards for Branches (Case 2)



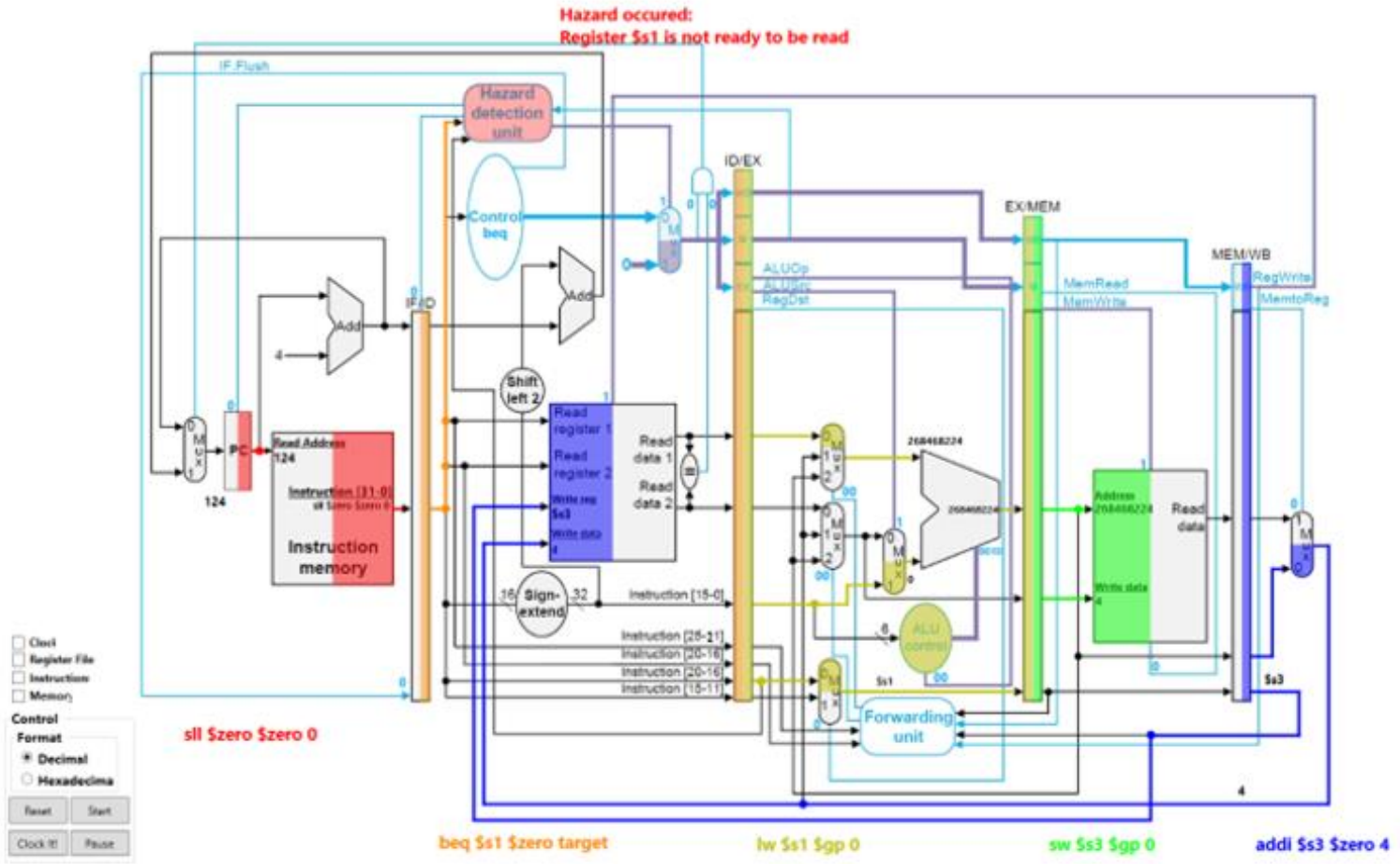
Data Hazards for Branches (Case 2)



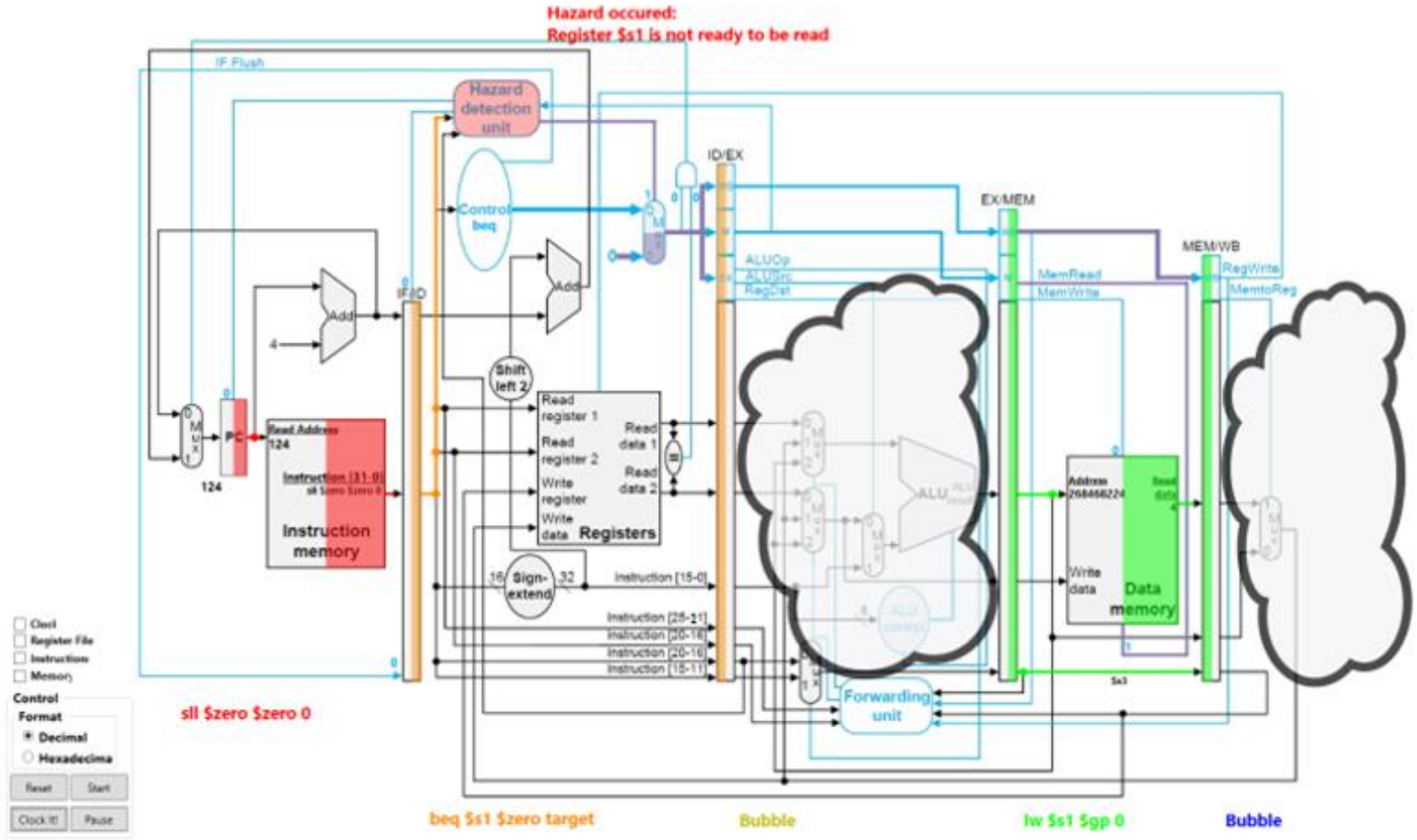
Data Hazards for Branches (Case 2)



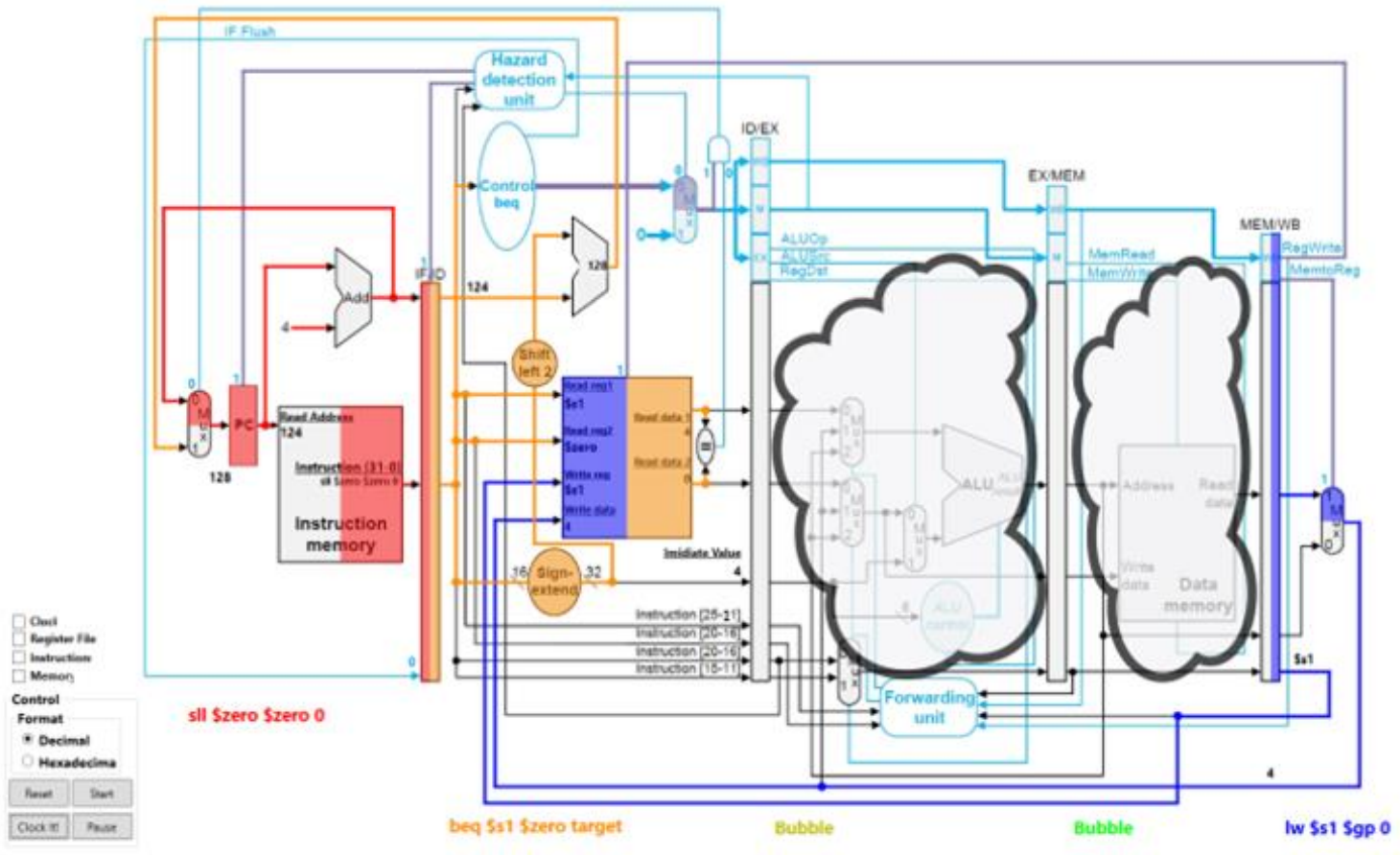
Data Hazards for Branches (Case 3)



Data Hazards for Branches (Case 3)

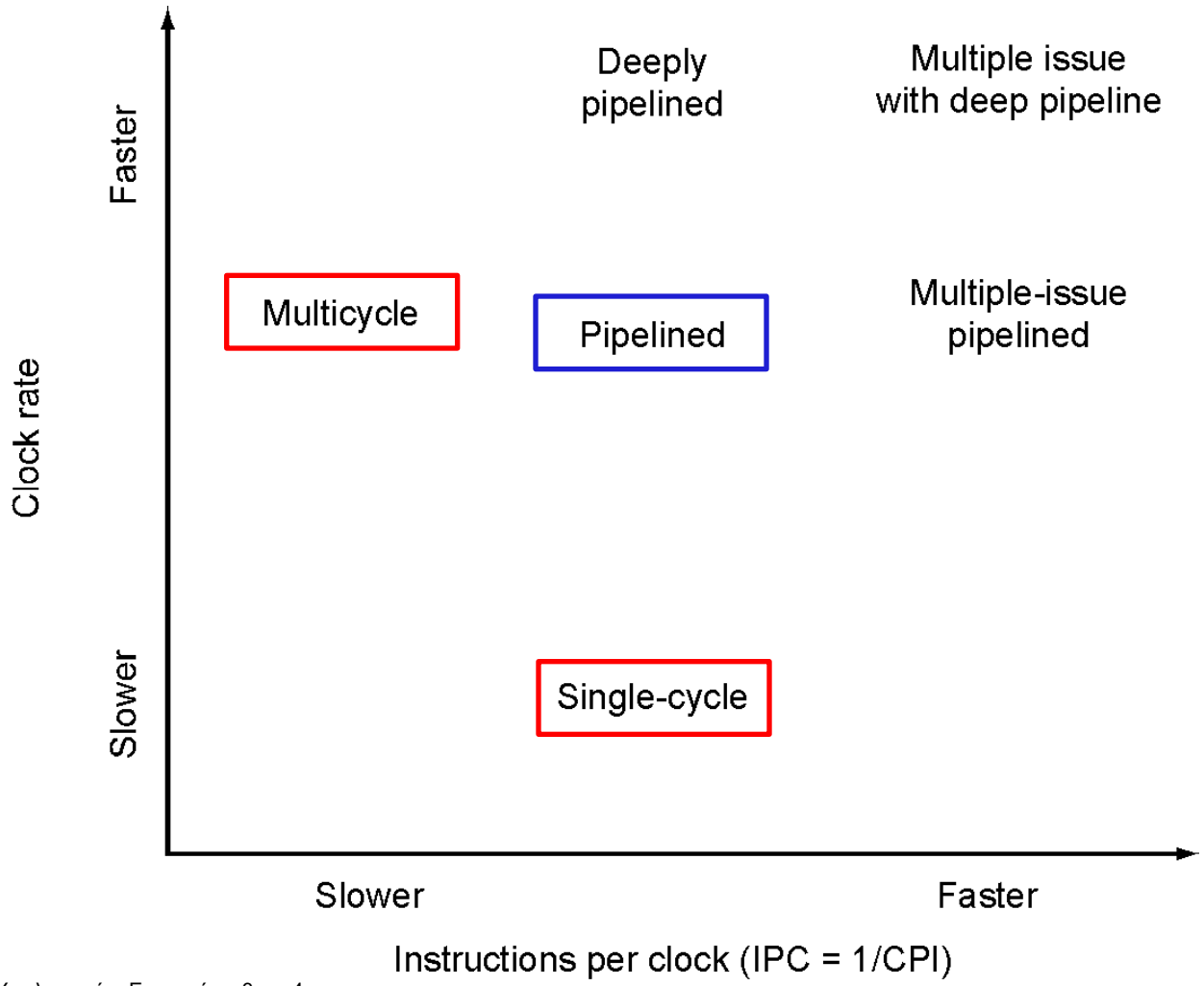


Data Hazards for Branches (Case 3)



Σύνοψη: Σύγκριση Απόδοσης

- Η διασωλήνωση δε βελτιώνει το χρόνο εκτέλεσης, αλλά την ικανότητα διεκπεραίωσης (*throughput*)



Σύνοψη: Εργαστηριακές Ασκήσεις 3 και 4