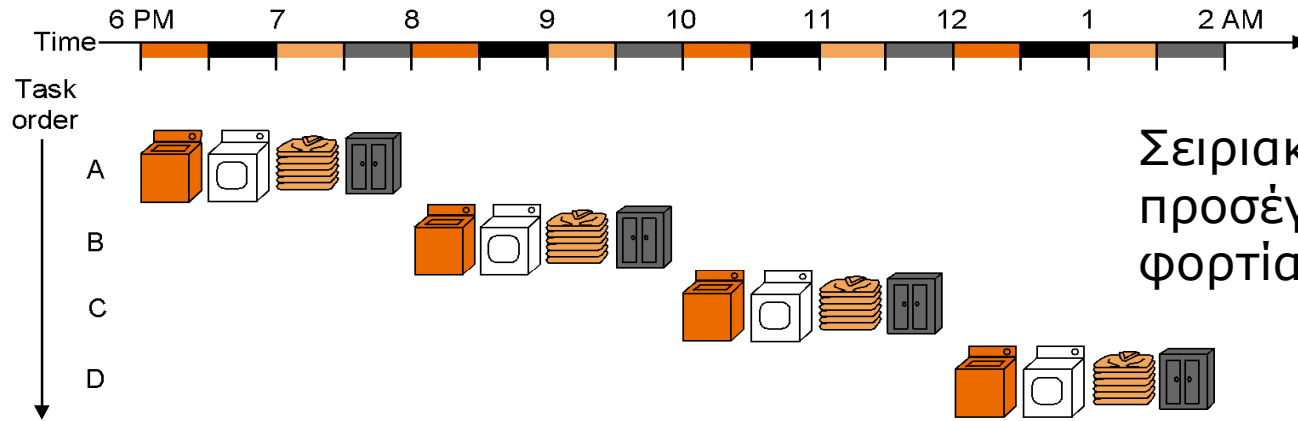
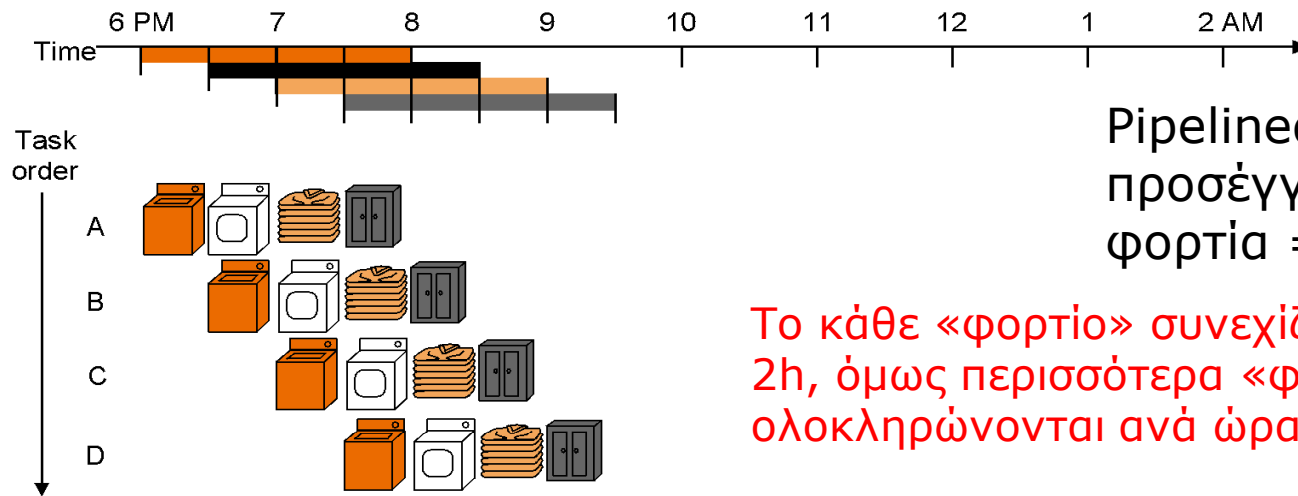


Pipeline: Ένα παράδειγμα από ...τη καθημερινή ζωή

1. Πλυντήριο
2. Στεγνωτήριο
3. Δίπλωμα
4. αποθήκευση



30 min κάθε «φάση»



Το κάθε «φορτίο» συνεχίζει να θέλει 2h, όμως περισσότερα «φορτία» ολοκληρώνονται ανά ώρα

Εντολές MIPS

Πέντε στάδια :

1. Φέρε την εντολή από τη μνήμη (IF-Instruction Fetch)
2. Διάβασε τους καταχωρητές, ενώ αποκωδικοποιείς την εντολή (ID+RegisterFile Read) (στο εξής θα λέμε: ID)
3. Εκτέλεση της εντολής ή υπολογισμός διεύθυνσης (μέσω ALU) (EX-execute)
4. Προσπέλαση μνήμης (MEM)
5. Εγγραφή αποτελέσματος στο RegisterFile (WB-write back)

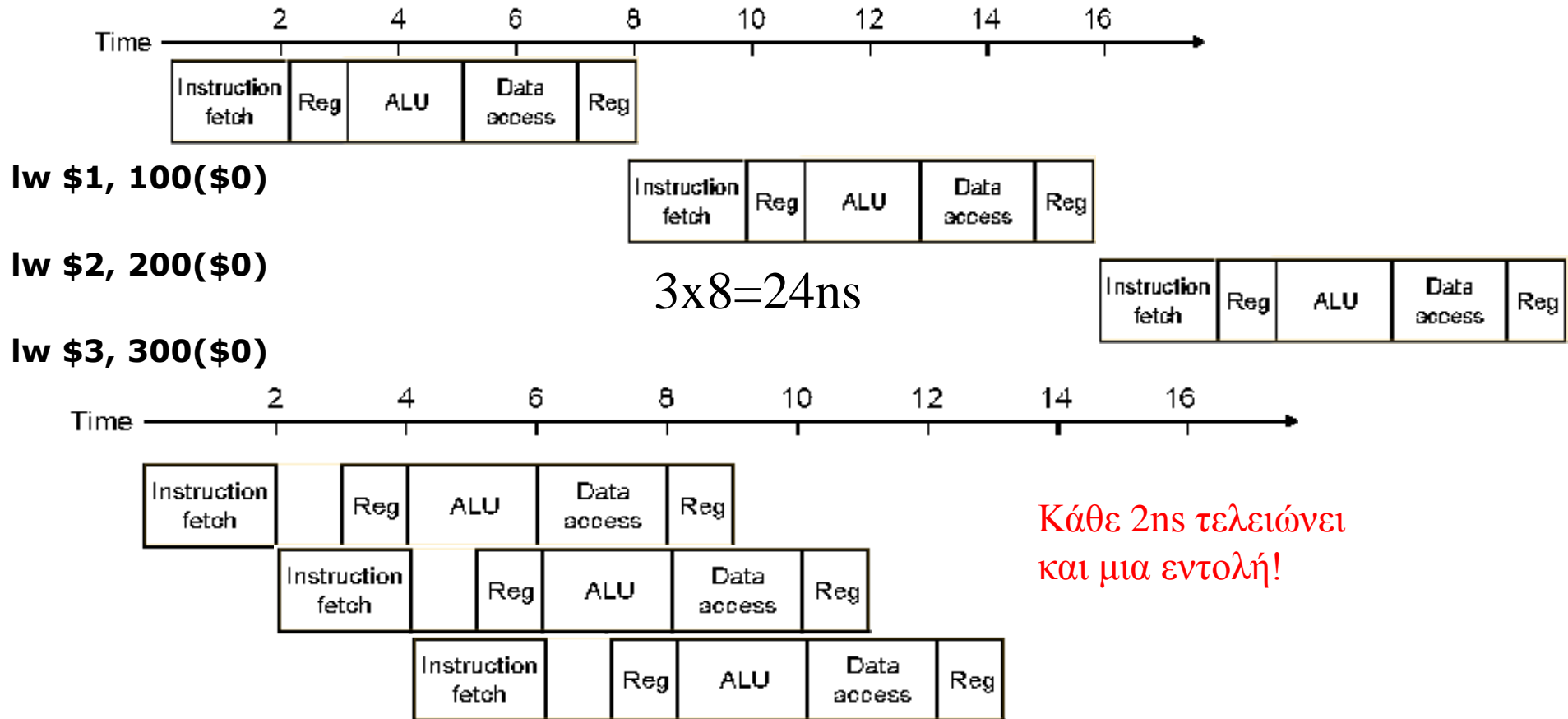
Single-cycle vs pipelined performance:

Single cycle: όλες οι εντολές διαρκούν ένα κύκλο ρολογιού, ίσο με το μήκος της πιο χρονοβόρου εντολής

Έστω: 2 ns για ALU, MEM ανάγνωση ή εγγραφή και 1 ns για register file ανάγνωση ή εγγραφή

Instruction class	Instruct. fetch	Register read	ALU operation	Data access	Register Write	Total Time
Load word (lw)	2ns	1ns	2ns	2ns	1ns	8ns
Store word (sw)	2ns	1ns	2ns	2ns		7ns
R-Type (add,sub,and, or, slt)	2ns	1ns	2ns		1ns	6ns
Branch(beq)	2ns	1ns	2ns			5ns

Έστω οι παρακάτω εντολές lw:



Εδώ οι βαθμίδες δεν είναι απόλυτα ίσες. Στην ιδανική περίπτωση:

$$\text{time}_{\text{pipelined}} = \text{time}_{\text{non_pipelined}} / \text{number of pipe stages}$$

Στη single cycle υλοποίηση, η εντολή διαρκεί ένα κύκλο ίσο με την πιο χρονοβόρα (εδω: 8 ns)

Στη pipeline υλοποίηση, το ρολόι κάθε φάσης (στάδιο-pipeline stage) διαρκεί (2 ns), ακόμα και αν υπάρχουν στάδια του 1ns

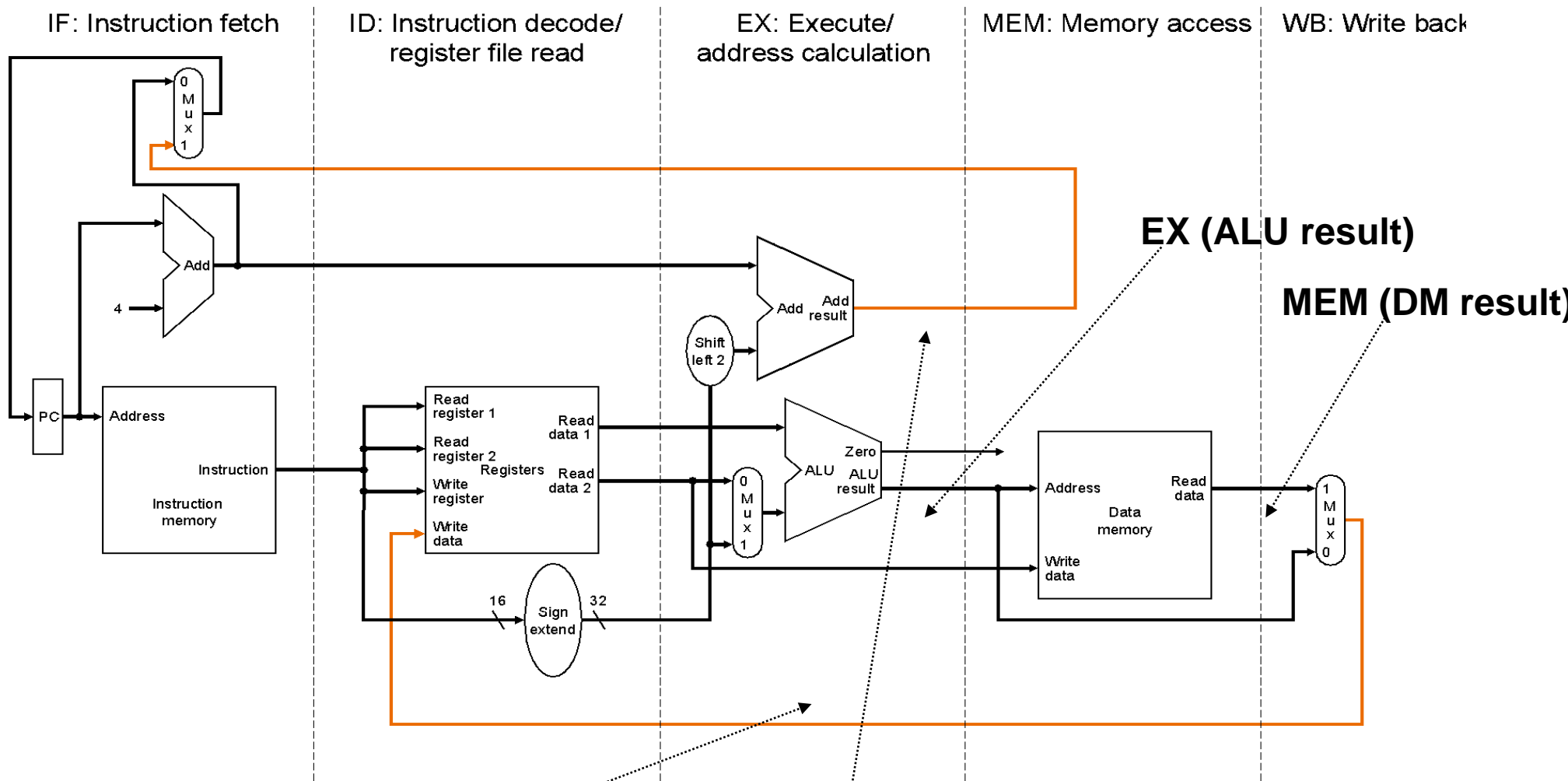
Στο προηγούμενο παράδειγμα 14 ns για pipeline, 24ns για single cycle, άρα 1,71 επιτάχυνση.

Αν είχαμε 1000 εντολές ακόμα: pipeline $1000 \times 2\text{ns} + 14\text{ ns} = 2014\text{ ns}$

Single cycle $1000 \times 8\text{ns} + 24\text{ ns} = 8024\text{ ns}$

Άρα επιτάχυνση: $8024/2014=3,98 \sim 4$ (8ns/2ns ratio μεταξύ εντολών)

Στάδια διόδου δεδομένων ενός κύκλου (single cycle datapath):



Ροή εκτέλεσης εντολών-δεδομένων: από αριστερά προς τα δεξιά

Εξαιρέση?

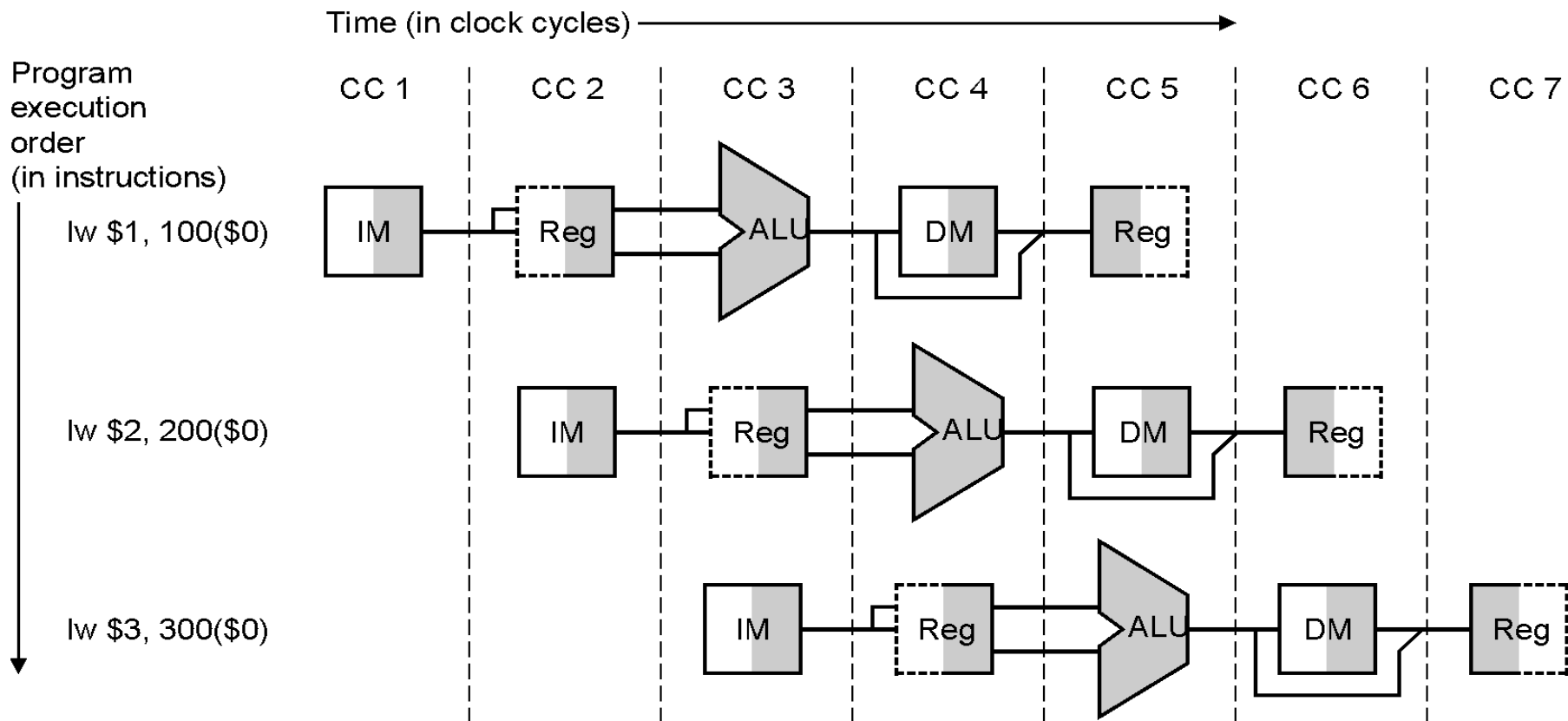
Write-back

και επιλογή νέου PC

κίνδυνος ελέγχου (control hazard)

κίνδυνος δεδομένων (data hazard)

Εκτέλεση αγωγού (pipelined execution)

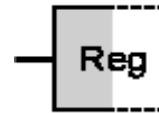


Τι θα γίνει αν χρησιμοποιούμε την ίδια λειτουργική μονάδα (Functional Unit), π.χ. IM, RegFile, ALU, DM σε **διαφορετικούς** κύκλους για **διαφορετικές** εντολές?

Μόνο για RegFile:

Μπορούμε να διαβάσουμε και να γράψουμε το RegFile στον ίδιο κύκλο: (θα μας βοηθήσει σε αποφυγή κινδύνων» hazards)

Στο πρώτο μισό του κύκλου γράφουμε (σκιασμένο αριστερά)



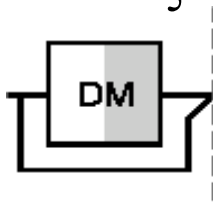
και στο δεύτερο μισό διαβάζουμε (σκιασμένο δεξιά)



κύκλος: γράφουμε-διαβάζουμε

Γενικά για Functional Units:

Όταν ένα functional unit ή ένας pipeline register είναι σκιασμένο σημαίνει ότι χρησιμοποιείται για ανάγνωση (σκιασμένο δεξιά) ή εγγραφή (σκιασμένο αριστερά)



Στην υλοποίηση του multicycle datapath, είχαμε την **ίδια** μονάδα να χρησιμοποιείται από την **ίδια** εντολή σε **διαφορετικούς** κύκλους, π.χ. ALU ή MEM

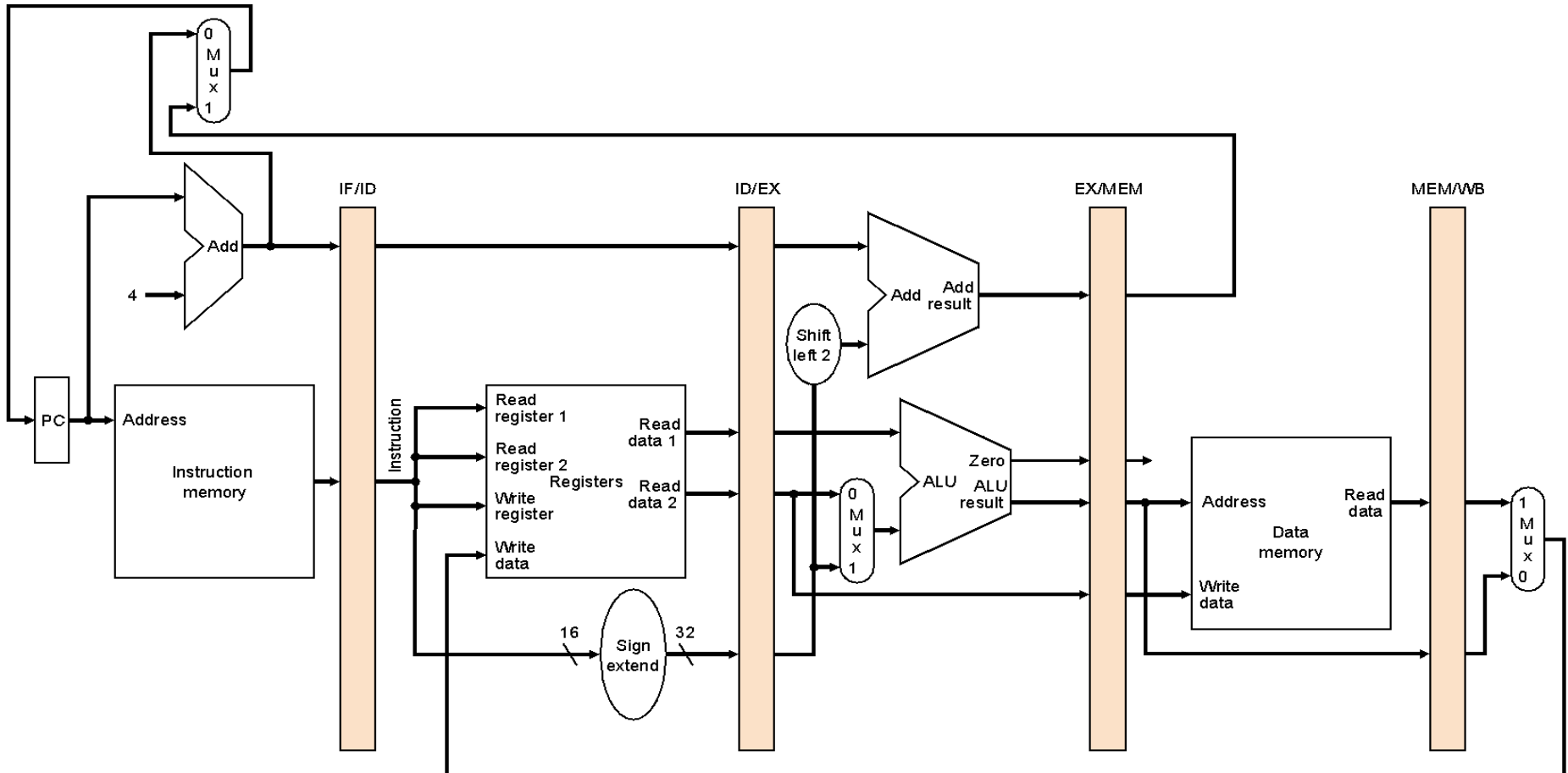
Στην pipelined υλοποίηση του datapath, έχουμε την **ίδια** μονάδα να χρησιμοποιείται από **διαφορετικές (διαδοχικές)** εντολές σε **διαφορετικούς (διαδοχικούς)** κύκλους

Πώς διασφαλίζεται **ορθότητα εκτέλεσης** κάθε εντολής;

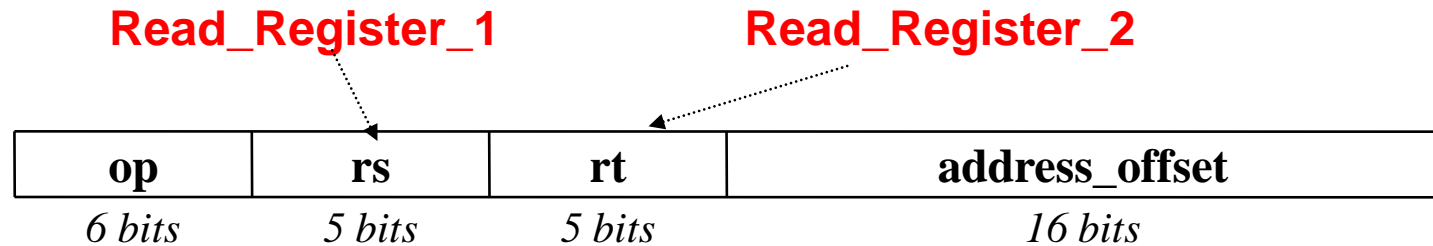
Καταχωρητές **κατάλληλου;;;** μεγέθους ανάμεσα σε διαδοχικά στάδια (pipeline stages)

Pipelined εκδοχή του single cycle datapath

(προσθέσαμε τους καταχωρητές-pipeline registers ανάμεσα σε διαδοχικά στάδια)

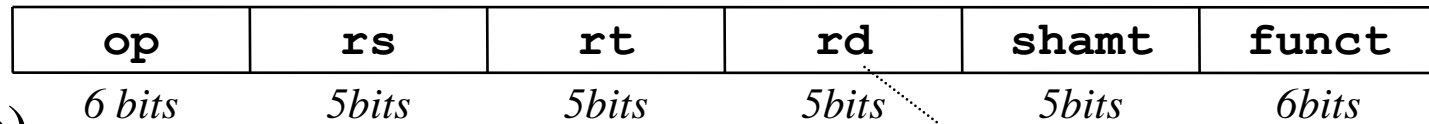


I-Type:



`lw $rt, address_offset($rs)`

R-Type: (register type)



`add $rd, $rs, $rt`

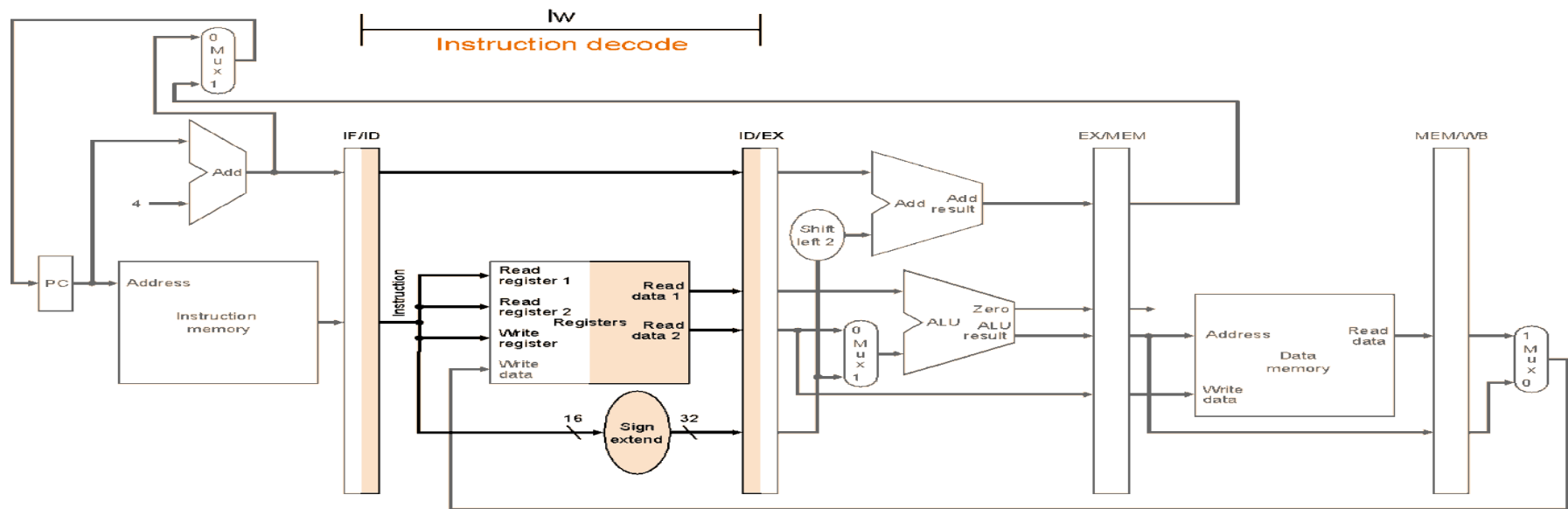
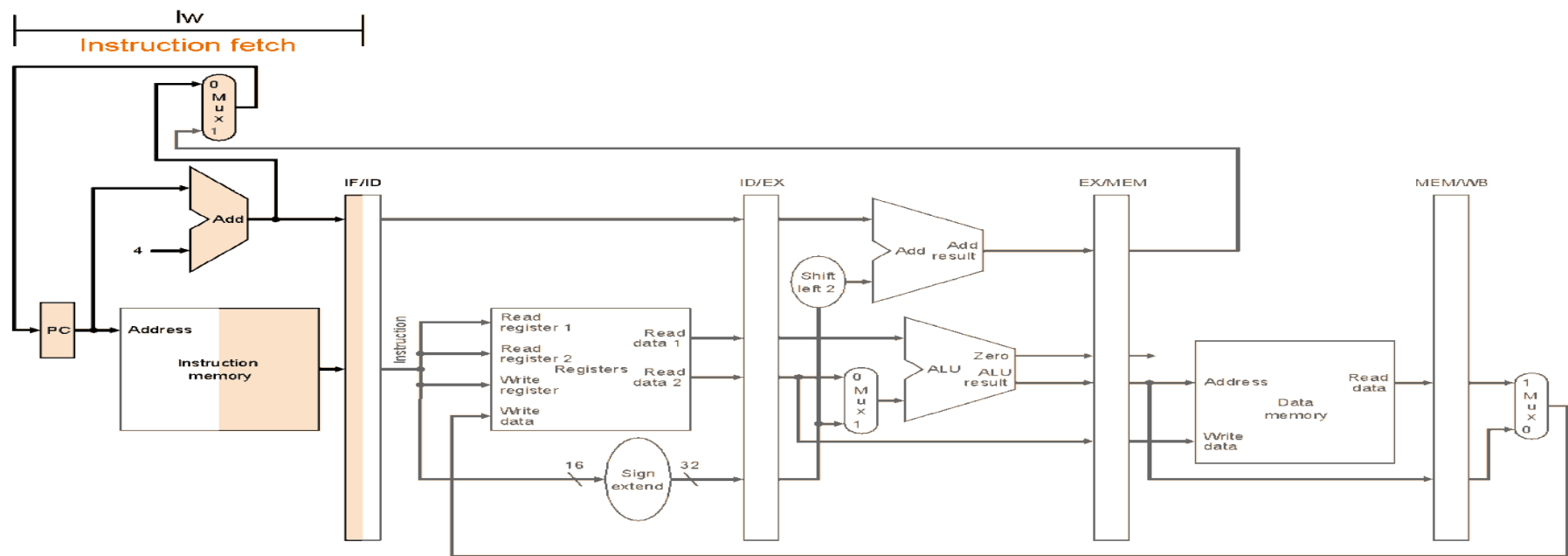
Op: opcode

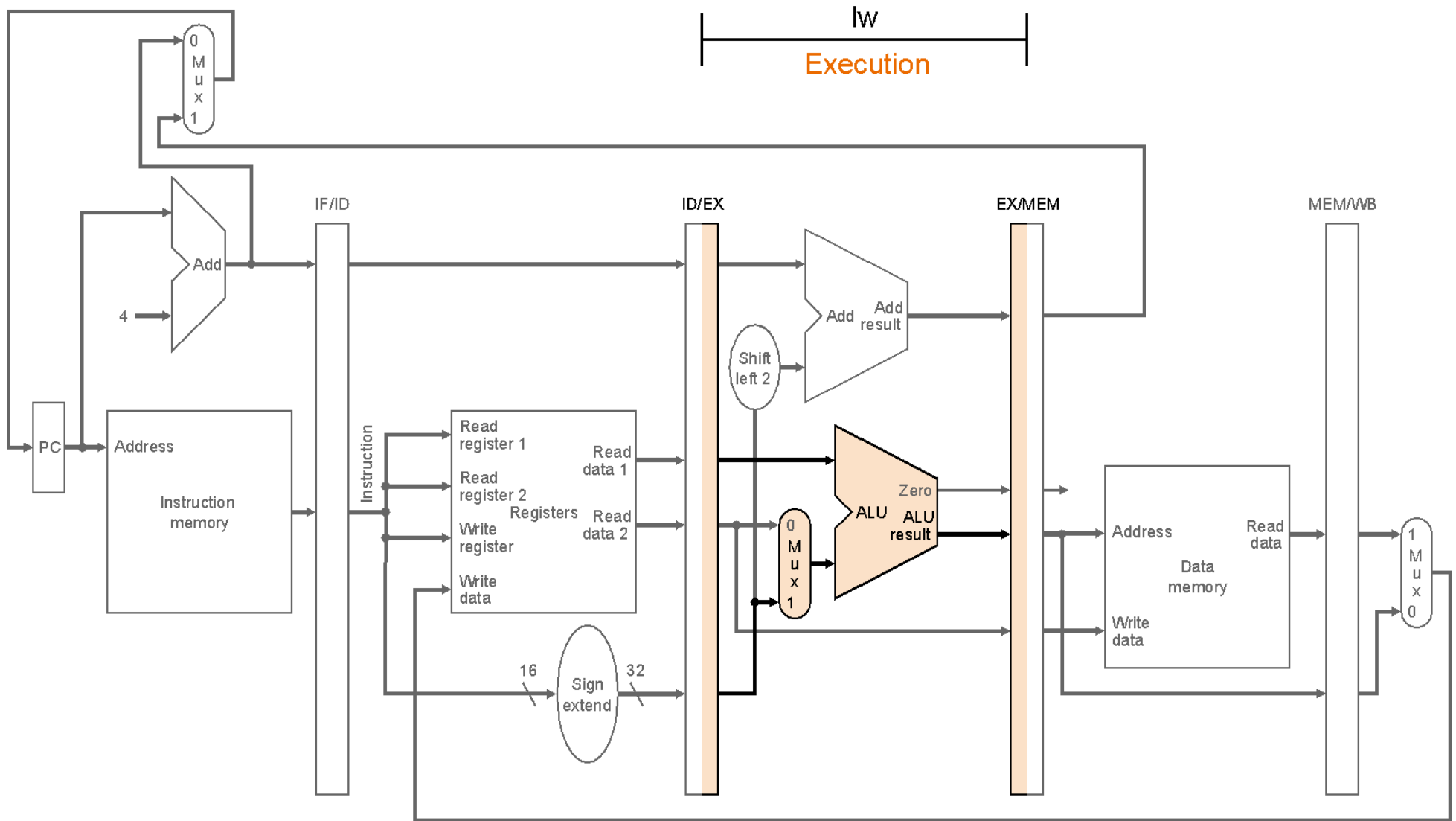
rs, rt: register source operands

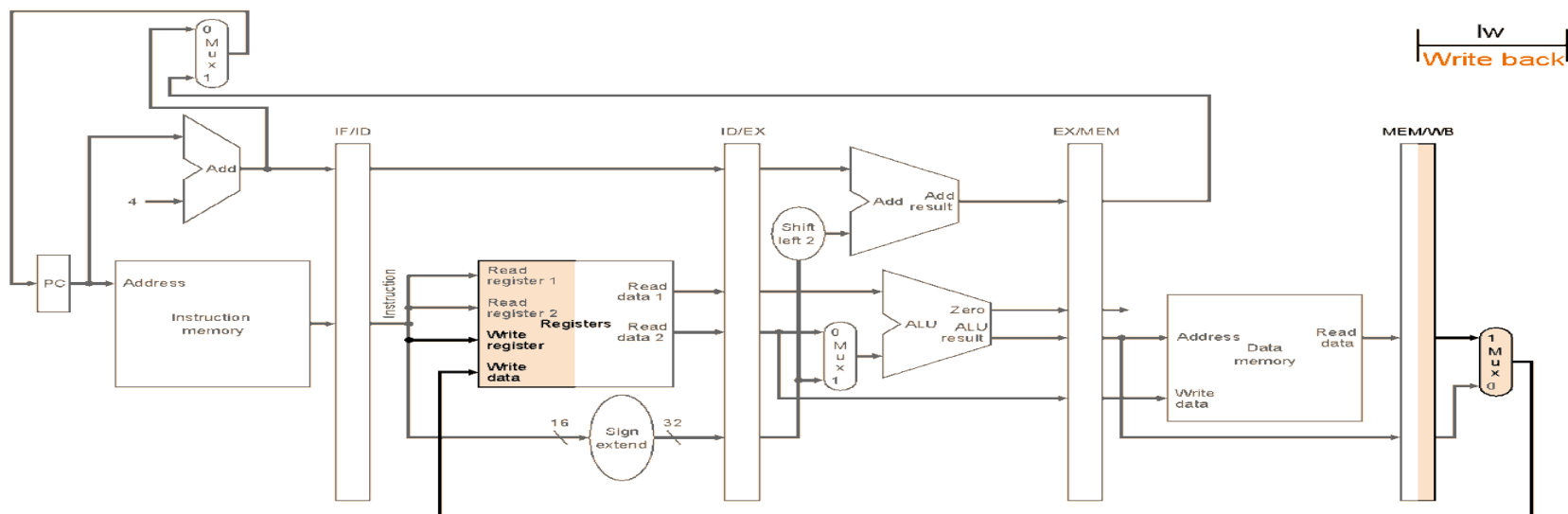
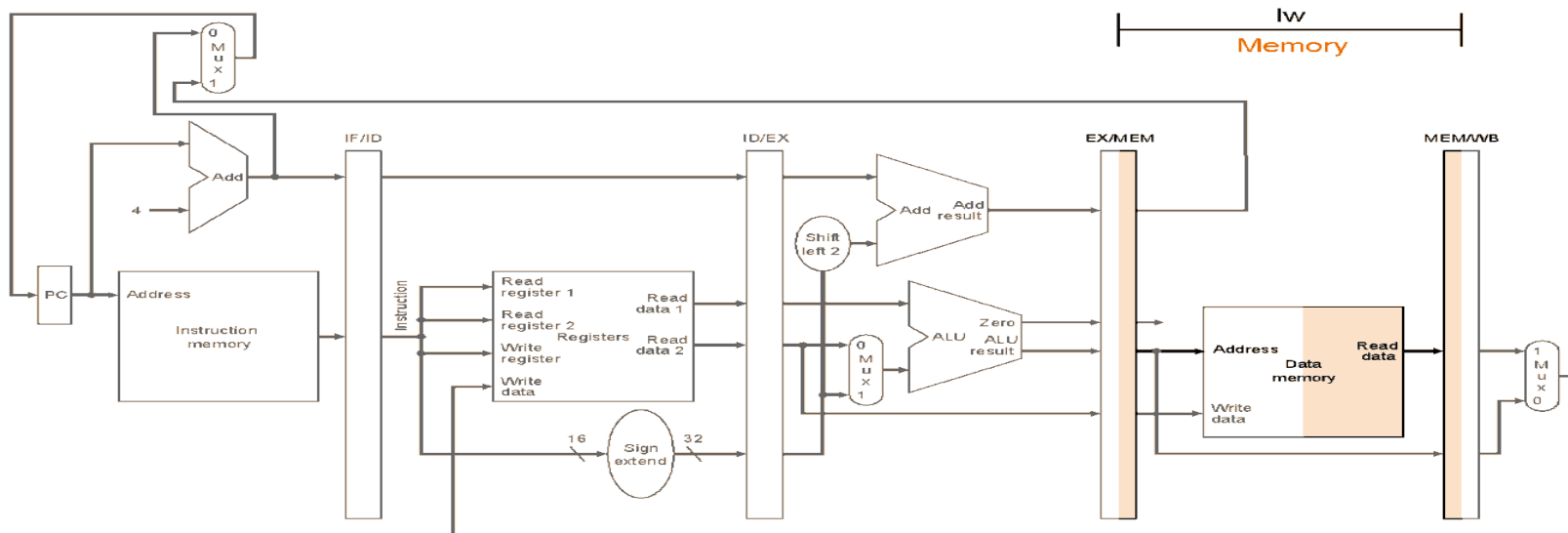
Rd: register destination operand

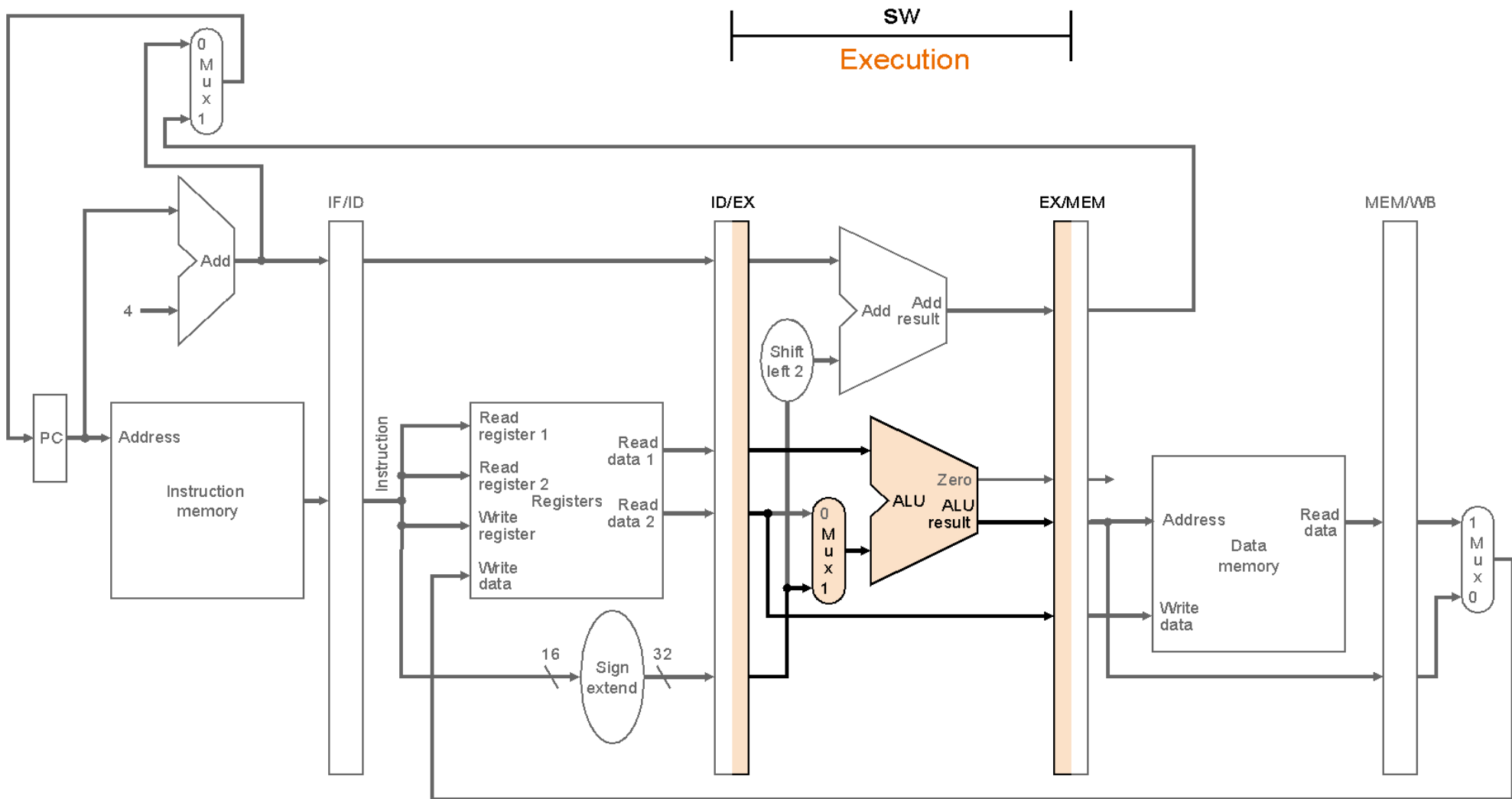
Shamt: shift amount

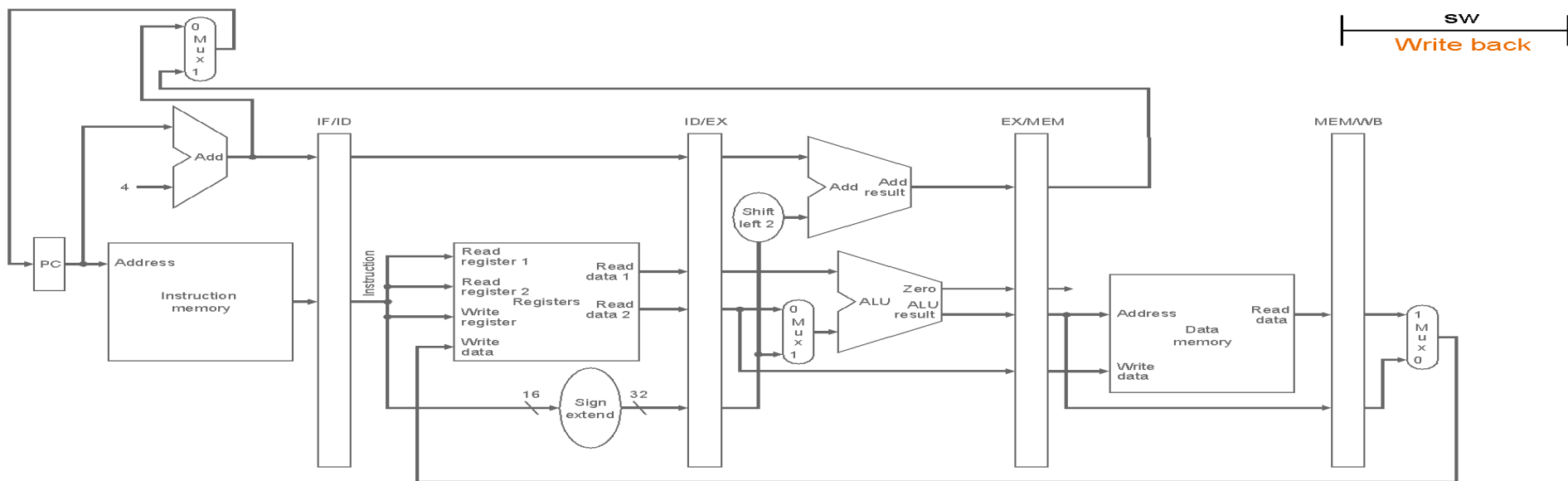
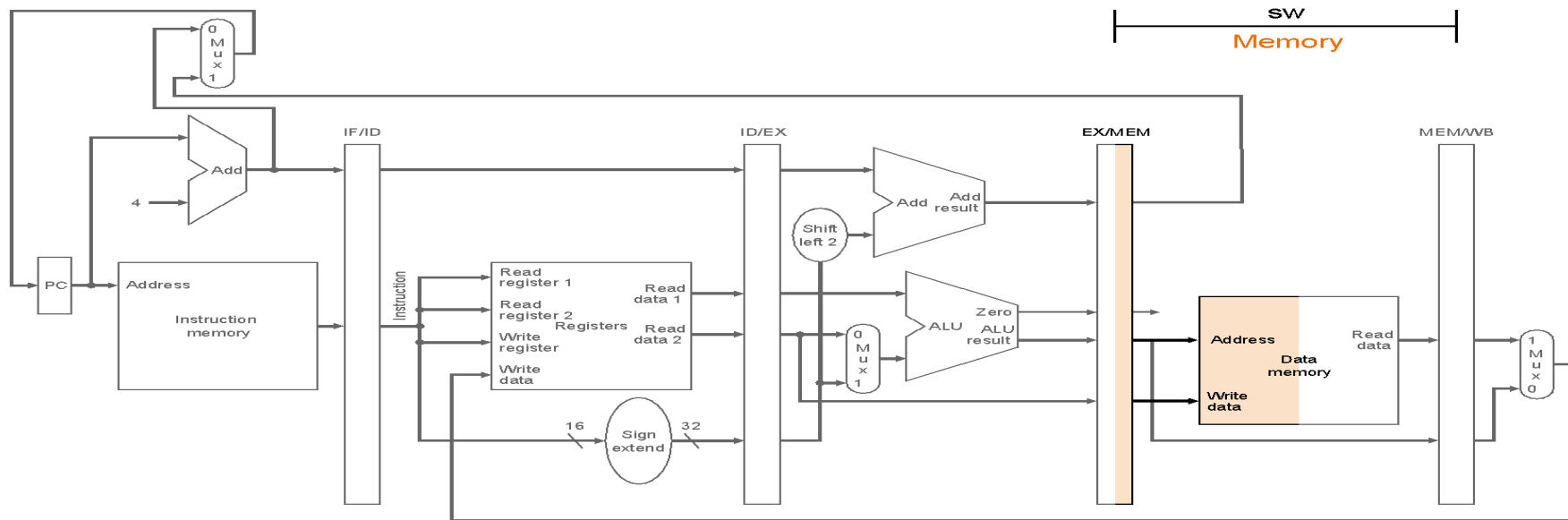
Funct : op specific (function code)



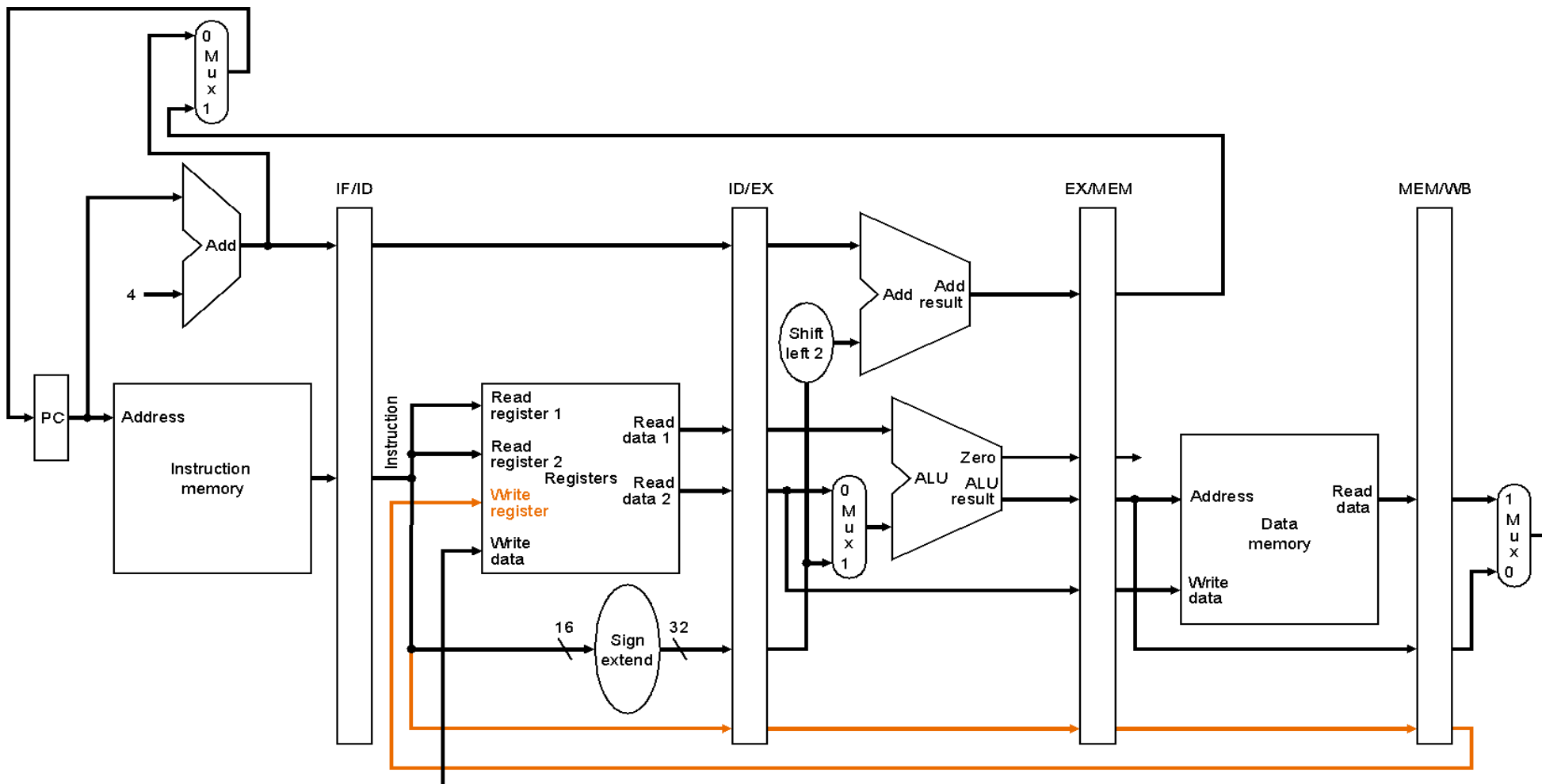






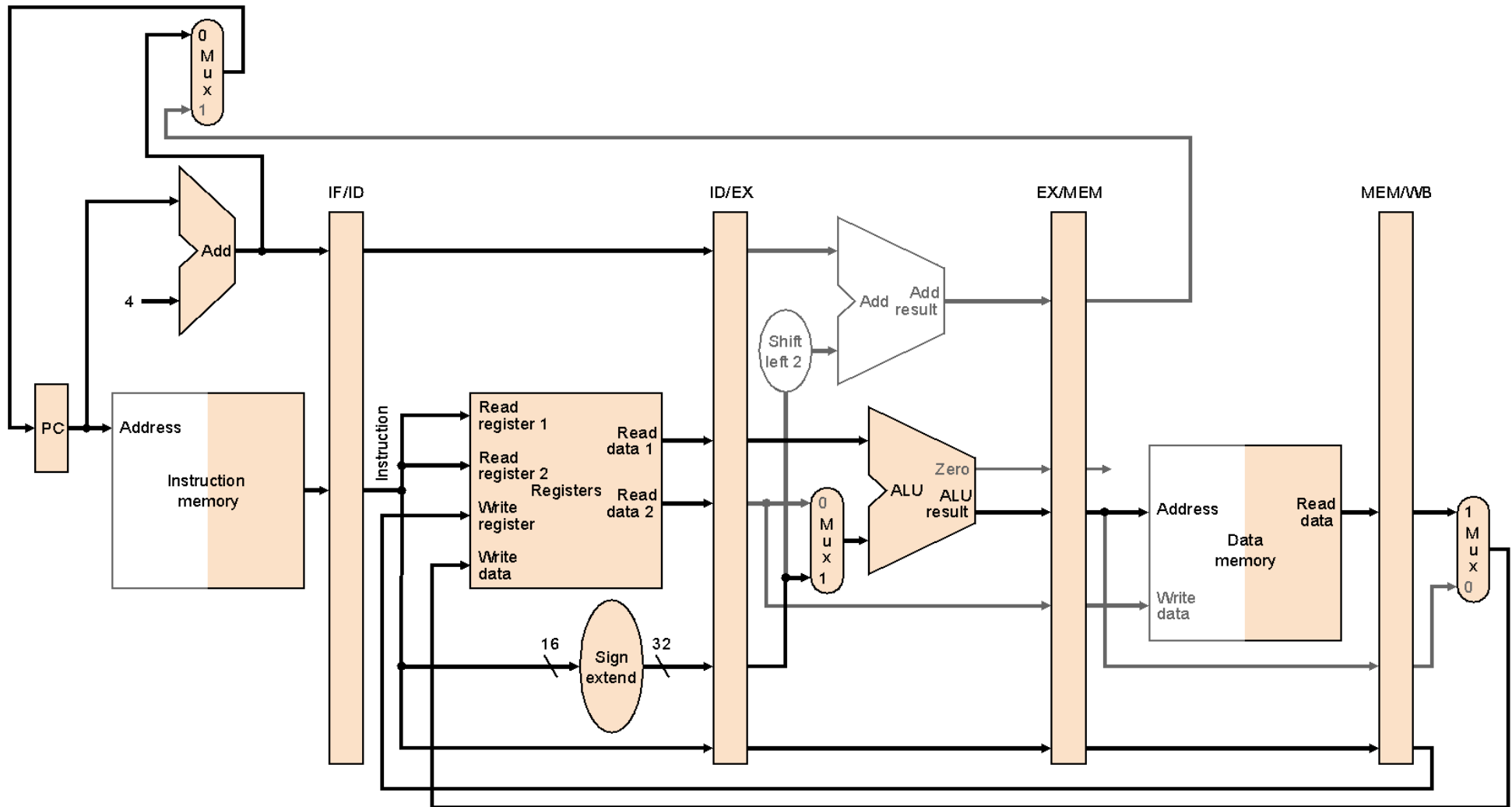


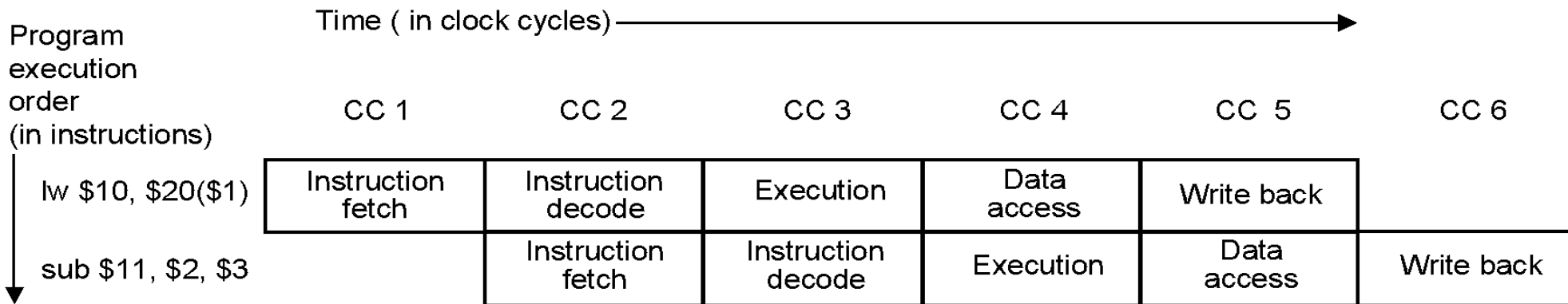
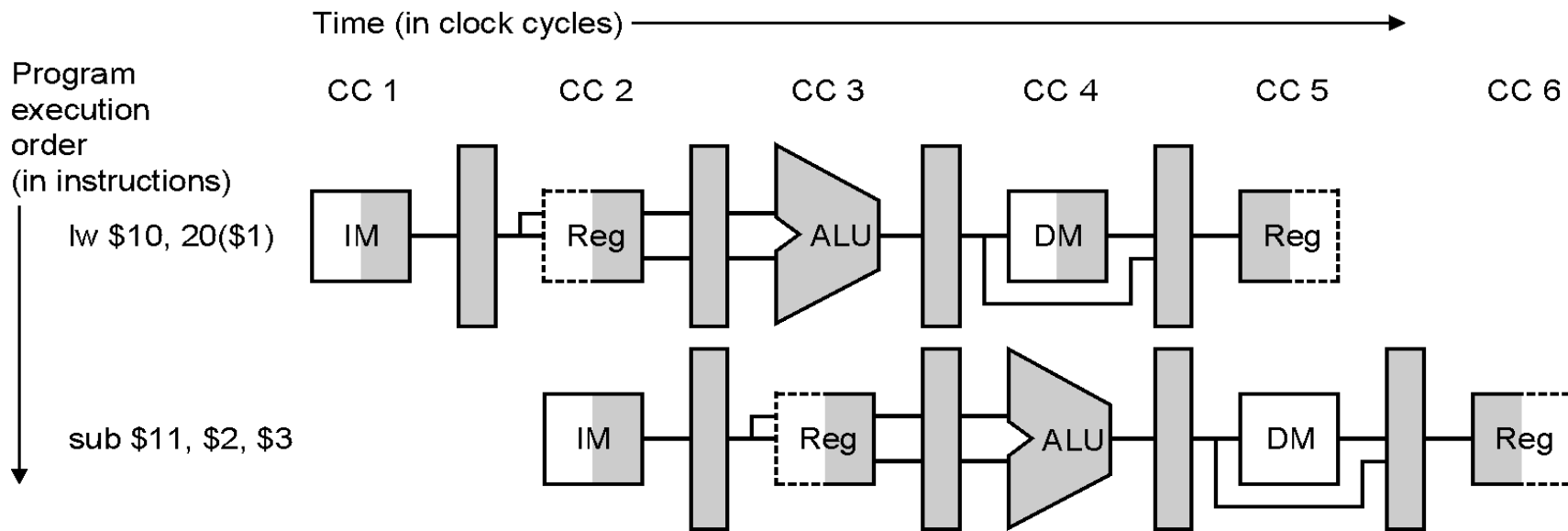
Το διορθωμένο pipeline για την lw:

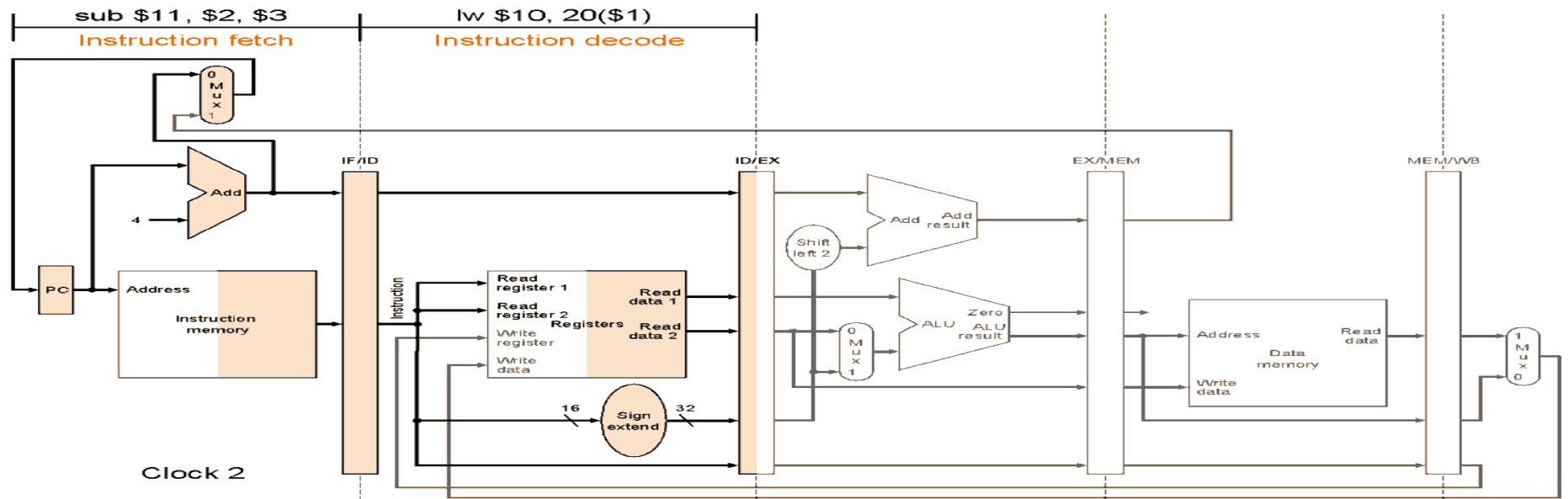
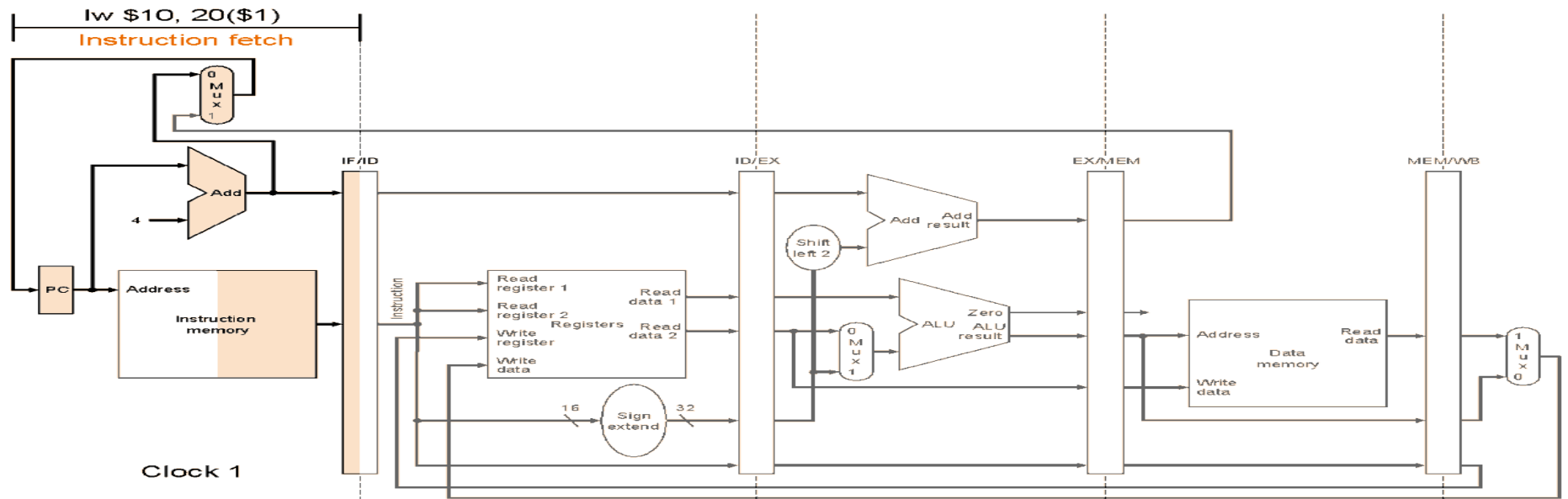


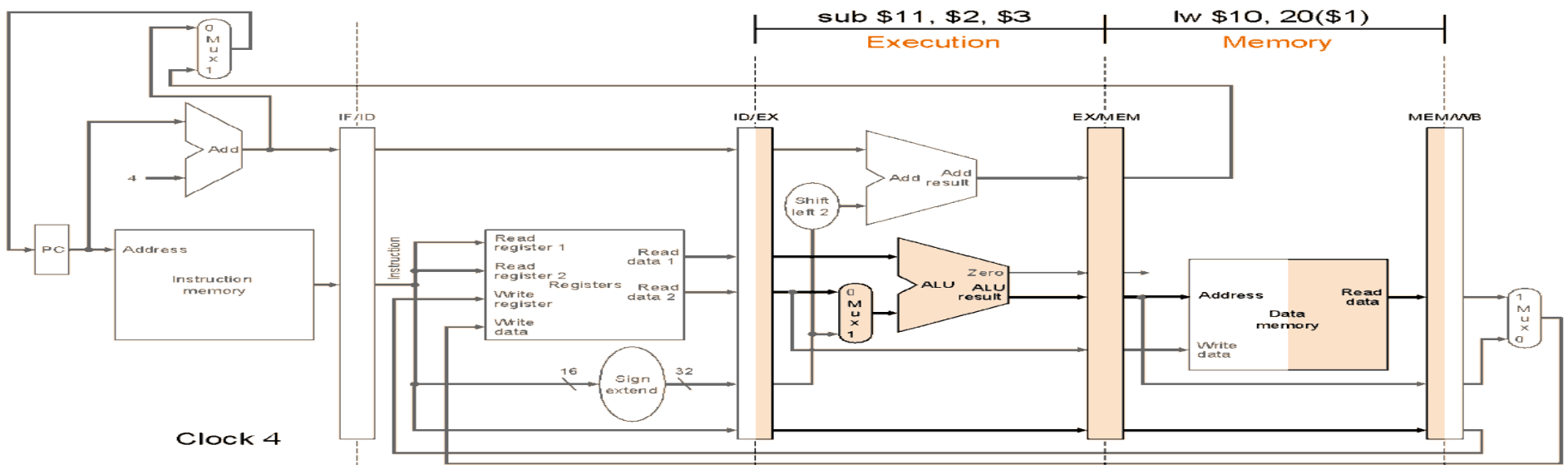
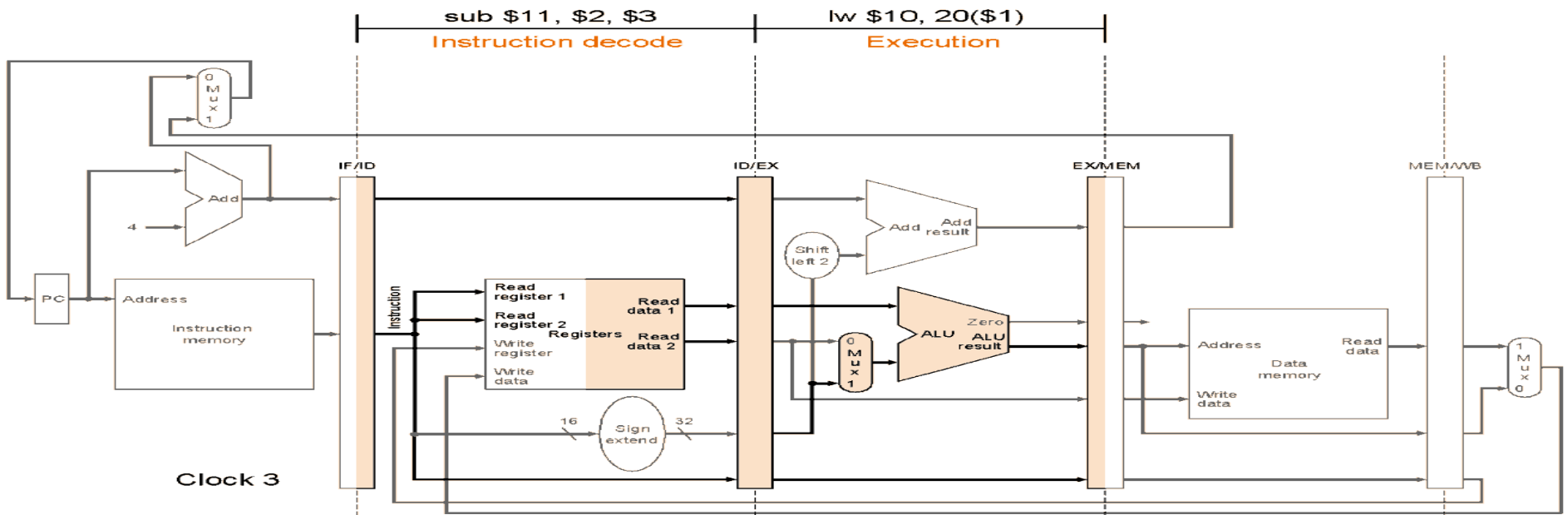
Ο αριθμός του write register έρχεται και αυτός μέσα από το pipeline τη σωστή στιγμή

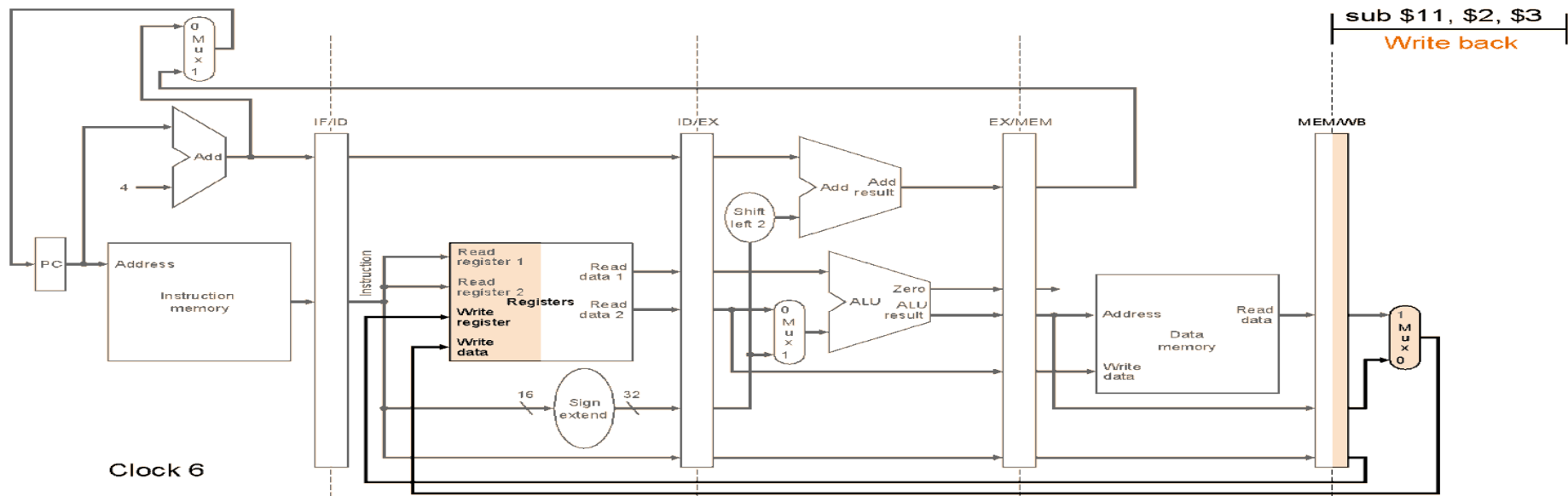
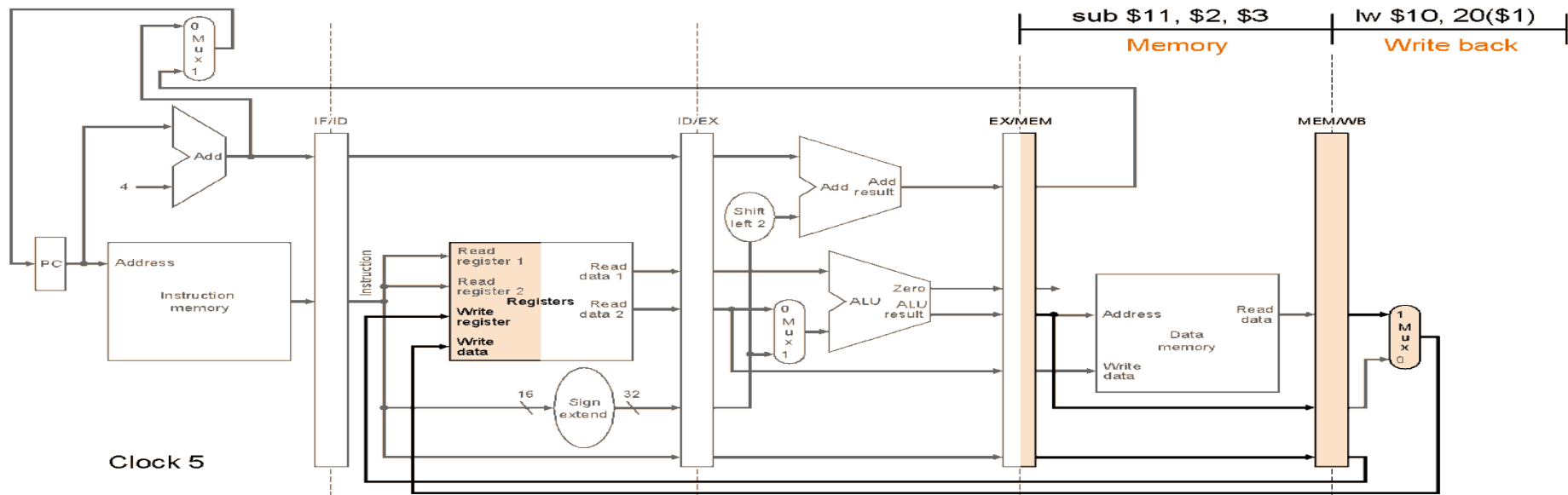
Τα τμήματα του datapath που χρησιμοποιήθηκαν κατά την εκτέλεση της lw:











Κίνδυνοι Σωλήνωσης (Pipeline Hazards)

- Δομικοί Κίνδυνοι (structural hazards)

Το υλικό δεν μπορεί να υποστηρίξει το συνδυασμό των εντολών που θέλουμε να εκτελέσουμε στον ίδιο κύκλο μηχανής. (π.χ. ενιαία L1 \$ για I & D)

- Κίνδυνοι Ελέγχου (control hazards)

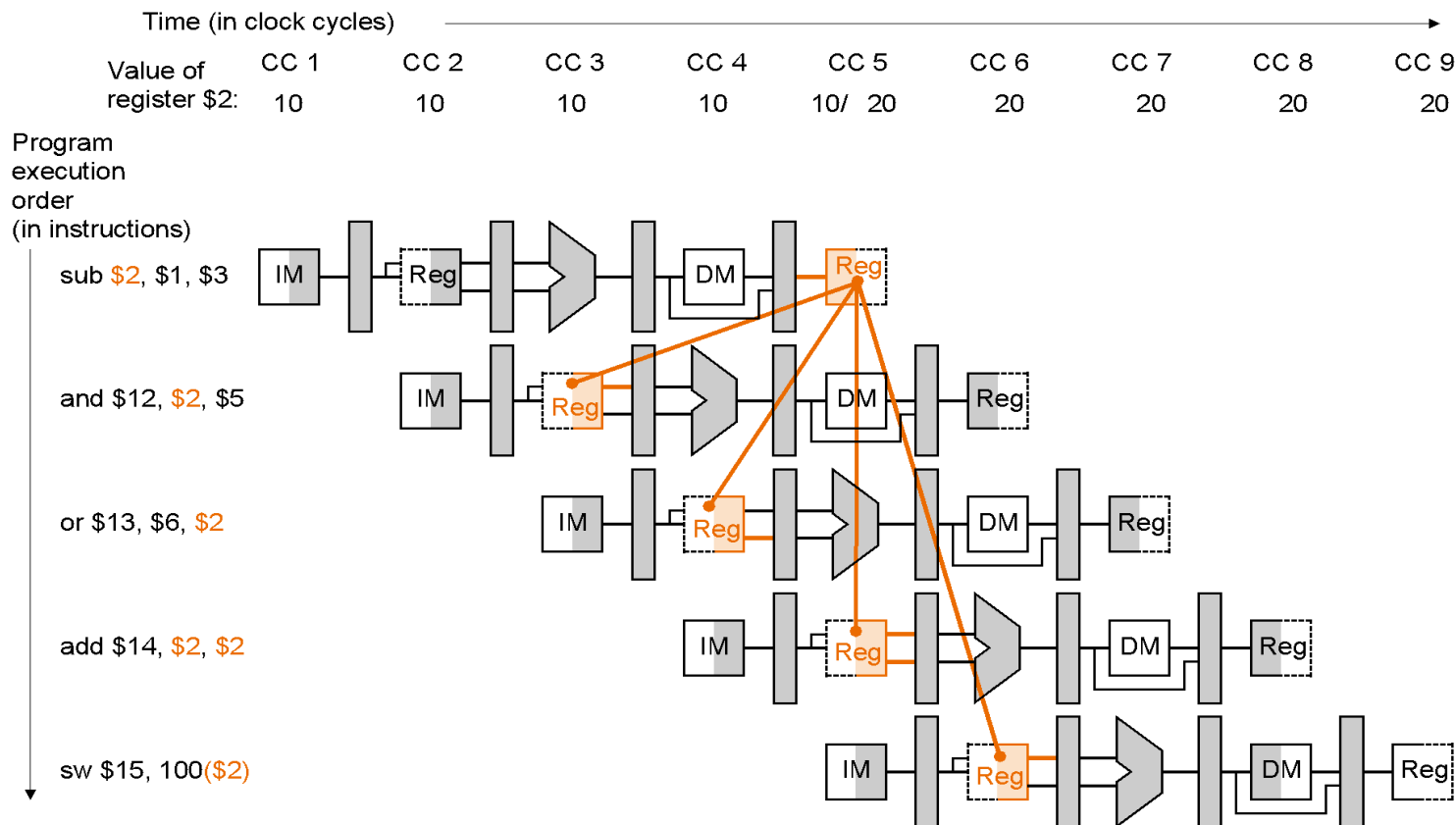
Το υλικό δεν μπορεί να προχωρήσει την εκτέλεση επόμενων εντολών καθώς αναμένεται η ολοκλήρωση της εκτέλεσης μιας εντολής (π.χ. Branches)

- Κίνδυνοι Δεδομένων (data hazards)

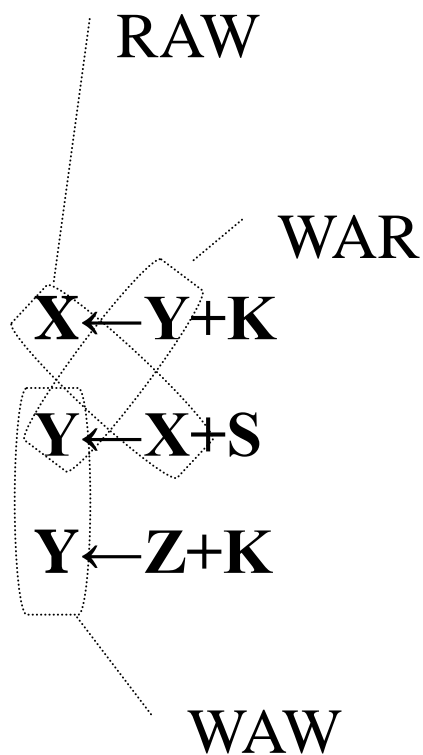
Κίνδυνοι Δεδομένων (Data Hazards) / Το σχήμα προώθησης (forwarding)

```

sub    $2, $1, $3    # καταχωρητής $2 γράφεται από τη sub
and    $12, $2, $5   # 1ος τελεστής ($2) εξαρτάται από sub
or     $13, $6, $2   # 2ος τελεστής ($2) εξαρτάται από sub
add    $14, $2, $2   # 1ος&2ος τελεστής $2) -//- από sub
sw     $15, 100($2)  # offset ($2)          -//- από sub
    
```



Εξαρτήσεις Δεδομένων



- RAW (Read-After-Write) (true-dependence)
Η ανάγνωση ενός καταχωρητή πρέπει να ακολουθεί την εγγραφή στον ίδιο καταχωρητή από προηγούμενη εντολή
- WAR: (Write-After-Read) (anti-dependence)
Η εγγραφή σε ένα καταχωρητή πρέπει να ακολουθεί την ανάγνωσή του από προηγούμενη εντολή
- WAW: (Write-After-Write) (output-dependence)
Η εγγραφή σε ένα καταχωρητή πρέπει να ακολουθεί όλες τις εγγραφές στον ίδιο καταχωρητή από προηγούμενες εντολές

RAW

add **\$t0**, \$s0, \$s1
sub \$t2, **\$t0**, \$s3
or \$s3, \$t7, \$s2
mult \$t2, \$t7, \$s0

True dependence –

WAR

add \$t0, \$s0, \$s1
sub \$t2, \$t0, **\$s3**
or **\$s3**, \$t7, \$s2
mult \$t2, \$t7, \$s0

Name dependence -
antidependence –

WAW

add \$t0, \$s0, \$s1
sub **\$t2**, \$t0, \$s3
or \$s3, \$t7, \$s2
mult **\$t2**, \$t7, \$s0

name dependence -
output dependence –

Identify all the dependencies

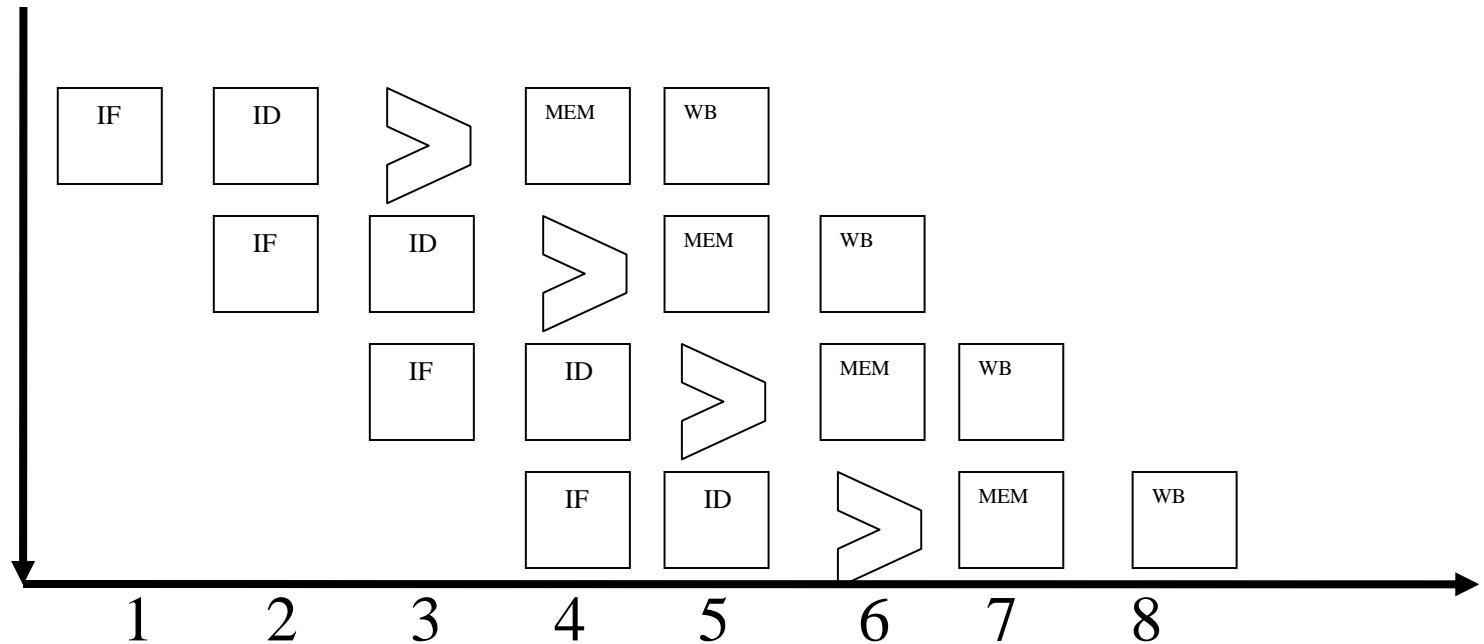
RAW
WAR
WAW

add \$t0, \$s0, \$s1

sub \$t2, \$t0, \$s3

or \$s3, \$t7, \$s2

mul \$t2, \$t7, \$s0



Which dependencies cause hazards? (stalls)

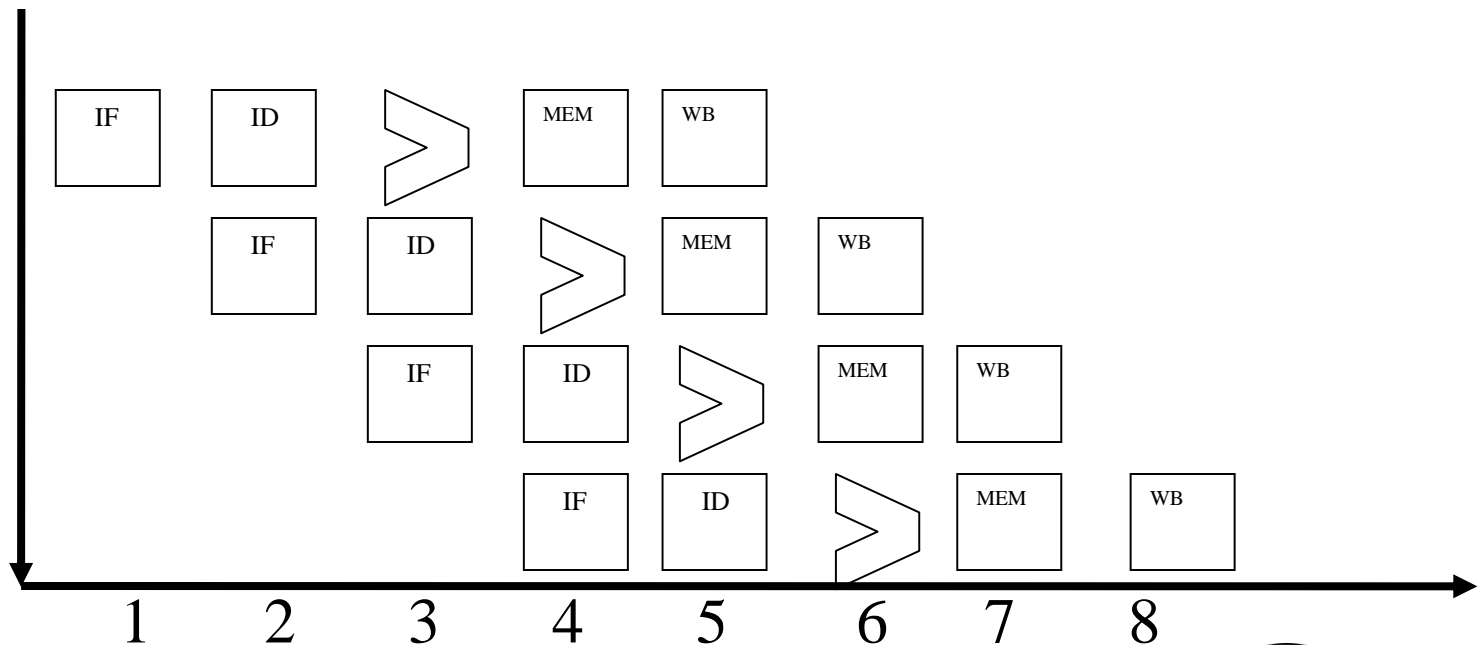
RAW
WAR
WAW

add \$t0, \$s0, \$s1

sub \$t2, \$t0, \$s3

or \$s3, \$t7, \$s2

mul \$t2, \$t7, \$s0



Let's reorder the or

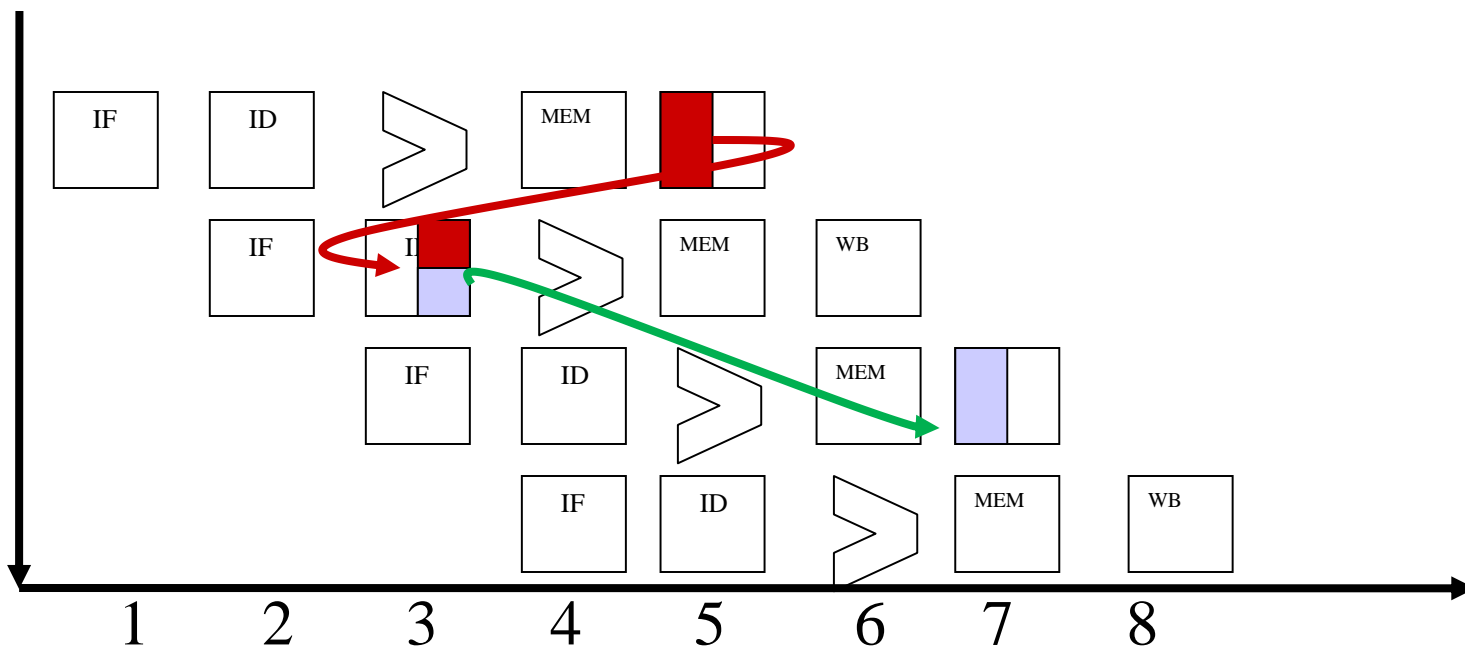
RAW
WAR
WAW

add \$t0, \$s0, \$s1

sub \$t2, \$t0, \$s3

or \$s3, \$t7, \$s2

mul \$t2, \$t7, \$s0



Let's reorder the or

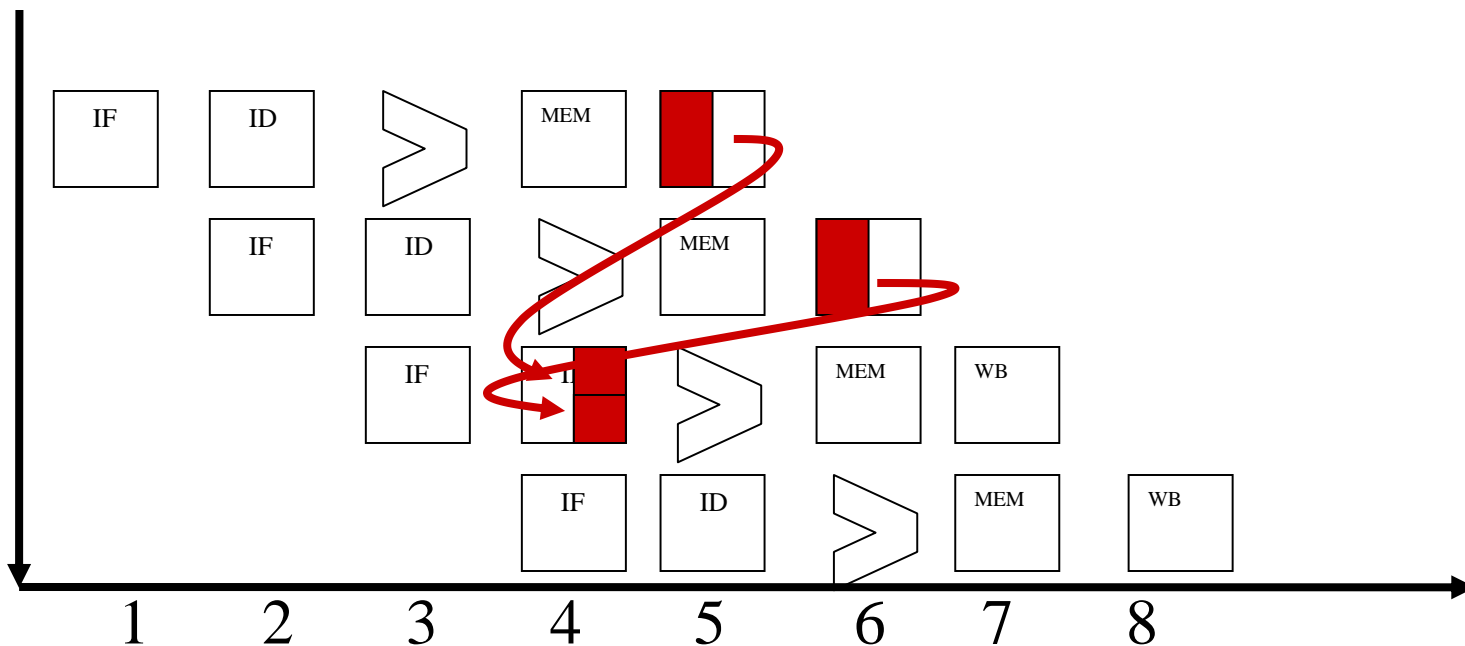
RAW
WAR
WAW

add \$t0, \$s0, \$s1

or \$s3, \$t7, \$s2

sub \$t2, \$t0, \$s3

mul \$t2, \$t7, \$s0



Let's reorder the mul

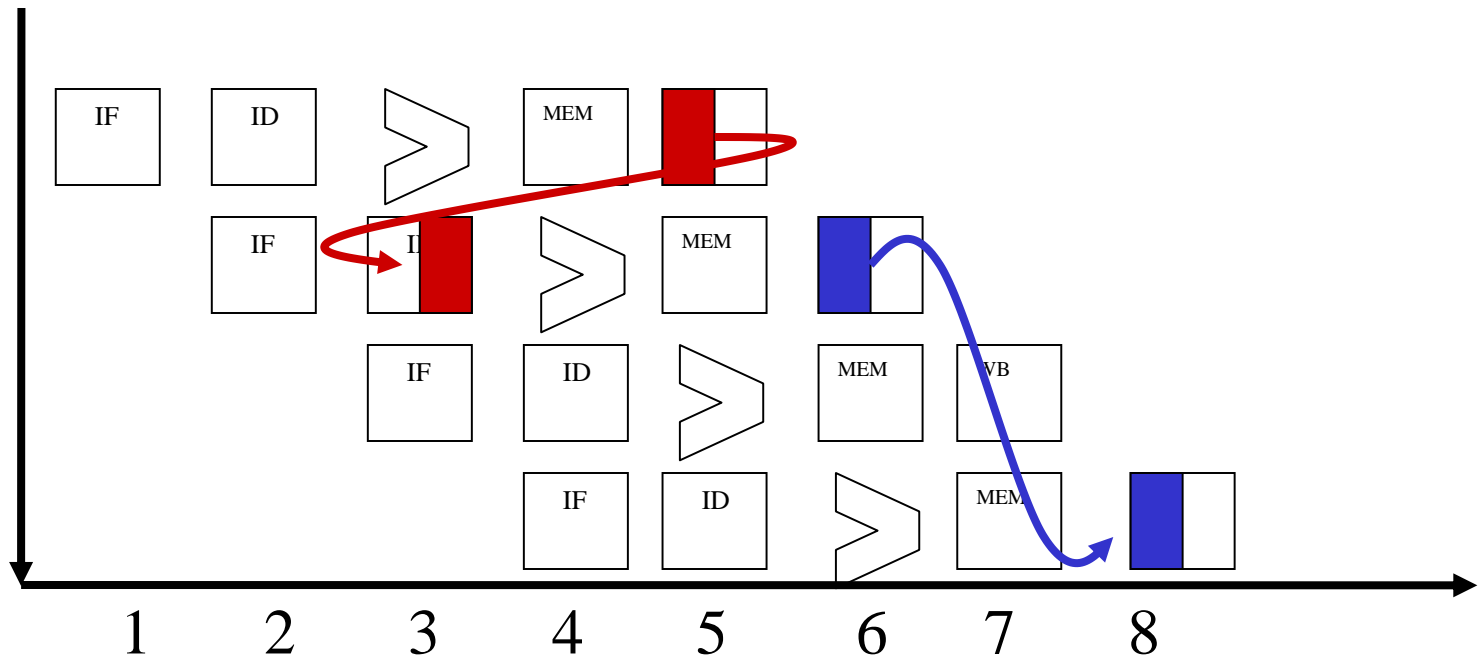
RAW
WAR
WAW

add \$t0, \$s0, \$s1

sub \$t2, \$t0, \$s3

or \$s3, \$t7, \$s2

mul \$t2, \$t7, \$s0



Let's reorder the mul

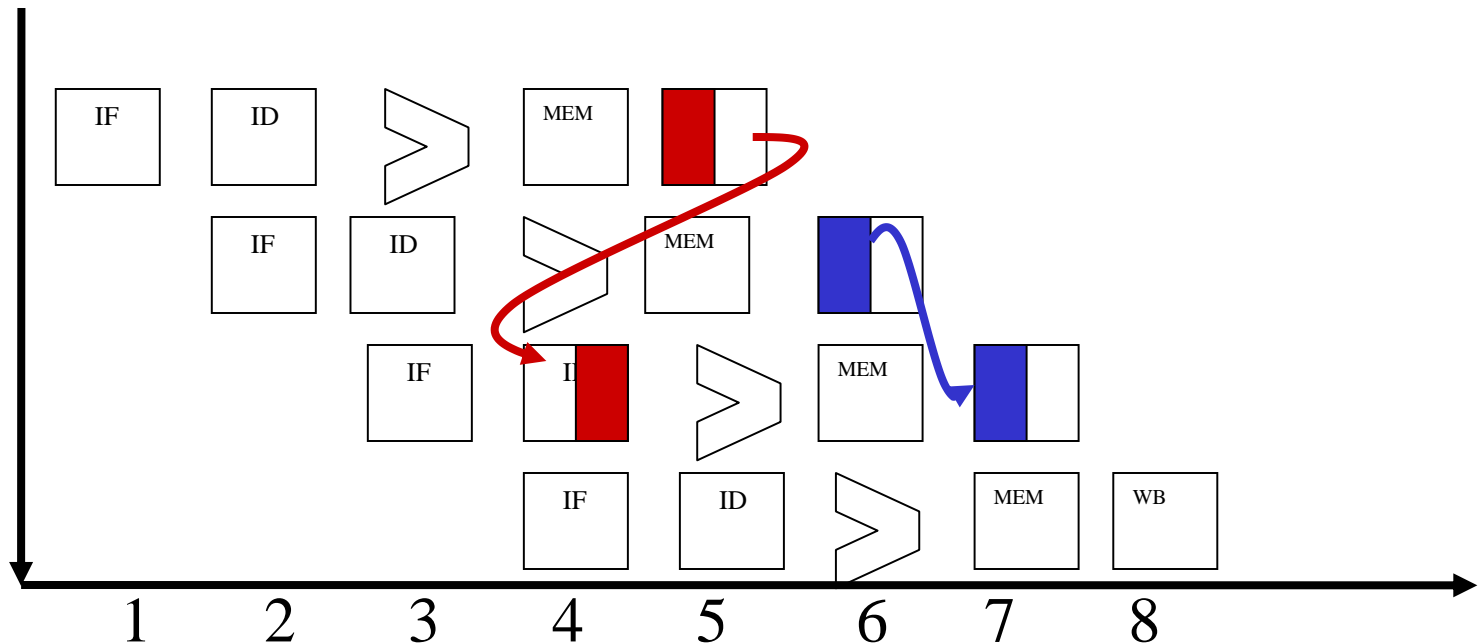
RAW
WAR
WAW

add \$t0, \$s0, \$s1

mul \$t2, \$t7, \$s0

sub \$t2, \$t0, \$s3

or \$s3, \$t7, \$s2



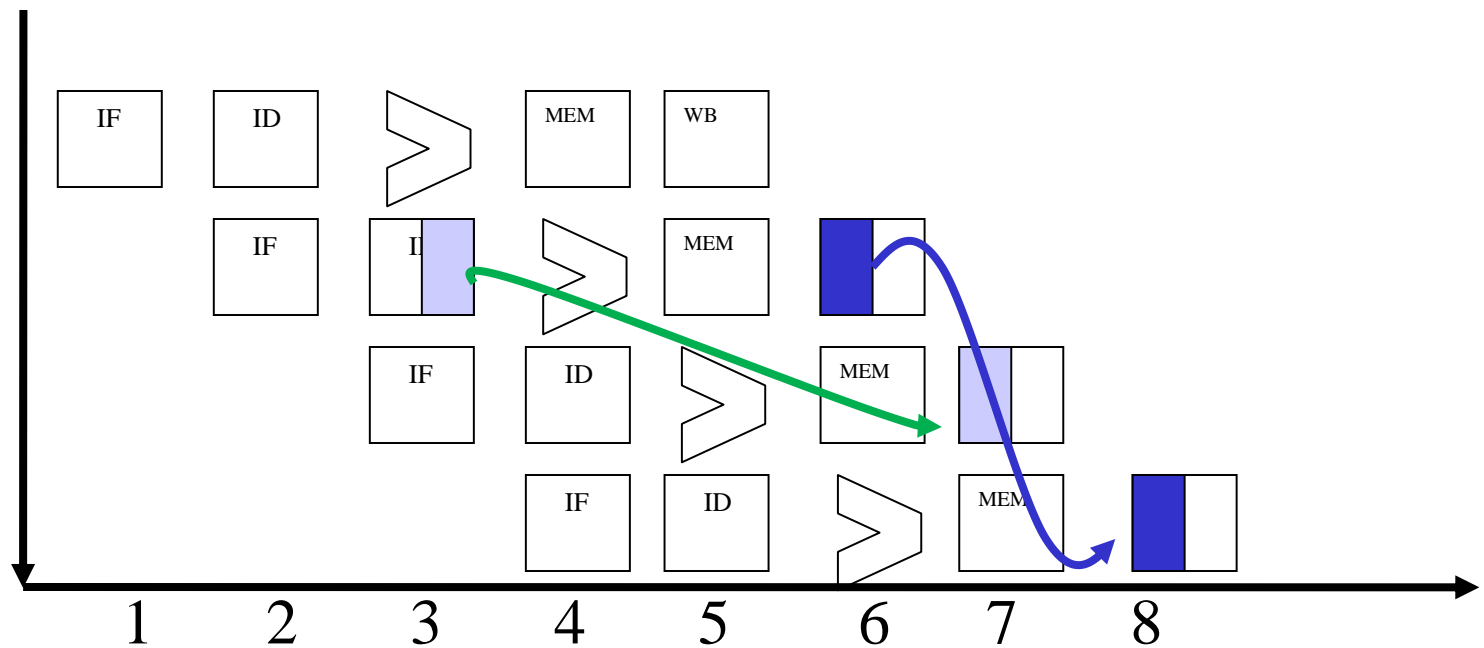
How to alleviate name dependencies?

add \$t0, \$s0, \$s1

sub \$t2, \$t0, \$s3

or \$s3, \$t7, \$s2

mul \$t2, \$t7, \$s0



Data Hazards:

Οι εντολές ανταλλάσσουν μεταξύ τους δεδομένα μέσω του Register File και της μνήμης.

Όταν η επόμενη εντολή-ες χρειάζεται για όρισμα (ανάγνωση) κάτι που δεν έχει προλάβει να γράψει η προηγούμενη

	1	2	3	4	5	6	7	8	9
sub \$2,\$1,\$3	IF	ID	EX	MEM	WB				
and \$12,\$2,\$5		IF	ID	EX	MEM	WB			
or \$13,\$6,\$2			IF	ID	EX	MEM	WB		
add \$14,\$2,\$2				IF	ID	EX	MEM	WB	
sw \$15, 100(\$2)					IF	ID	EX	MEM	WB

Μία λύση είναι η καθυστέρηση(stall) του αγωγού (pipeline):

Προσθέτω δύο
εντολές NOP:

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

STALL
(κύκλοι
αναμονής)

	1	2	3	4	5	6	7	8	9	10	11
sub \$2,\$1,\$3	IF	ID	EX	MEM	WB						
nop		IF	ID								
nop			IF	ID							
and \$12,\$2,\$5				IF	ID	EX	MEM	WB			
or \$13,\$6,\$2					IF	ID	EX	MEM	WB		
add \$14,\$2,\$2						IF	ID	EX	MEM	WB	
sw \$15, 100(\$2)							IF	ID	EX	MEM	WB

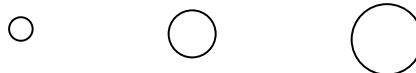
Πιο κομψή λύση είναι η προώθηση (forwarding):

Έχουμε εξάρτηση δεδομένων RAW. Τα αποτελέσματα γράφονται είτε στην μνήμη είτε στο register file.

Στην περίπτωση R-TYPE:

Αποθηκεύονται στο RegFile, παράγονται όμως μετά την ALU, άρα είναι διαθέσιμα στον EX/MEM

Πώς θα βρουν η επόμενη και η μεθεπόμενη εντολή στη φάση EX τα σωστά ορίσματα:



1. Προωθούμε το EX/MEM αποτέλεσμα ως είσοδο για την ALU πράξη της επόμενης εντολής

2. Το ίδιο κάνουμε για τη μεθεπόμενη εντολή, προωθούμε το MEM/WB αποτέλεσμα, ως είσοδο για την ALU πράξη της μεθεπόμενης εντολής

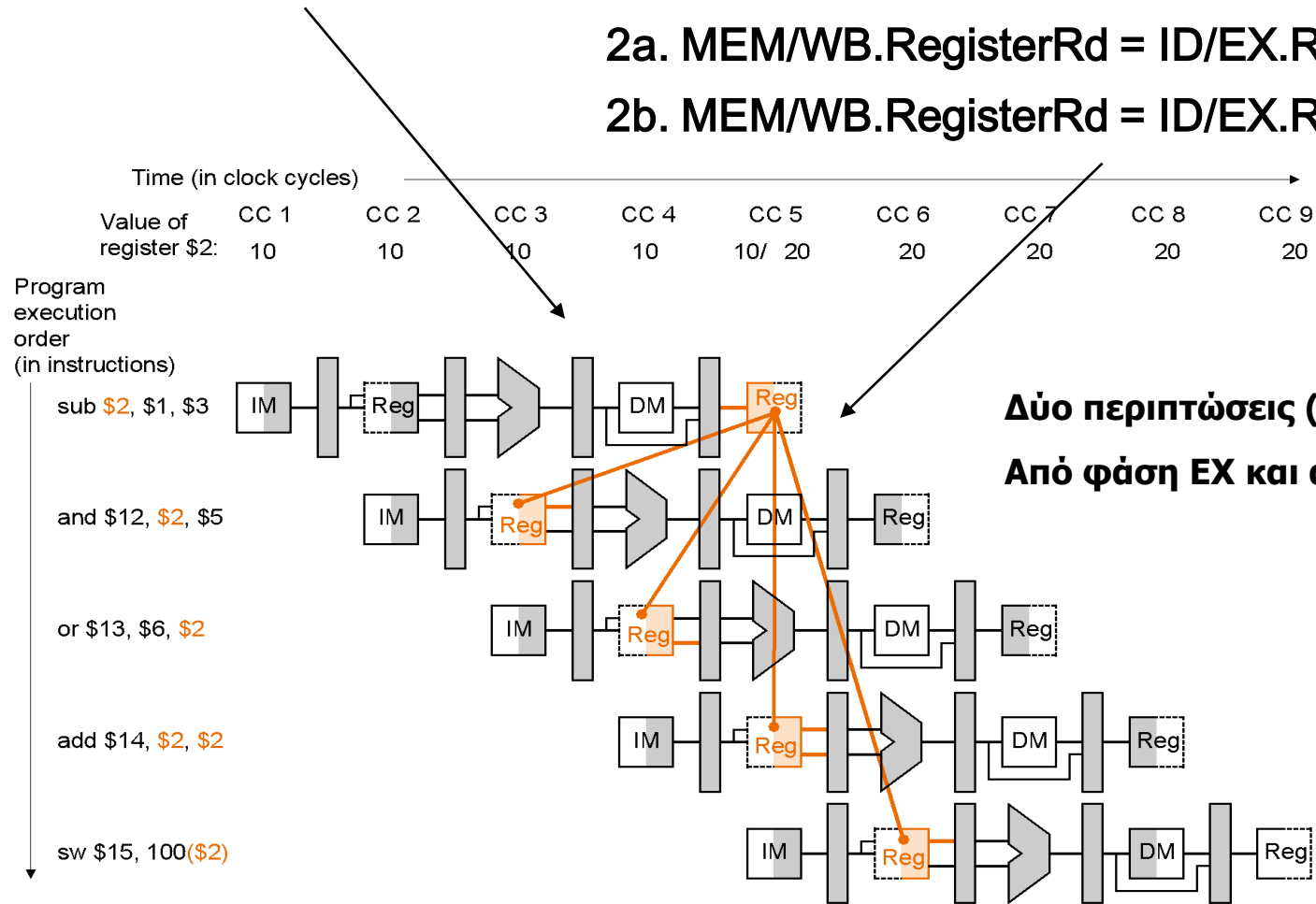
Είσοδο στην ALU όχι μόνο από ID/EX καταχωρητή!!

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs

2b. MEM/WB.RegisterRd = ID/EX.RegisterRt



Δύο περιπτώσεις (αιτίες) για hazard:
Από φάση EX και από φάση MEM

sub-and hazard: EX/MEM.RegisterRd = ID/EX.RegisterRs = \$2

sub-or hazard: MEM/WB.RegisterRd = ID/EX.RegisterRt = \$2

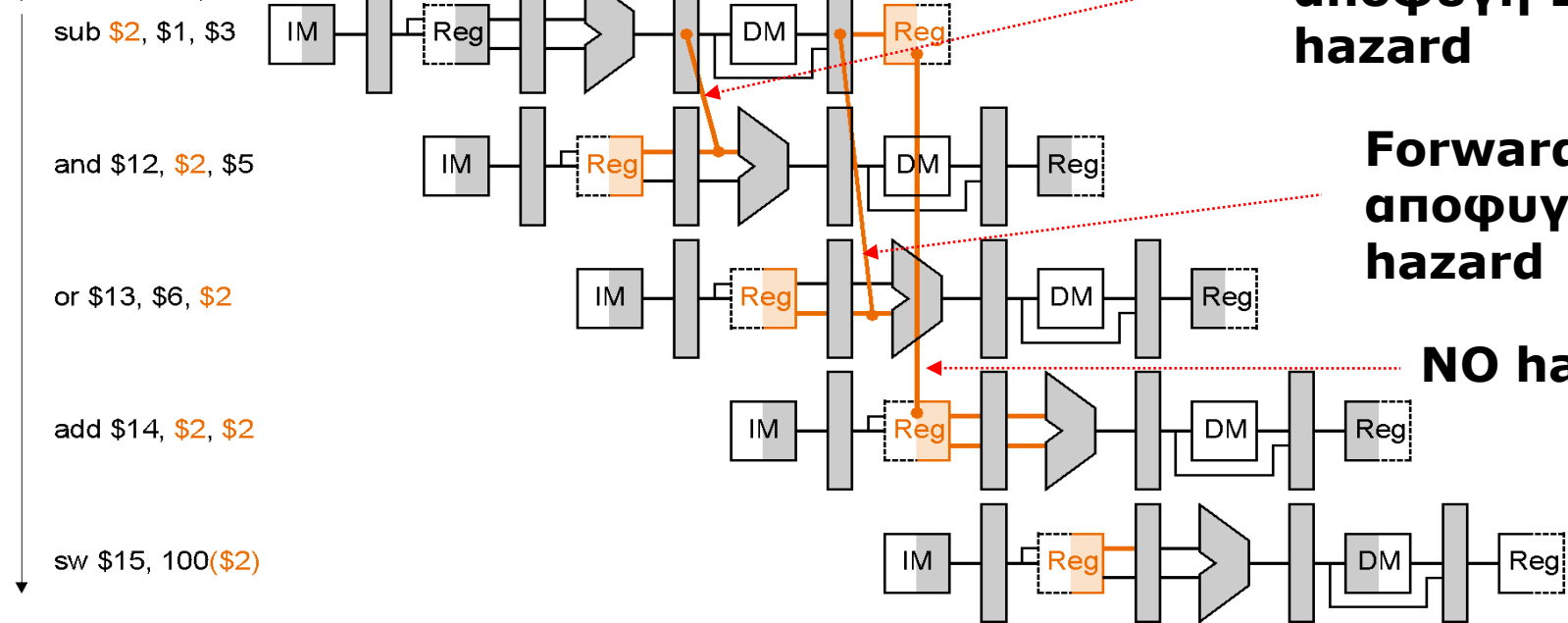
Forwarding:

Πρώτα ανιχνεύουμε την πιθανή αιτία κινδύνου
 Μετά κάνουμε προώθηση της κατάλληλης τιμής

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

Program execution order (in instructions)



Forwarding για αποφυγή EX hazard

Forwarding για αποφυγή MEM hazard

NO hazard!!

Συνθήκες ελέγχου των κινδύνων:

1. EX hazard:

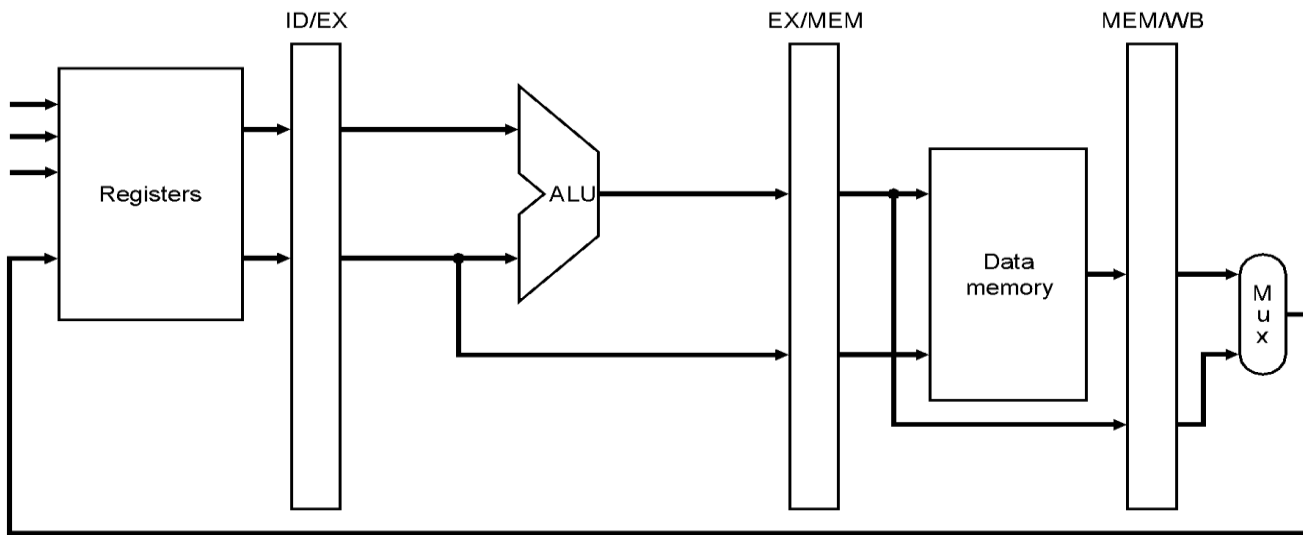
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd≠0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd≠0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB=10

2. MEM hazard:

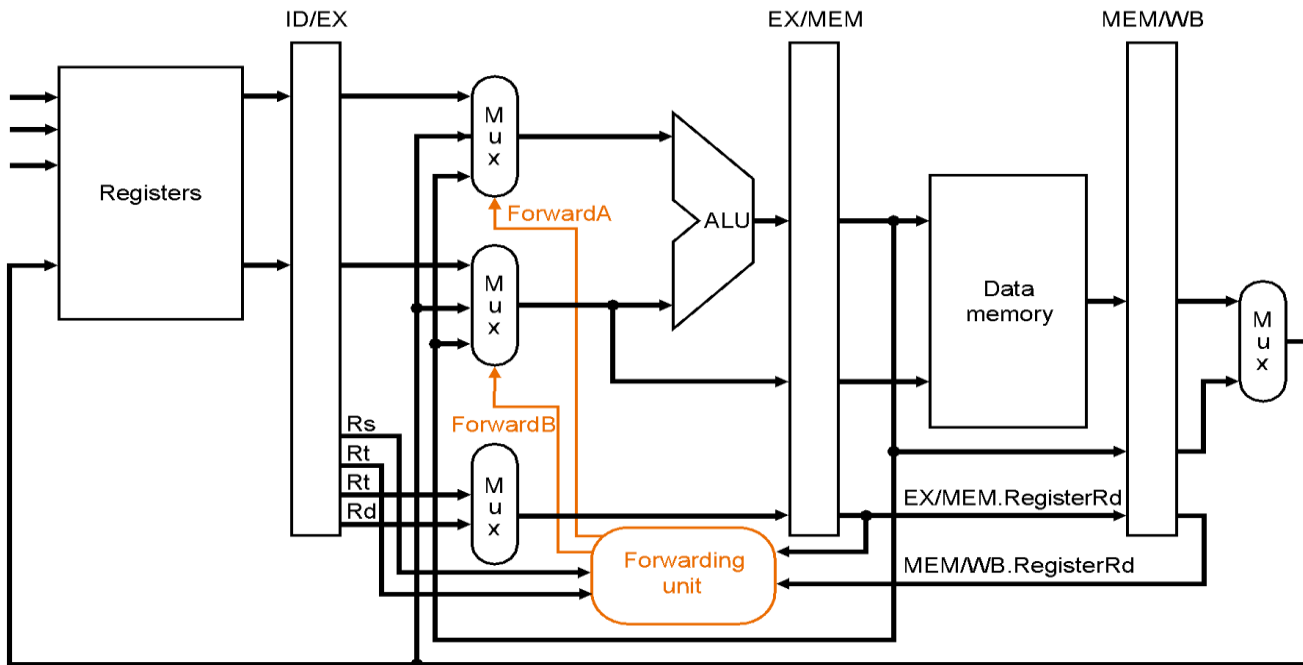
```
if (MEM/WB.RegWrite  
and (MEM/WB.RegisterRd≠0)  
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01
```

```
if (MEM/WB.RegWrite  
and (MEM/WB.RegisterRd≠0)  
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01
```



a. No forwarding

Pipelining χωρίς forwarding



b. With forwarding

Forwarding paths:

Από:

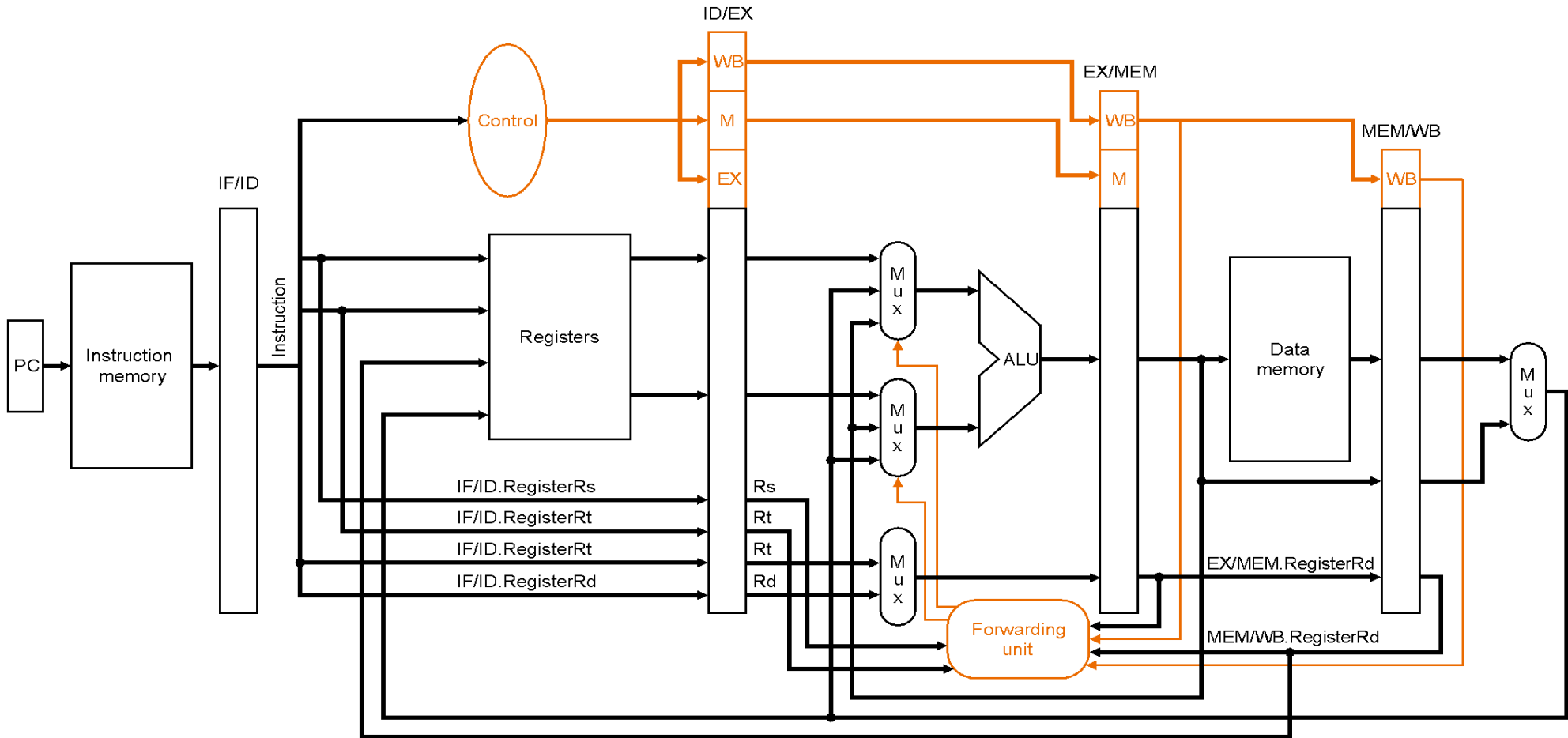
a) **EX/MEM register**

και από:

b) **MEM/WB register**

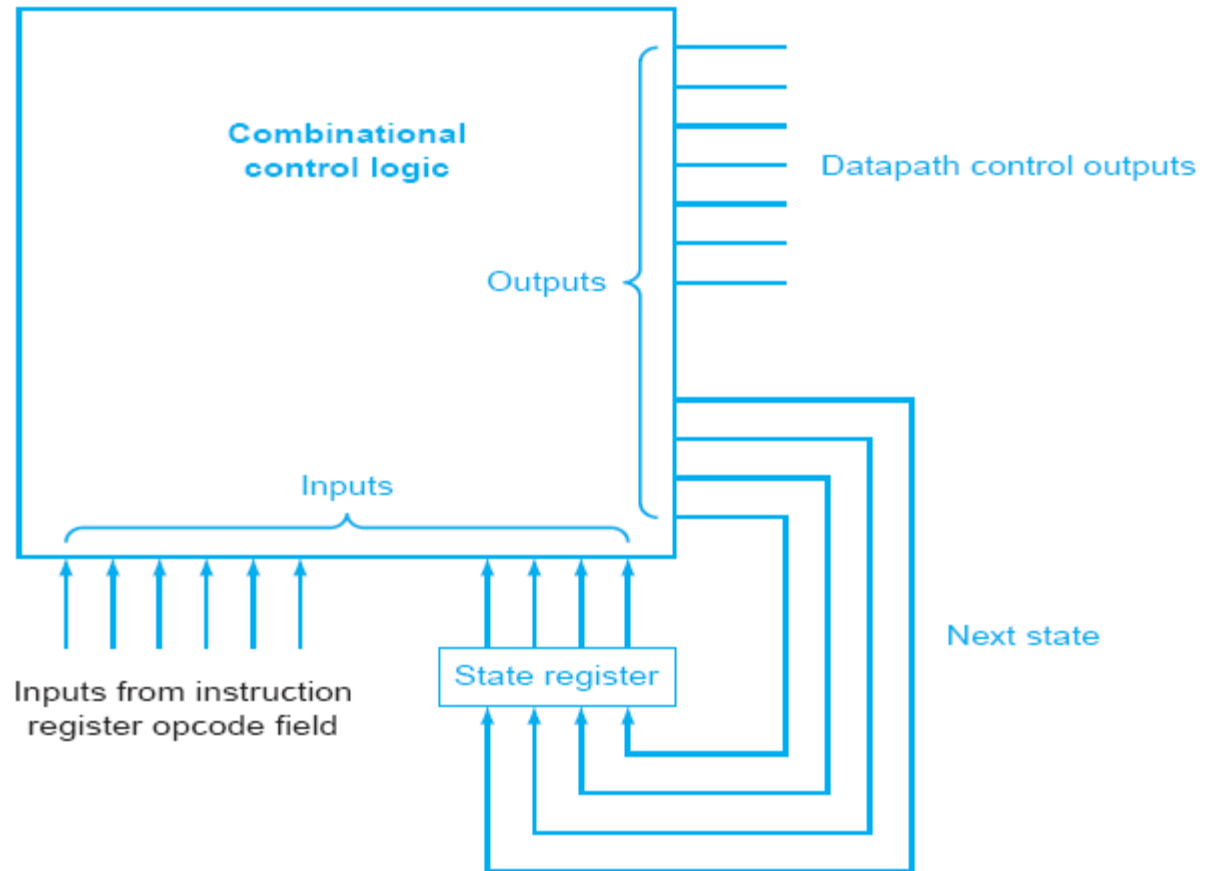
προς τις εισόδους της ALU

Datapath to resolve hazards via forwarding:

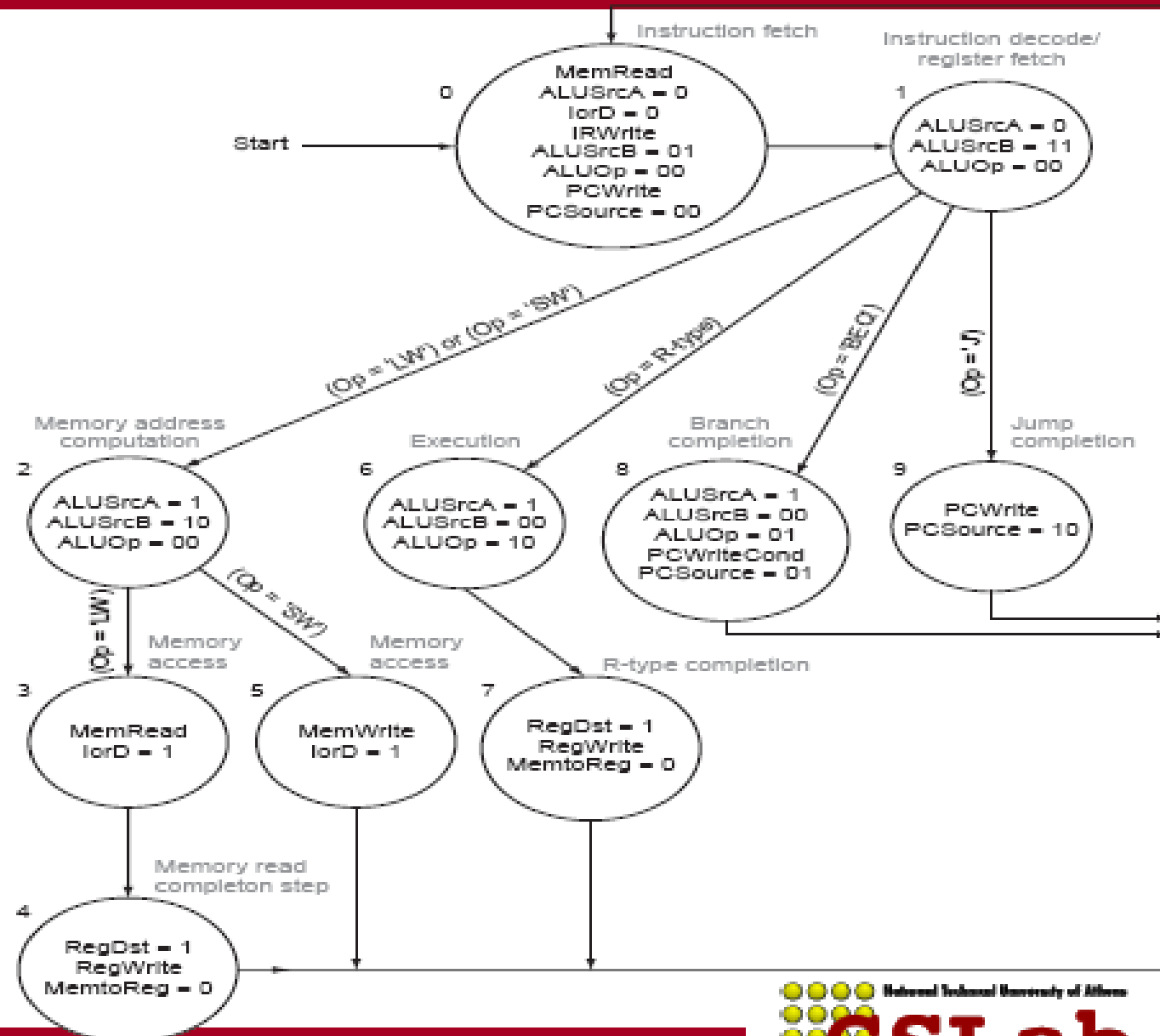


Σχεδιασμός Control

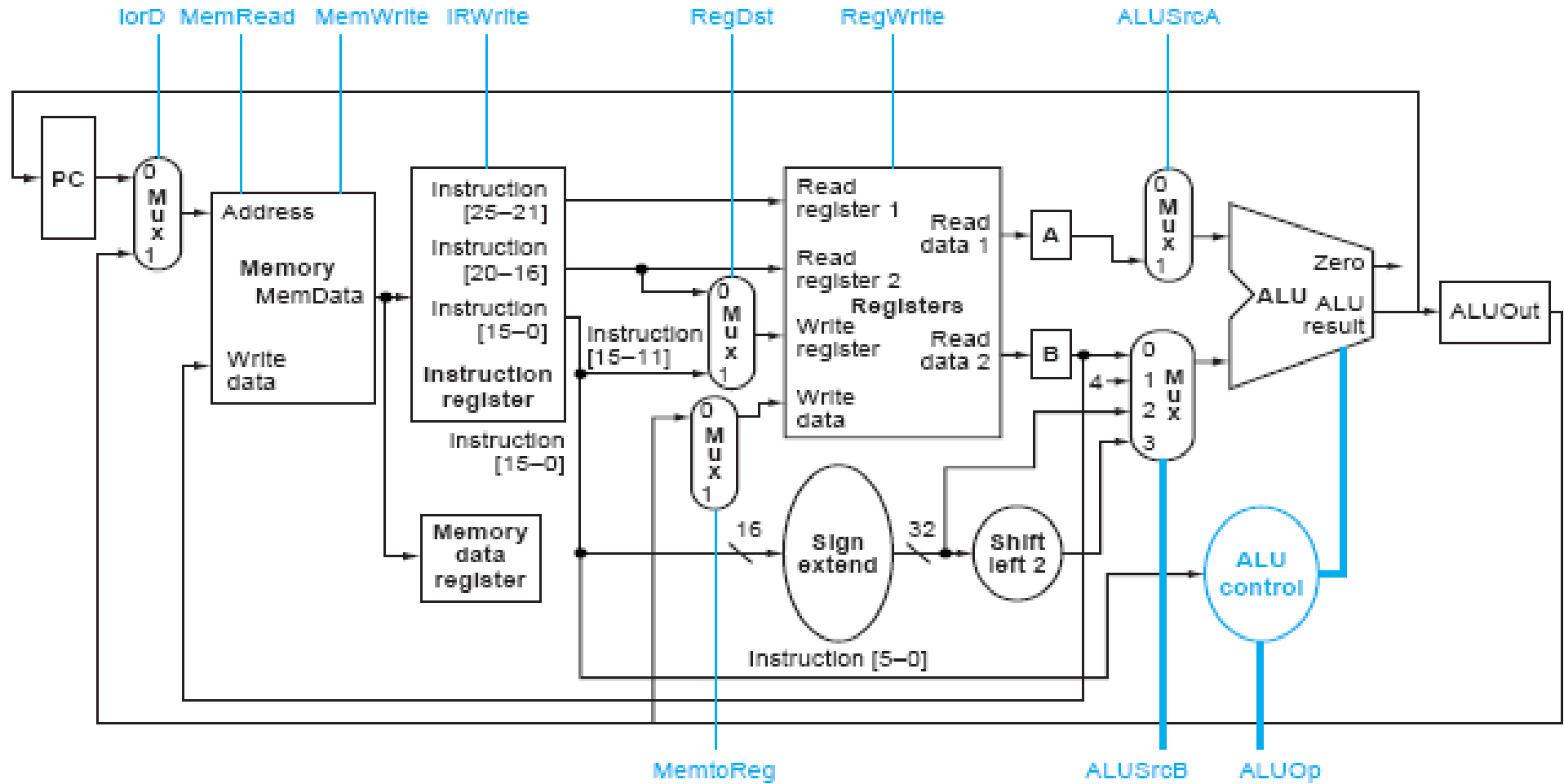
- Είσοδοι
 - Πεδία του instruction
- Έξοδοι
 - Σήματα ελέγχου ALU
 - Σήματα ελέγχου μνήμης
 - Πολυπλέκτες
- Finite State Machine



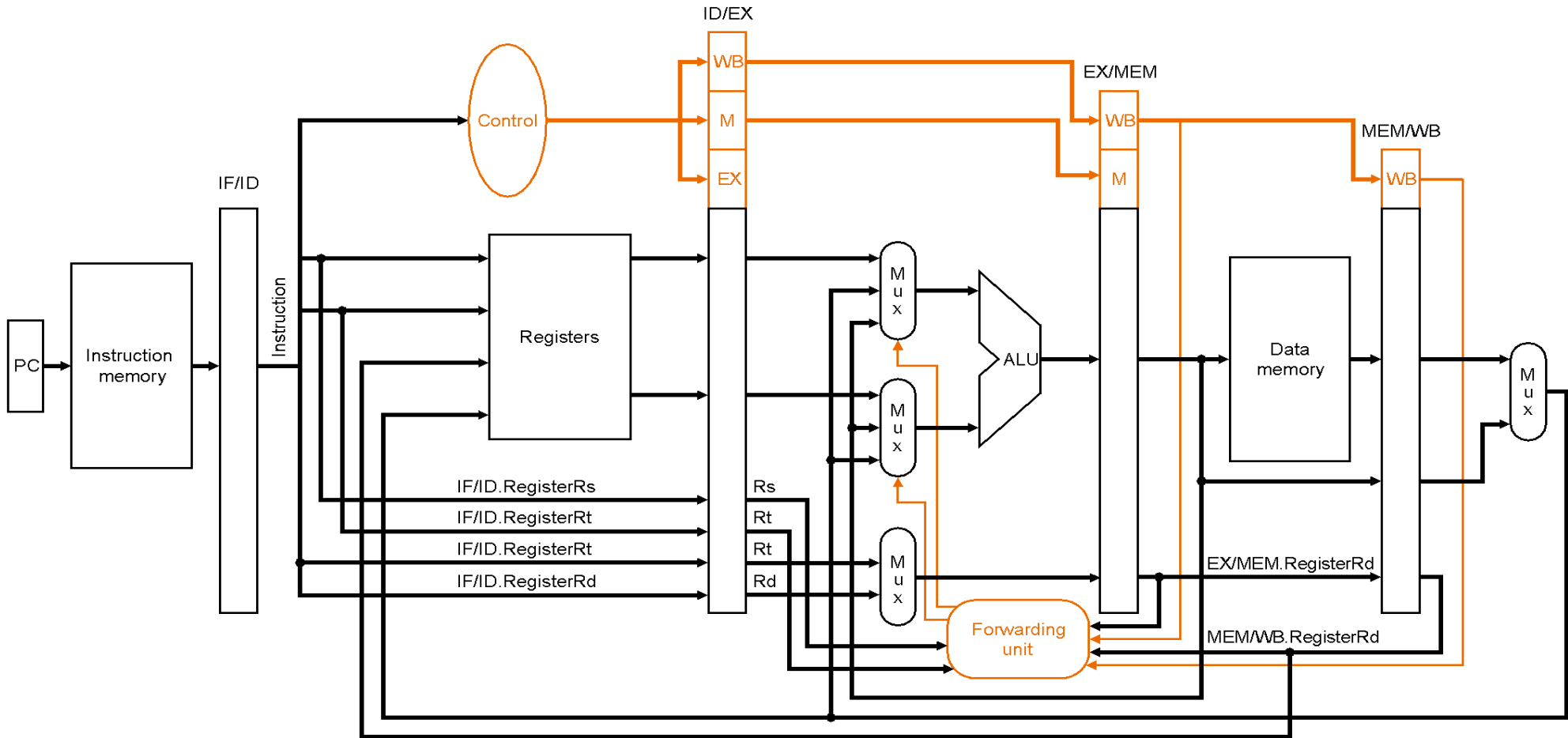
Σχεδιασμός Control FSM (Multicycle datapath)



Multicycle datapath



Datapath to resolve hazards via forwarding:



or \$4, \$4, \$2

and \$4, \$2, \$5

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

before<1>

before<2>

Clock 3

add \$9, \$4, \$2

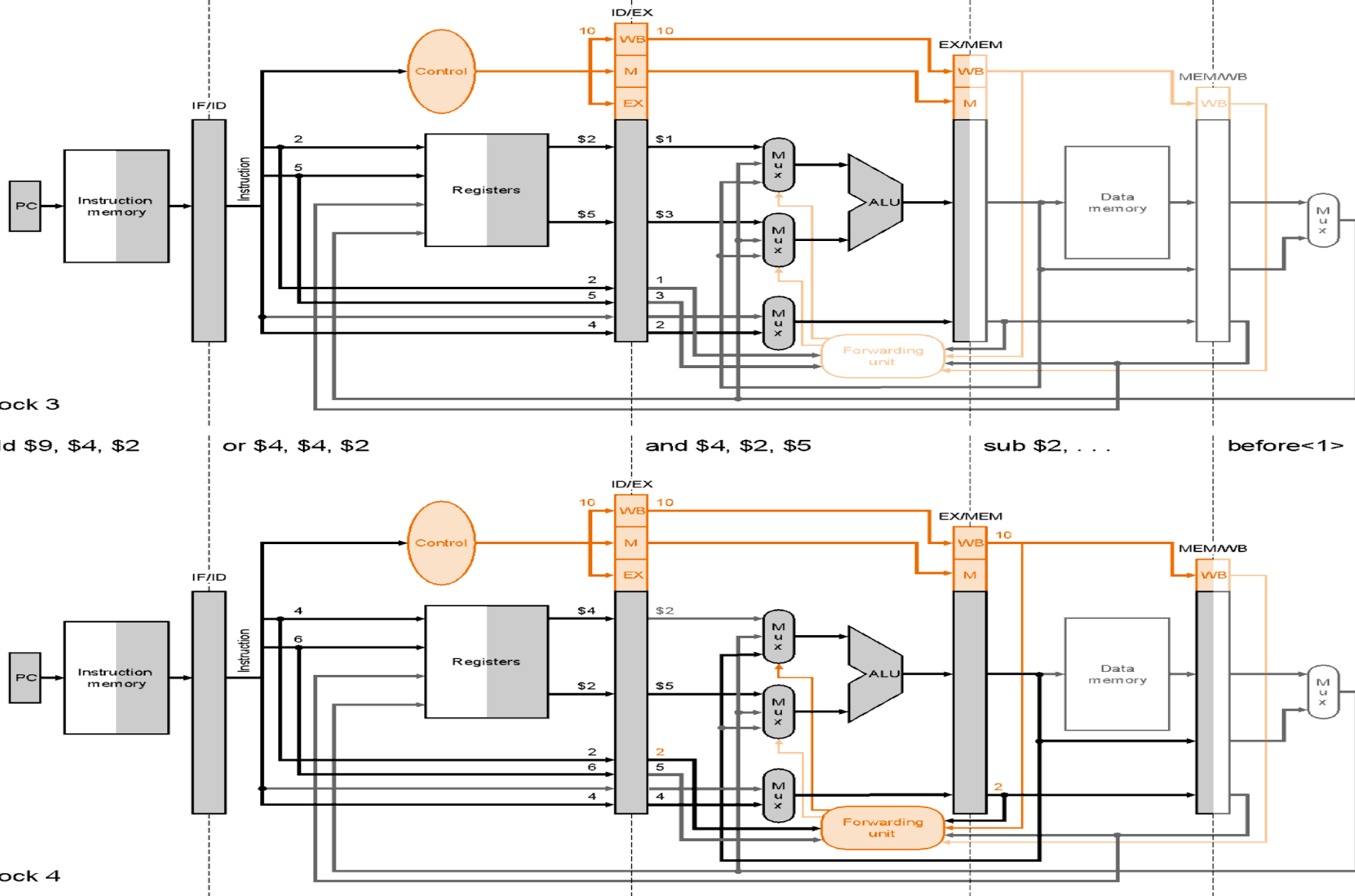
or \$4, \$4, \$2

and \$4, \$2, \$5

sub \$2, ...

before<1>

Clock 4



after<1>

add \$9, \$4, \$2

or \$4, \$4, \$2

and \$4, ...

sub \$2, ...

Clock 5

after<2>

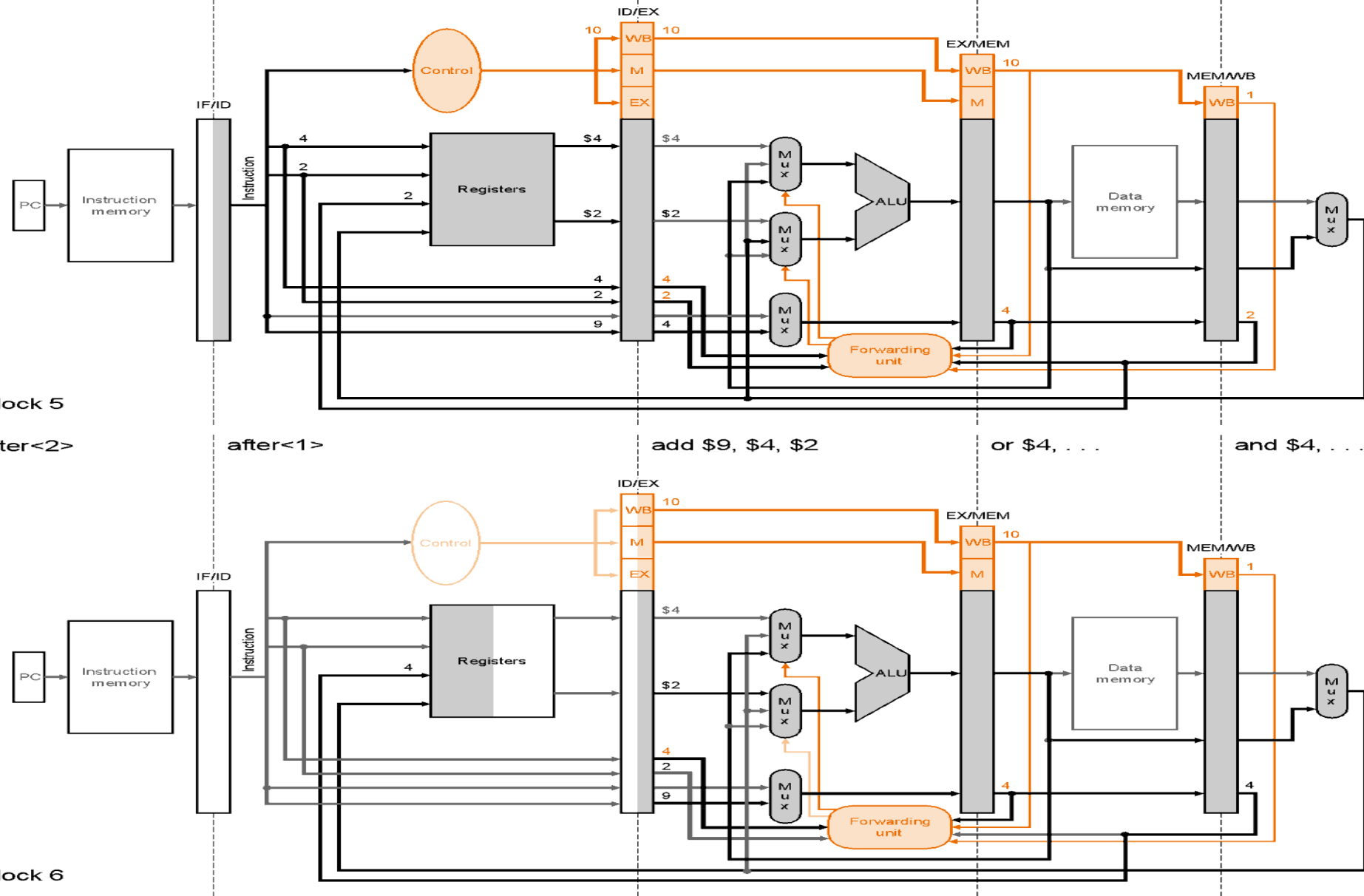
after<1>

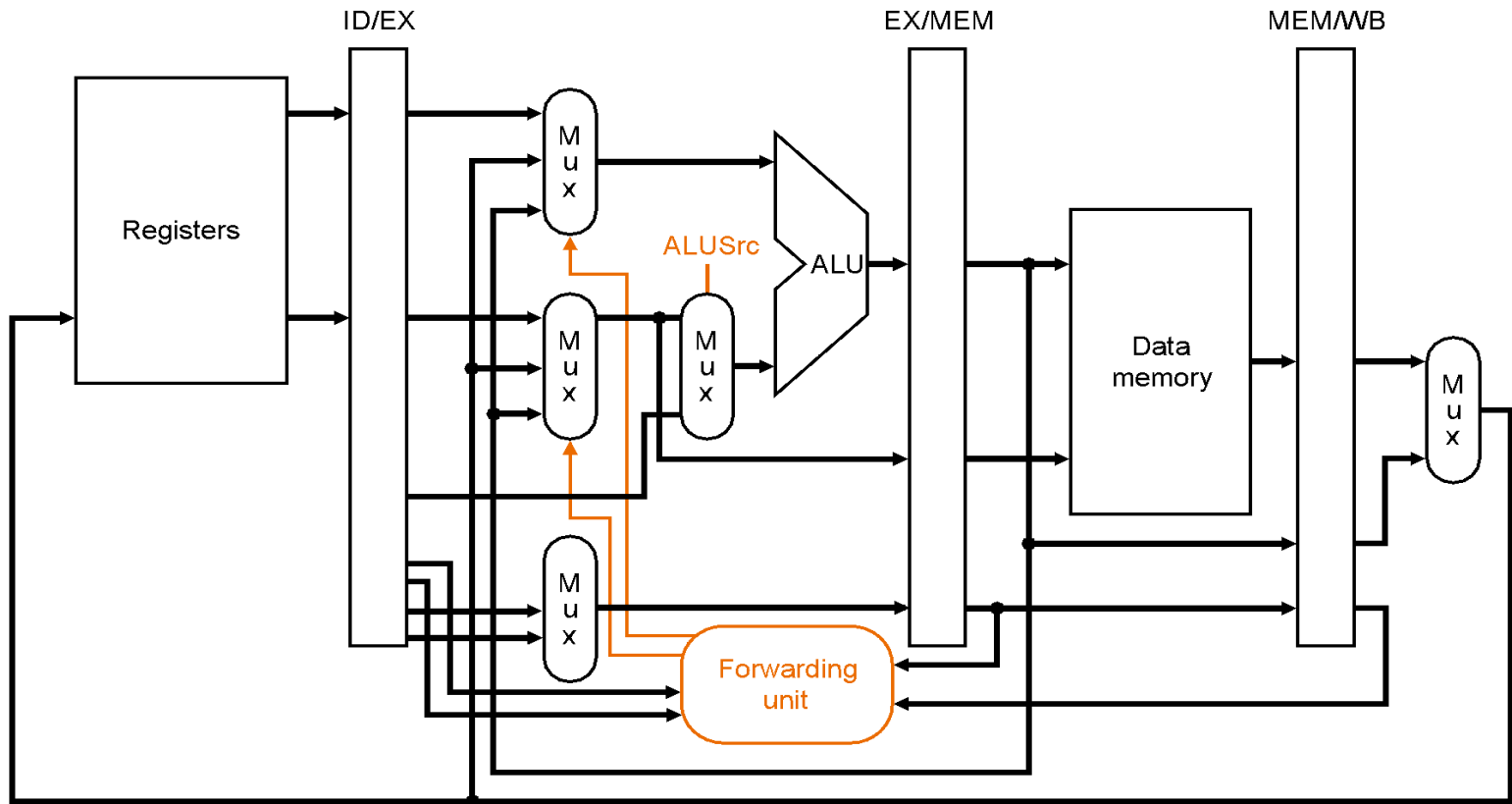
add \$9, \$4, \$2

or \$4, ...

and \$4, ...

Clock 6





Κίνδυνοι δεδομένων (data hazards) και αναπόφευκτες καθυστερήσεις (stalls) εξ' αιτίας τους

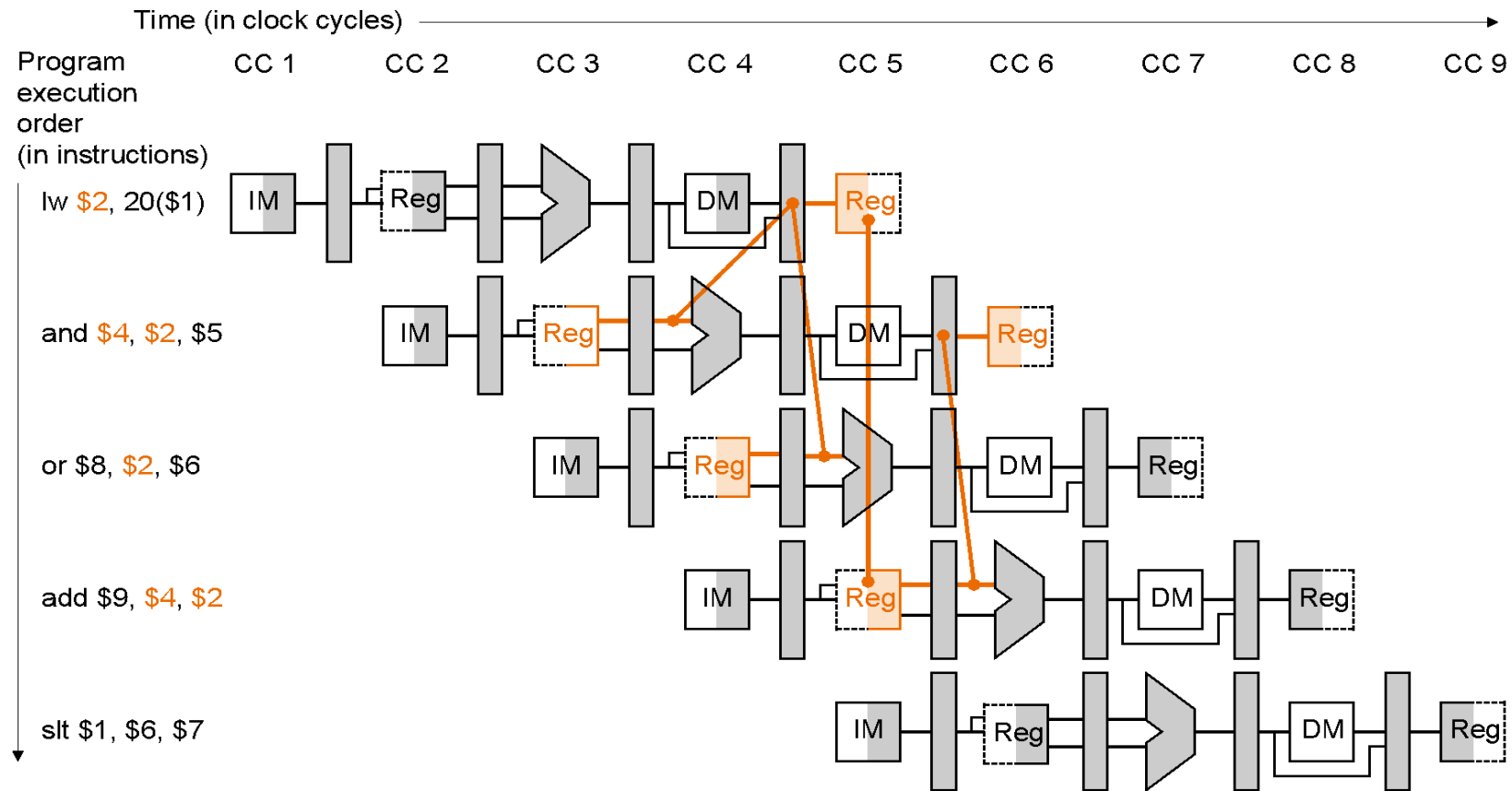
Όταν η εντολή που κάνει write είναι R-TYPE, τότε το αποτέλεσμα είναι έτοιμο στην φάση EX (έξοδος ALU) και αποθηκεύεται, στο τέλος του κύκλου, στον EX/MEM καταχωρητή.

Οι επόμενες χρειάζονται τα ορίσματα στην φάση EX οπότε το forwarding δουλεύει. (αφού η εντολή που κάνει write θα βρίσκεται στις MEM και WB)

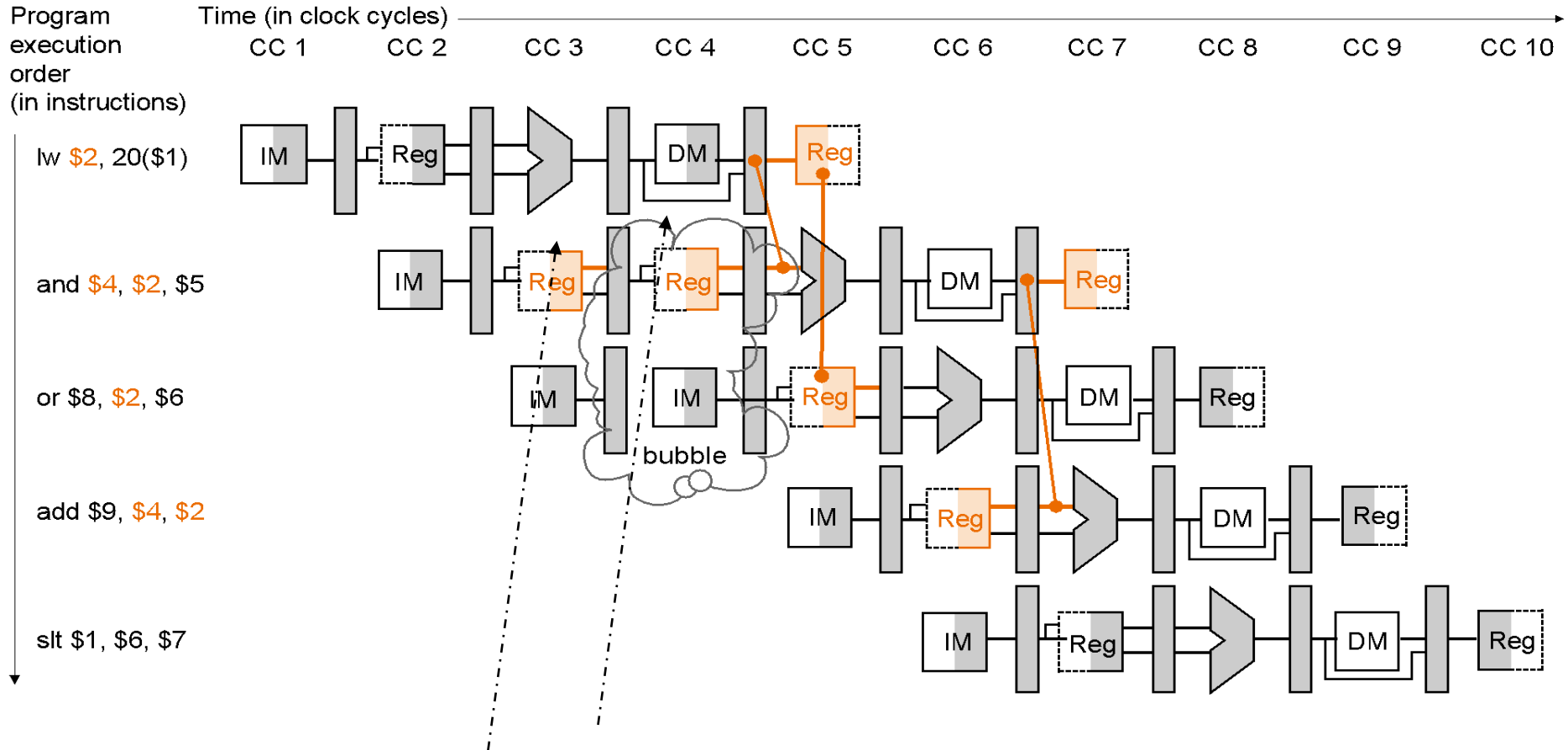
Το forwarding δεν δίνει όμως πάντα λύση!!

αναπόφευκτες καθυστερήσεις (stalls):

Όταν η προηγούμενη εντολή (που κάνει write) είναι load ή store, τότε το **αποτέλεσμα** είναι έτοιμο στο τέλος της φάσης MEM ή και WB (ακόμα χειρότερα)



Λύση: αναπόφευκτες καθυστερήσεις (stalls) στο pipeline



Επαναλαμβάνουμε τις ίδιες φάσεις: ID για την and και IF για την or.

Πως ανιχνεύουμε τις αναπόφευκτες καθυστερήσεις:

Hazard detection unit

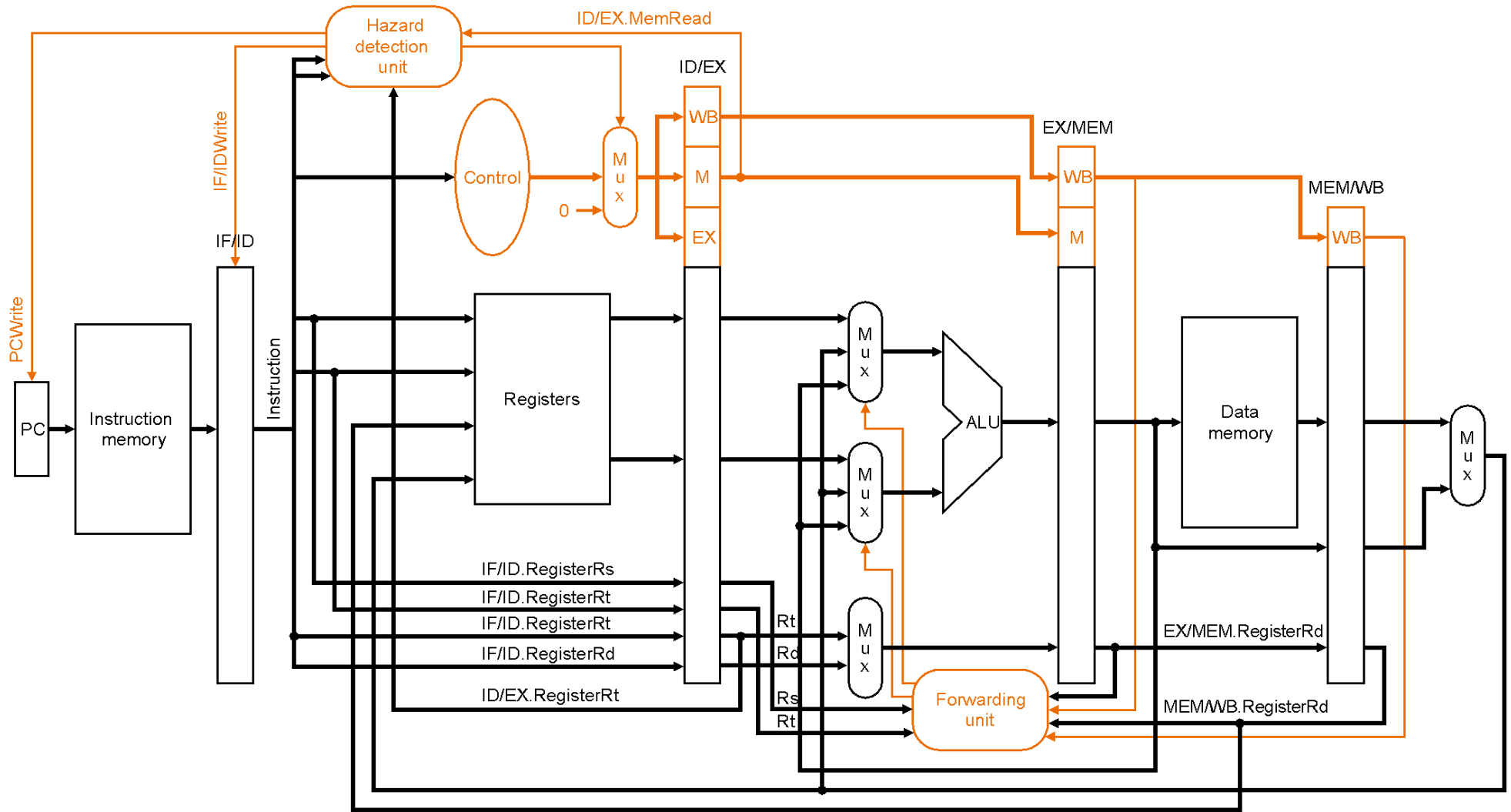
Λειτουργεί στη φάση ID ώστε να βάλει καθυστέρηση μεταξύ του load και της χρησιμοποίησης των αποτελεσμάτων του.

```
If (ID/EX.MemRead and
  ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
  (ID/EX.RegisterRt = IF/ID.RegisterRt)))
  Stall the pipeline
```

Τι σημαίνει stall:

Εμποδίζουμε το PC και τον IF/ID να αλλάξουν-
άρα διαβάζεται σε δύο διαδοχικούς κύκλους η
ίδια εντολή και αποκωδικοποιείται η ίδια
επόμενη της δύο φορές συνεχόμενα

Final Pipeline:



Άσκηση Pipeline

Δεδομένα

Έχουμε ένα loop...

Rep:

```
lw $2, 100($3)
sub $2, $2, $5
sw $2, 100($3)
sub $3, $3, $6
sub $1, $1, $7
bne $1, $0, Rep
```

Exit:

Άσκηση Pipeline

Δεδομένα

Έχουμε ένα loop...

και αυτή την αρχική κατάσταση στους καταχωρητές

```
Rep:
  lw $2, 100($3)
  sub $2, $2, $5
  sw $2, 100($3)
  sub $3, $3, $6
  sub $1, $1, $7
  bne $1, $0, Rep
```

```
$1: 500
$7: 5
```

- Δεν υπάρχει cache miss
- Cache hit σε lcc
- branches γίνονται resolve στο MEM stage

```
Exit:
```

Ζητούμενο

Για το 1^ο LOOP (μέχρι και το 1w του 2^{ου} LOOP)

Να δείξετε τα **διάφορα στάδια του pipeline** (Διάγραμμα χρονισμού) που περνάνε οι εντολές.

Υποδείξτε και εξηγήστε τα **πιθανά hazards** που μπορούν να προκύψουν κατά την εκτέλεση, καθώς και τον τρόπο που αντιμετωπίζονται.

Κύκλος	1	2	3	4	5	6	7
Εντολή 1	IF	ID	EX	MEM	WB		
Εντολή 2		IF	ID	EX	MEM	WB	
Εντολή 3		
...							

Ζητούμενο (2)

Πόσοι κύκλοι απαιτούνται συνολικά για να ολοκληρωθεί ο βρόχος (για όλες τις επαναλήψεις του, όχι μόνο για την 1η);

Rep:

```
lw $2, 100($3)
sub $2, $2, $5
sw $2, 100($3)
sub $3, $3, $6
sub $1, $1, $7
bne $1, $0, Rep
```

Exit:

Κύκλος	1	2	3	4	5	6	7
Εντολή 1	IF	ID	EX	MEM	WB		
Εντολή 2		IF	ID	EX	MEM	WB	
Εντολή 3		
...							

Απάντηση

```
Rep:                                     $1: 500
    lw $2, 100($3)                       $7: 5
    sub $2, $2, $5                        Δεν υπάρχει cache miss
    sw $2, 100($3)                       cache hit σε lcc
    sub $3, $3, $6
    sub $1, $1, $7 → $1 = 500, 495, 490, 485, ...
    bne $1, $0, Rep
Exit:
```

Ο βρόχος θα εκτελεστεί για $500 / 5 = 100$ επαναλήψεις.

Το διάγραμμα χρονισμού του pipeline για τη χρονική διάρκεια που ζητείται είναι το ακόλουθο:



























Κύκλος	1	2	3	4	5	6	7	8	9	10	11	12	13
lw \$2,100(\$3)	IF	ID	EX	MEM	WB								
sub \$2,\$2,\$5		IF	ID			EX	MEM	WB					
sw \$2,100(\$3)			IF			ID			EX	MEM	WB		
sub \$3,\$3,\$6						IF			ID	EX	MEM	WB	
sub \$1,\$1,\$7									IF	ID	EX	MEM	WB
bne \$1,\$0, Rep										IF	ID		
lw \$2,100(\$3)													

Κύκλος	14	15	16	17	18	19	20
lw \$2,100(\$3)							
sub \$2,\$2,\$5							
sw \$2,100(\$3)							
sub \$3,\$3,\$6							
sub \$1,\$1,\$7							
bne \$1,\$0, Rep	EX	MEM	WB				
lw \$2,100(\$3)			IF	ID	EX	MEM	WB

Ή πιο συμπιεσμένο για να χωράει σε μια σελίδα 😊

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID	☁	☁	EX	M	WB												
sw			IF	☁	☁	ID	☁	☁	EX	M	WB									
sub				☁	☁	IF	☁	☁	ID	EX	M	WB								
sub				☁	☁		☁	☁	IF	ID	EX	M	WB							
bne				☁	☁		☁	☁		IF	ID	☁	☁	EX	M	WB				
lw				☁	☁		☁	☁				☁	☁			IF	ID	EX	M	WB

Hazards

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID			EX	M	WB												
sw			IF			ID			EX	M	WB									
sub						IF			ID	EX	M	WB								
sub									IF	ID	EX	M	WB							
bne										IF	ID			EX	M	WB				
lw																IF	ID	EX	M	WB

Rep:

lw \$2, 100 (\$3)

sub \$2, \$2, \$5

sw \$2, 100 (\$3)

sub \$3, \$3, \$6

sub \$1, \$1, \$7

bne \$1, \$0, Rep

Exit:

Stalls στους κύκλους 4,5:

Ο καταχωρητής \$2 για την εντολή **sub \$2,\$2,\$5** (η ανάγνωση του οποίου γίνεται στο στάδιο ID) γίνεται διαθέσιμος στο τέλος του κύκλου 5 (στάδιο WB) από την εντολή **lw**.

Hazards

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID	☁	☁	EX	M	WB												
sw			IF	☁	☁	ID	☠	☠	EX	M	WB									
sub				☁	☁	IF	☠	☠	ID	EX	M	WB								
sub				☁	☁		☠	☠	IF	ID	EX	M	WB							
bne				☁	☁		☠	☠		IF	ID	☁	☁	EX	M	WB				
lw				☁	☁		☠	☠				☁	☁			IF	ID	EX	M	WB

Rep:

lw \$2, 100 (\$3)

sub \$2, \$2, \$5

sw \$2, 100 (\$3)

sub \$3, \$3, \$6

sub \$1, \$1, \$7

bne \$1, \$0, Rep

Exit:

Stalls στους κύκλους 7,8:

Ο \$2 για την εντολή sw \$2,100(\$3)
γίνεται διαθέσιμος στο τέλος του κύκλου 8.

Hazards

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
lw	IF	ID	EX	M	WB																
sub		IF	ID	☁	☁	EX	M	WB													
sw			IF	☁	☁	ID	☁	☁	EX	M	WB										
sub				☁	☁	IF	☁	☁	ID	EX	M	WB									
sub				☁	☁		☁	☁	IF	ID	EX	M	WB								
bne				☁	☁		☁	☁		IF	ID	☁	☁	EX	M	WB					
lw				☁	☁		☁	☁				☁	☁				IF	ID	EX	M	WB

Rep:

```
lw $2,100($3)
sub $2,$2,$5
sw $2,100($3)
sub $3,$3,$6
sub $1,$1,$7
bne $1,$0, Rep
```

Exit:

Stalls στους κύκλους 12,13:

Ο \$1 για την εντολή bne \$1,\$0,Rep γίνεται διαθέσιμος στο τέλος του κύκλου 13.

Hazards

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID			EX	M	WB												
sw			IF			ID			EX	M	WB									
sub						IF			ID	EX	M	WB								
sub									IF	ID	EX	M	WB							
bne									IF	ID				EX	M	WB				
lw																IF	ID	EX	M	WB

Το δεύτερο στιγμιότυπο της εντολής lw \$2,100(\$3) αρχίζει να εκτελείται από τον κύκλο 16, διότι η απόφαση για την διακλάδωση ελήφθη στον κύκλο 15.

Υπολογισμός χρόνου

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID	cloud	cloud	EX	M	WB												
sw			IF	cloud	cloud	ID	cloud	cloud	EX	M	WB									
sub				cloud	cloud	IF	cloud	cloud	ID	EX	M	WB								
sub				cloud	cloud		cloud	cloud	IF	ID	EX	M	WB							
bne				cloud	cloud		cloud	cloud		IF	ID	cloud	cloud	EX	M	WB				
lw				cloud	cloud		cloud	cloud				cloud	cloud			IF	ID	EX	M	WB



15 κύκλοι ρολογιού

Για τα πρώτα 99 loops έχουμε $99 \times 15cc = 1485cc$.

Για την 100^η επανάληψη έχουμε 16cc.

Συνολικά απαιτούνται $1485cc + 16cc = 1501cc$ για την εκτέλεση του βρόχου.

Pipeline 2

Για την ίδια ακολουθία εντολών, δείξτε και εξηγήστε τον χρονοισμό του pipeline, θεωρώντας τώρα ότι **υπάρχει σχήμα προώθησης**. Θεωρείστε ότι οι αποφάσεις για τις διακλαδώσεις λαμβάνονται στο στάδιο MEM.

Πόσοι κύκλοι απαιτούνται συνολικά για να ολοκληρωθεί ο βρόχος;

Rep:

```
lw $2, 100($3)
sub $2, $2, $5
sw $2, 100($3)
sub $3, $3, $6
sub $1, $1, $7
bne $1, $0, Rep
```

Exit:

Με / Χωρίς προώθηση

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID			EX	M	WB												
sw			IF			ID			EX	M	WB									
sub						IF			ID	EX	M	WB								
sub									IF	ID	EX	M	WB							
bne									IF	ID				EX	M	WB				
lw																IF	ID	EX	M	WB

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID		EX	M	WB													
sw			IF		ID	EX	M	WB												
sub					IF	ID	EX	M	WB											
sub						IF	ID	EX	M	WB										
bne							IF	ID	EX	M	WB									
lw											IF	ID	EX	M	WB					

Hazards

Rep:

lw \$2, 100 (\$3)

sub \$2, \$2, \$5

sw \$2, 100 (\$3)

sub \$3, \$3, \$6

sub \$1, \$1, \$7

bne \$1, \$0, Rep

Exit:

Στον κύκλο 4 υπάρχει stall. **Γιατί;**

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw	IF	ID	EX	MEM	WB										
sub		IF	ID		EX	MEM	WB								
sw			IF		ID	EX	MEM	WB							
sub					IF	ID	EX	MEM	WB						
sub						IF	ID	EX	MEM	WB					
bne							IF	ID	EX	MEM	WB				
lw											IF	ID	EX	MEM	WB

Hazards







Η τιμή της θέσης μνήμης 100(\$3), που θα αποθηκευτεί στον \$2, δεν μπορεί να είναι διαθέσιμη πριν το στάδιο **MEM**.

Όταν όμως θα γίνει διαθέσιμη στο τέλος του κύκλου αυτού θα προωθηθεί στις εισόδους της ALU, ώστε να εκτελεστεί η εντολή sub \$2,\$2,\$5 χωρίς να περιμένουμε να γραφτεί η τιμή στον \$2.

Αποφεύγεται το stall που είχαμε στην προηγούμενη περίπτωση στην κύκλο 5 😊

```
Rep:
lw $2, 100($3)
sub $2, $2, $5
sw $2, 100($3)
sub $3, $3, $6
sub $1, $1, $7
bne $1, $0, Rep
```

Exit:

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
lw	IF	ID	EX	MEM	WB										
sub		IF	ID		EX	MEM	WB								
sw			IF		ID	EX	MEM	WB							
sub					IF	ID	EX	MEM	WB						
sub						IF	ID	EX	MEM	WB					
bne							IF	ID	EX	MEM	WB				
lw											IF	ID	EX	MEM	WB

Το ίδιο ισχύει και για τα stalls που είχαμε στους κύκλους 7, 8

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID			EX	M	WB												
sw			IF			ID			EX	M	WB									
sub						IF			ID	EX	M	WB								
sub									IF	ID	EX	M	WB							
bne										IF	ID			EX	M	WB				
lw																IF	ID	EX	M	WB

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID		EX	M	WB													
sw			IF		ID	EX	M	WB												
sub					IF	ID	EX	M	WB											
sub						IF	ID	EX	M	WB										
bne							IF	ID	EX	M	WB									
lw											IF	ID	EX	M	WB					

Το ίδιο ισχύει και για τα stalls που είχαμε στους κύκλους 12, 13.

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID			EX	M	WB												
sw			IF			ID			EX	M	WB									
sub						IF			ID	EX	M	WB								
sub									IF	ID	EX	M	WB							
bne										IF	ID			EX	M	WB				
lw																IF	ID	EX	M	WB

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID		EX	M	WB													
sw			IF		ID	EX	M	WB												
sub					IF	ID	EX	M	WB											
sub						IF	ID	EX	M	WB										
bne							IF	ID	EX	M	WB									
lw											IF	ID	EX	M	WB					

Υπολογισμός χρόνου

CC	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw	IF	ID	EX	M	WB															
sub		IF	ID	☁	EX	M	WB													
sw			IF	☁	ID	EX	M	WB												
sub				☁	IF	ID	EX	M	WB											
sub				☁		IF	ID	EX	M	WB										
bne				☁			IF	ID	EX	M	WB									
lw				☁							IF	ID	EX	M	WB					



10 κύκλοι ρολογιού

Για τα πρώτα 99 loops έχουμε $99 \times 10cc = 990cc$.

Για την 100η επανάληψη έχουμε 11cc.

Συνολικά απαιτούνται $990cc + 11cc = 1001cc$ για την εκτέλεση του βρόχου.

Χωρίς την προώθηση χρειαστήκαμε **1501cc**.