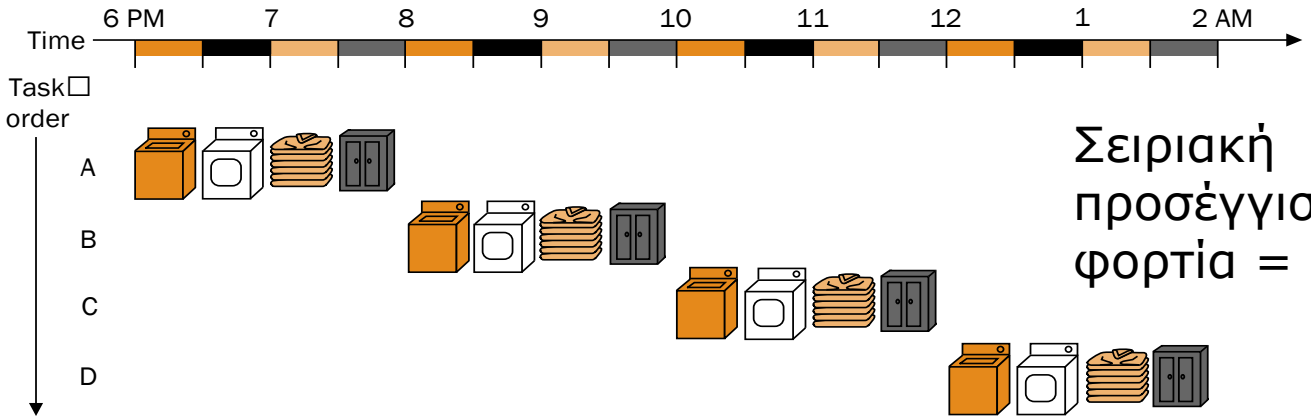
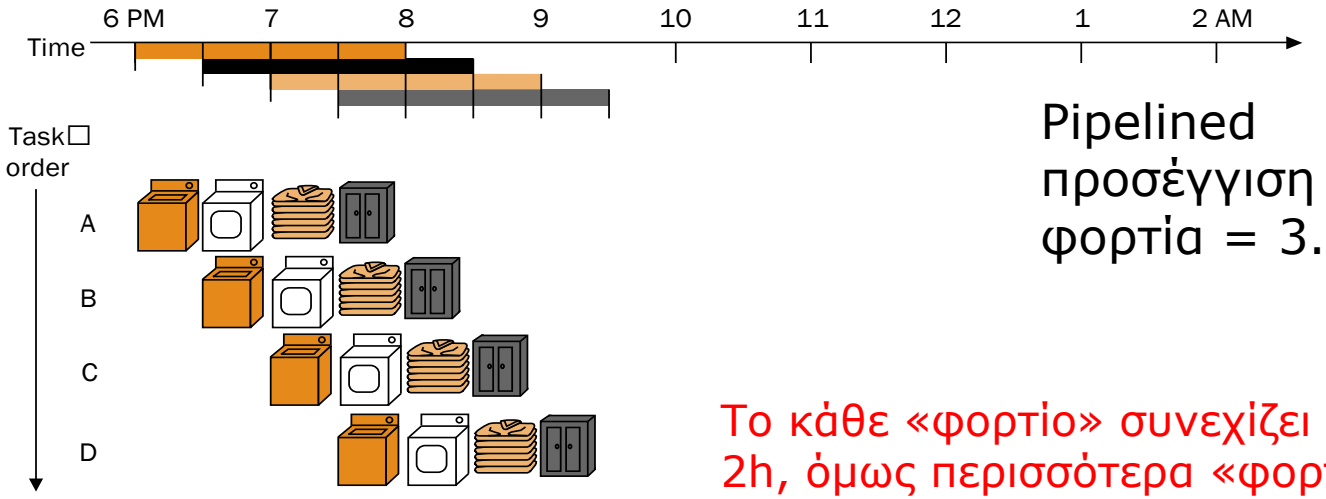
Pipeline: Ένα παράδειγμα από ….τη καθημερινή ζωή

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



Σειριακή προσέγγιση για 4 φορτία = 8h

30 min κάθε «φάση»



Pipelined προσέγγιση για 4 φορτία = 3.5h

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

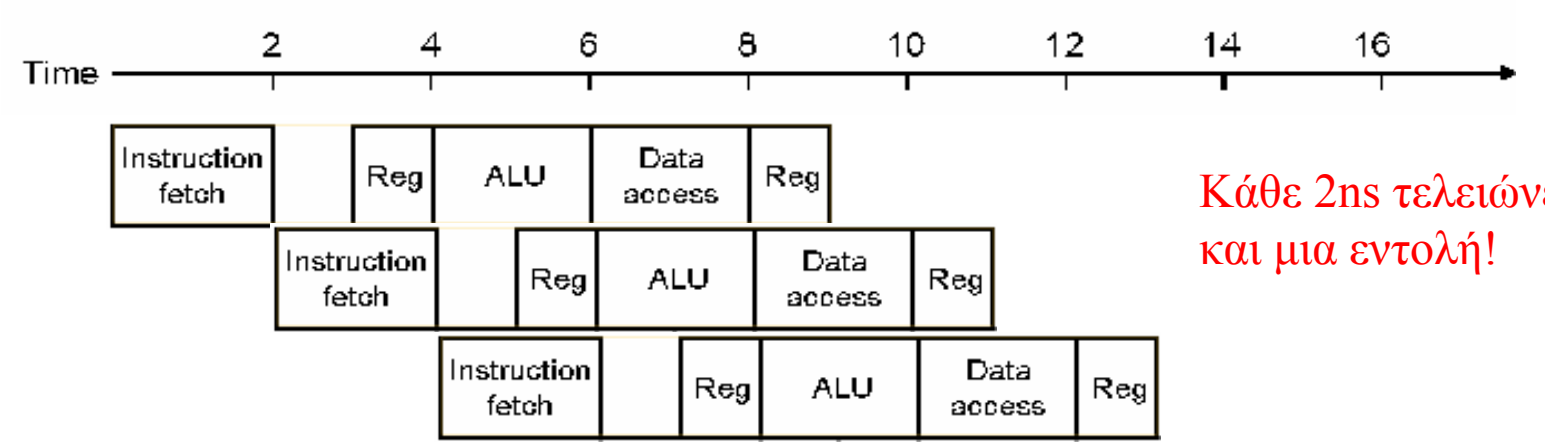
Instruction class	Instruct. fetch	Register read	ALU operation	Data access	Register Write	Total Time
Load word (lw)	2ns	1ns	2ns	2ns	1ns	8ns
Store word (sw)	2ns	1ns	2ns	2ns		7ns
R-Type (add,sub,and, or, slt)	2ns	1ns	2ns		1ns	6ns
Branch(beq)	2ns	1ns	2ns			5ns

Έστω οι παρακάτω εντολές lw:

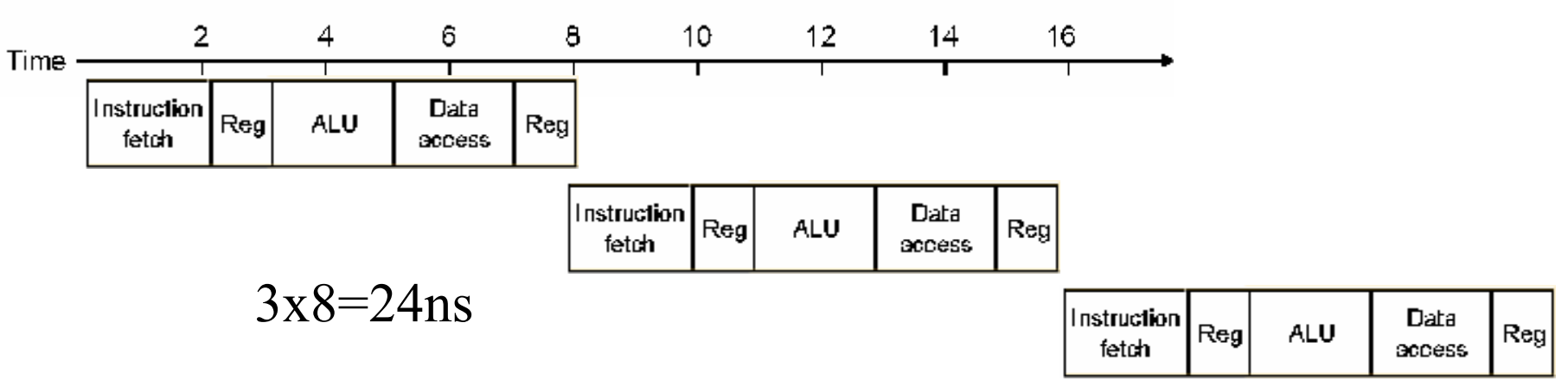
lw $1, 100($0)

lw $2, 200($0)

lw $3, 300($0)



Κάθε 2ns τελειώνει και μια εντολή!



3x8=24ns

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

$time\_between\_instructions_{pipelined} =$

$time\_between\_instructions_{non\_pipelined} /$ 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

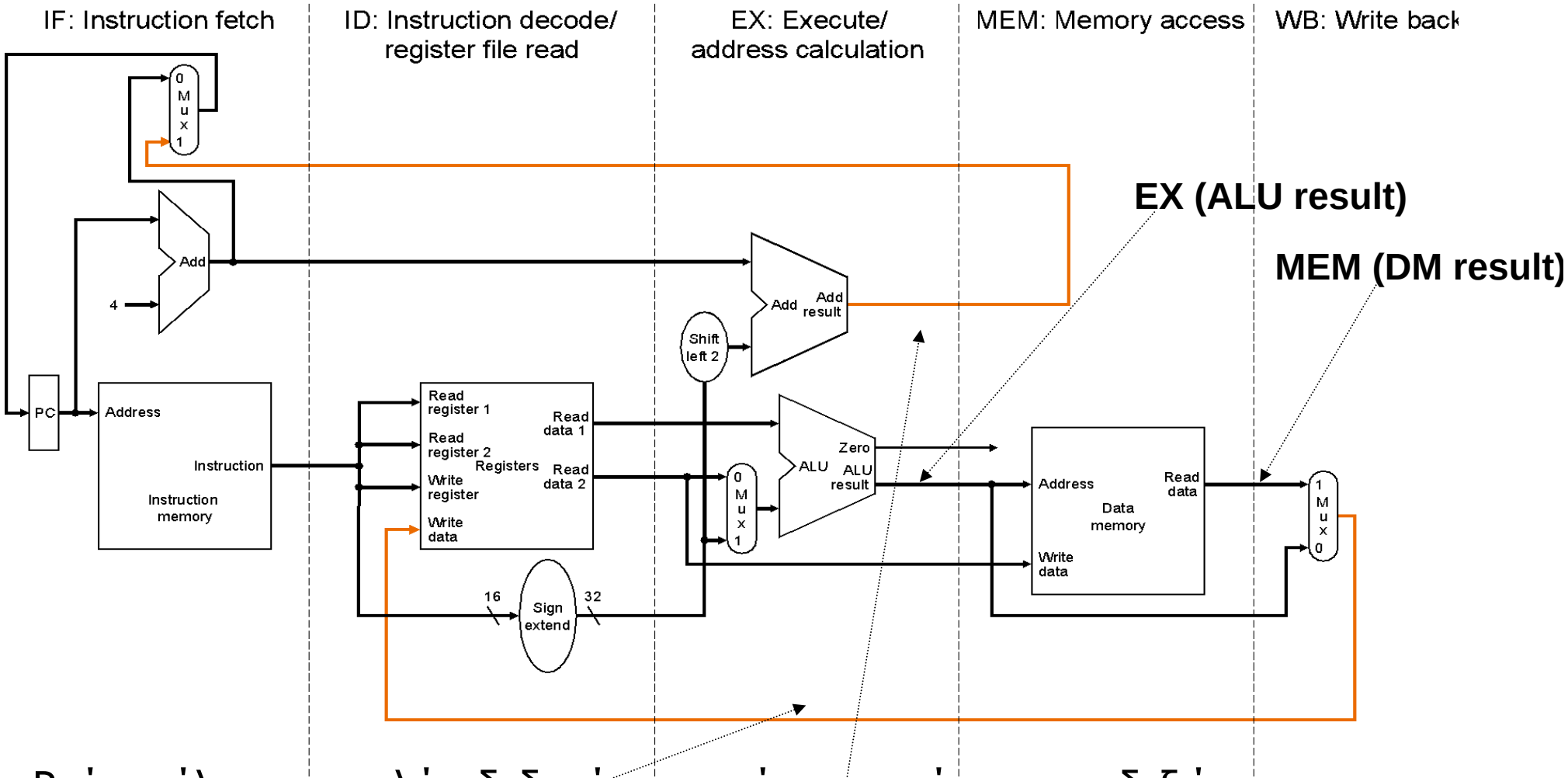
Στο προηγούμενο παράδειγμα 14 ns για pipeline, 24ns για single cycle, άρα 1,71 επιτάχυνση.

Άν είχαμε 1000 εντολές ακόμα: pipeline 1000x2ns + 14 ns = 2014 ns

Single cycle 1000x8ns + 24 ns = 8024 ns

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

Στάδια διόδου δεδομένων ενός κύκλου (single cycle datapath):

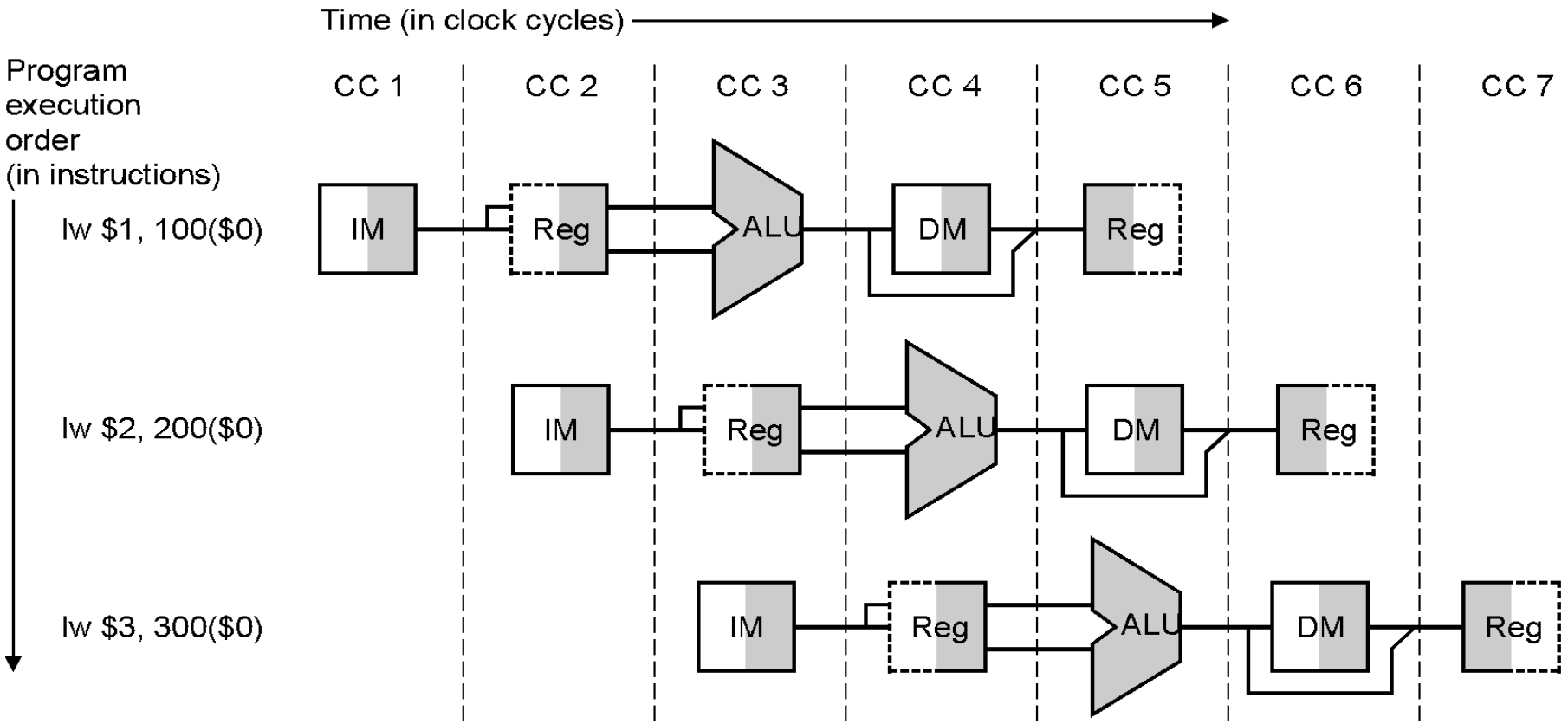


Ροή εκτέλεσης εντολών-δεδομένων: από αριστερά προς τα δεξιά

Εξαίρεση? Write-back και επιλογή νέου PC

κίνδυνος δεδομένων (data hazard) κίνδυνος ελέγχου (control hazard)

Εκτέλεση αγωγού (pipelined execution)

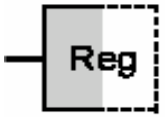


Τι θα γίνει αν χρησιμοποιούμε την ίδια λειτουργική μονάδα (Functional Unit), π.χ. IM, RegFile, ALU, DM σε διαφορετικούς κύκλους για διαφορετικές εντολές?

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)

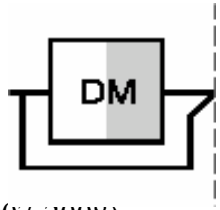


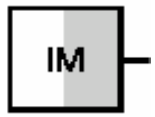
και στο δεύτερο μισό διαβάζουμε (σκιασμένο δεξιά)

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

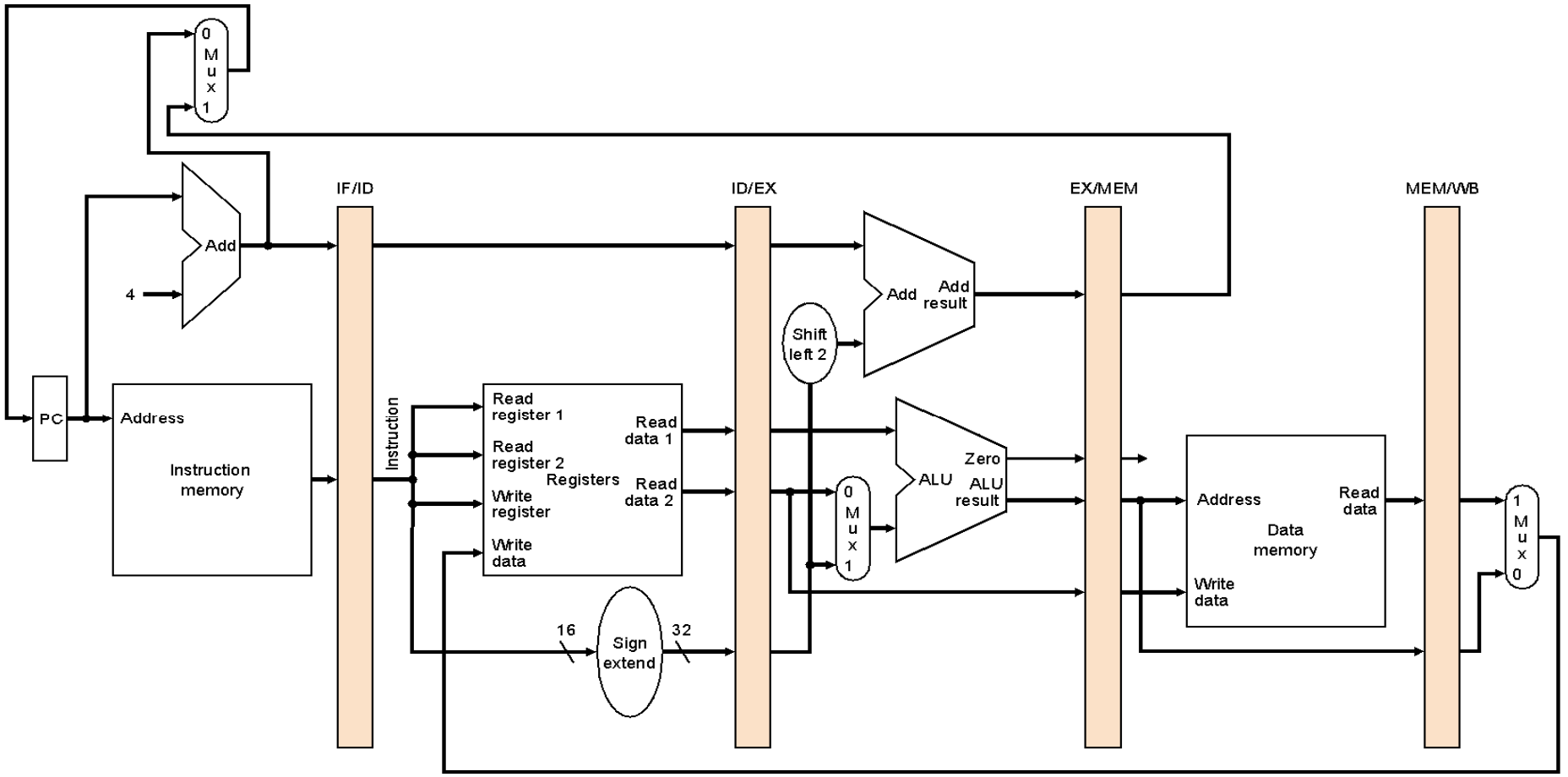
Στην pipelined υλοποίηση του datapath, έχουμε την ίδια μονάδα να χρησιμοποιείται από διαφορετικές (διαδοχικές) εντολές σε διαφορετικούς (διαδοχικούς) κύκλους

Πώς διασφαλίζεται ορθότητα εκτέλεσης κάθε εντολής;

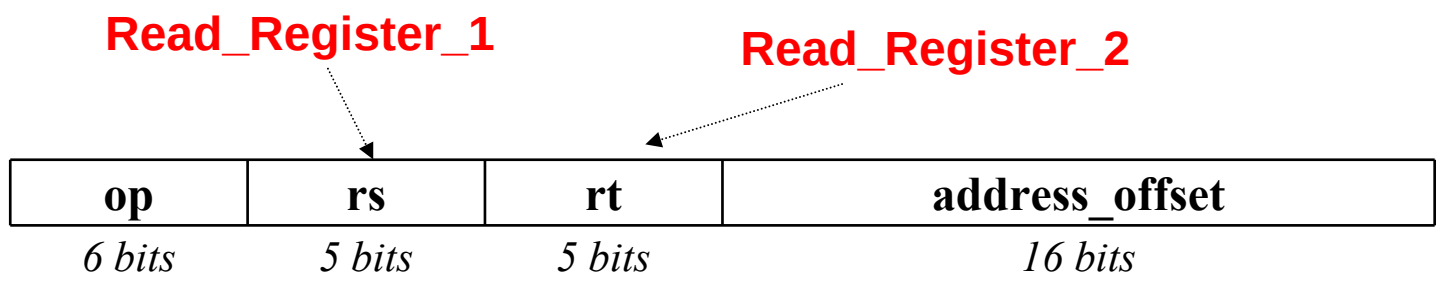
Καταχωρητές *κατάλληλου;;;* μεγέθους ανάμεσα σε διαδοχικά στάδια (pipeline stages)

Pipelined εκδοχή του single cycle datapath

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

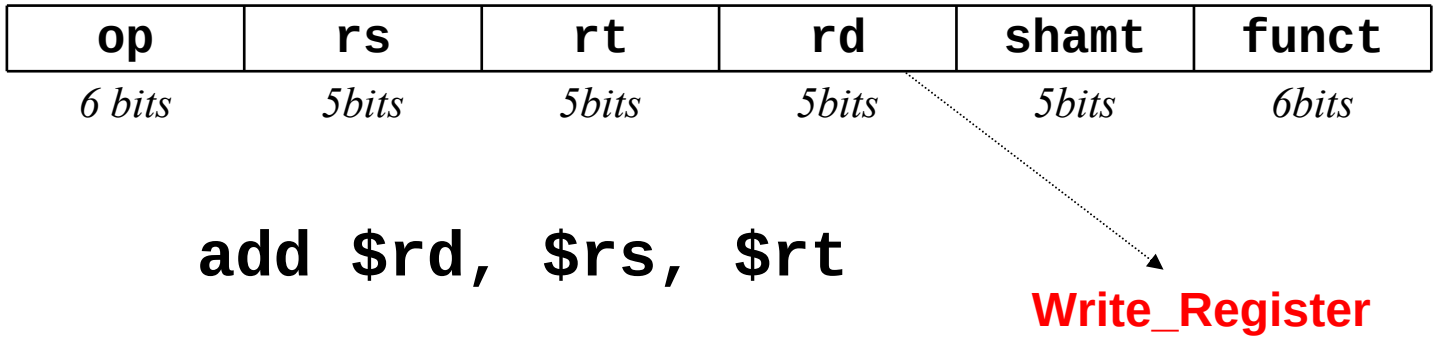


I-Type:




```
lw $rt, address_offset($rs)
```

R-Type:
(register type)




```
add $rd, $rs, $rt
```

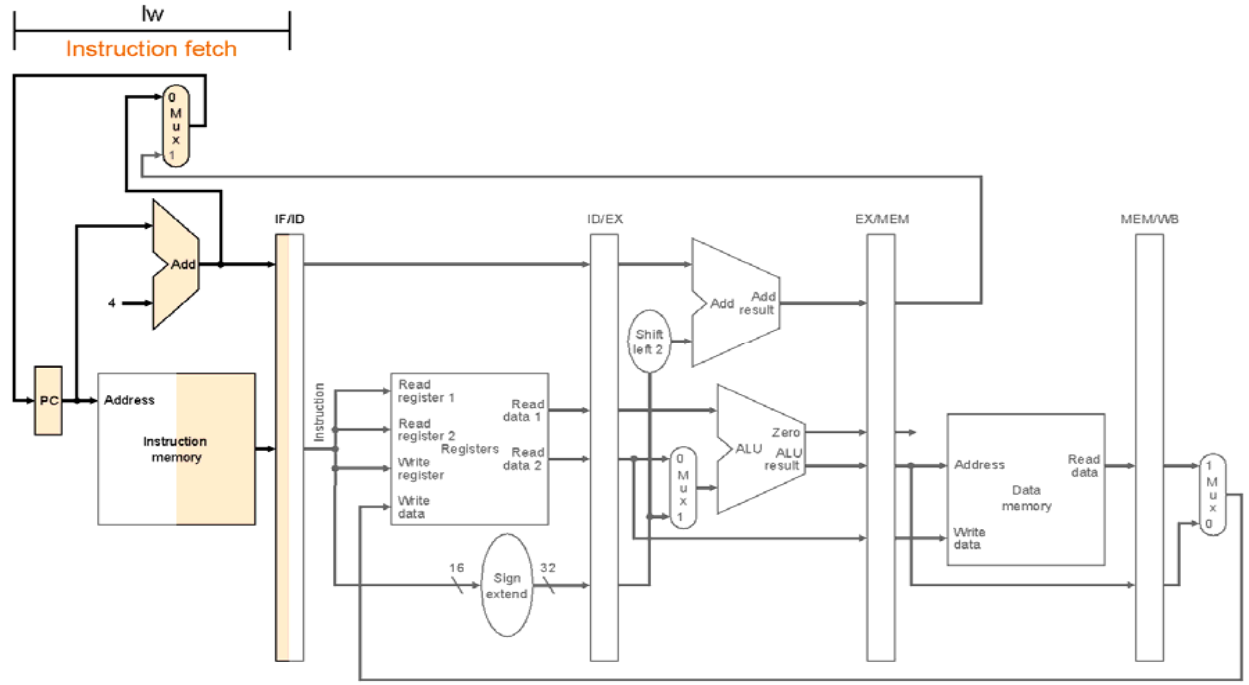
`Op`: opcode

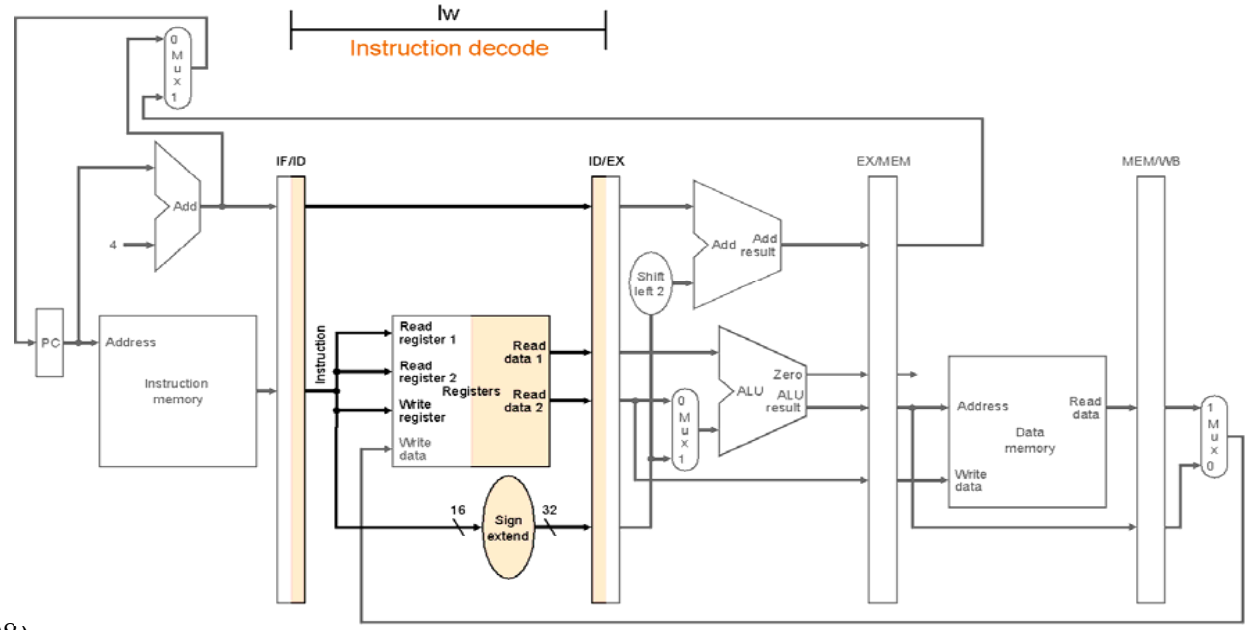
`rs`,`rt`: register source operands

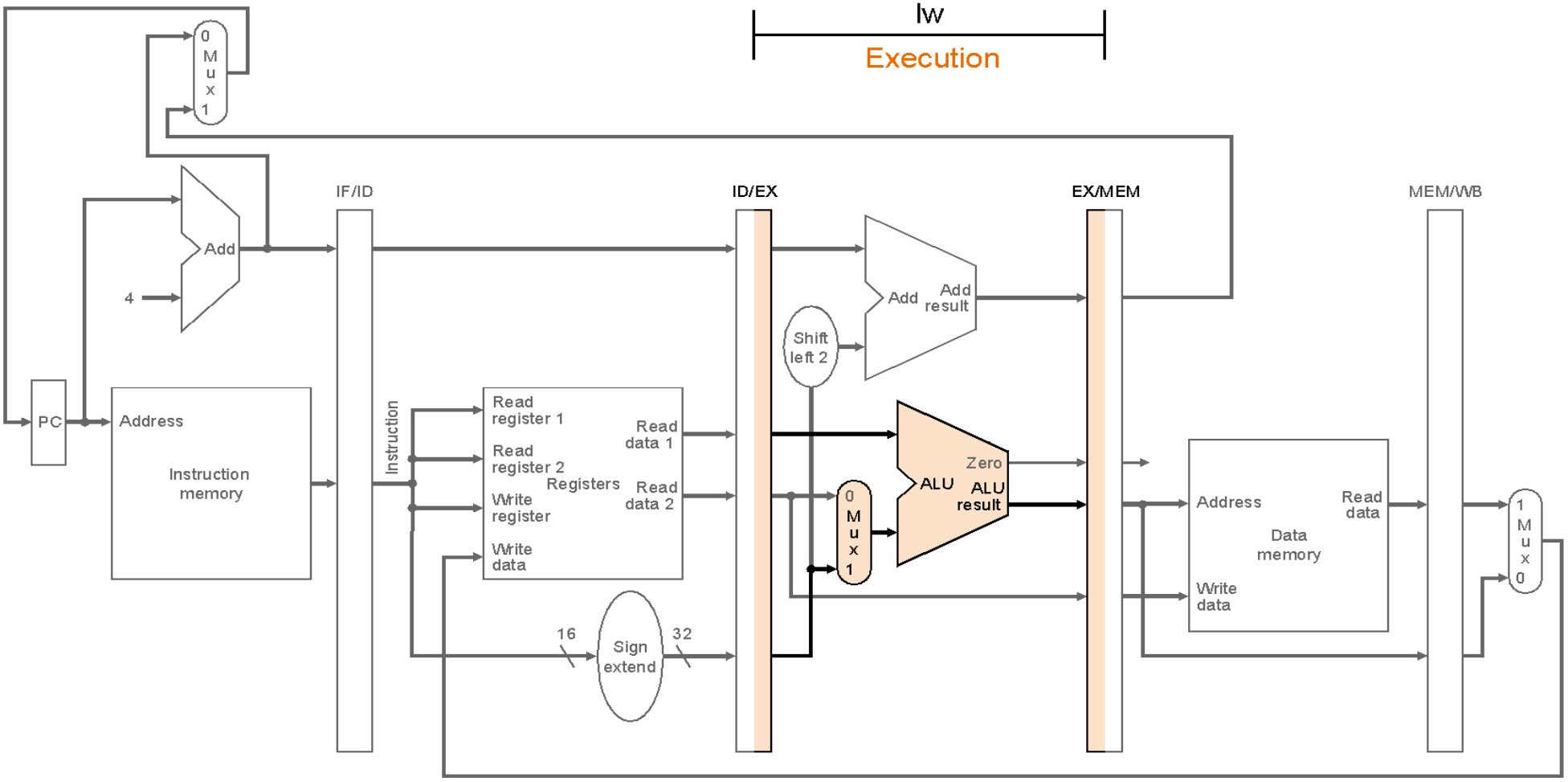
`Rd`: register destination operand

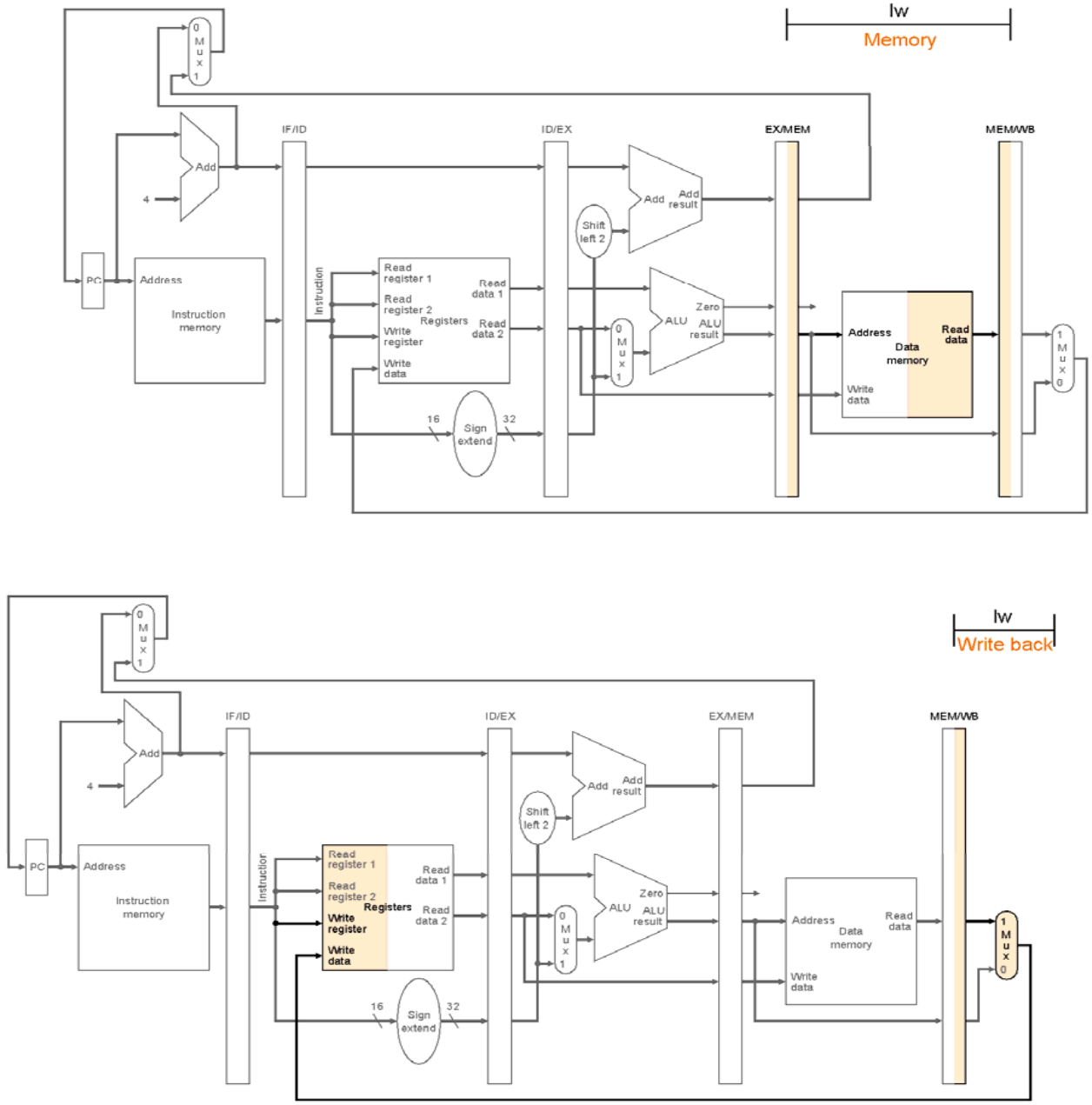
`Shamt`: shift amount

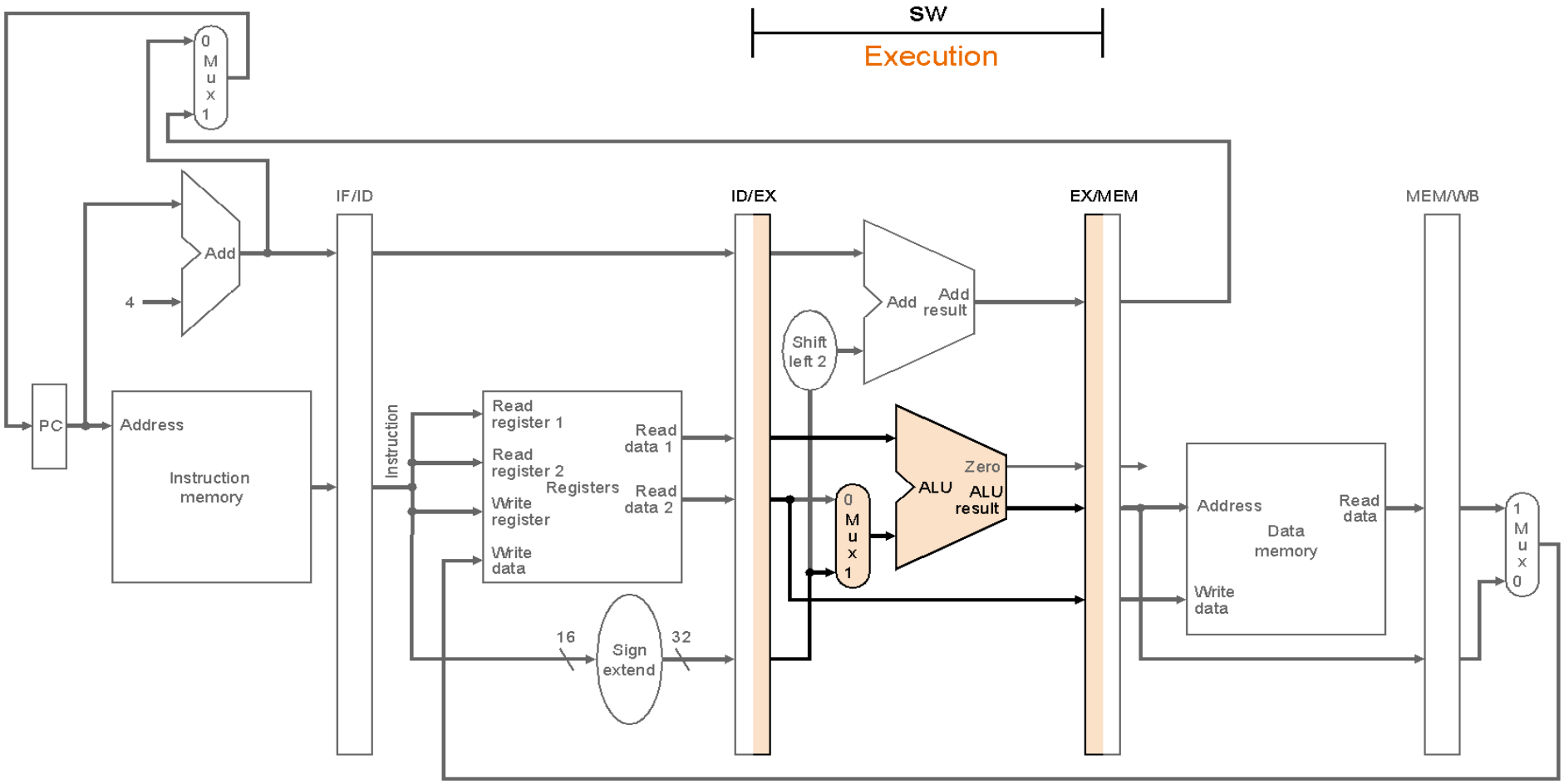
`Funct`: op specific (function code)

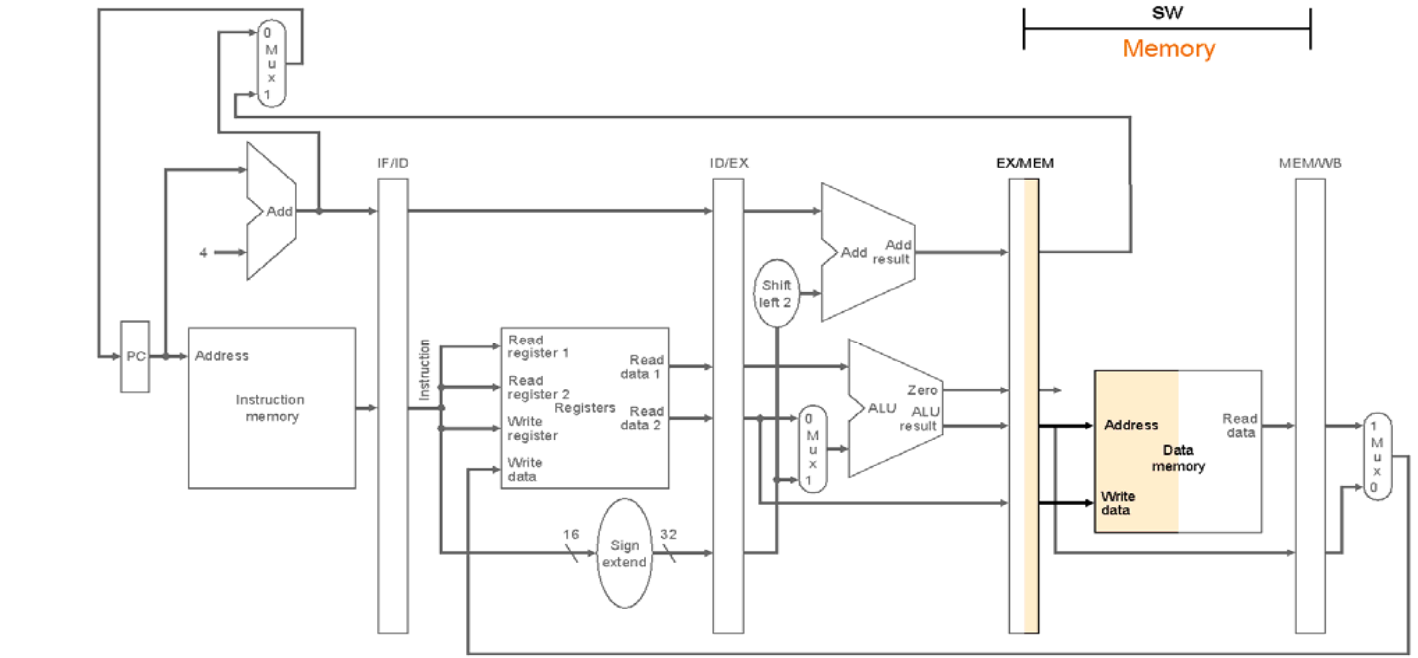

lw
Instruction fetch
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
M u x

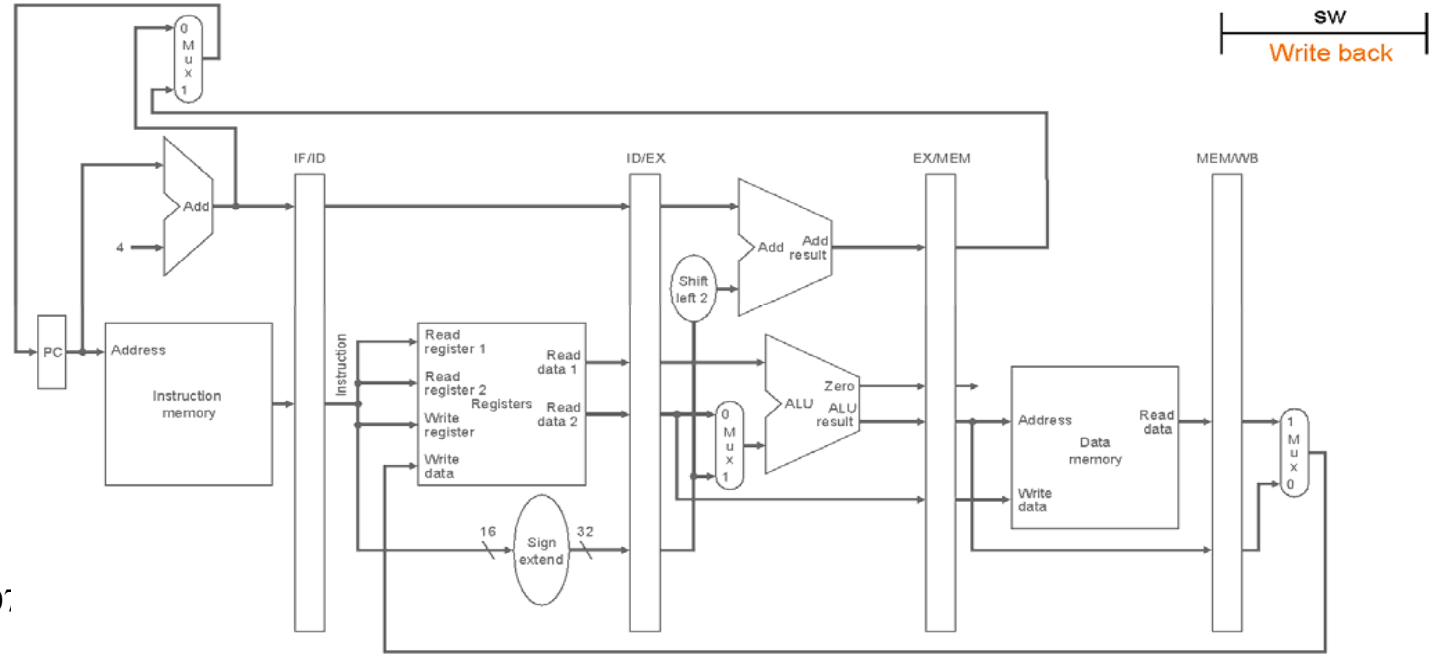

lw
Instruction decode
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
M u x


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

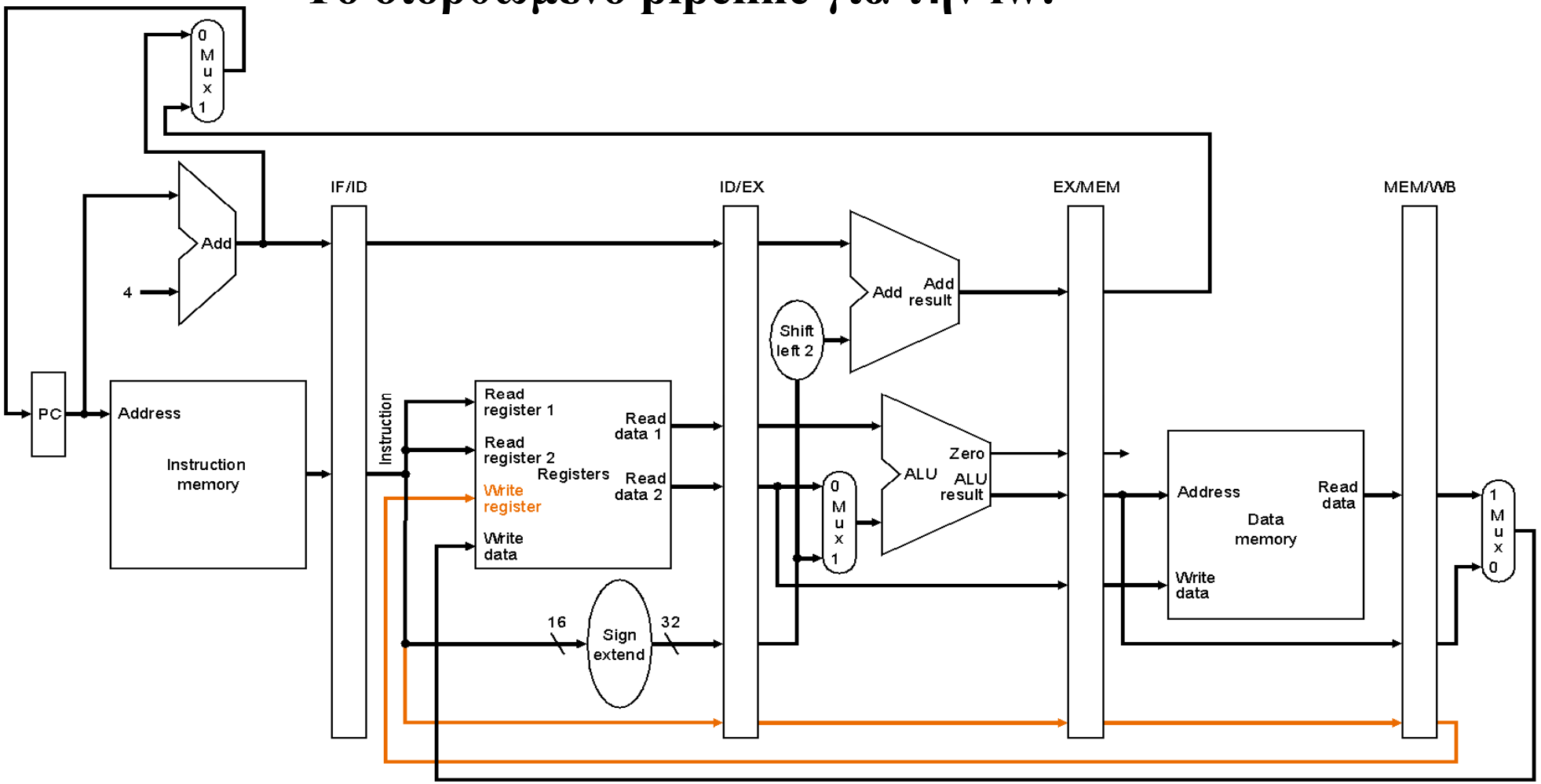

lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
M u x
0
1
lw
Write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


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


SW
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
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
Data memory
Read data
Write data
16
Sign extend
32
M u x

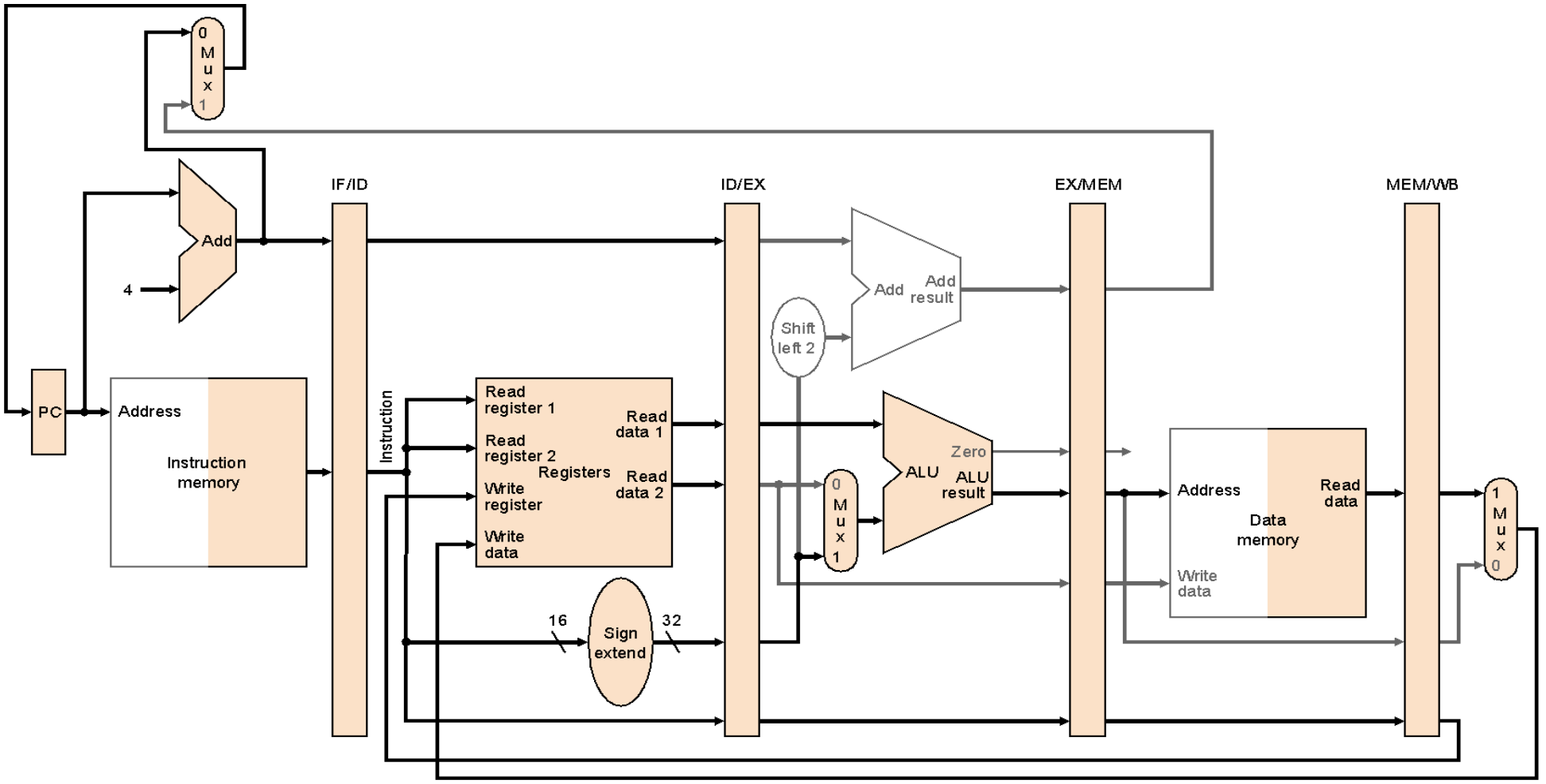

SW
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
M u x

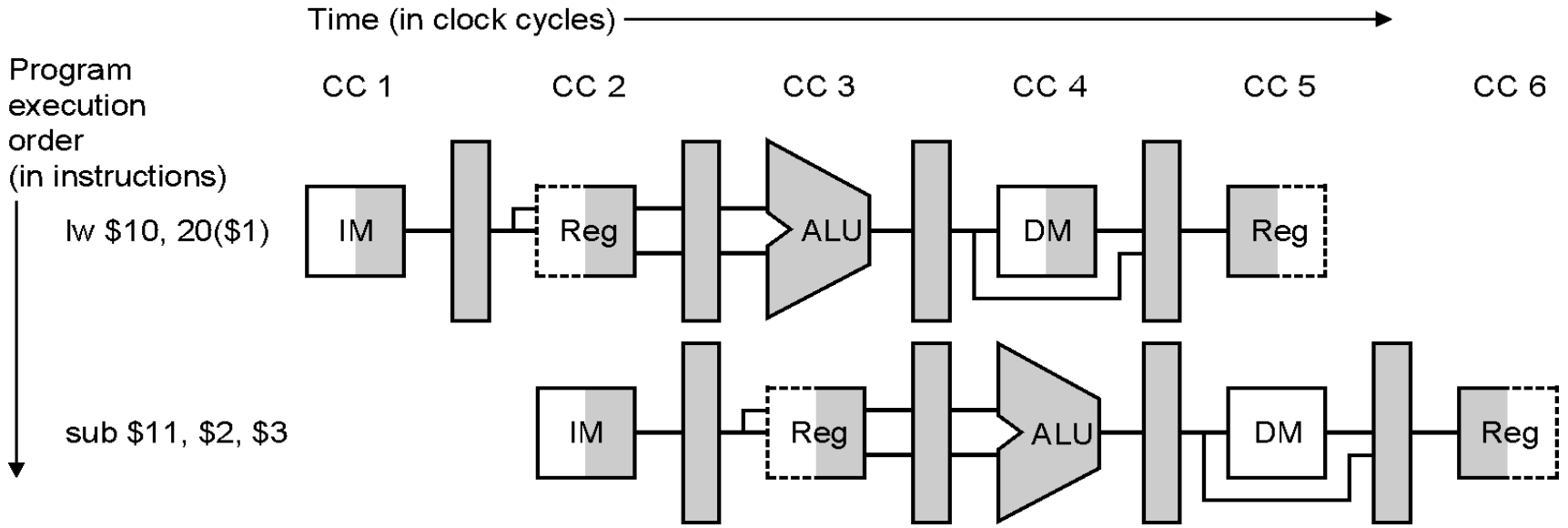
Το διορθωμένο pipeline για την lw:

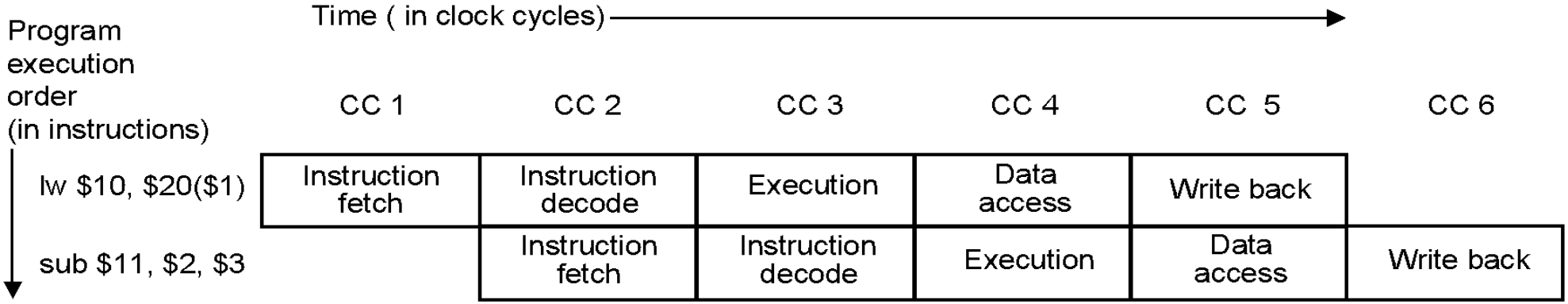


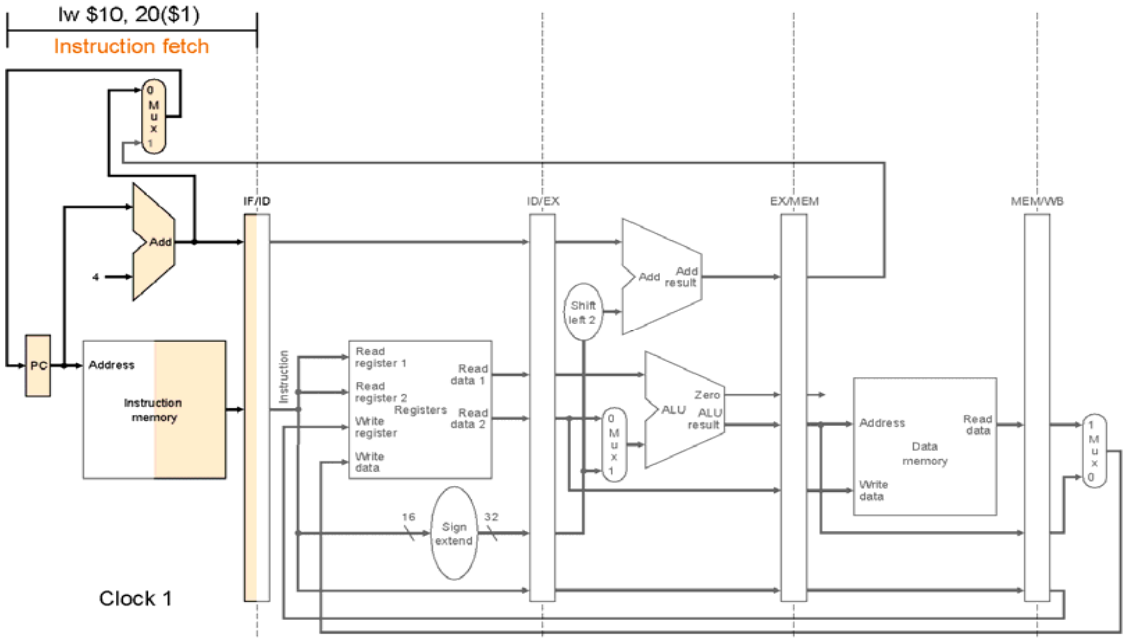
Ο αριθμός του write register έρχεται και αυτός μέσα από το pipeline τη σωστή στιγμή

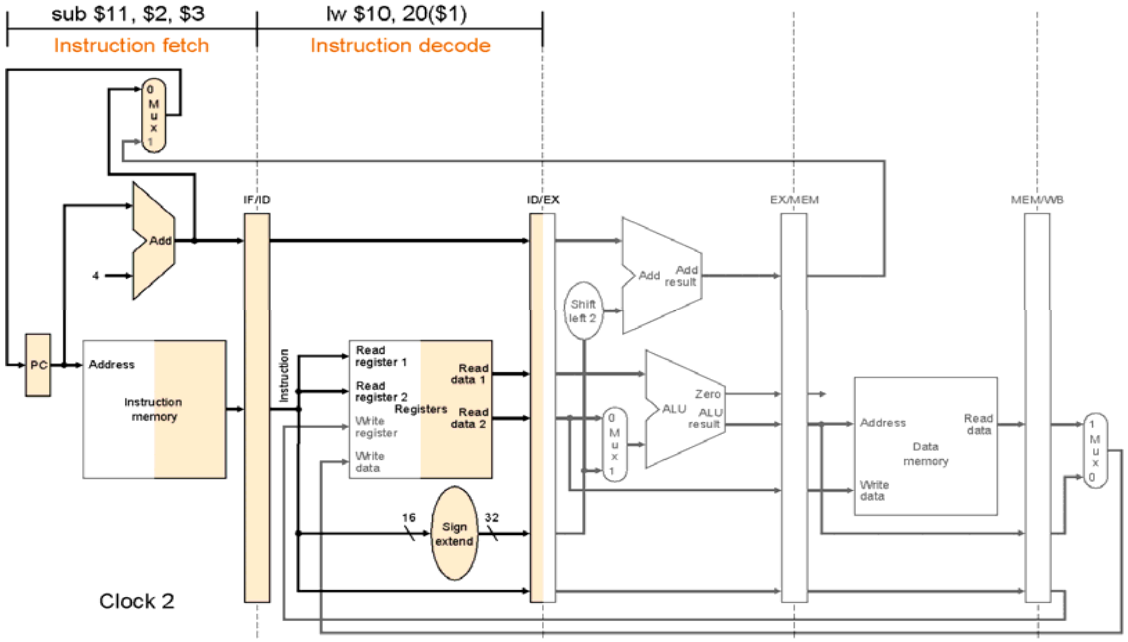
Τα τμήματα του datapath που χρησιμοποιήθηκαν κατά την εκτέλεση της lw:

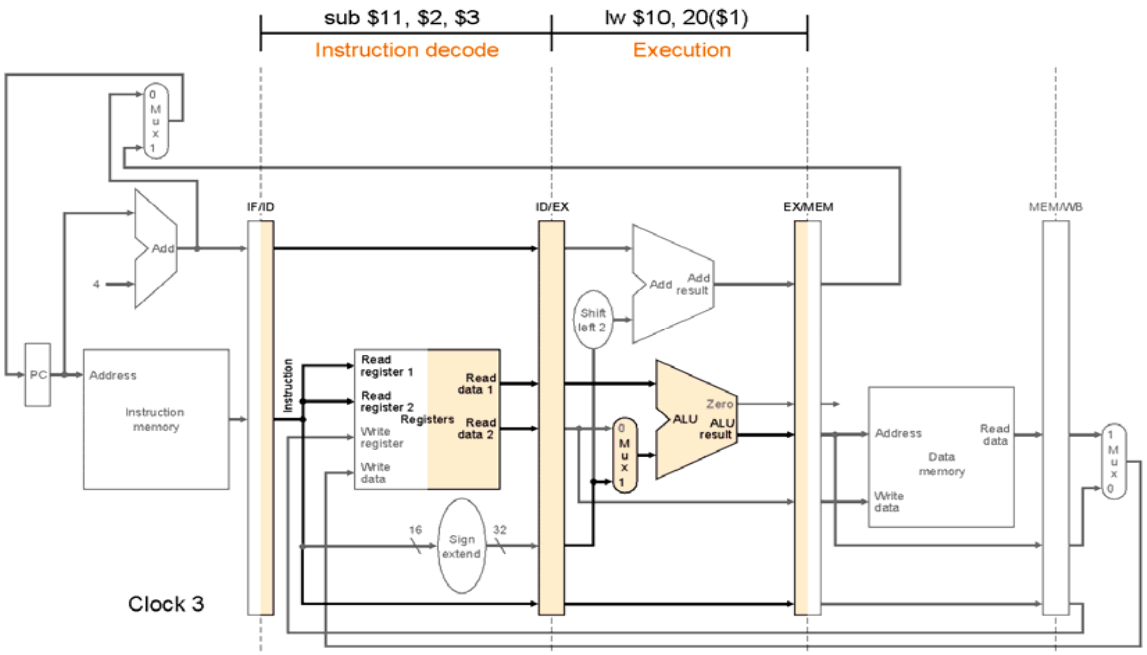


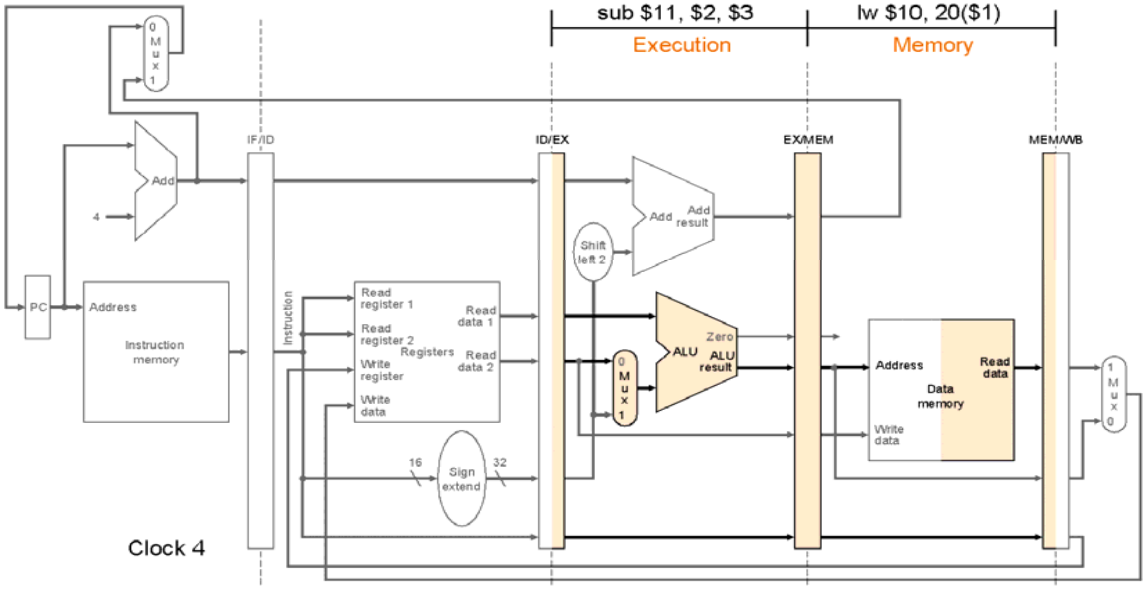

Time (in clock cycles)
Program execution order (in instructions)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
lw $10, 20($1)
IM
Reg
ALU
DM
Reg
sub $11, $2, $3
IM
Reg
ALU
DM
Reg

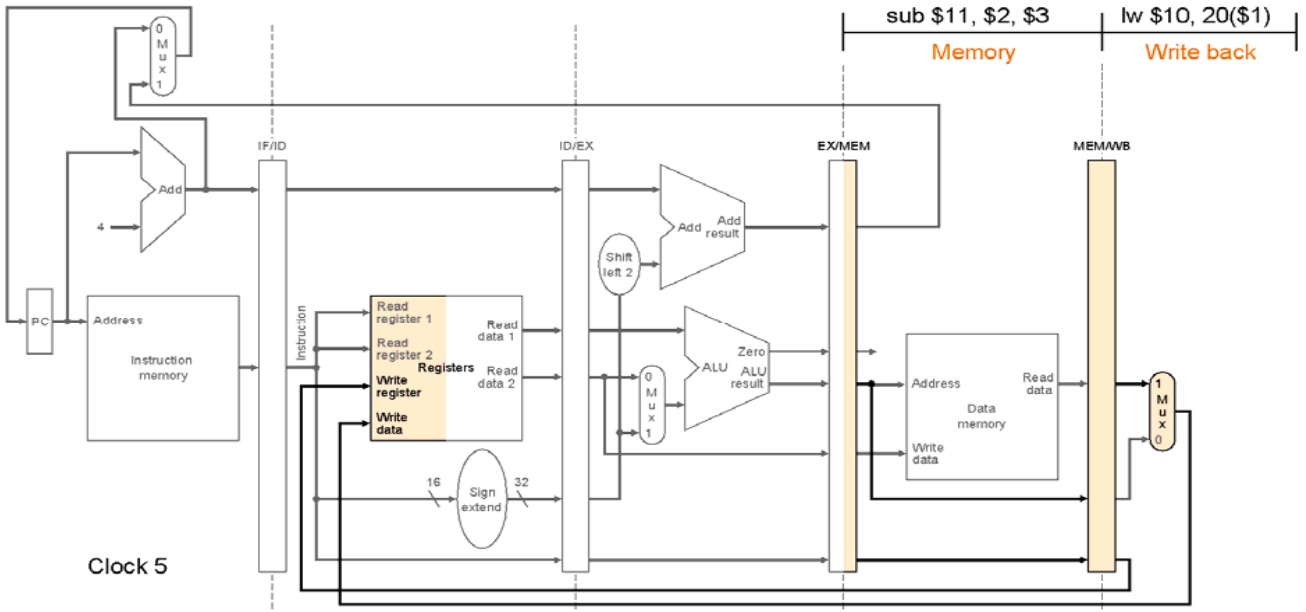

Time (in clock cycles)
Program execution order (in instructions)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
lw $10, $20($1)
Instruction fetch
Instruction decode
Execution
Data access
Write back
sub $11, $2, $3
Instruction fetch
Instruction decode
Execution
Data access
Write back

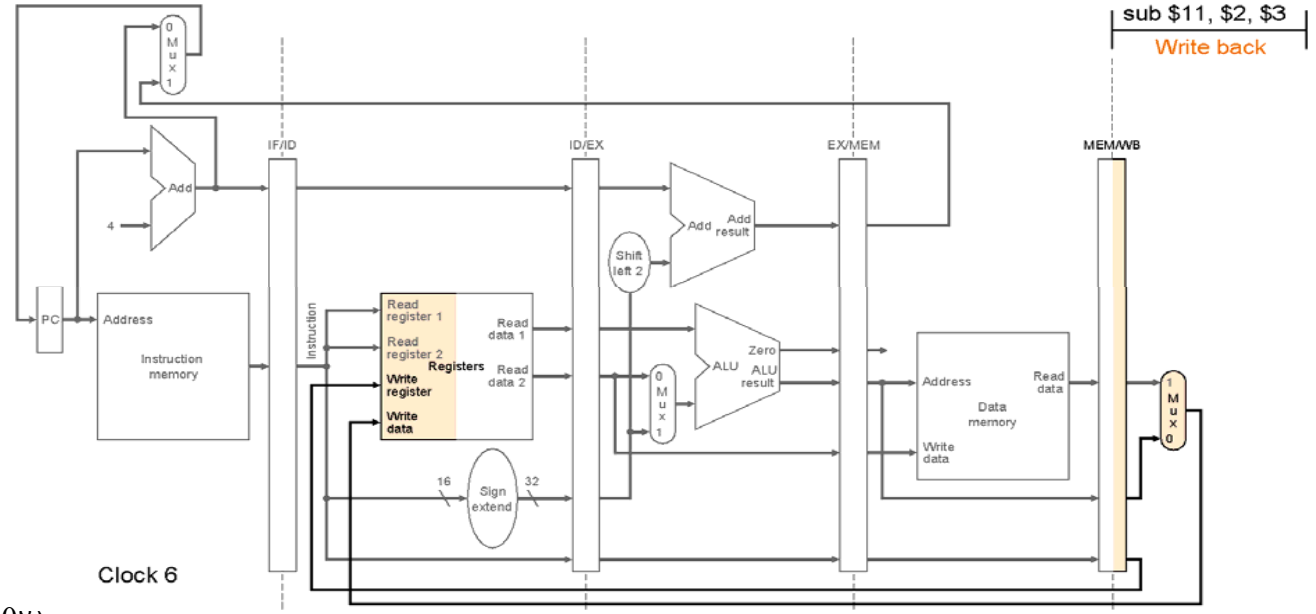

lw $10, 20($1)
Instruction fetch
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign extend
32
Address
Data memory
Read data
Write data
Clock 1


sub $11, $2, $3
Instruction fetch
lw $10, 20($1)
Instruction decode
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
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
16
Sign extend
32
Address
Data memory
Read data
Write data
Clock 2


sub $11, $2, $3
lw $10, 20($1)
Instruction decode
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
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 result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
Clock 3


sub $11, $2, $3
lw $10, 20($1)
Execution
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
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 result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
16
Sign extend
32
Clock 4


sub $11, $2, $3
lw $10, 20($1)
Memory
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
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
Data memory
Read data
Write data
16
Sign extend
32
M u x
Clock 5


sub $11, $2, $3
Write back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
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
Data memory
Read data
Write data
16
Sign extend
32
M u x
Clock 6

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)

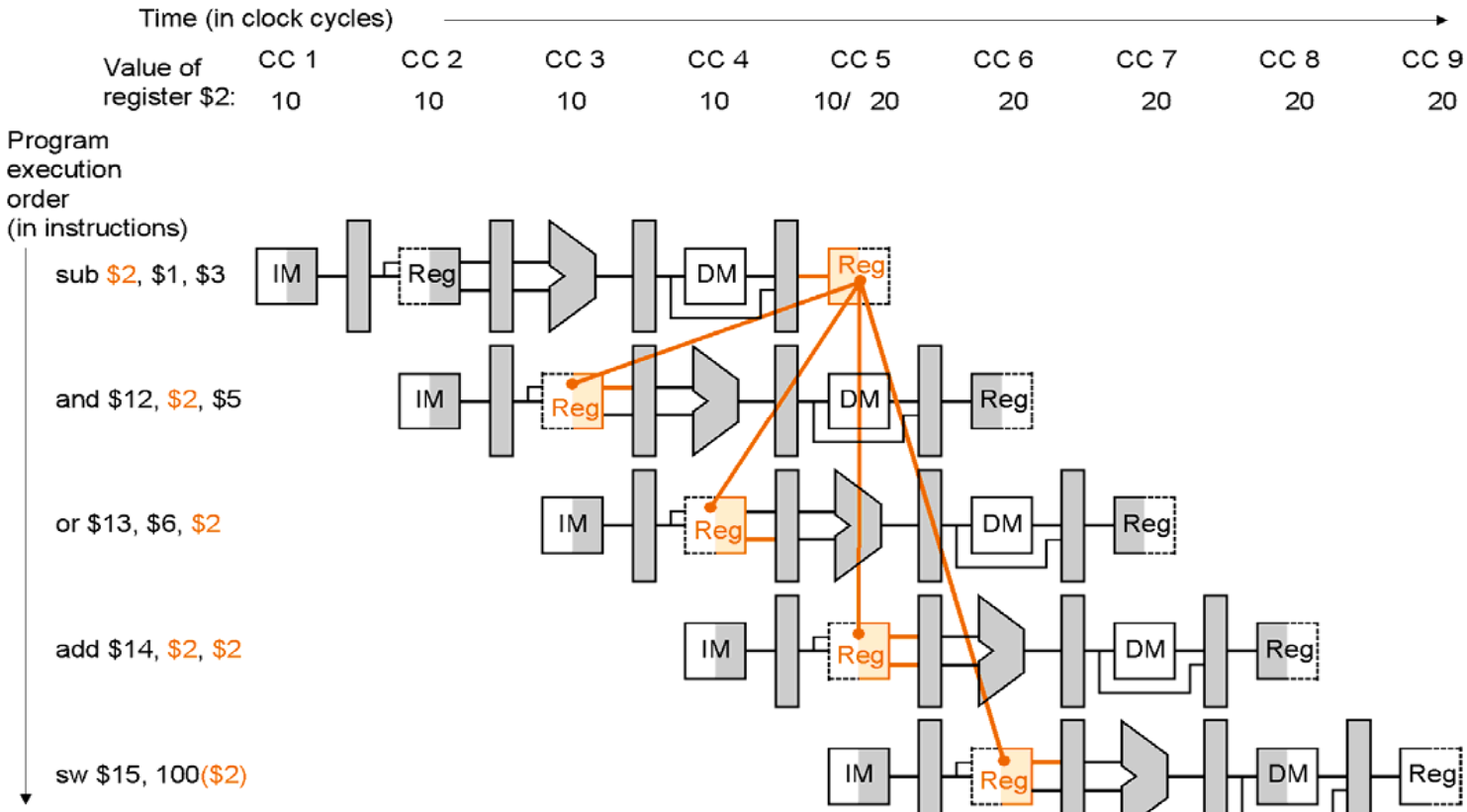
- Κίνδυνοι Ελέγχου (control hazards)

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

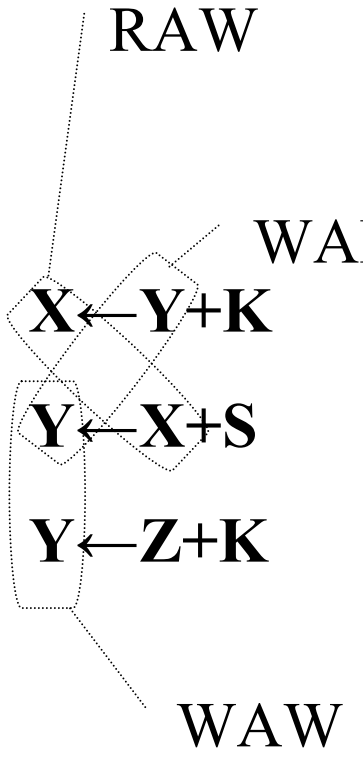
- Κίνδυνοι Δεδομένων (data hazards)

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

```
sub    $2,  $1, $3   # καταχωρητής $2 γράφεται από τη sub
and    $12, $2, $5   # 1ος τελεστέος ($2) εξαρτάται από sub
or     $13, $6, $2   # 2ος τελεστέος ($2) εξαρτάται από sub
add    $14, $2, $2   # 1ος&2ος τελεστέος $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 –

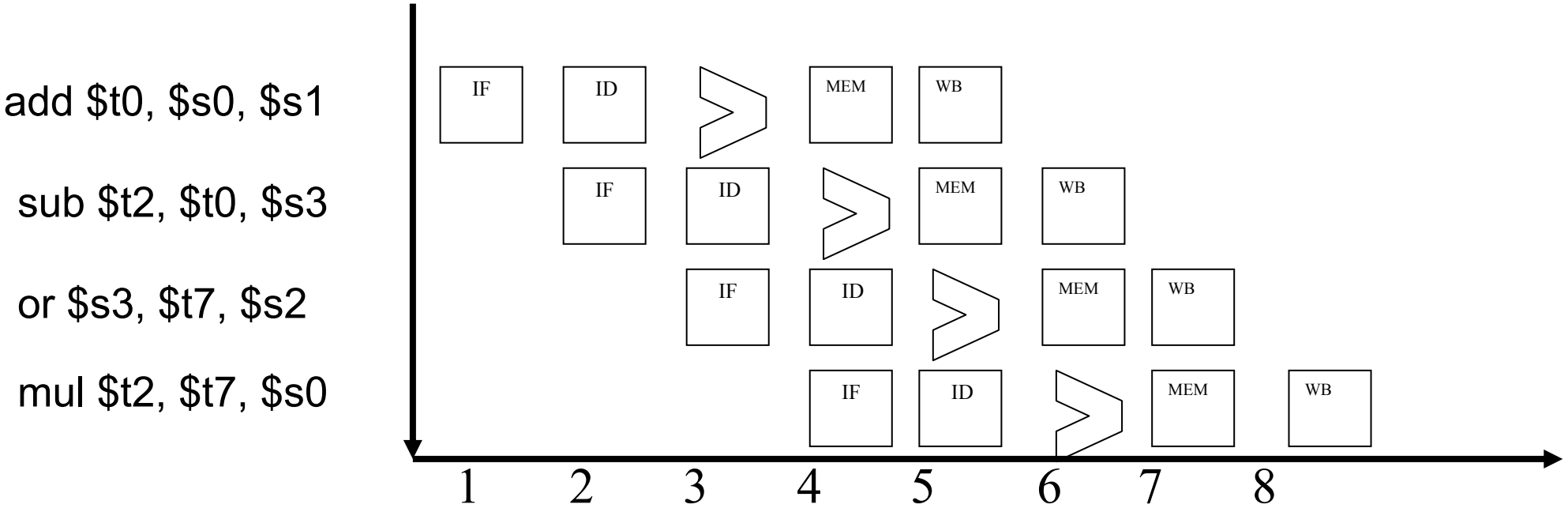
WAW -

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

name dependence -
output dependence –

Identify all of the dependencies

RAW
WAR
WAW



Which dependencies cause hazards? (stalls)

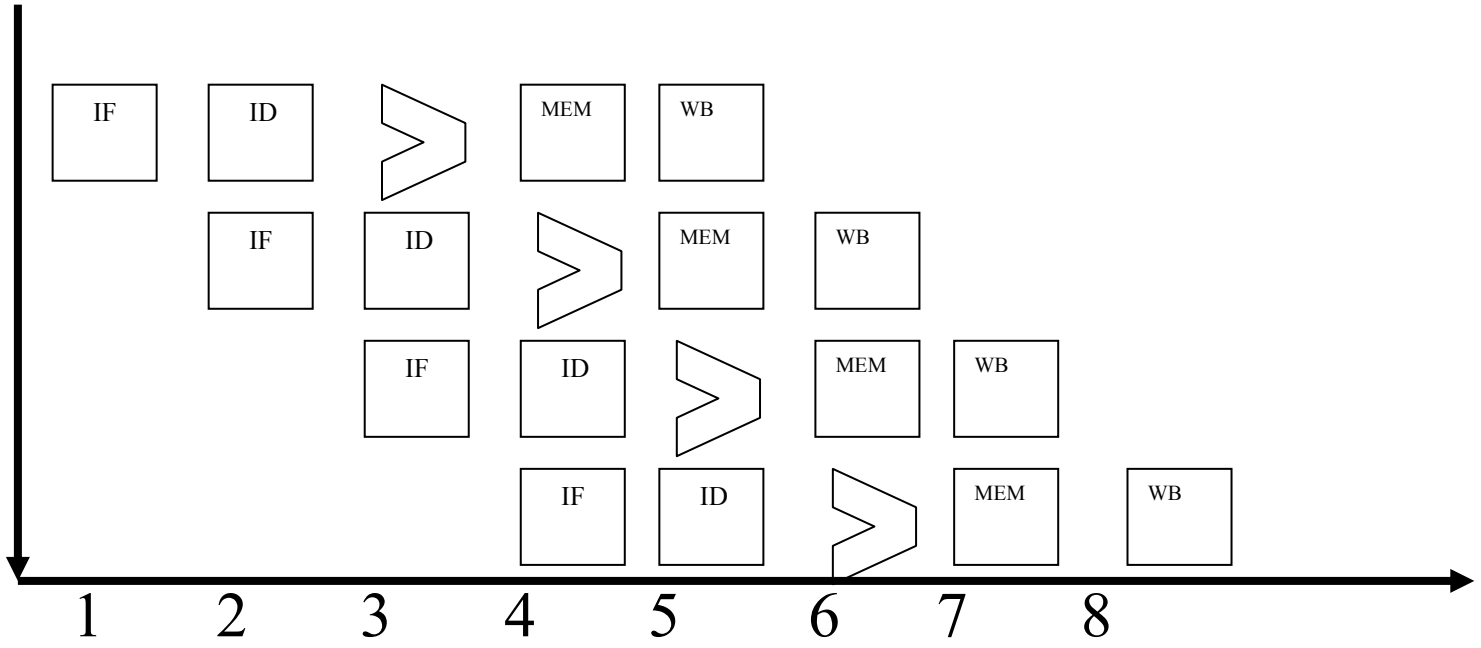
RAW
WAR
WAW

add $t0, $s0, $s1

sub $t2, $t0, $s3

or $s3, $t7, $s2

mul $t2, $t7, $s0



Let's reorder the or

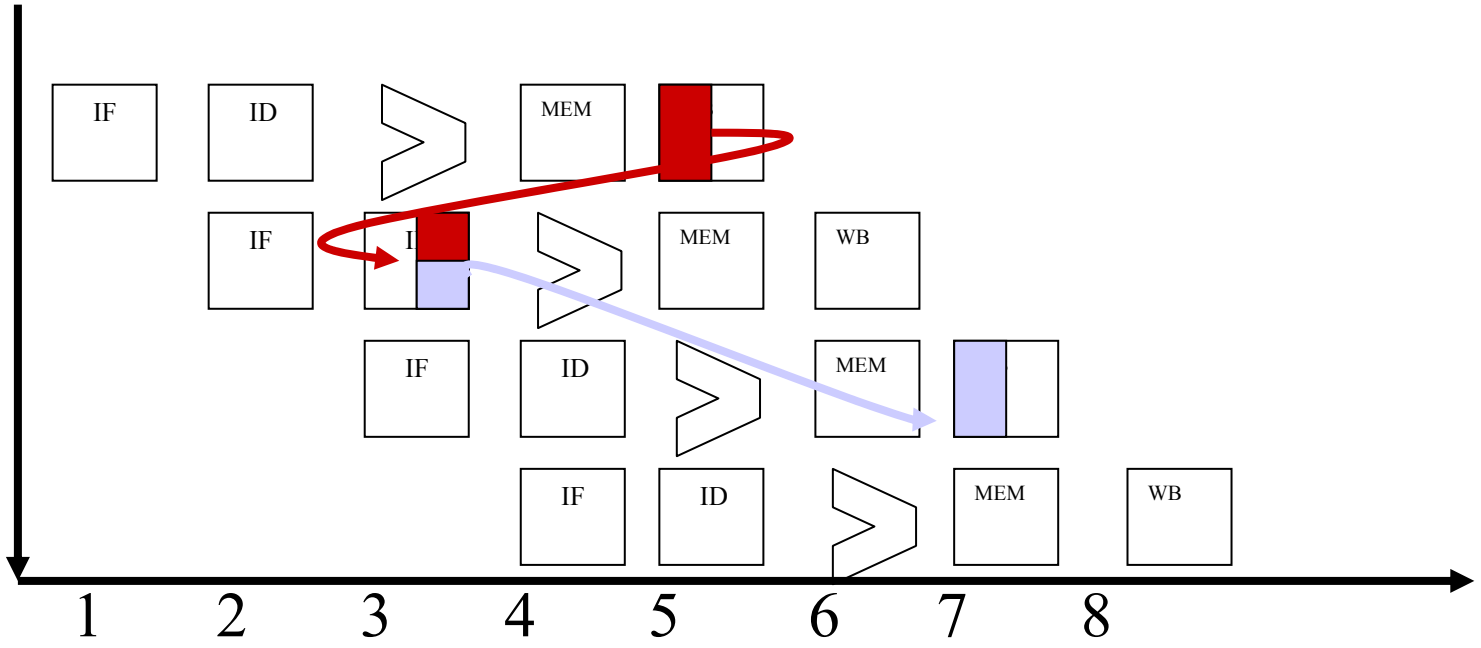
RAW
WAR
WAW

add $t0, $s0, $s1

sub $t2, $t0, $s3

or $s3, $t7, $s2

mul $t2, $t7, $s0



Let's reorder the or

RAW
WAR
WAW

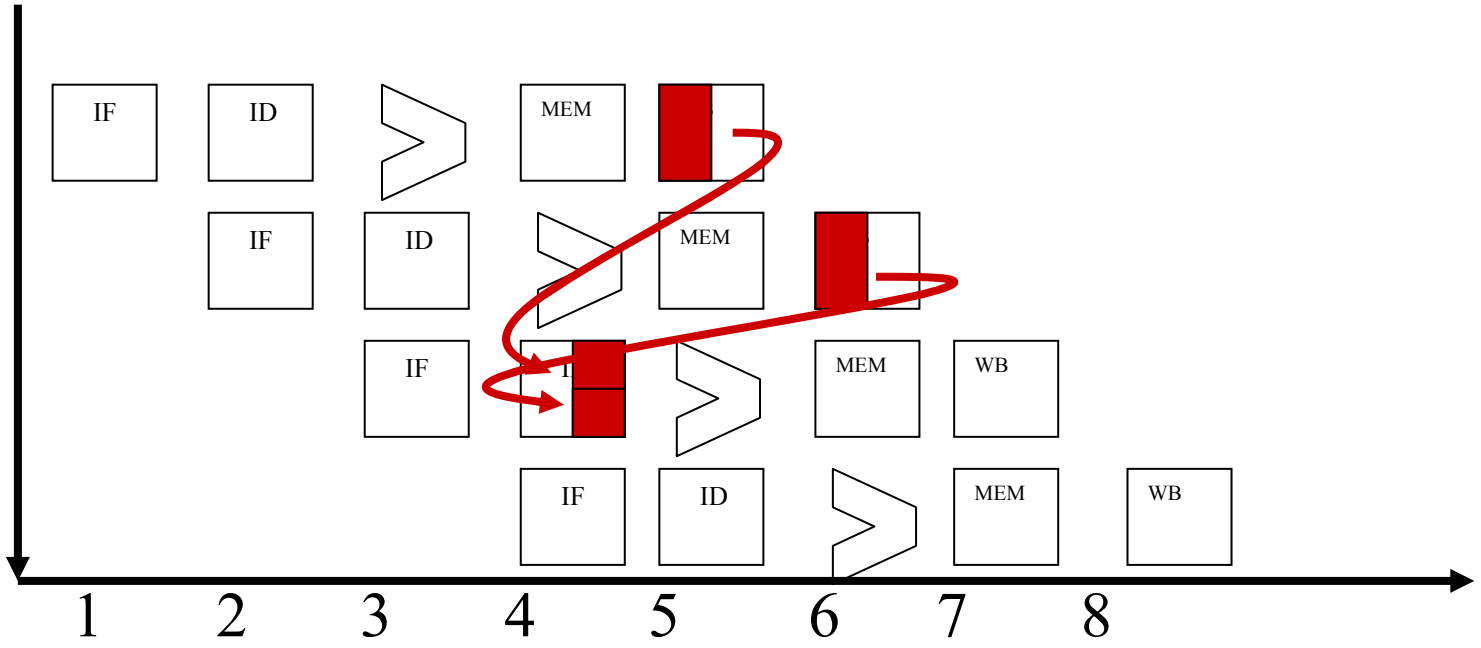
add $t0, $s0, $s1

or $s3, $t7, $s2

RAW

sub $t2, $t0, $s3

mul $t2, $t7, $s0



Let's reorder the mul

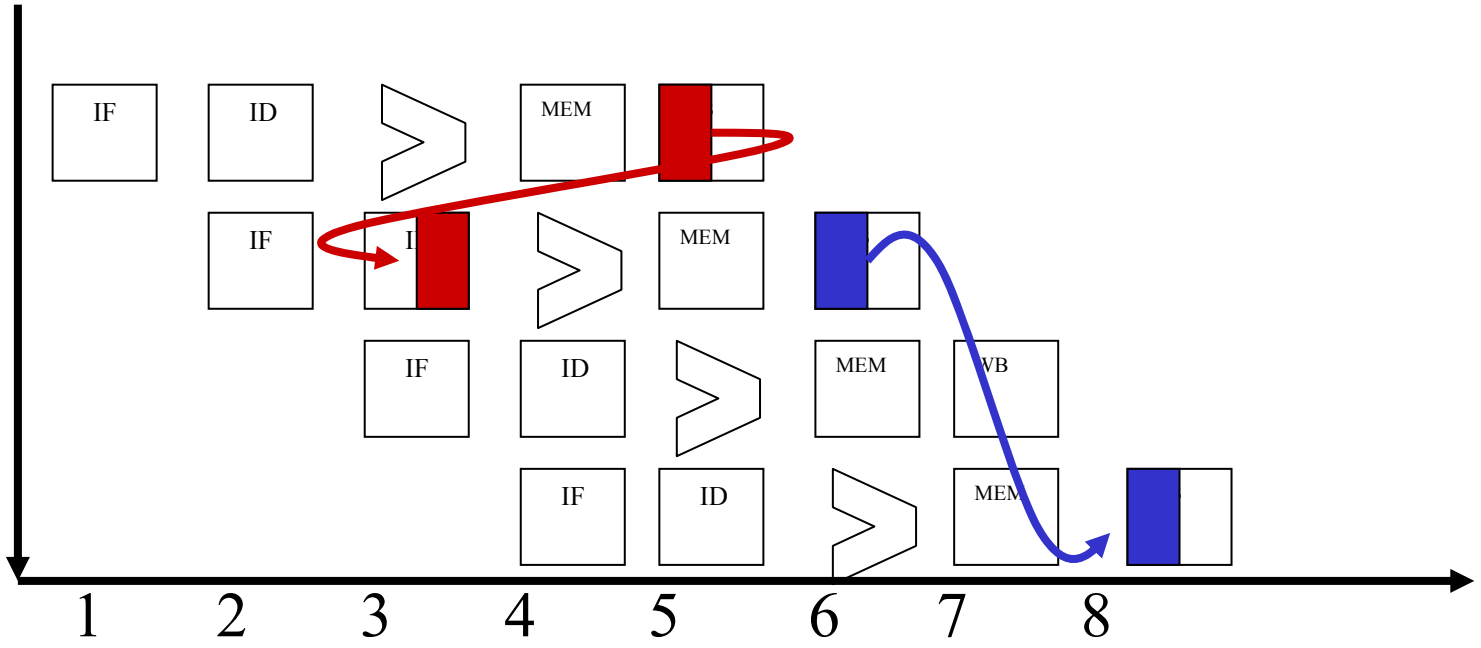
RAW
WAR
WAW

add $t0, $s0, $s1

sub $t2, $t0, $s3

or $s3, $t7, $s2

mul $t2, $t7, $s0



Let's reorder the mul

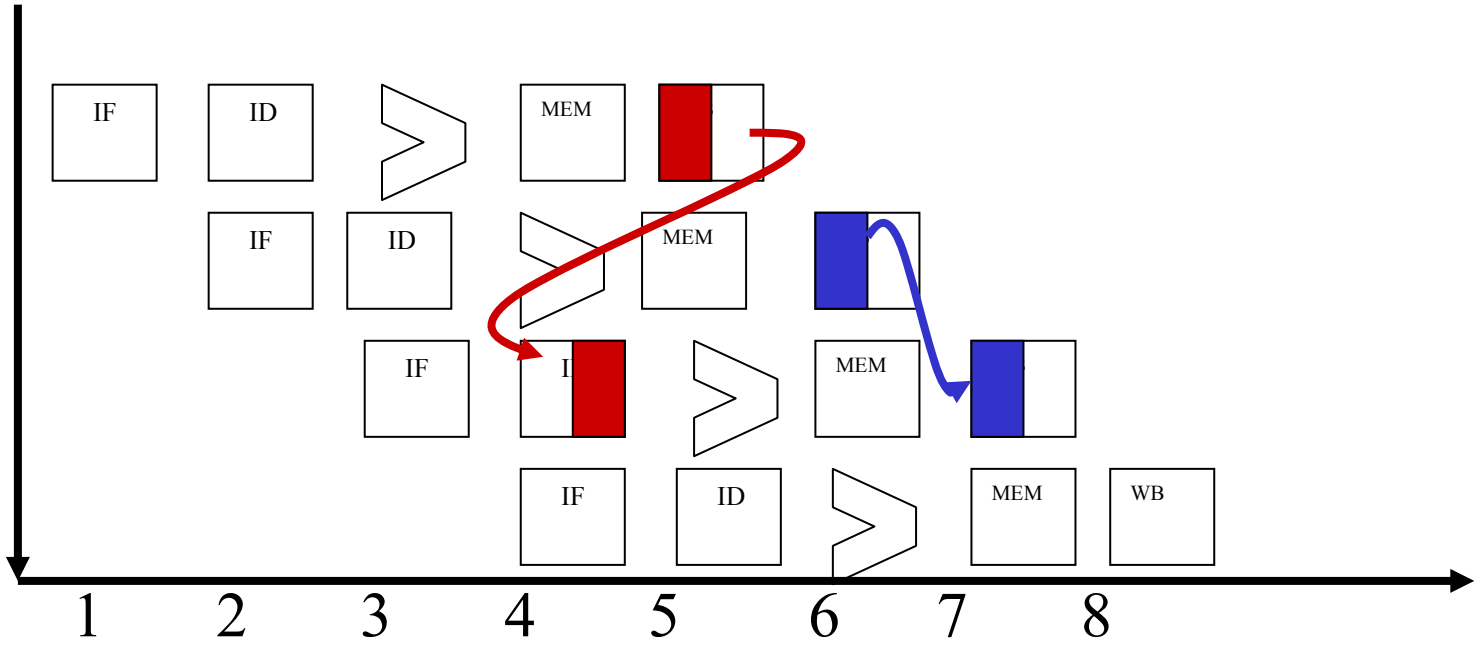
RAW
WAR
WAW

add $t0, $s0, $s1

mul $t2, $t7, $s0

sub $t2, $t0, $s3

or $s3, $t7, $s2



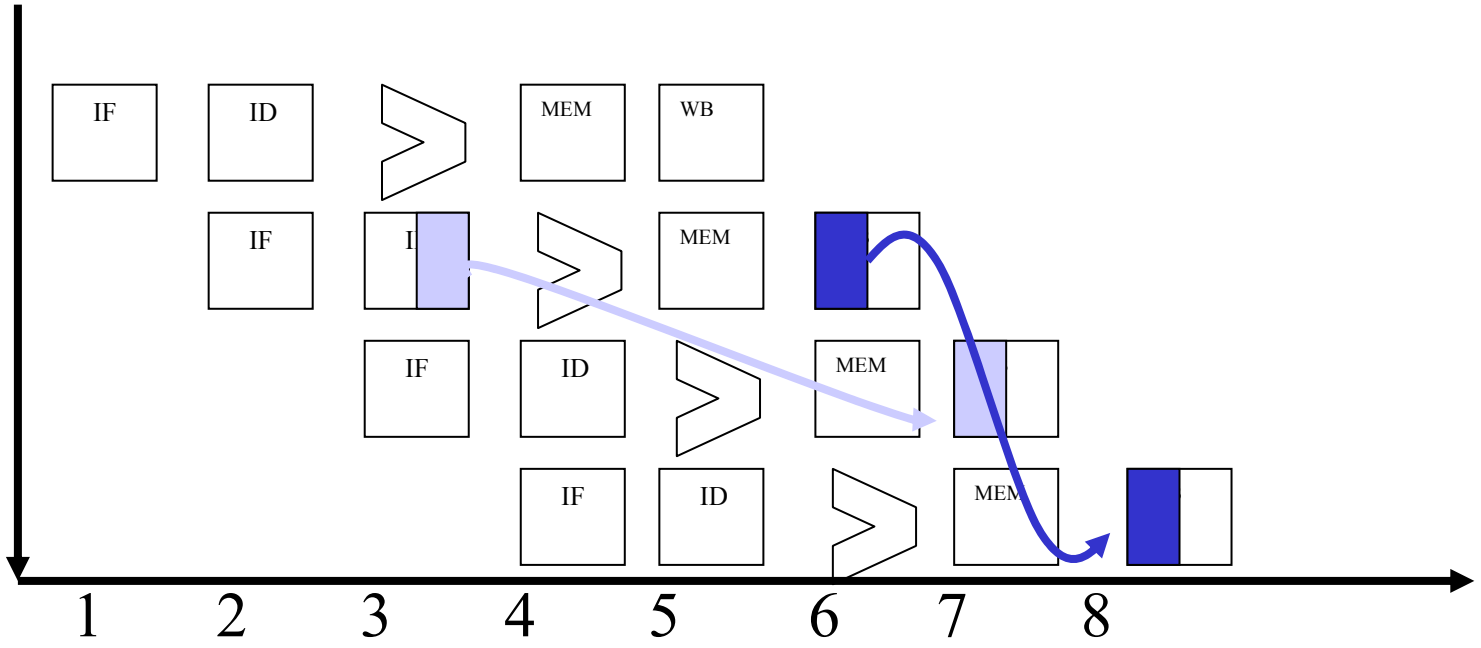
How to alleviate name dependencies?

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 βρούν 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 μέθεπόμενη εντολή, προωθούμε το MEM/WB αποτέλεσμα, ως είσοδο για την ALU πράξη της μεθεπόμενης εντολή

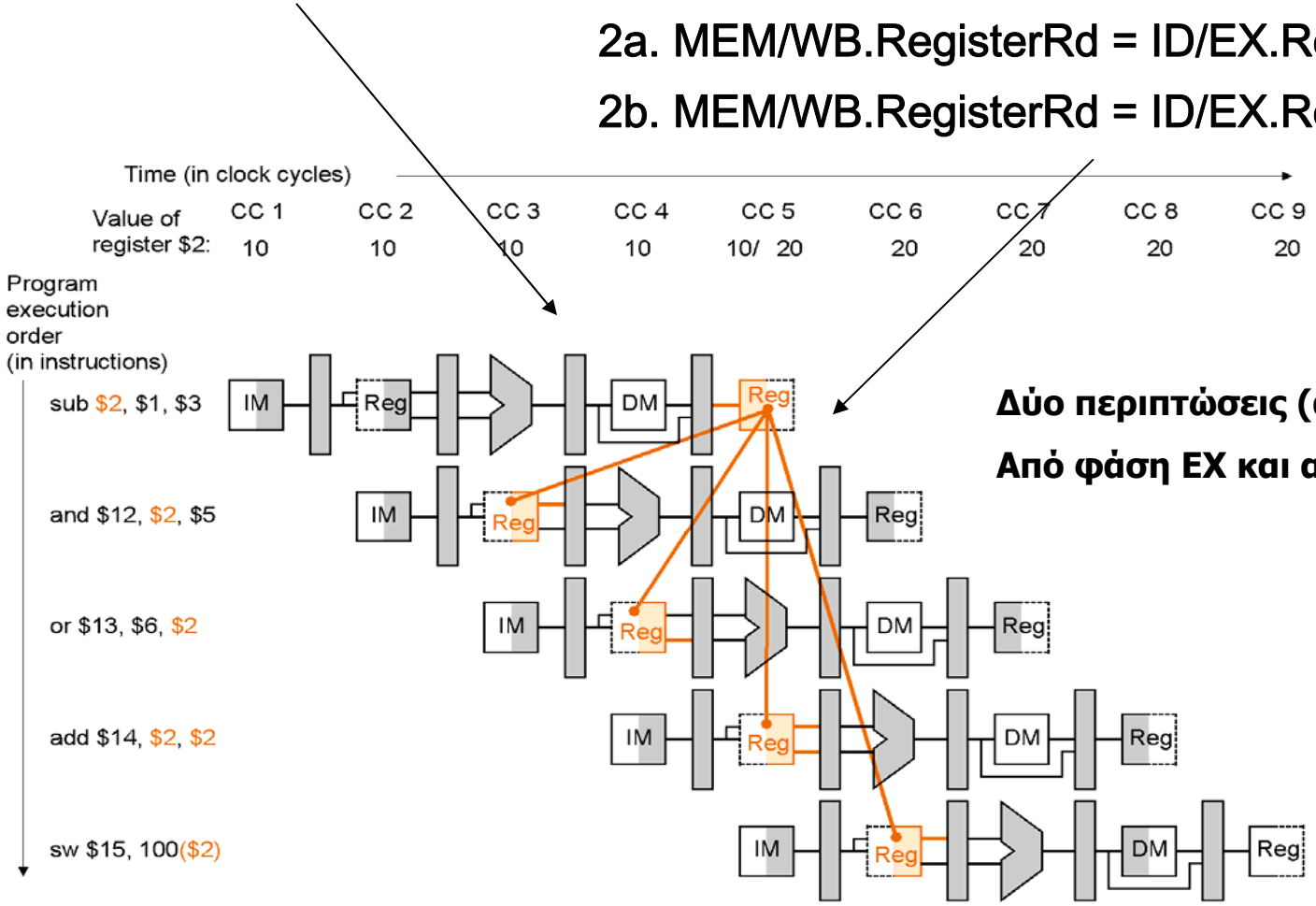
Είσοδο στην ALU όχι μόνο από ID/EX καταχωρητή!!

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:

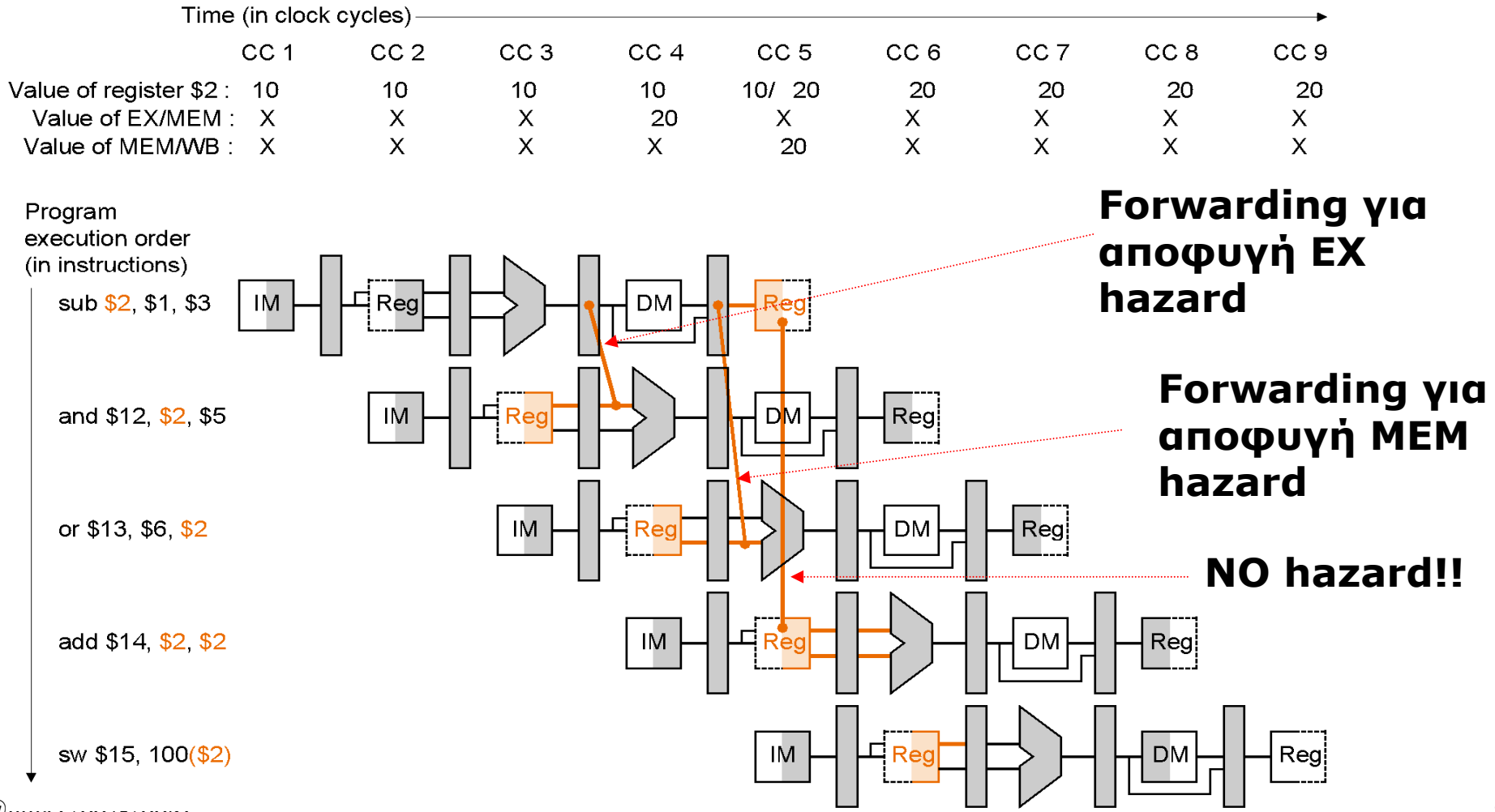
Από φάση EX και από φάση MEM

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



Συνθήκες ελέγχου των κινδύνων:

1. EX hazard:

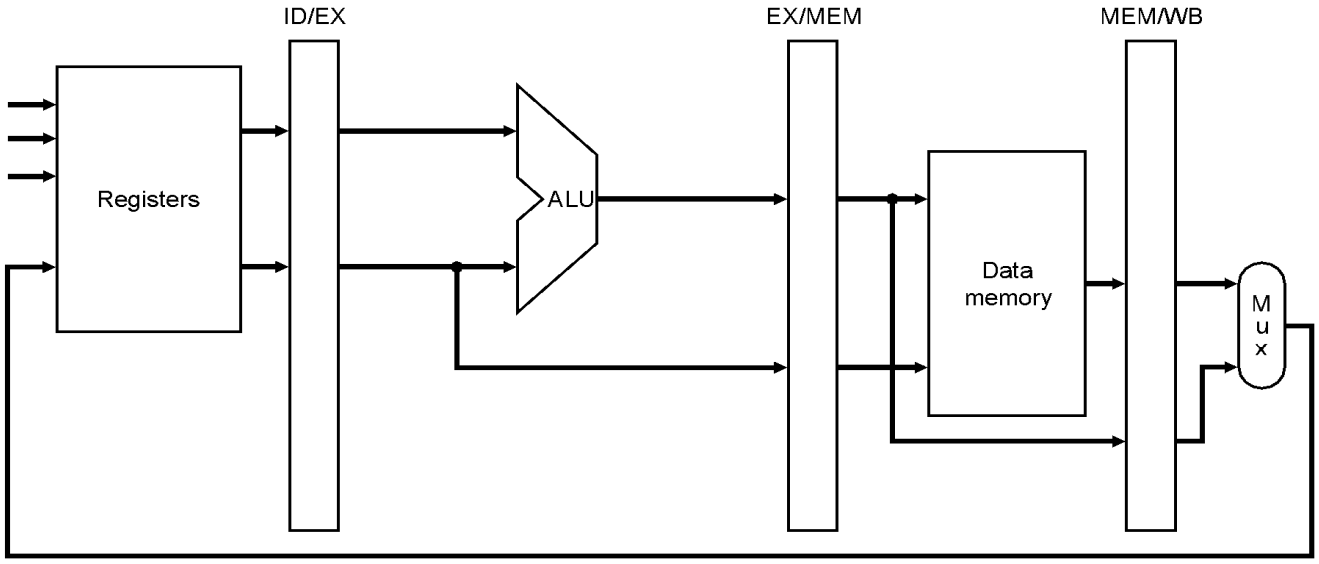
```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd≠0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

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

2. MEM hazard:

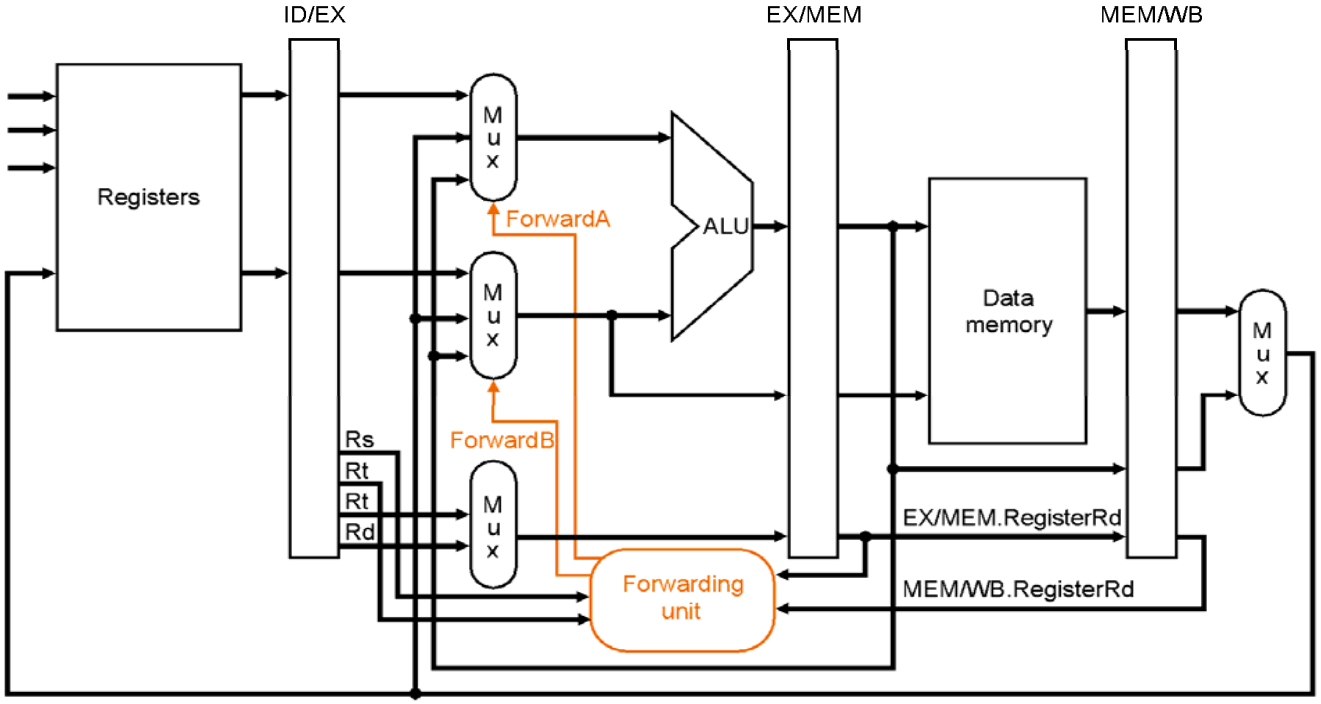
`if` (MEM/WB.RegWrite
`and` (MEM/WB.RegisterRd≠0)
`and` (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

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



a. No forwarding

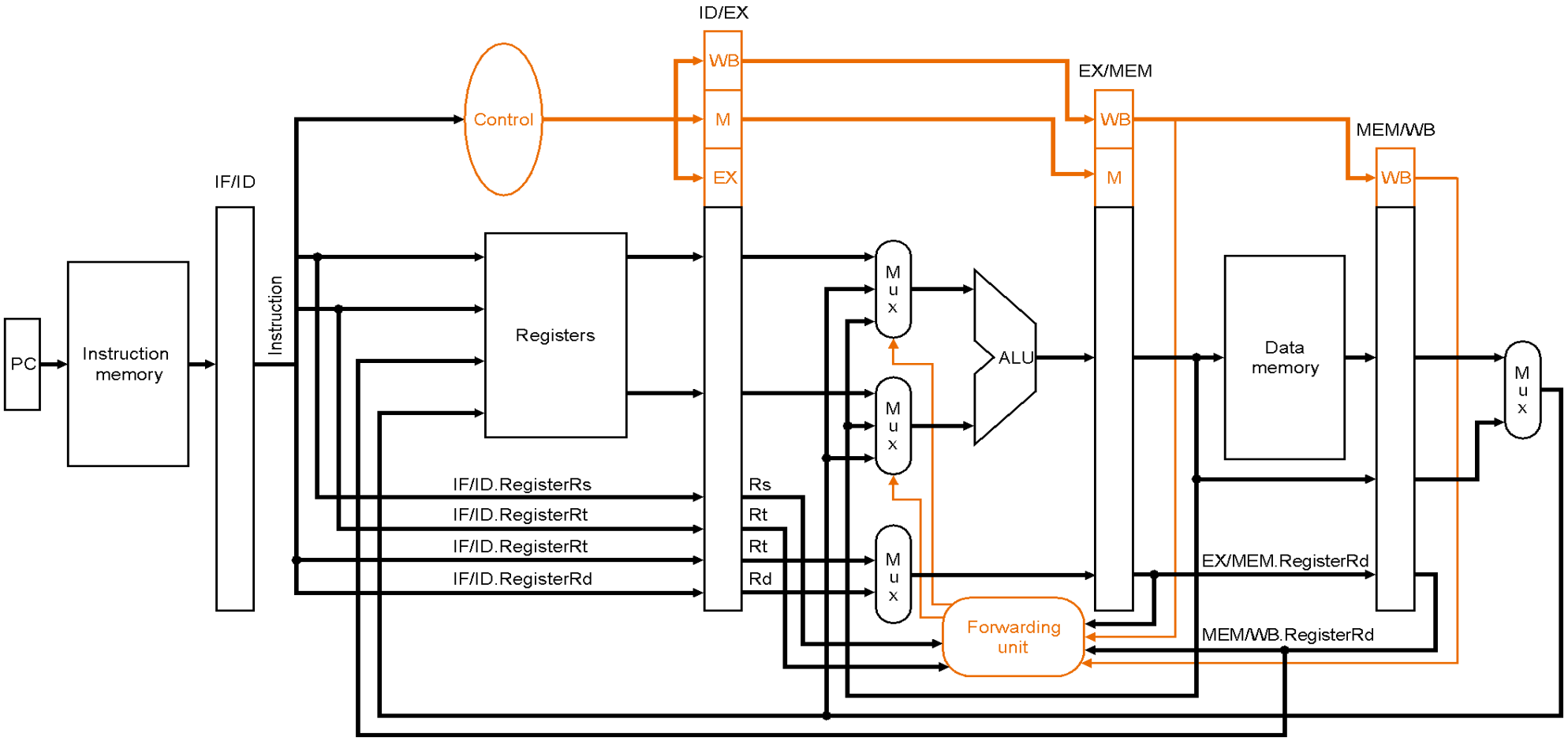
Pipelining χωρίς forwarding

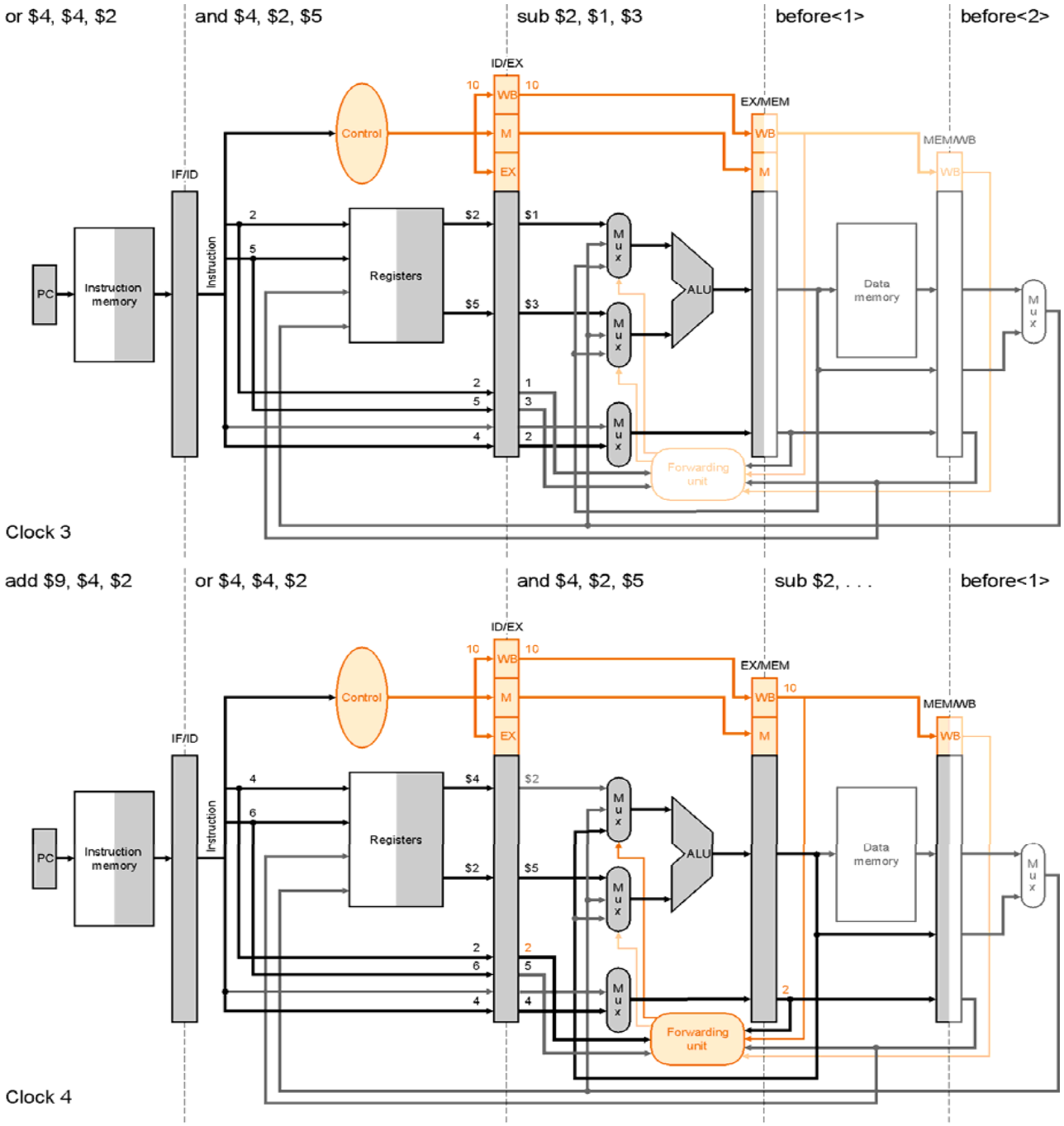


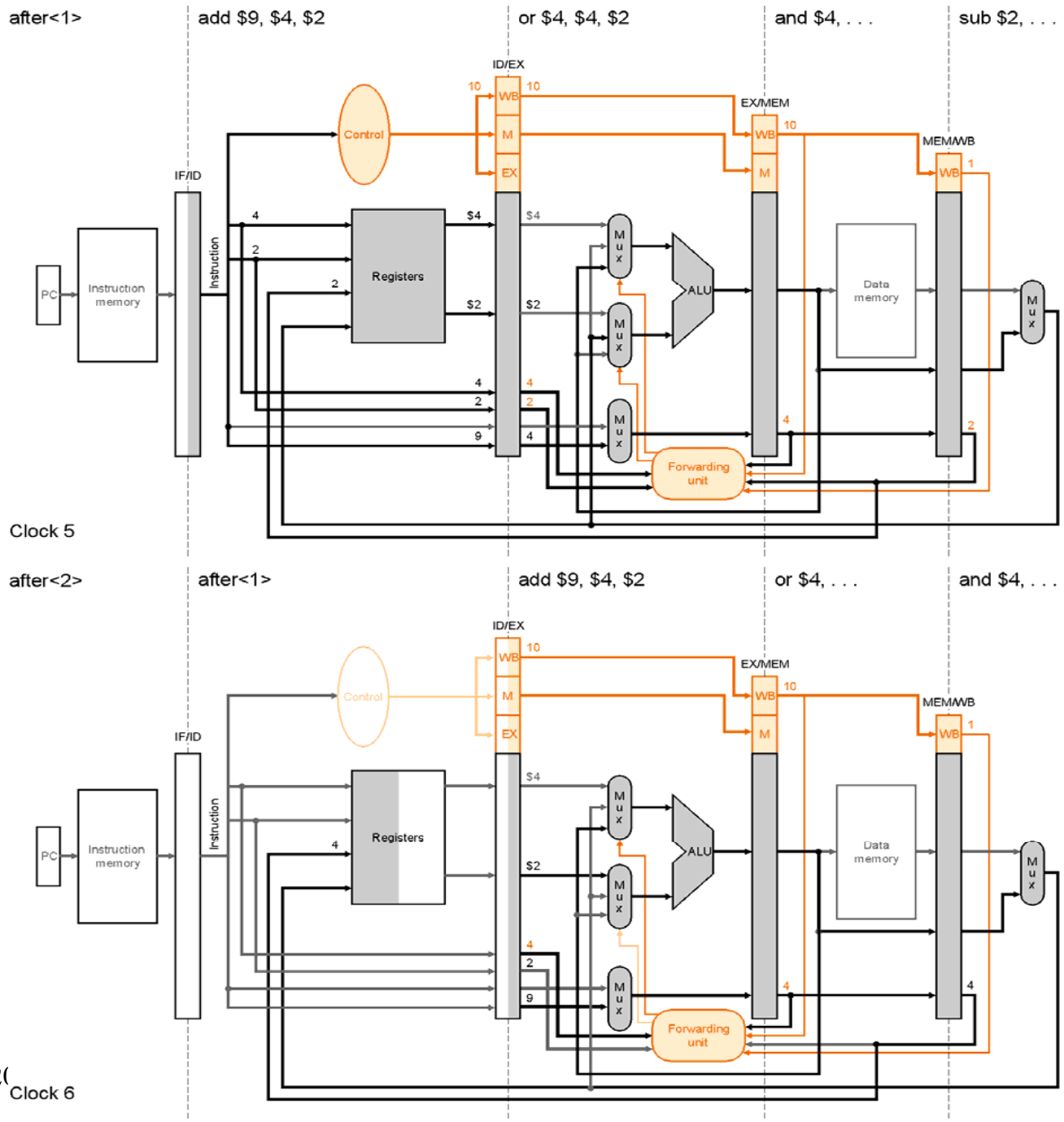
b. With forwarding

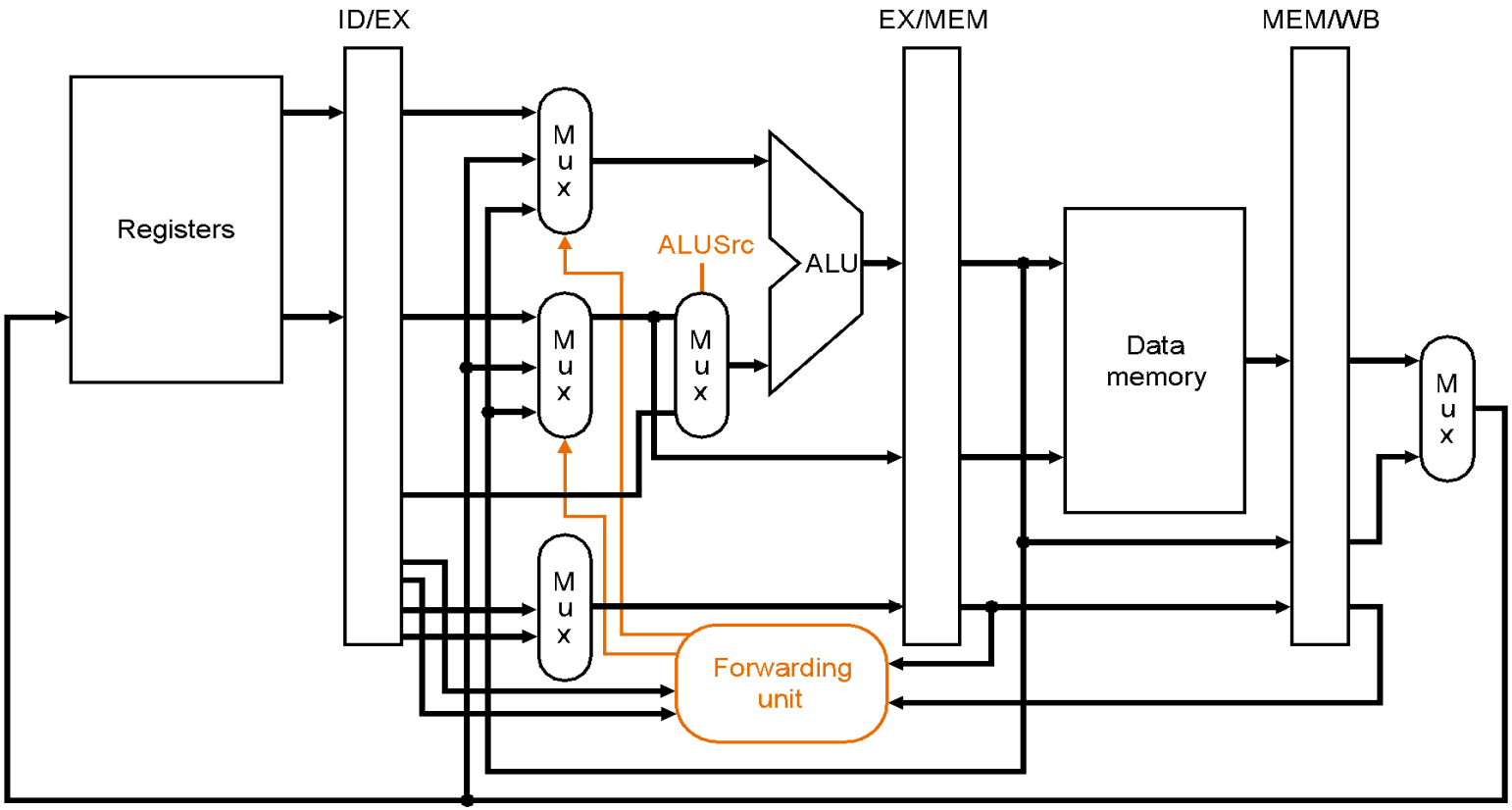
<u>Forwarding paths</u>:
Από:
a) EX/MEM register
και από:
b) MEM/WB register
προς τις εισόδους της ALU

Datapath to resolve hazards via forwarding:




or $4, $4, $2
and $4, $2, $5
sub $2, $1, $3
before<1>
before<2>
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
IF/ID
Instruction
PC
Instruction memory
Registers
ALU
Data memory
Forwarding unit
Mux
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, . . .
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
IF/ID
Instruction
PC
Instruction memory
Registers
$4
$2
ALU
Mux
Data memory
Forwarding unit
Clock 5
after<2>
after<1>
add $9, $4, $2
or $4, . . .
and $4, . . .
Clock 6


ID/EX
EX/MEM
MEM/WB
Registers
M
u
x
ALUSrc
ALU
Data
memory
Forwarding
unit

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

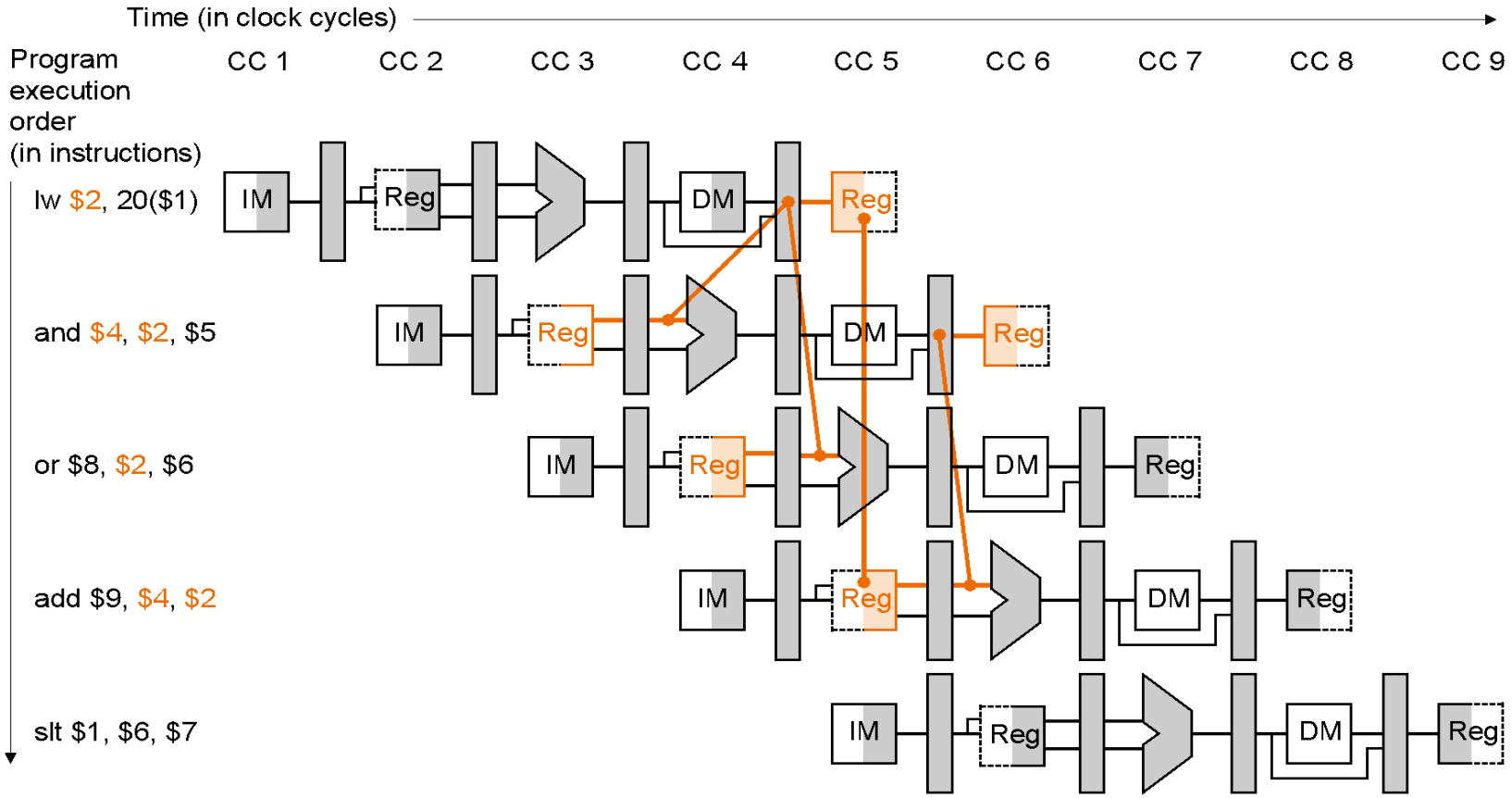
Όταν η εντολή που κάνει write είναι R-TYPE, τότε το **αποτέλεσμα** είναι *έτοιμο* στην φάση EX (έξοδος ALU) και αποθηκέυεται, στο τέλος του κύκλου, στον EX/MEM καταχωρητή.

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

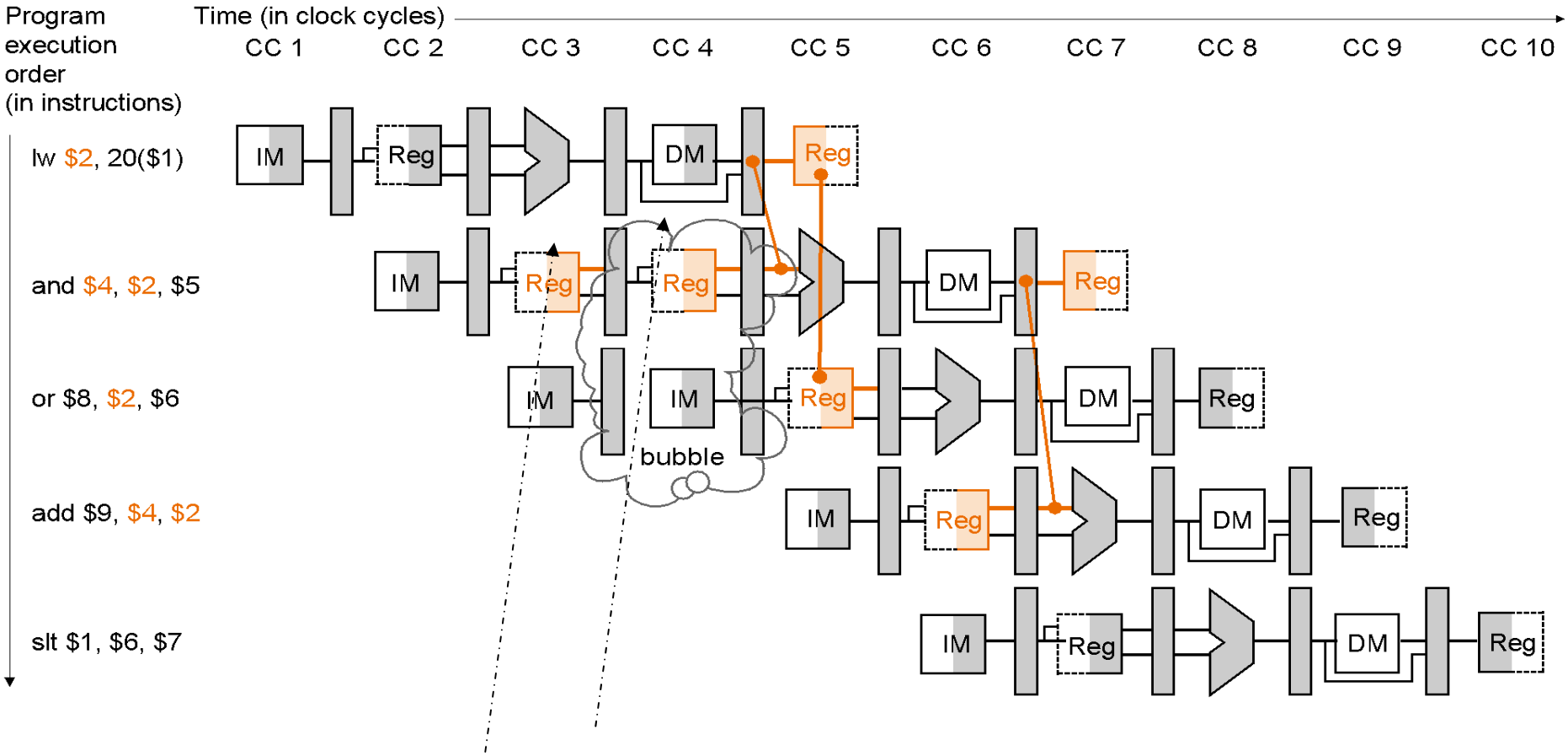
Το forwarding δεν δίνει όμως πάντα λύση!!

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 επόμενή της δύο φορές συνεχόμενα

Pipelined Control: