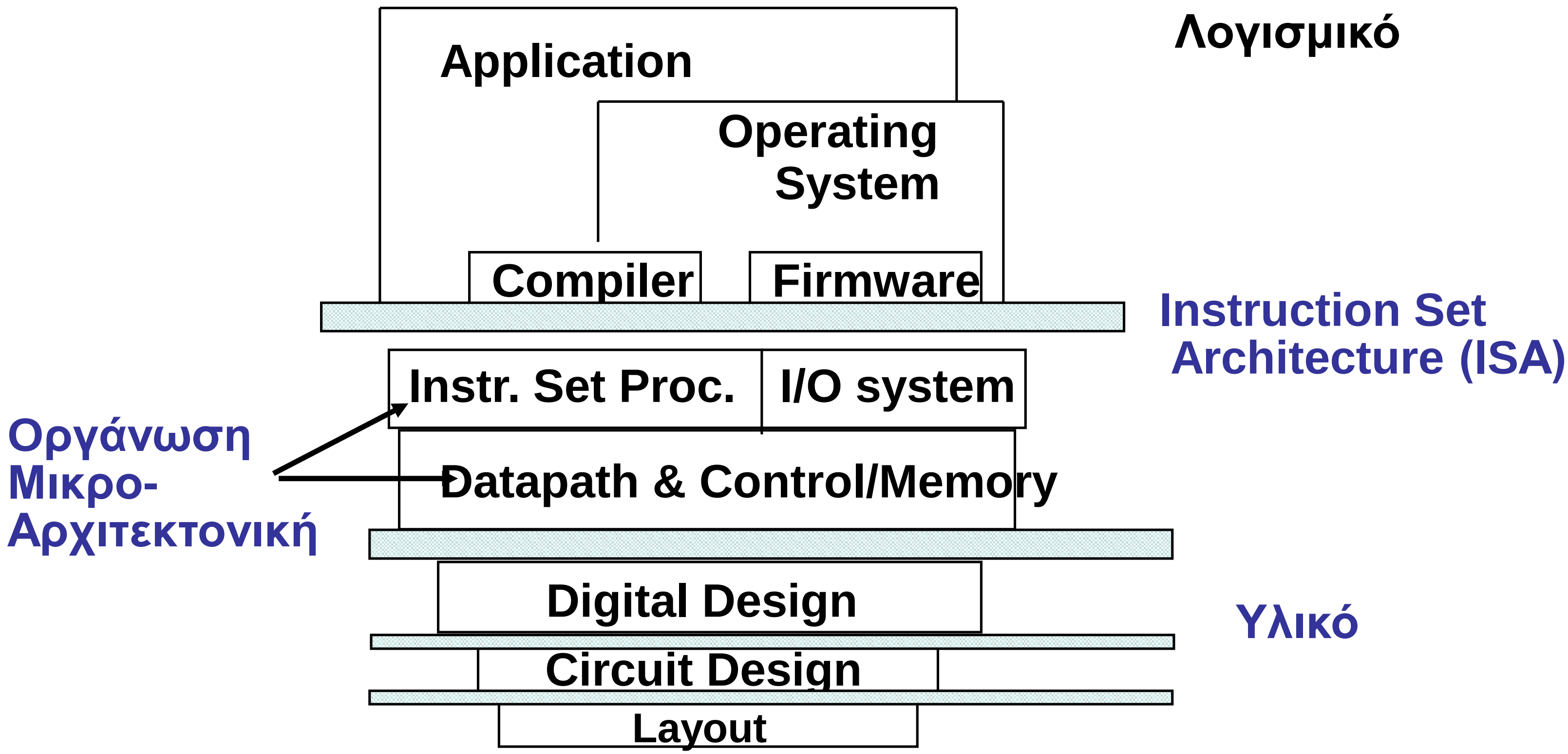


- Η ΑΣΕ (ISA) διασύνδεση μεταξύ λογισμικού και υλικού
- Η οργάνωση περιγραφή ροής και αποθήκευσης πληροφοριών και ελέγχου ξεχωριστά από την τεχνολογία υλοποίησης

- Instruction Set Architecture (ISA) design
  - i.e. decisions regarding:
    - registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding
  - example ISAs: x86, ARM, MIPS, IBM
    - 32 bit and 64 bit
    - CISC, RISC
  - Extending an ISA: new operations, e.g. bit manipulation to facilitate security, new data types e.g. 256-bit operands

MIPS ISA

Name	Number	Use	Preserved across a call?
$zero	0	The constant value 0	N.A.
$at	1	Assembler temporary	No
$v0–$v1	2–3	Values for function results and expression evaluation	No
$a0–$a3	4–7	Arguments	No
$t0–$t7	8–15	Temporaries	No
$s0–$s7	16–23	Saved temporaries	Yes
$t8–$t9	24–25	Temporaries	No
$k0–$k1	26–27	Reserved for OS kernel	No
$gp	28	Global pointer	Yes
$sp	29	Stack pointer	Yes
$fp	30	Frame pointer	Yes
$ra	31	Return address	Yes

PC: program counter points to next instruction to execute

Operations, operands (registers+memory), encoding of operations, syscalls

Instruction type/opcode
Data transfers
LB, LBU, SB
LH, LHU, SH
LW, LWU, SW
LD, SD
L.S, L.D, S.S, S.D
MFC0, MTC0
MOV.S, MOV.D
MFC1, MTC1
Arithmetic/logical
DADD, DADDI, DADDU, DADDIU
DSUB, DSUBU
DMUL, DMULU, DDIV, DDIVU, MADD
AND, ANDI
OR, ORI, XOR, XORI
LUI
DSLL, DSRL, DSRA, DSLLV, DSRLV, DSRAV
SLT, SLTI, SLTU, SLTIU
Control
BEQZ, BNEZ
BEQ, BNE
BC1T, BC1F
MOVN, MOVZ
J, JR
JAL, JALR
TRAP
ERET
Floating point
ADD.D, ADD.S, ADD.PS
SUB.D, SUB.S, SUB.PS
MUL.D, MUL.S, MUL.PS
MADD.D, MADD.S, MADD.PS
DIV.D, DIV.S, DIV.PS
CVT._._
C.__.D, C.__.S

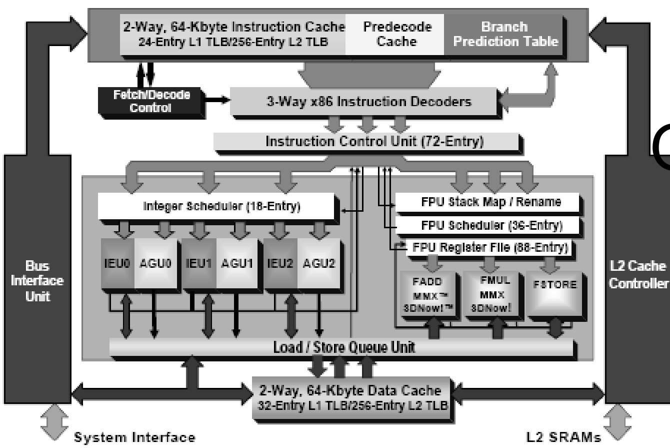
- Computer organization:
  - microarchitecture and hardware implementation of the ISA
    - Number of pipe stages, size of caches, number of functional units, prediction, prefetching…
- Design to maximize performance within constraints: cost, power, and availability
- Many options for implementing the ISA (huge and challenging design space)
  - Critical making right choices
  - Hardware configurable and dynamic: challenging how to configure it and use it efficiently

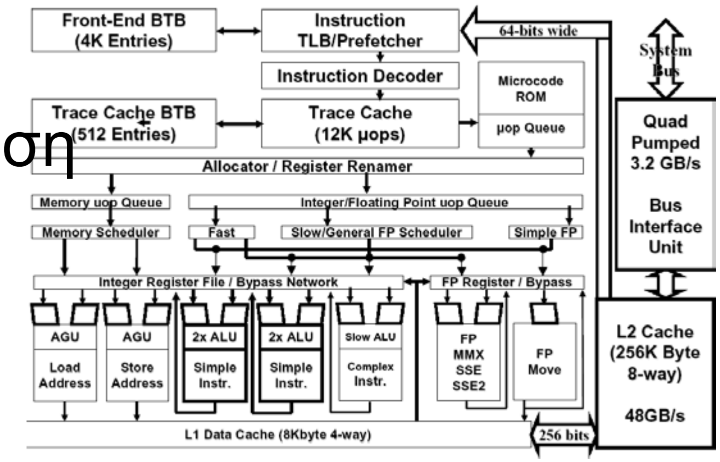
Implementation Technology

- Type of transistors
  - mosfet, finfet, number of fins
- Size of transistors, wires, …
- Metal Layers
- Placement and routing
- …

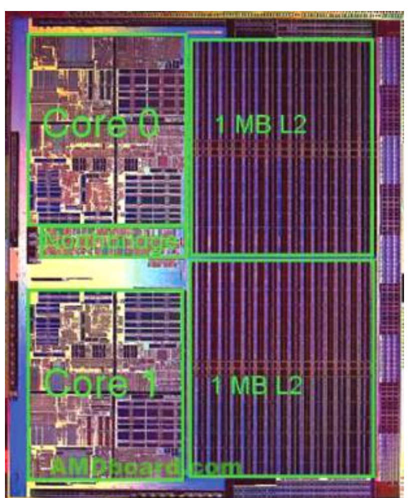
ΑΣΕ (ISA)

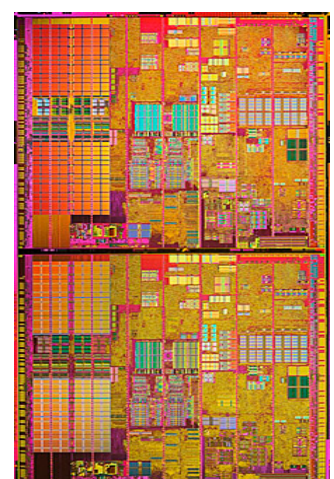
Οργάνωση



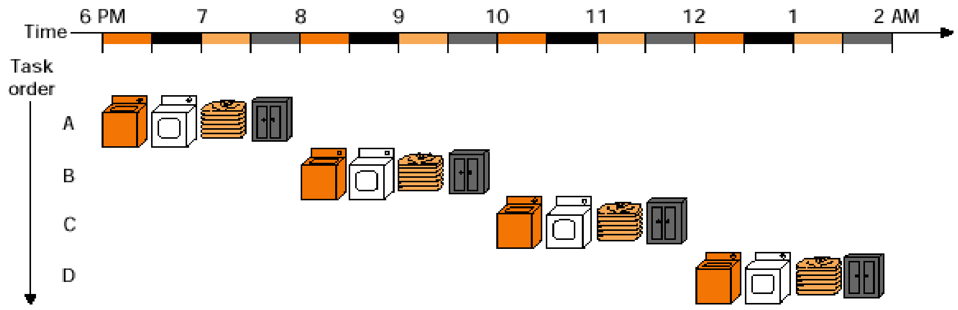


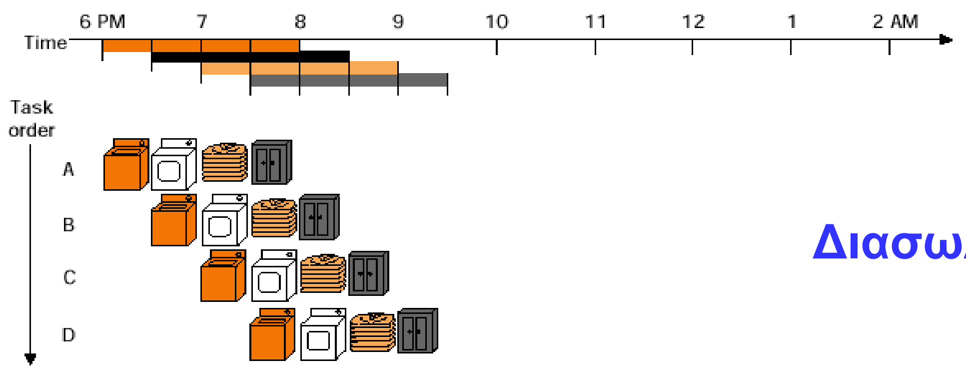
Υλικό/Hardware





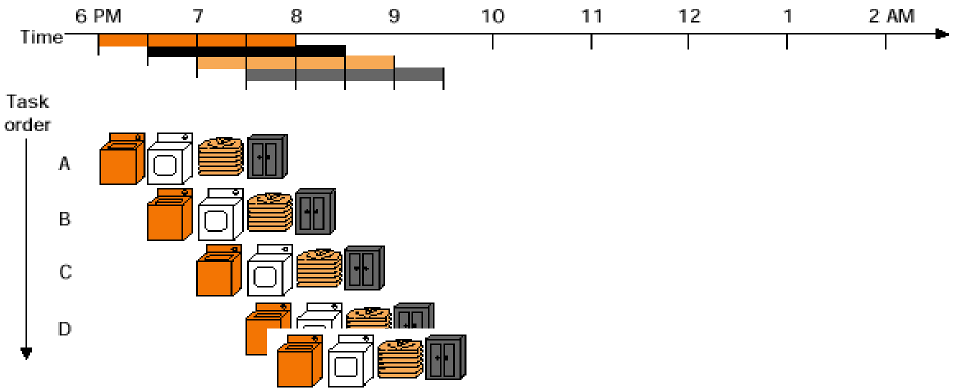
Επανάληψη
Σύστημα Διασωλήνωσης
(Pipelining)
How to organize a processor



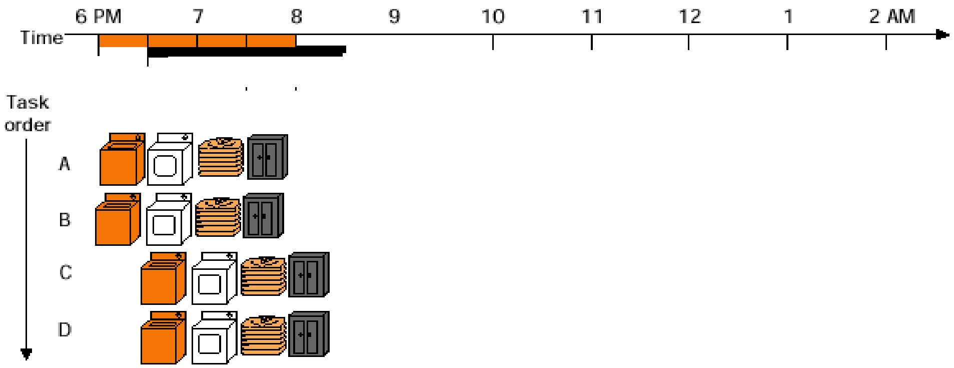


Διασωλήνωση

•Pipelining



•Παραλληλισμός + Pipelining



Χρόνος Εκτέλεσης = I x CPI x Cycle Time

I dynamic instructions

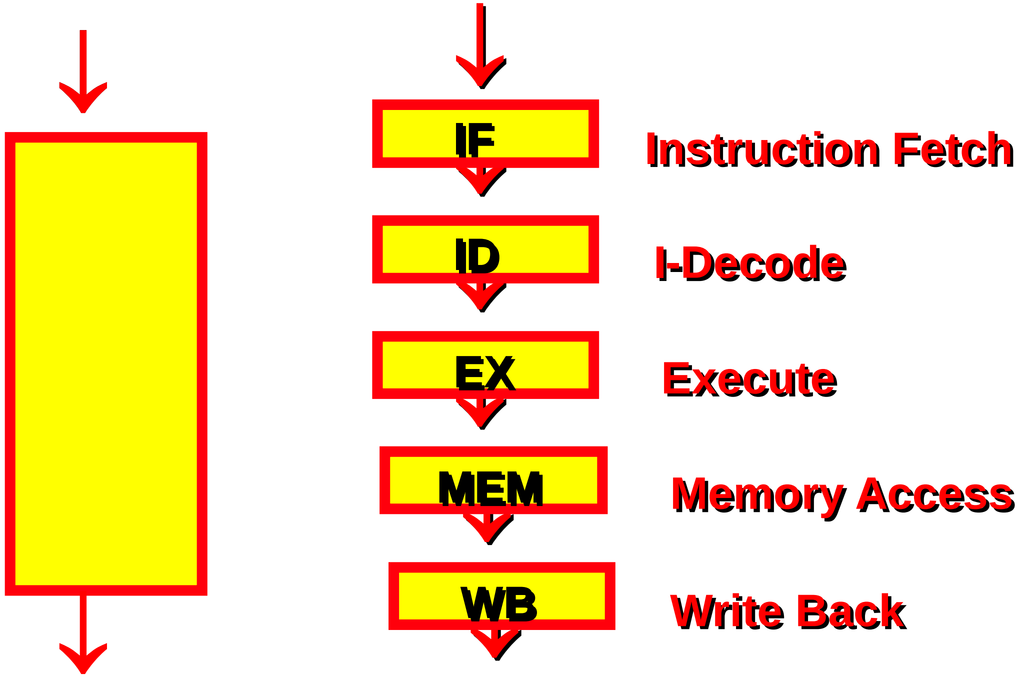
Cycles Per Instruction

Cycle Time

Με ή χωρις pipeline το I είναι το ίδιο όπως και το CPI = 1 (CPI=1 ιδεατά).

Το Cycle Time;

Το ρολόι πρέπει να δουλεύει με την ταχύτητα του πιο αργού τμήματος

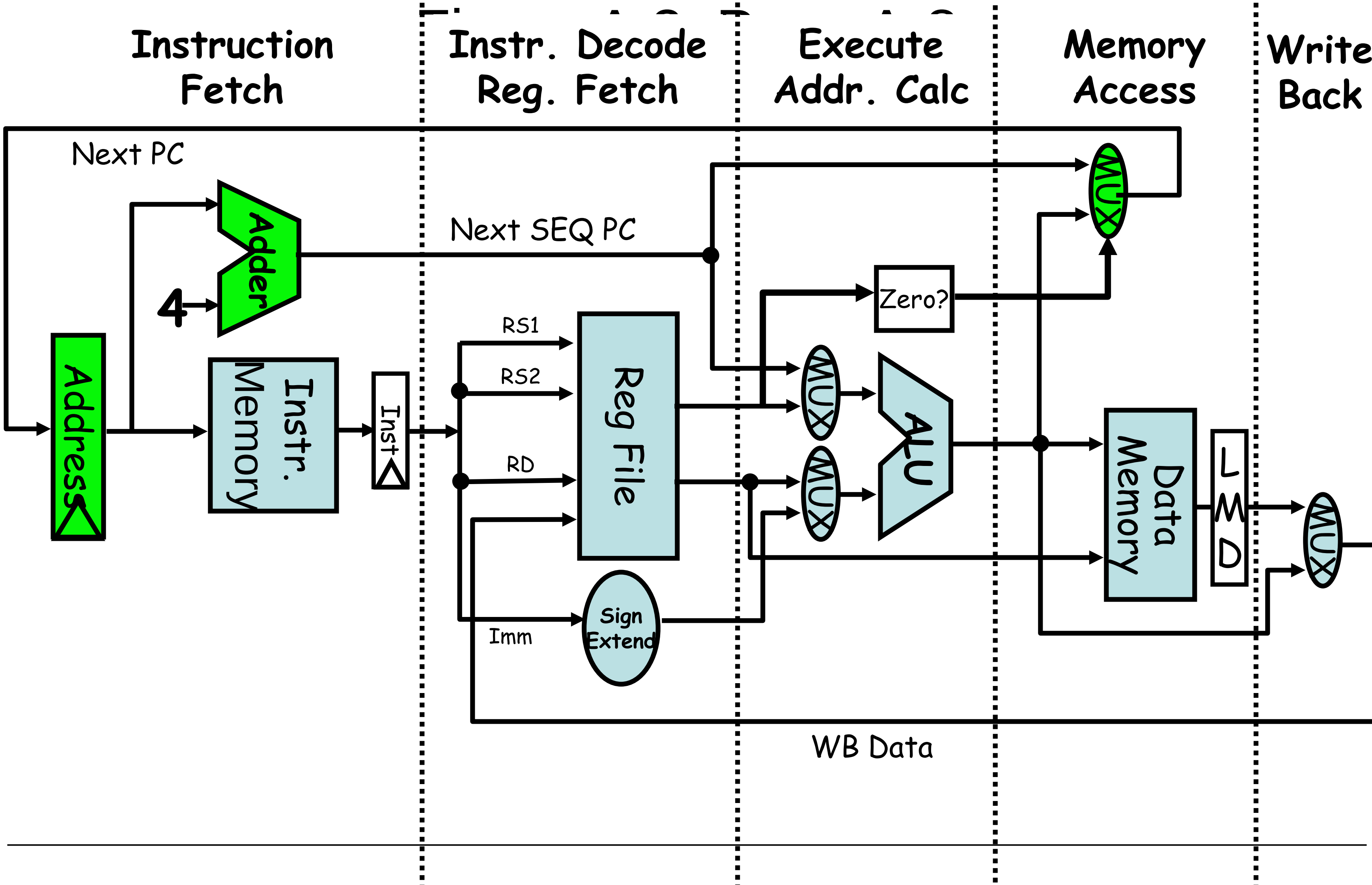


Χωρίς pipeline Cycle Time = κρίσιμο μονοπάτι του διαδρόμου δεδομένων
(ίδιο για όλες τις εντολές)

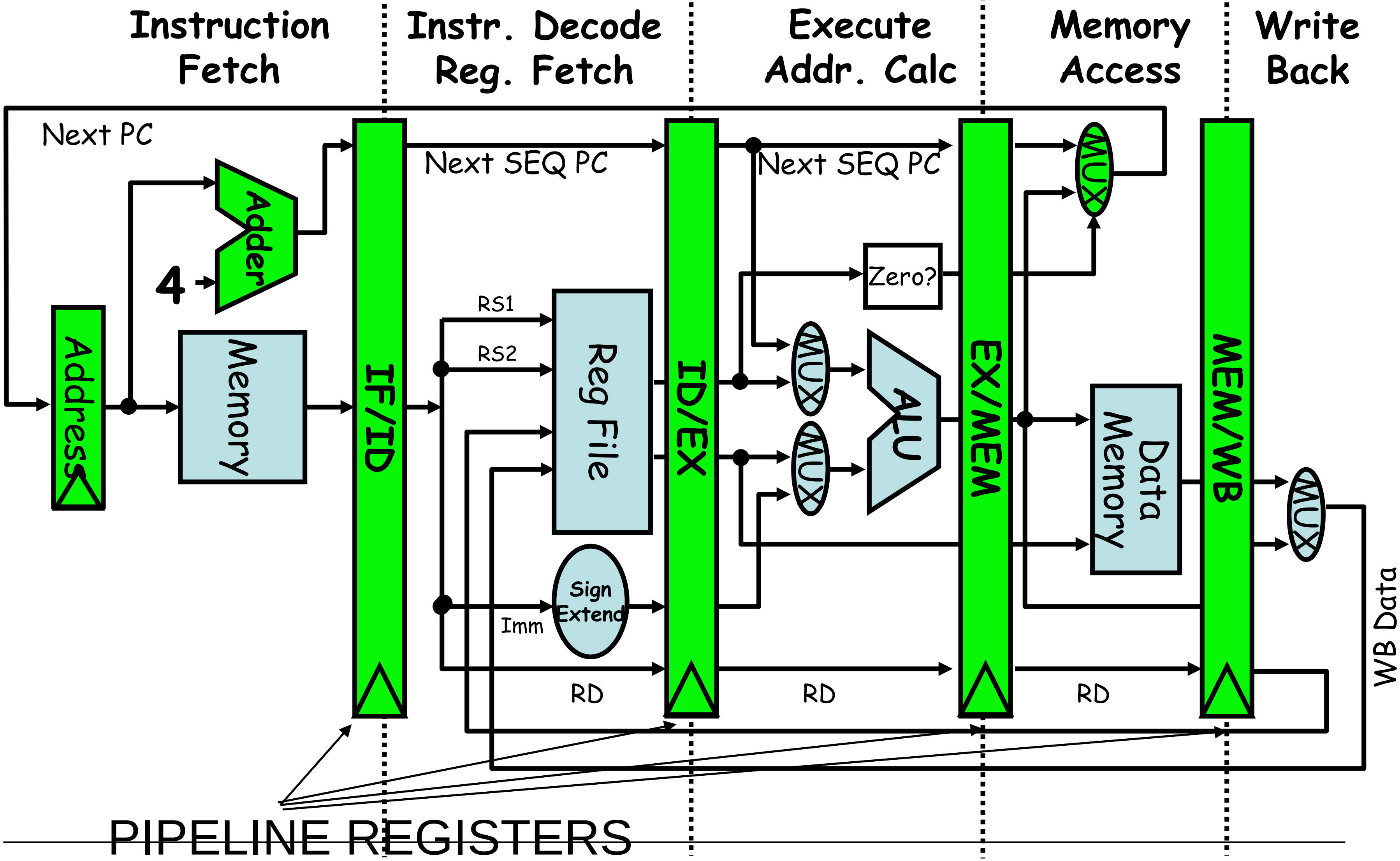
Με pipeline Cycle Time = Maximum (κρίσιμο μονοπάτι κάθε σταδίου)

Στην καλύτερη περίπτωση
pipeline Cycle Time = Cycle Time χωρίς pipeline / αριθμός σταδίων

MIPS Datapath



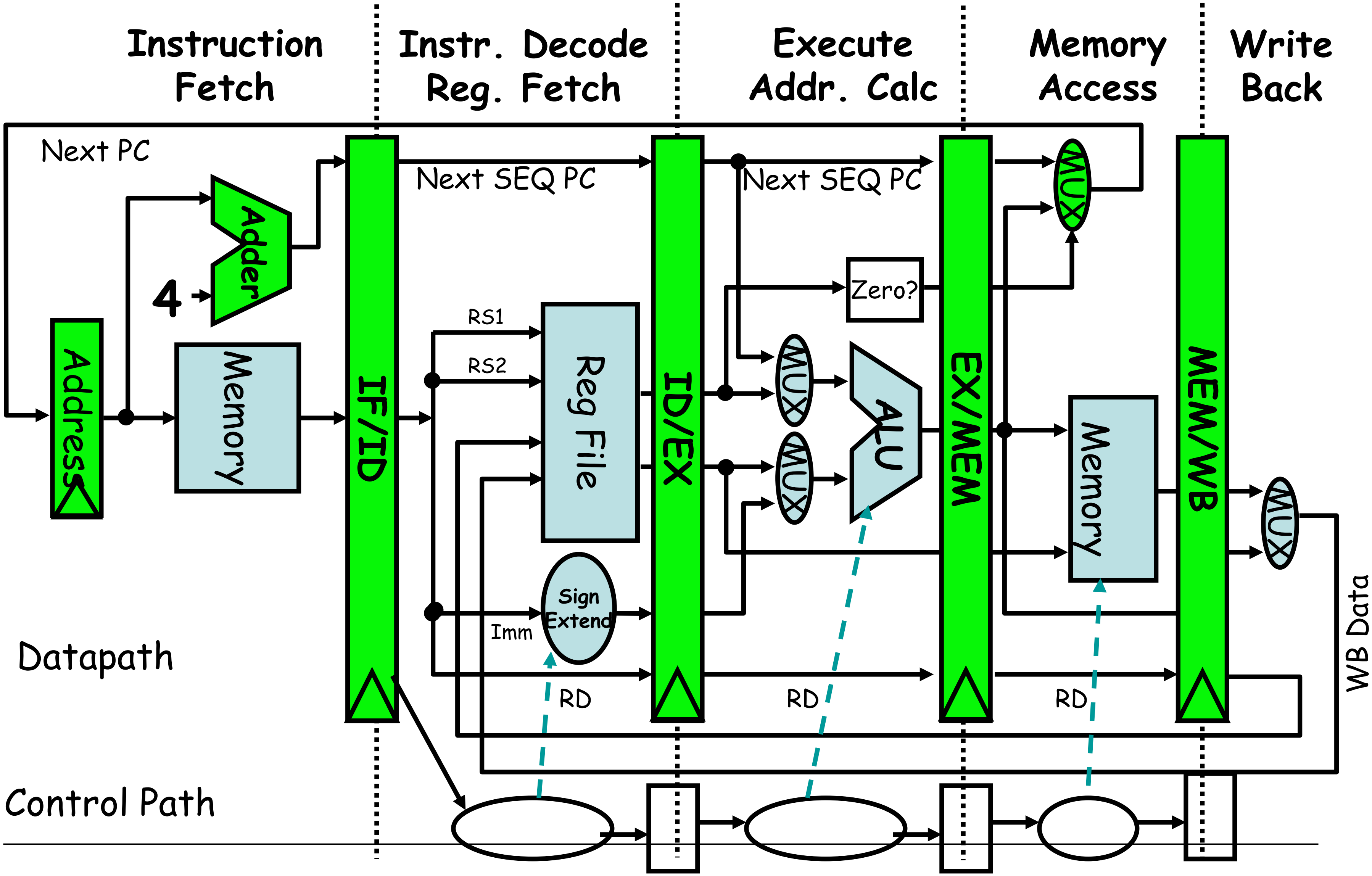
Pipelined MIPS Datapath



PIPELINE REGISTERS

Καταχωρητές Διασωλήνωσης

(Datapath) του Pipeline με control



Καταχωρητές Διασωλήνωσης

F/D:

- PC, instruction

D/E:

- Src1,Src2, instruction, PC, imm(sign ext)
- control for other stages (muxes, ALUop, Mem signals)

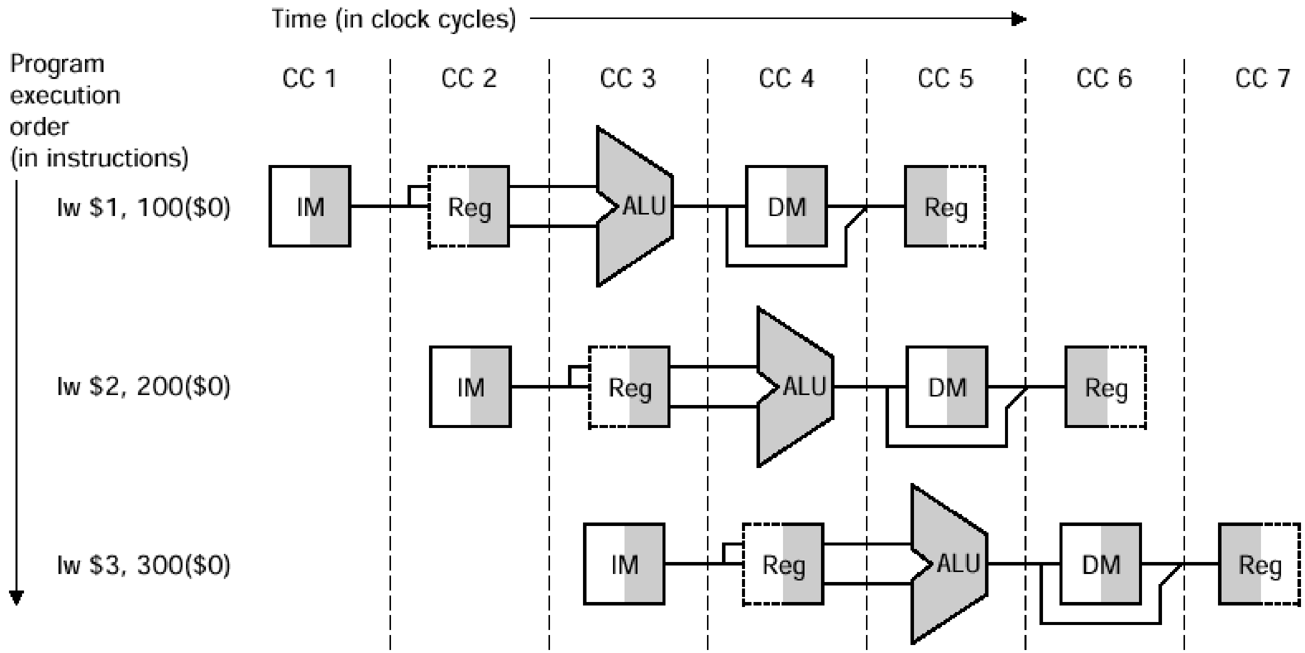
E/W

- ….

W/M

- ….

•Εκτέλεση με διασωλήνωση



•Κάθε εντολή (ανεξάρτητα τύπου) πρέπει να περάσει από όλα τα στάδια
•Χρειάζεται 5 κύκλους για να ολοκληρωθεί μια εντολή
•Δυνατότητα να ολοκληρώνεται μια εντολή κάθε κύκλο

Κίνδυνοι Διασωλήνωσης (Pipeline Hazards)

ΙΔΕΑΤΑ: Το σύστημα διασωλήνωσης εργάζεται χωρίς διακοπές εάν η κάθε εντολή που εκτελείται στη διασωλήνωση είναι ανεξάρτητη από κάθε άλλη εντολή που εκτελείται στη Διασωλήνωση

Στην πράξη υπάρχουν αλληλεξαρτήσεις μεταξύ των εντολών που εκτελούνται στην διασωλήνωσης

Κίνδυνοι Διασωλήνωσης: Εμποδίζουν την επόμενη ακολουθία εντολών από του να εκτελεστεί κατά τη διάρκεια του προκαθορισμένου κύκλου μηχανής. (στάση διασωλήνωσης ή φυσαλίδες διασωλήνωσης)

Stalls και Bubbles: μείωση της βελτίωσης από την χρήση του pipeline

Ύπαρξη κινδύνου προαπαιτεί εξάρτηση αλλά όχι το αντίστροφο

- Δομικοί Κίνδυνοι (Structural Hazards)
- Κίνδυνοι Δεδομένων (Data Hazards)
- Κίνδυνοι Ροής (Control Hazards)

Speed Up Equation for Pipelining

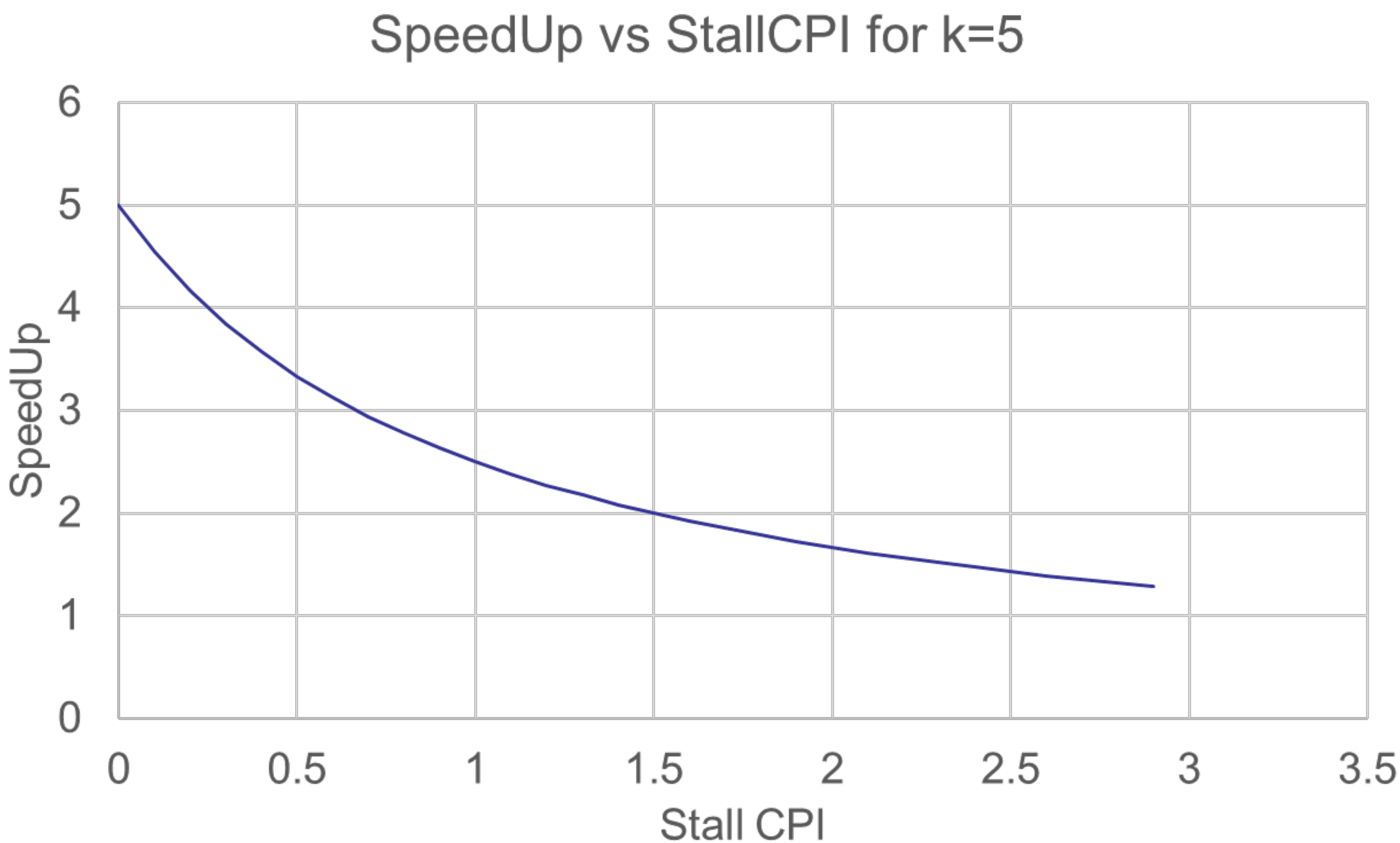
$$CPI_{pipelined} = \text{Ideal CPI} + \text{Average Stall cycles per Inst}$$

$$\text{Speedup} = \frac{1}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{unpipelined}}{\text{Cycle Time}_{pipelined}}$$

Ideal CPI = 1

$$\text{Speedup} = \frac{1}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{unpipelined}}{\text{Cycle Time}_{pipelined}}$$

k= Cycle Non-pipelined/Cycle Time Pipelined (ideally equal to # of stages)



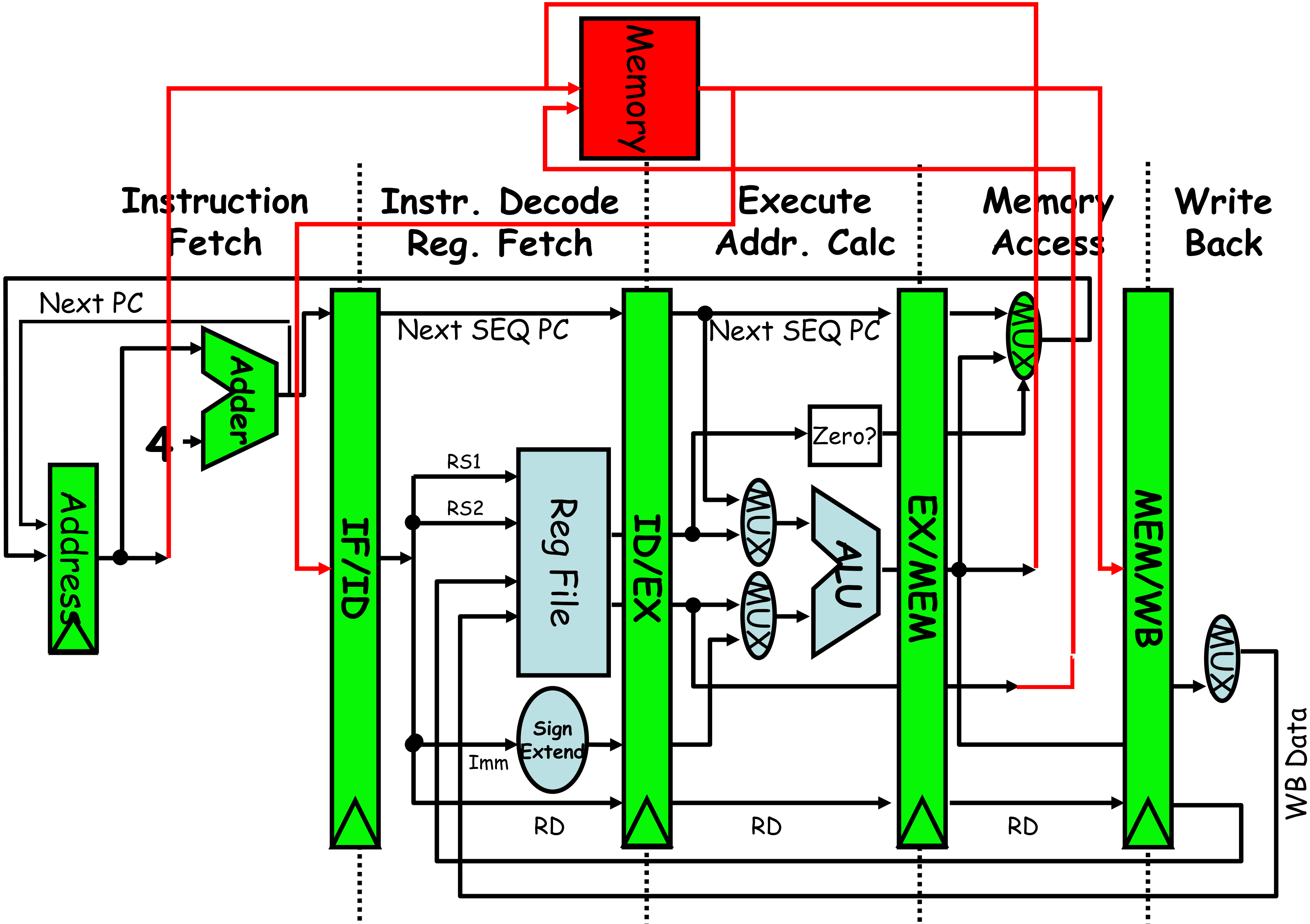
Δομικοί Κίνδυνοι (Structural Hazards)

• Ορισμένοι συνδυασμοί εντολών δεν μπορούν να ικανοποιηθούν διότι χρειάζονται περισσότερους πόρους από αυτούς που διαθέτει η μηχανή

1. Ορισμένες λειτουργικές μονάδες δεν είναι διασωληνωμένες

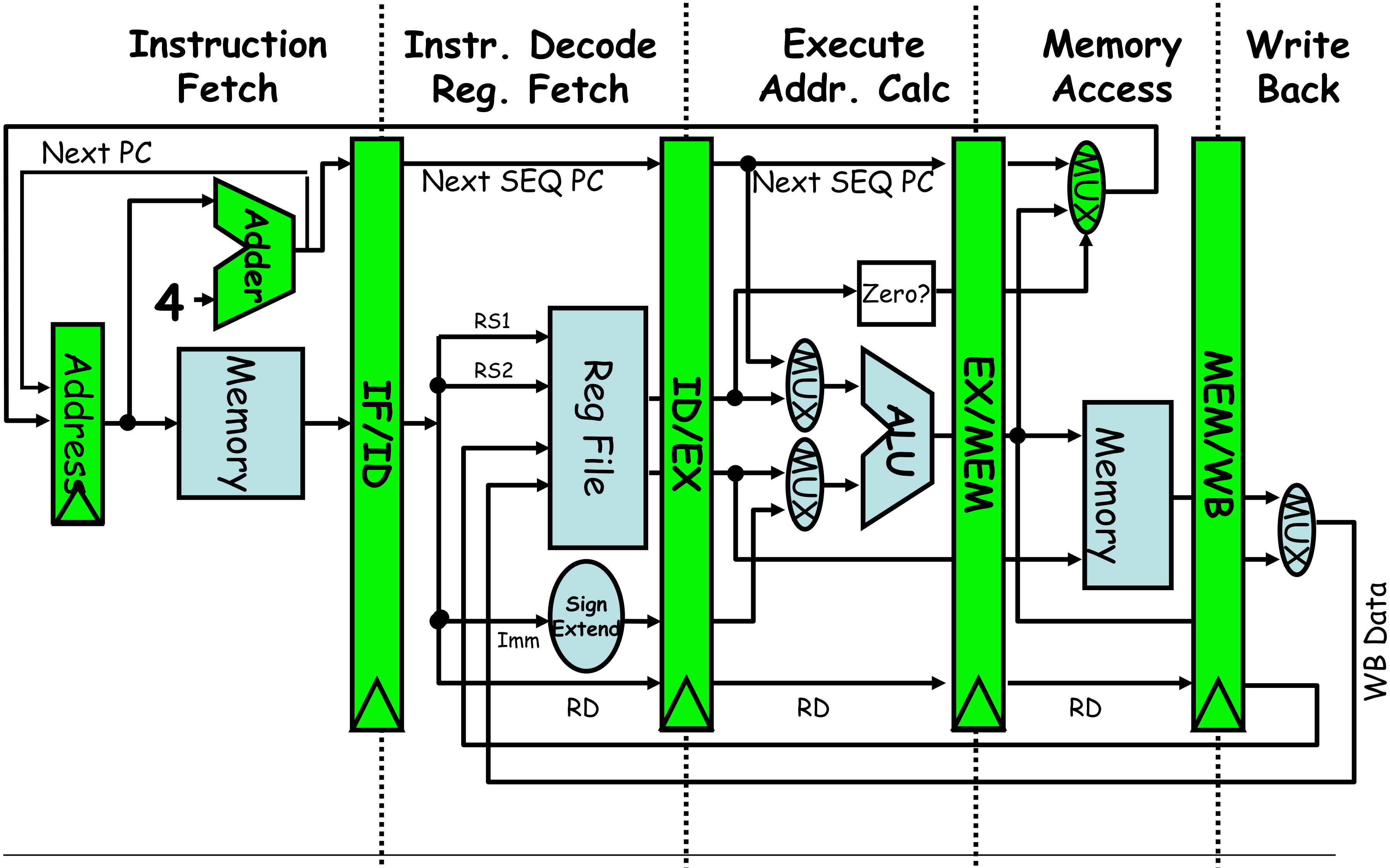
2. Ορισμένες μονάδες δεν αναπαράγονται αρκετά για να ικανοποιήσουν όλους τους πιθανούς συνδυασμούς από εντολές.

- Αρχείο καταχωρητών που επιτρέπει μόνο μια πρόσβαση (Only one register-file port)
- Μια πόρτα μνήμης



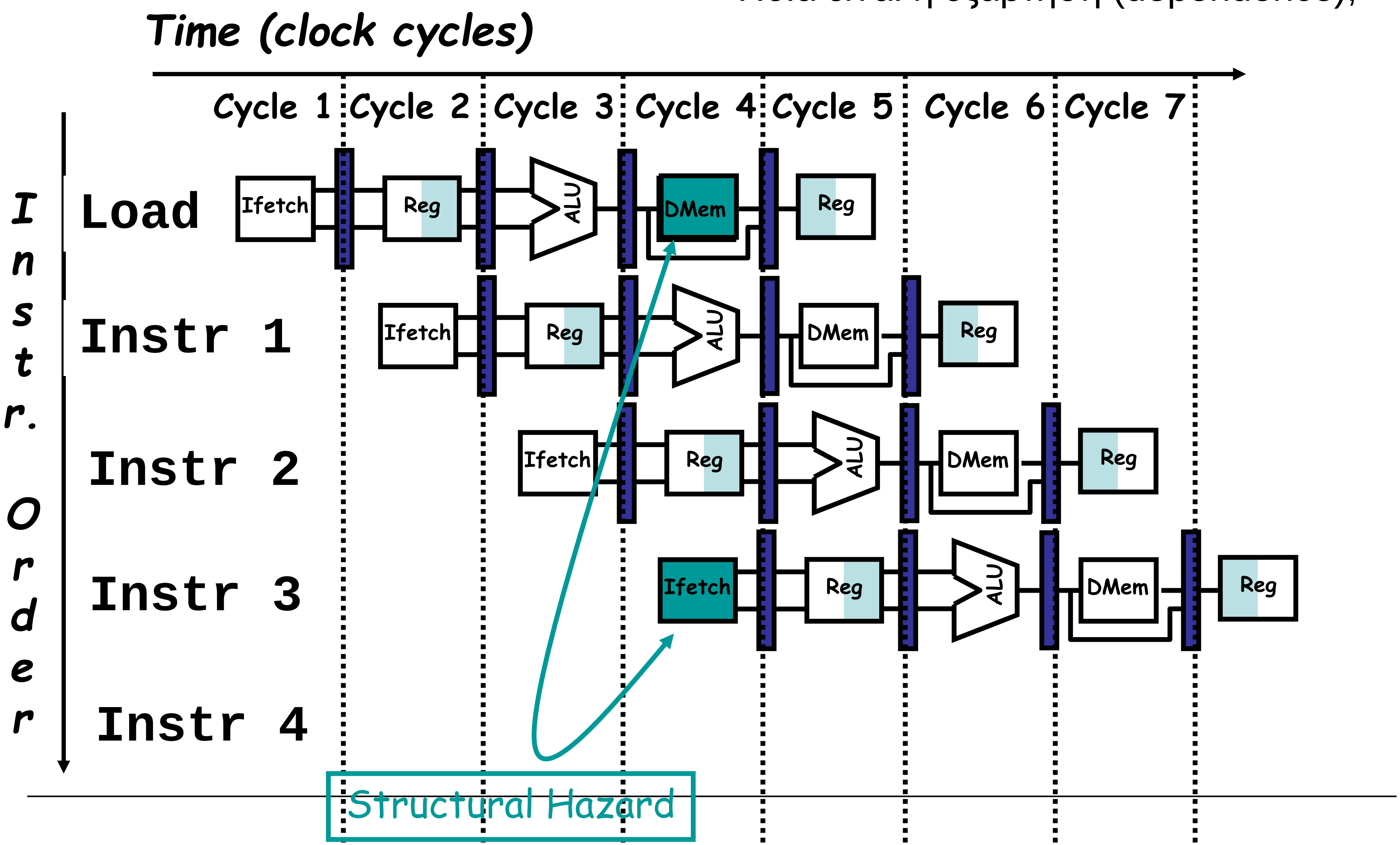
Παράδειγμα: μια πόρτα μνήμης

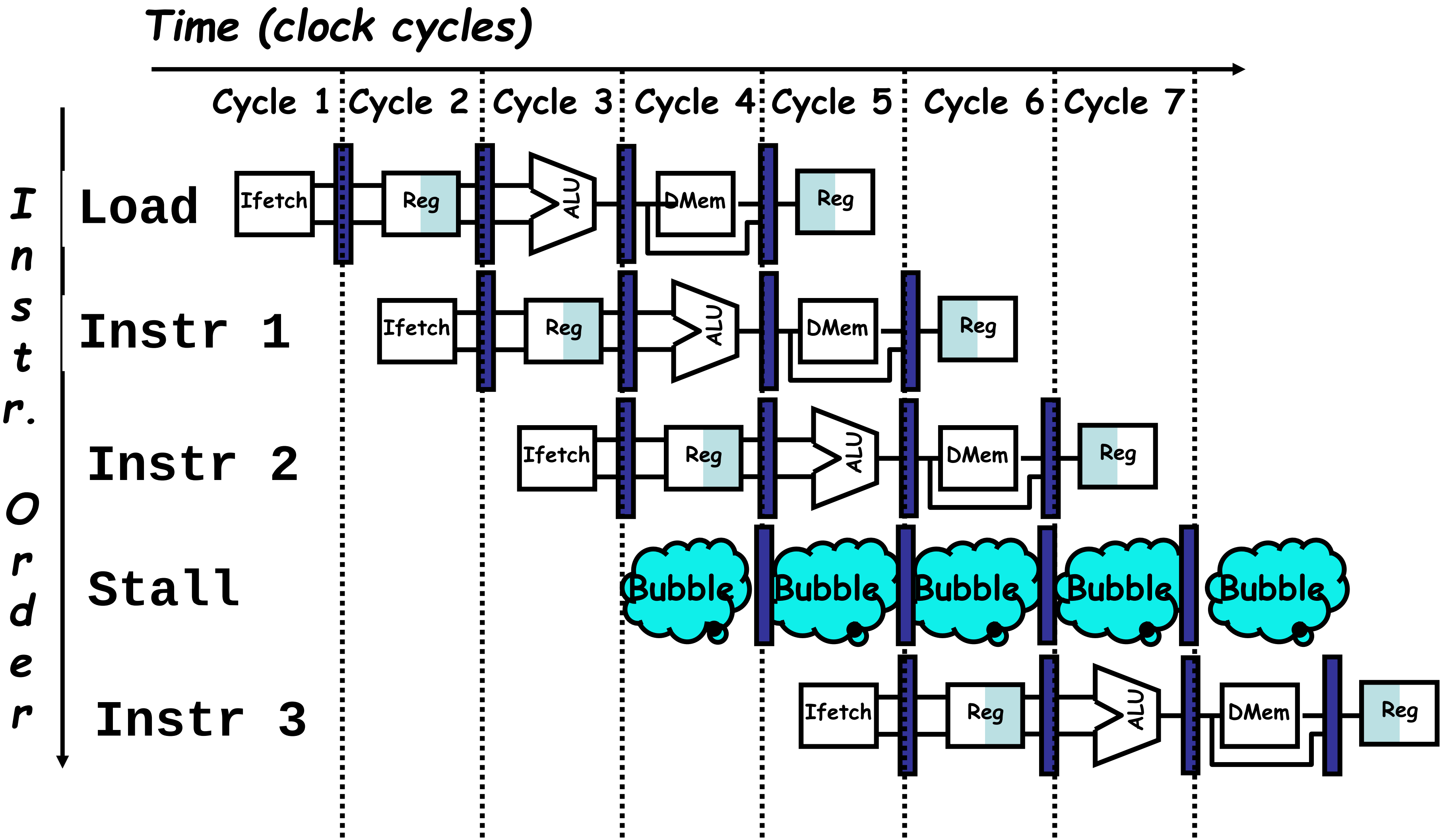
Για ευκολία… (show 2 memory boxes)



Παράδειγμα: μια πόρτα μνήμης

Ποια είναι η εξάρτηση (dependence);





Πως προκάλεις "bubble" στο pipeline?

Bubble στο FETCH

Στο στάδιο fetch ελέγχει το πιο κάτω:

```
if (Optype[E/M]==memory)
    PC = PC                      // stall
    Instruction[F/D]=nop         // bubble
else
    normal
```

Bubble/Stall Γενικά

- Εάν υπάρχει σε στάδιο εντολή που έχει hazzard, τότε
  - ΠΡΟΗΓΟΥΜΕΝΑ ΣΤΑΔΙΑ STALL (διατηρούν κατάσταση)
  - ΣΤΟ ΕΠΟΜΕΝΟ ΣΤΑΔΙΟ NOP (Bubble)
  - ΤΑ ΑΛΛΑ ΣΤΑΔΙΑ ΠΡΟΧΩΡΟΥΝ ΚΑΝΟΝΙΚΑ

Μηχανισμός Εξασφάλισης της Διασωλήνωσης [Pipeline Interlocks]

Επισημαίνει κινδύνους και Σταματά την τροφοδότηση στη διασωλήνωση της εντολής που θα χρησιμοποιήσει ένα δεδομένο μέχρι αυτό να παραχθεί από την παραγωγό εντολή

Μονάδα ελέγχου κινδύνων για κάθε στάδιο.

Λύσεις για κίνδυνους δομής

Λύση 1: Wait

⇒must detect the hazard

⇒must have mechanism to stall

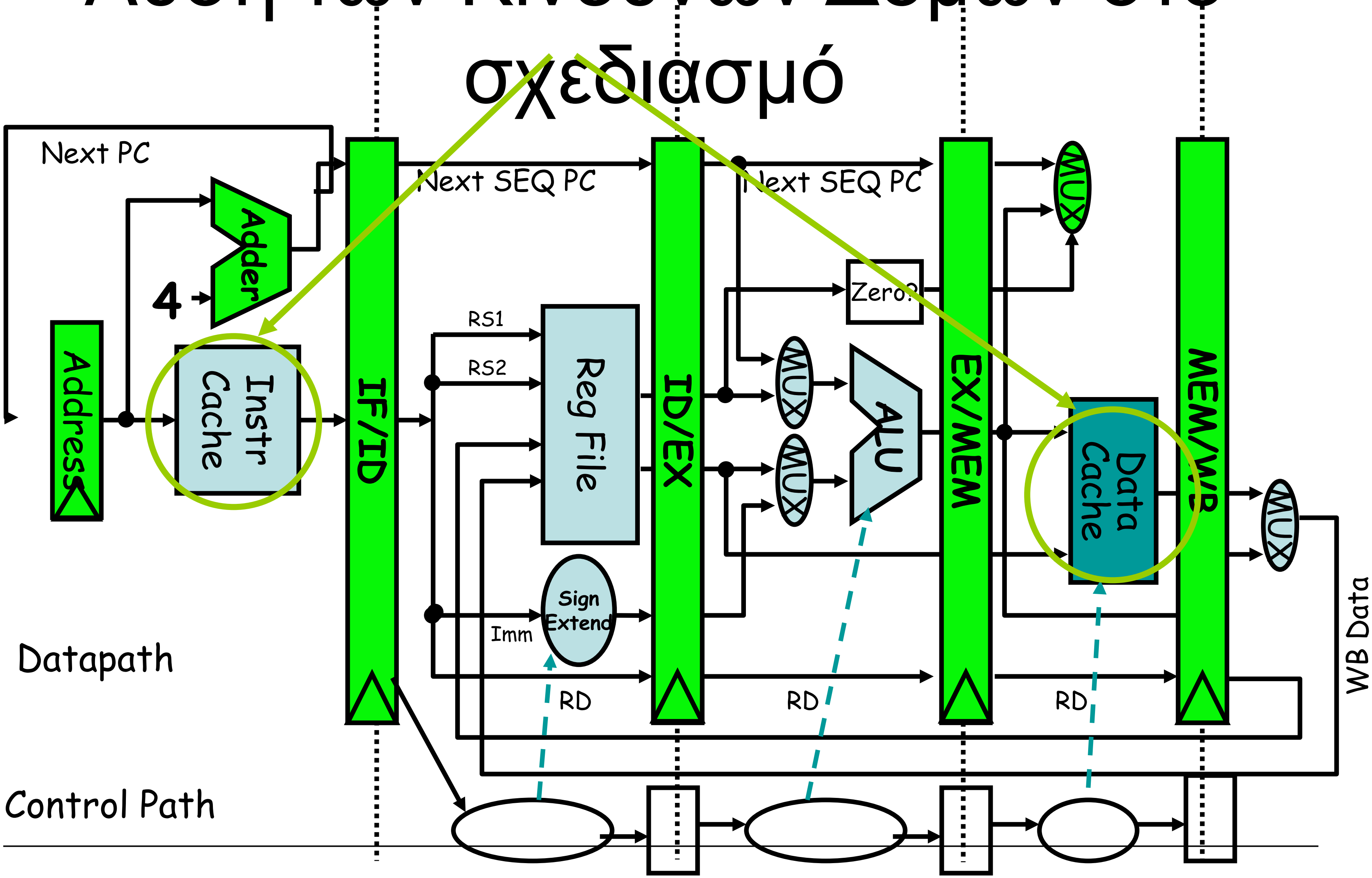
⇒Χάνεις επίδοση

Εάν οι εντολές πρόσβασης στη μνήμη είναι το 30% του ολικού αριθμού των εντολών. Πόσο πιο γρήγορη είναι η ιδανική μηχανή σε σύγκριση με την πραγματική μηχανή;

Λύση 2: Throw more hardware at the problem

Επίδοση με δύο πόρτες σε σχέση με ιδεατή;

Λύση των Κίνδυνων Δομών στο σχεδιασμό



Εξάρτηση Δεδομένων: RAW

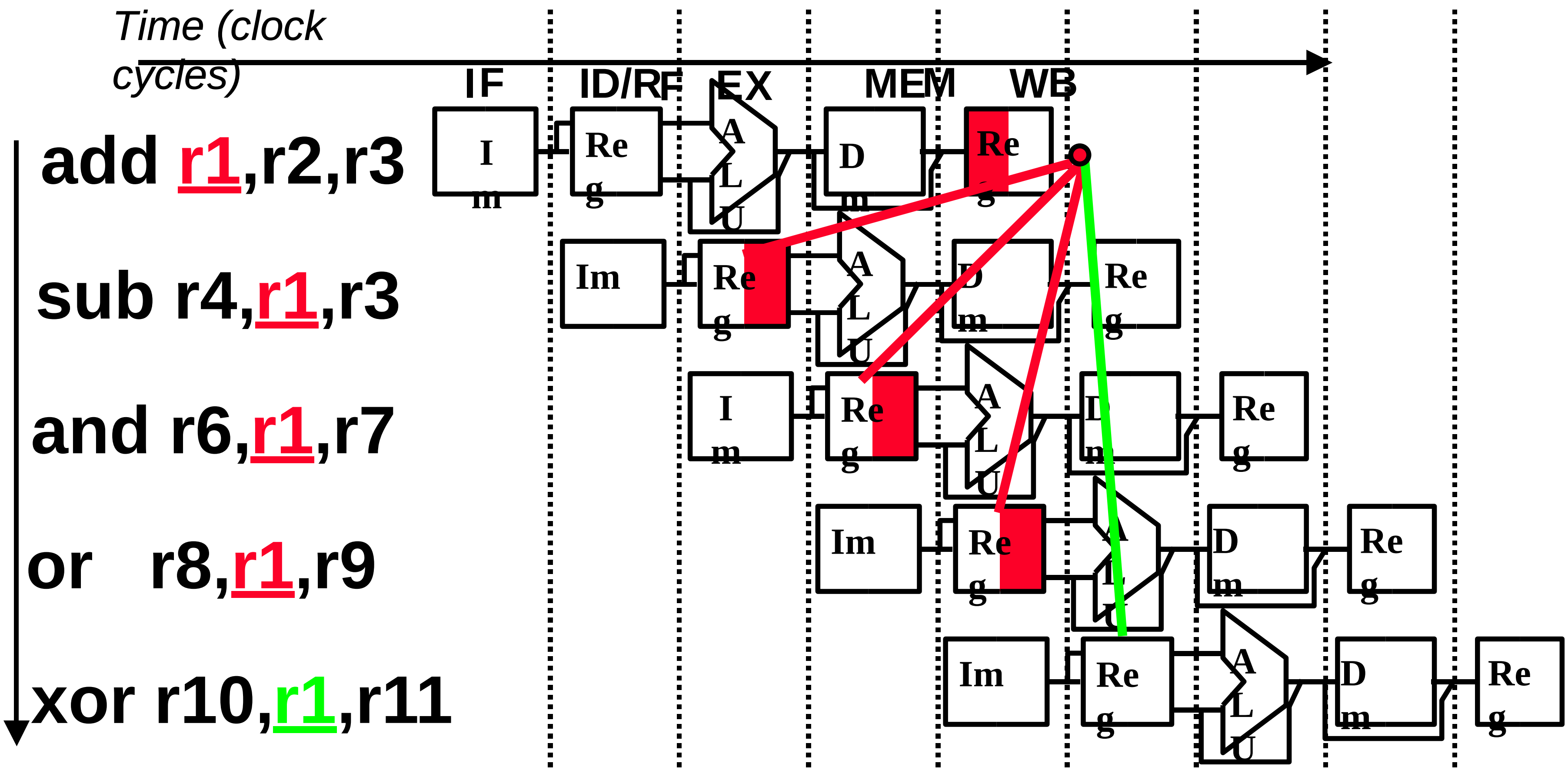
Εξάρτηση Read After Write (RAW)

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

Αυτό μπορεί να προκαλέσει κίνδυνο σε ένα pipeline. Πότε;

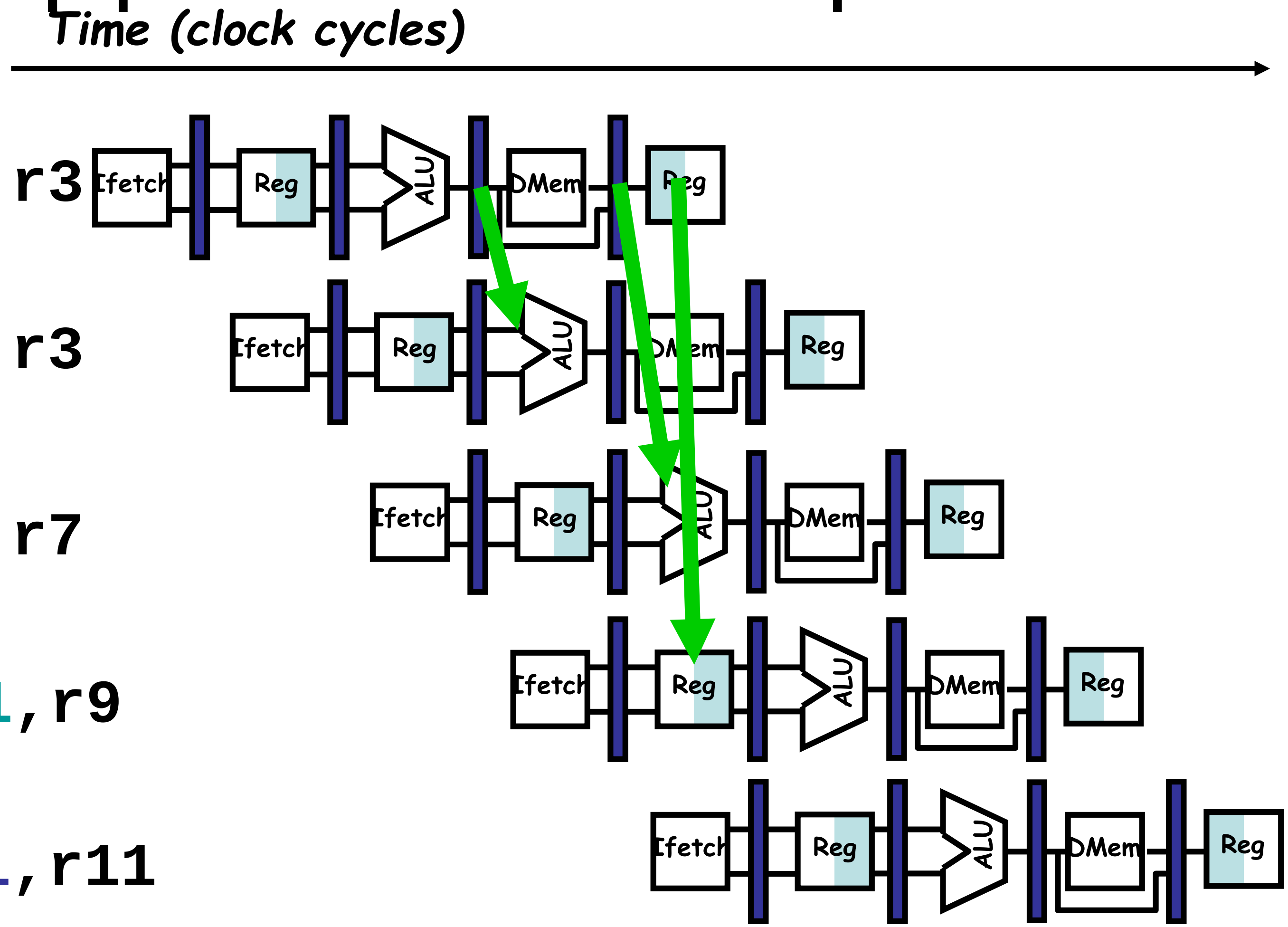
Κίνδυνοι-Δεδομένων (Data Hazards)

Κίνδυνοι-Δεδομένων δημιουργούνται όταν η διασωλήνωση διαφοροποιεί τη σειρά πρόσβασης σε τελεστές σε σύγκριση με τη σειρά πρόσβασης χωρίς διασωλήνωση



Ποιες είναι οι εξαρτήσεις (dependences) και ποιοι οι κίνδυνοι (hazards);

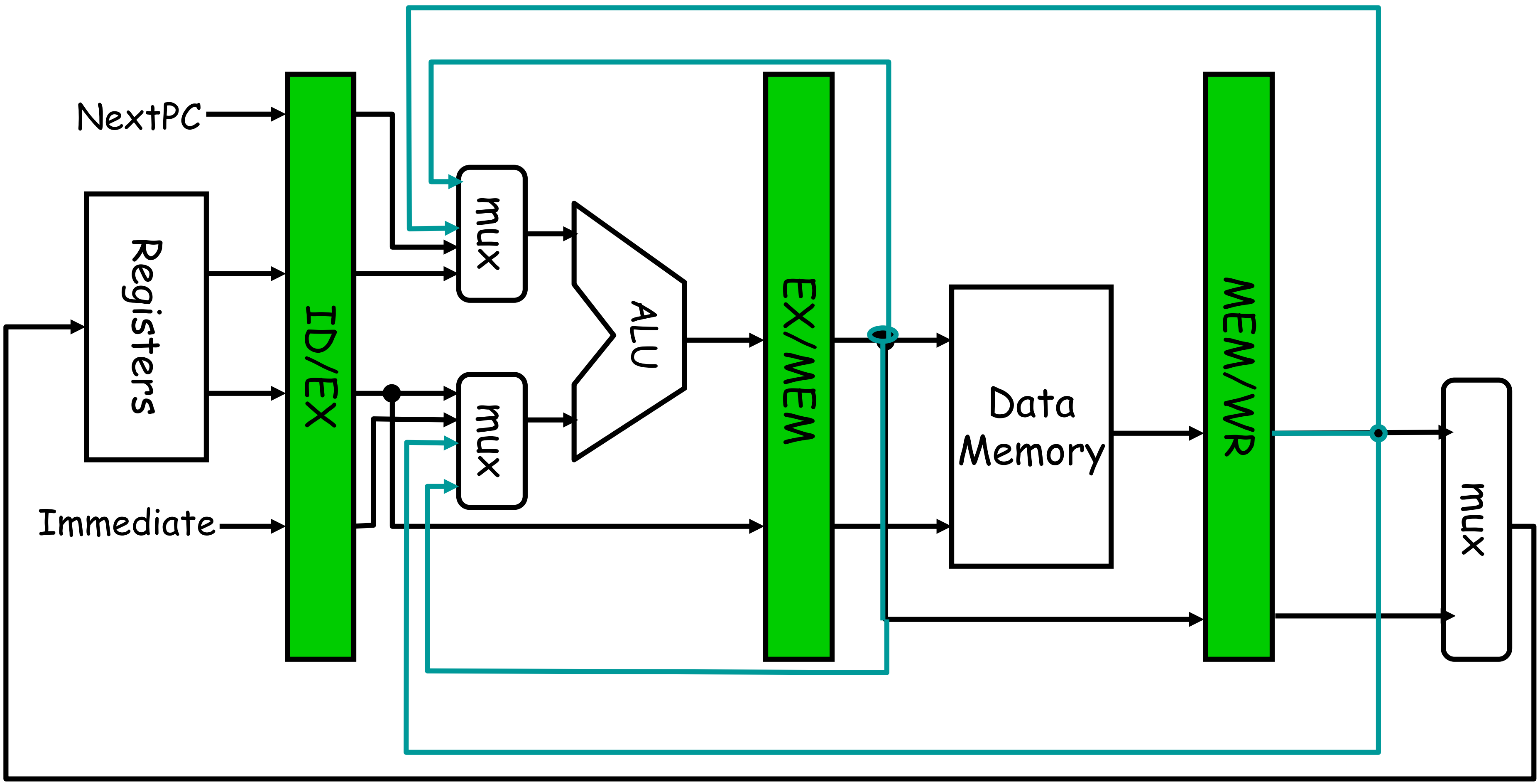
Μεταβίβαση (Forwarding) για αποφυγή Κινδύνου Δεδομένων



Instr. Order

```
add r1,r2,r3
sub r4,r1,r3
and r6,r1,r7
or   r8,r1,r9
xor r10,r1,r11
```

Αλλαγές στο υλικό για Forwarding



Τι σημαίνει μεταβίβαση (forwarding);

- Μεταχείριση πεδίων των pipeline registers σαν πιθανά inputs στις εντολές
- Πχ: στο στάδιο EXEC εκτελούμε το πιο κάτω:

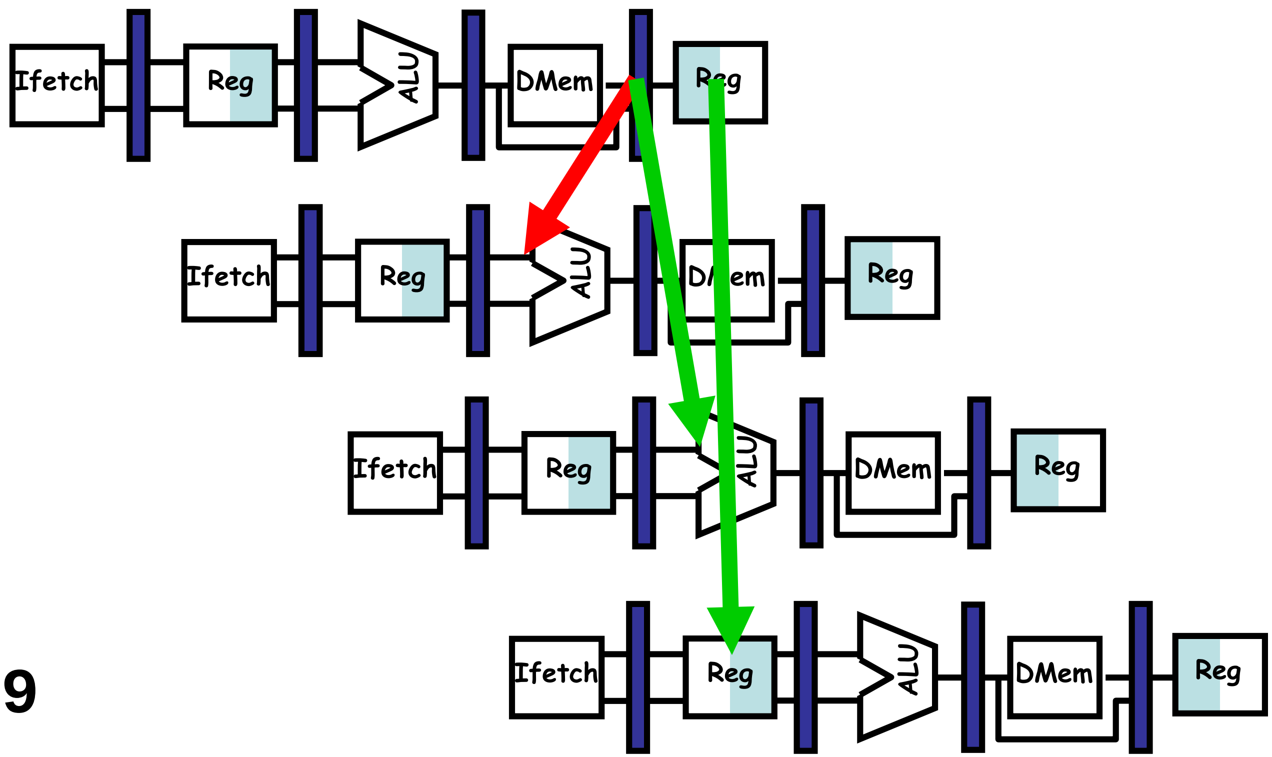
```
if (SrcReg1[D/E] != NULL)
    if (DestReg[E/M]==SrcReg1[D/E])
        ALUSrc1= DestValue[E/M]
    else if (DestReg[M/W]==SrcReg1[D/E])
        ALUSrc1= DestValue[M/W]
    else
        ALUSrc1= SrcValue1[D/E]
```

Κίνδυνοι Δεδομένων και με Forwarding

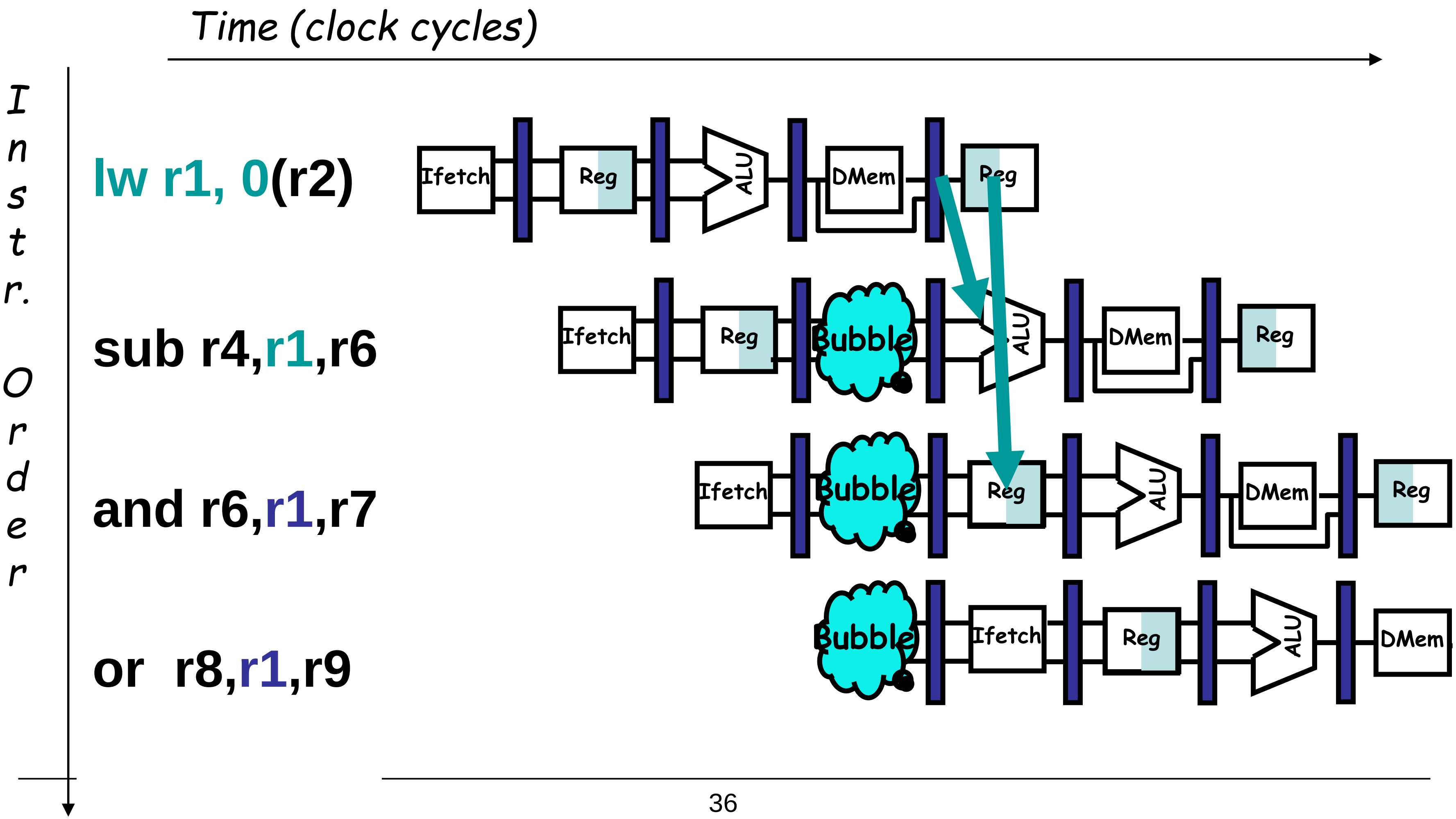
Time (clock cycles)

Instr. Order

```
lw r1,0(r2)
sub r4,r1,r6
and r6,r1,r7
or   r8,r1,r9
```



Λύσεις του κινδύνου της εντολής φόρτωσης



Παράδειγμα:

- Το 20% των εντολών είναι φόρτωσης (loads)
- Στο 50% των περιπτώσεων η επόμενη εντολή χρησιμοποιεί το αποτέλεσμα της εντολής φόρτωσης Ο κίνδυνος αυτός δημιουργεί μια στάση στη διασωλήνωση

• Ερώτηση: Πόσο πιο γρήγορη είναι η διασωλήνωση χωρίς τον κίνδυνο αυτό σε σύγκριση με την πραγματική μηχανή

Παράδειγμα:

– Το 20% των εντολών είναι φόρτωσης

– Στο 50% των περιπτώσεων η επόμενη εντολή χρησιμοποιεί το αποτέλεσμα της εντολής φόρτωσης. Ο κίνδυνος αυτός δημιουργεί μια στάση στη διασωλήνωση

• Ερώτηση: Πόσο πιο γρήγορη είναι η διασωλήνωση χωρίς τον κίνδυνο αυτό σε σύγκριση με την πραγματική μηχανή

Το CPI της εντολής που ακολουθεί την εντολή φόρτωσης και χρησιμοποιεί το αποτέλεσμα της φόρτωσης είναι 1.5

CPI= 1 + 0.2 * 0.5 *1 = 1.1

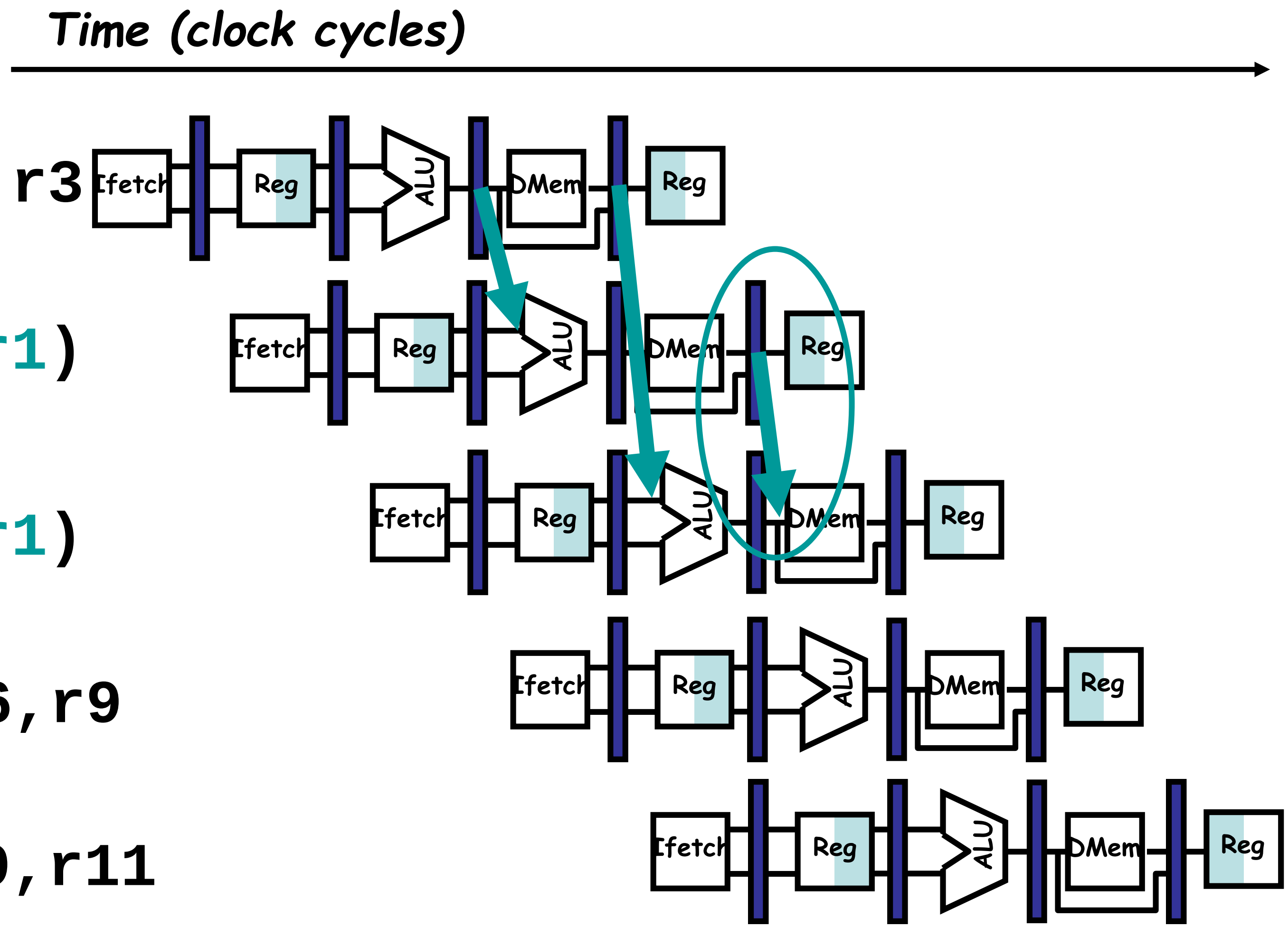
Performance ratio = 1.1 / 1 i.e. 10% faster

Forwarding to Avoid LW-SW Data Hazard

Figure A.8, Page A-20

Instr. Order

```
add r1,r2,r3
lw r4, 0(r1)
sw r4,12(r1)
or   r8,r6,r9
xor r10,r9,r11
```



Διαβίβαση Δεδομένων (Data Forwarding)

- Το σύστημα διασωλήνωσης πρέπει να διαβιβάζει δεδομένα μεταξύ διαφόρων σταδίων

- Κίνδυνοι Δεδομένων με εντολές μνήμης

#Ποτέ υπάρχει εξάρτηση;

sw r4,0(r5)

..

..

lw r3,4(r6)

Τι είναι το πιθανό είδος εξάρτησης;   Memory Dependences

Ο κίνδυνος αυτός δεν παρουσιάζεται διότι οι προσπελάσεις στη μνήμη γίνονται πάντοτε στη σωστή σειρά. (Μόνο το τμήμα MEM διαβάζει ή γράφει στη μνήμη)

Data Dependences: WAR

Write After Read (WAR)

Instr$_J$ writes operand *before* Instr$_I$ reads it

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

Called an "anti-dependence".

This results from reuse of the name "r1".

Can't cause a hazard in simple pipeline because:

- Reads are always in stage 2, and
- Writes are always in stage 5

Data Dependences: WAW

Write After Write (WAW)

Instr$_J$ writes operand *before* Instr$_I$ writes it.

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

Called an "output dependence"

This also results from the reuse of name "r1".

Can't cause a hazard in simple pipeline because:

- Writes are always in stage 5

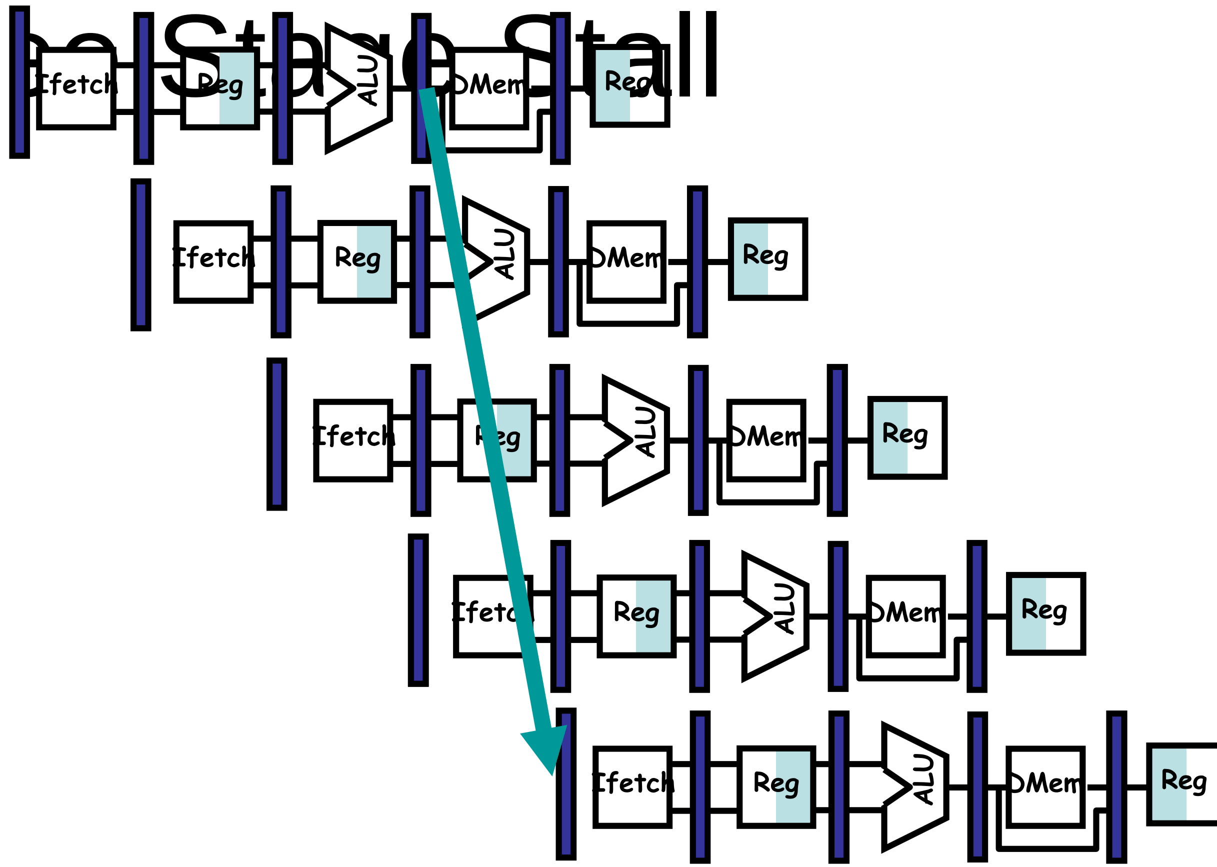
Will see WAR and WAW in more complicated pipes

Κίνδυνοι Διακλαδώσεων

```
10: beq r1,r3,36

14: and r2,r3,r5

18: or  r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11
```

Κίνδυνοι Διακλαδώσεων

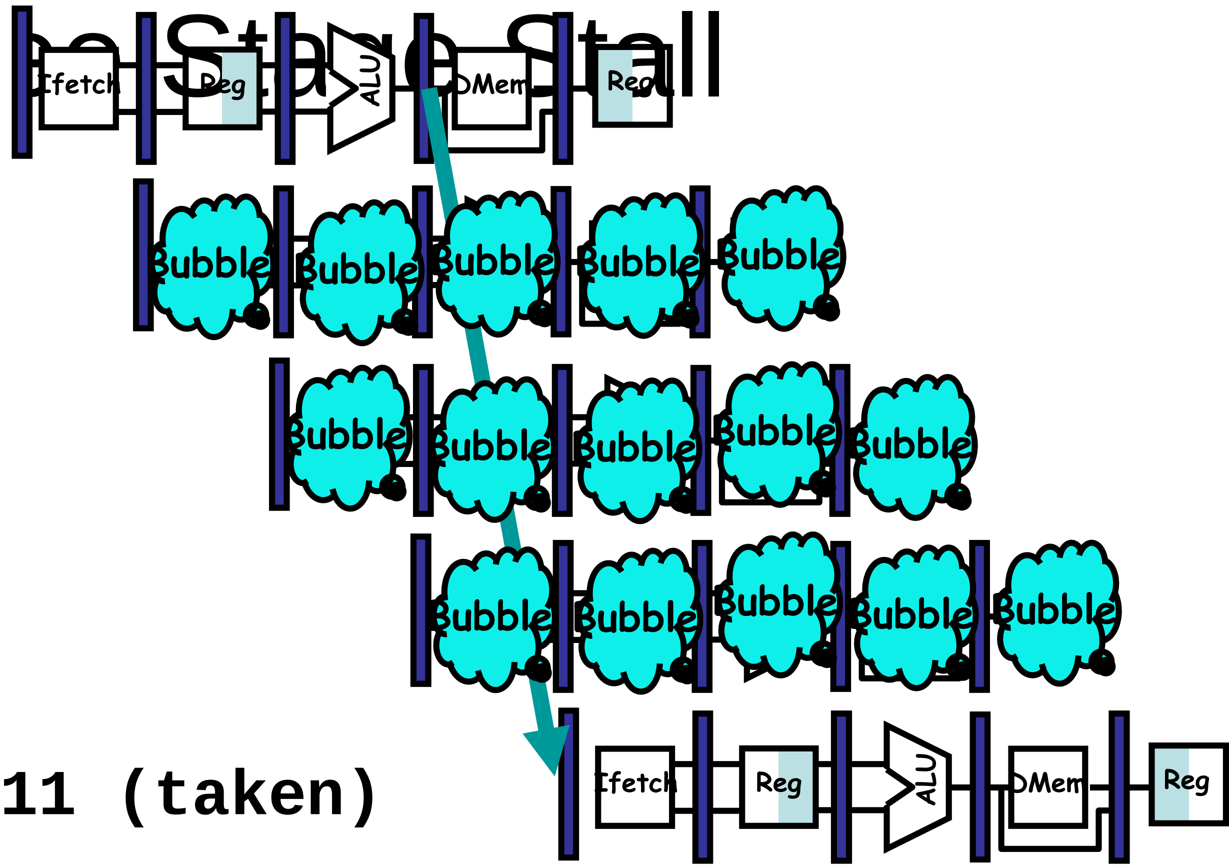
10: beq r1,r3,36



Τι θα γίνει όταν διαβάσουμε εντολή διακλάδωσης?

Κίνδυνοι Διακλαδώσεων

```
10: beq r1,r3,36

stall

stall

stall

36: xor r10,r1,r11 (taken)
      ή
14: and r2,r3,r5 (not taken)
```



Κίνδυνοι Διακλαδώσεων (Κίνδυνοι Ροής Ελέγχου)

Οι Κίνδυνοι Διακλαδώσεων δημιουργούνται όταν έχουμε μια διακλάδωση (η οποία μπορεί και να βασίζεται σε αποτέλεσμα άλλης εντολής) στην διασωλήνωση που ακολουθείτε από άλλες εντολές.

Παράδειγμα: Υπολογίστε την επιτάχυνση του pipeline εάν το 30% των εντολών είναι διακλαδώσεις

- **Οι διακλαδώσεις στο** MIPS χρειάζονται 3 τμήματα (IF, ID,EXE)
- **3 stalls (στάσεις, φυσσαλίδες)**
- **Ideal CPI = 1.**
- **CPIreal= 1 + 0.3 x 3 = 1.9**
- **Μόνο 50% της ιδανικής επιτάχυνσης είναι εφικτή**.

- Η μείωση του κόστους της επιτάχυνσης είναι εξαιρετικά σημαντική Ειδικά για μηχανές υψηλής απόδοσης
- Πολλαπλά στάδια μεταξύ fetch και execution)
- Τρόποι για μείωση κινδύνων λόγο διακλαδώσεων:

#1: Stall until branch direction is known

#2: Determine target and direction early

#3: Predict Branch Not Taken

#4: Predict Branch Taken

#5

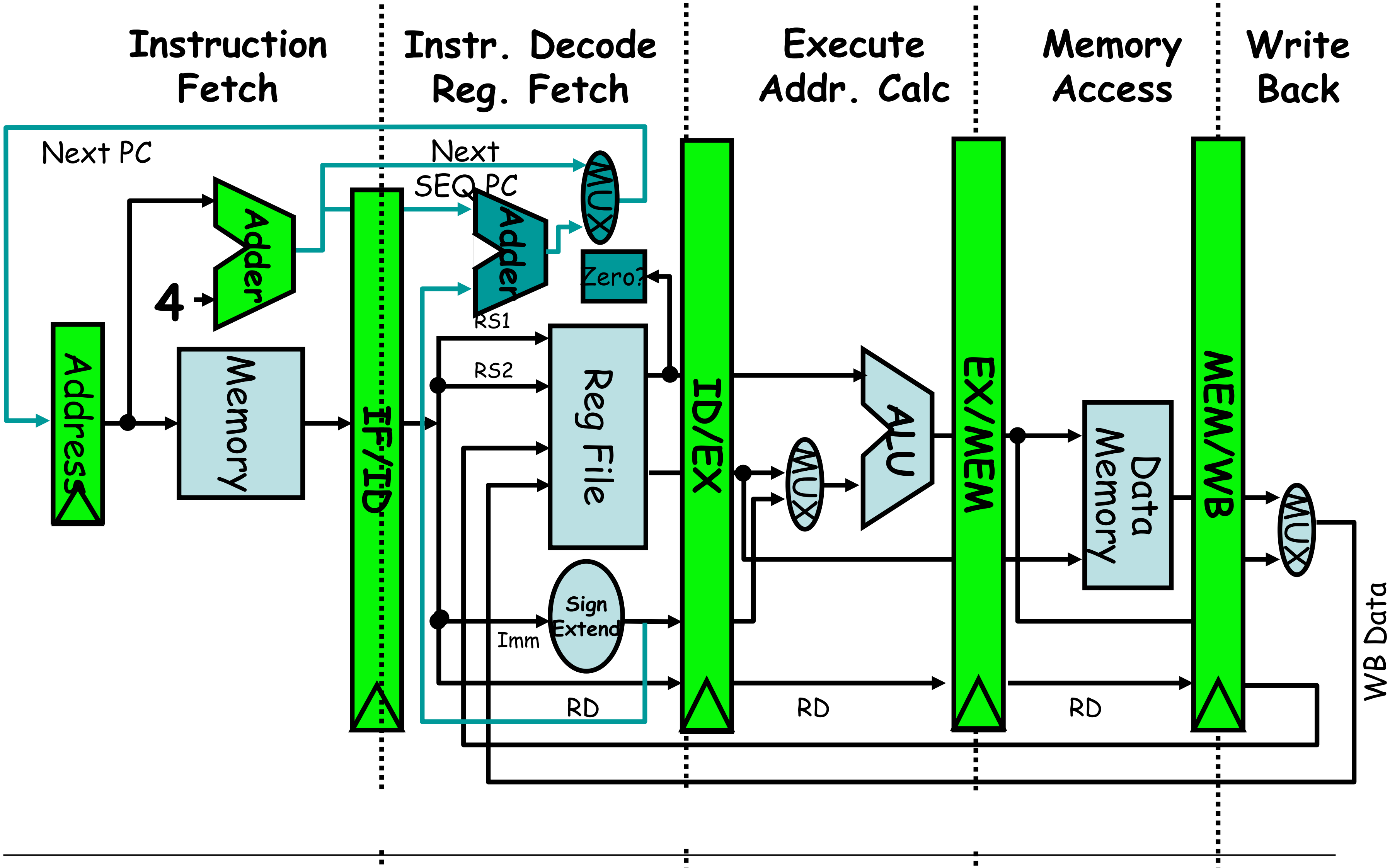
#6

…

- Οι στάσεις που οφείλονται σε κινδύνους διακλαδώσεων μπορούν να ελαττωθούν εάν:

1. Προσδιορίσουμε πιο νωρίς στην διασωλήνωση εάν η διακλάδωση θα είναι taken, και
2. Υπολογίσουμε το νέο PC πιο νωρίς

Μείωση stalls λόγο διακλαδώσεων



Η μείωση επιτυγχάνεται με:

- Την προσθήκη ειδικού κυκλώματος για να ελέγχει τη συνθήκη διακλάδωσης (test-for-zero) στο τμήμα ID

Την προσθήκη ειδικού αθροιστή στο τμήμα ID για τον υπολογισμό της Διεύθυνσης του Προορισμού της Διακλάδωσης(BTA: branch target address).

Γιατί όχι στο στάδιο fetch;

Το κόστος της διακλάδωσης τώρα ένας κύκλος

Μειονεκτήματα;

Μείωση του κόστους των διακλαδώσεων σε διασωληνωμένες μηχανές

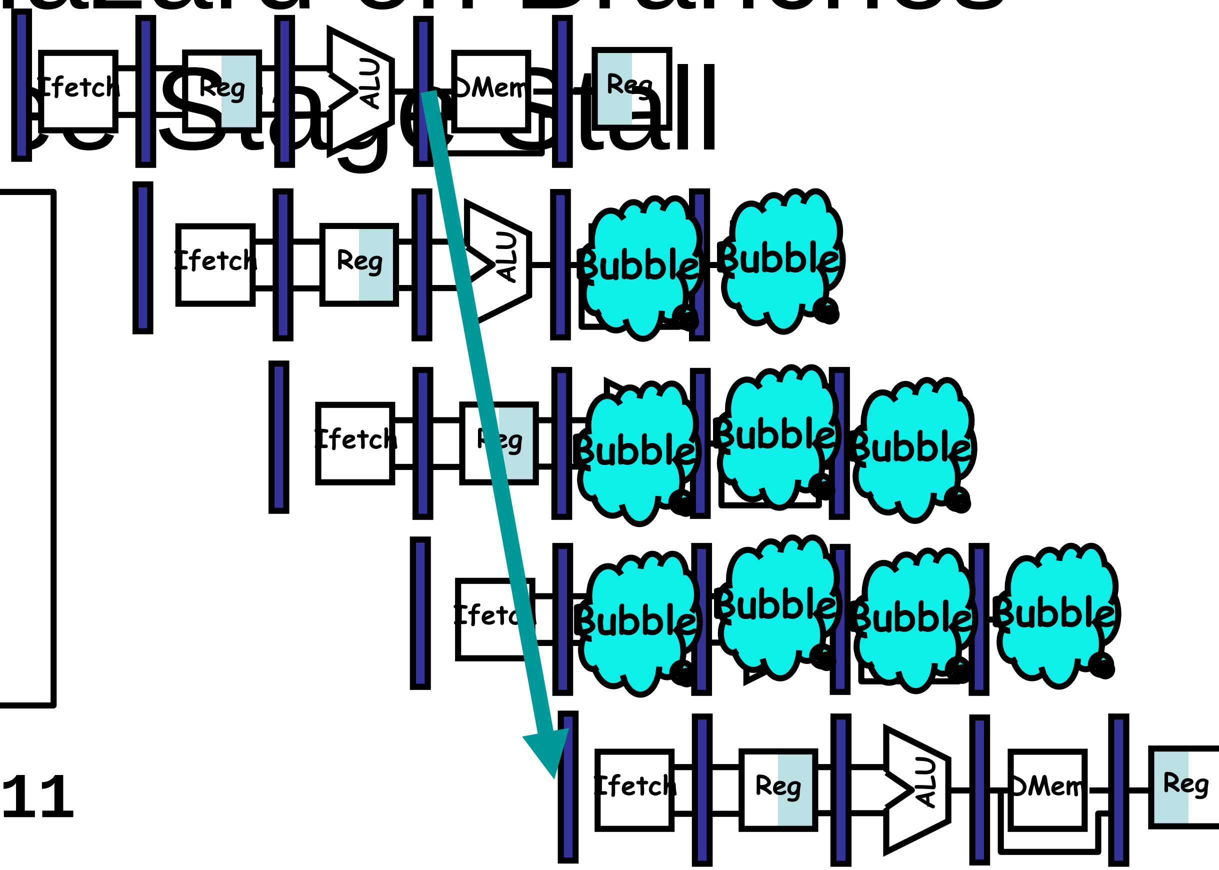
- Στατικές Προβλέψεις
  - Πρόβλεψη ότι η διακλάδωση θα είναι επιτυχής (taken)
  - Πρόβλεψη ότι η διακλάδωσης δεν θα είναι επιτυχής (not taken)

#3: Predict Branch Not Taken

- Execute successor instructions in sequence
- “Flush/Squash” instructions in pipeline if branch actually taken
  - **Τι σημαίνει squash?**
- 47% branches not taken on average
- PC+4 already calculated, so use it to get next instruction

Control Hazard on Branches

```
10: beq r1,r3,36

14: and r2,r3,r5

18: or  r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11
```



Squash Pipeline

- Σε όλα τα στάδια πριν το branch εισαγωγή bubble.

#4: Predict Branch Taken

- 53% branches taken on average
- But haven’t calculated branch target address
- Still incur 1 cycle branch penalty
  - Other machines: branch target known before outcome

Διακλαδώσεις με καθυστέρηση [Delayed branch]

- Διακλαδώσεις με καθυστέρηση *n* κύκλων

Η Διακλάδωση με καθυστέρηση δεν λαμβάνει χώρα μέχρις ότου ένας αριθμός εντολών (n) που ακολουθούν τη διακλάδωση εκτελεστούν

Branch Instruction
Sequential Successor 1
Sequential Successor 2

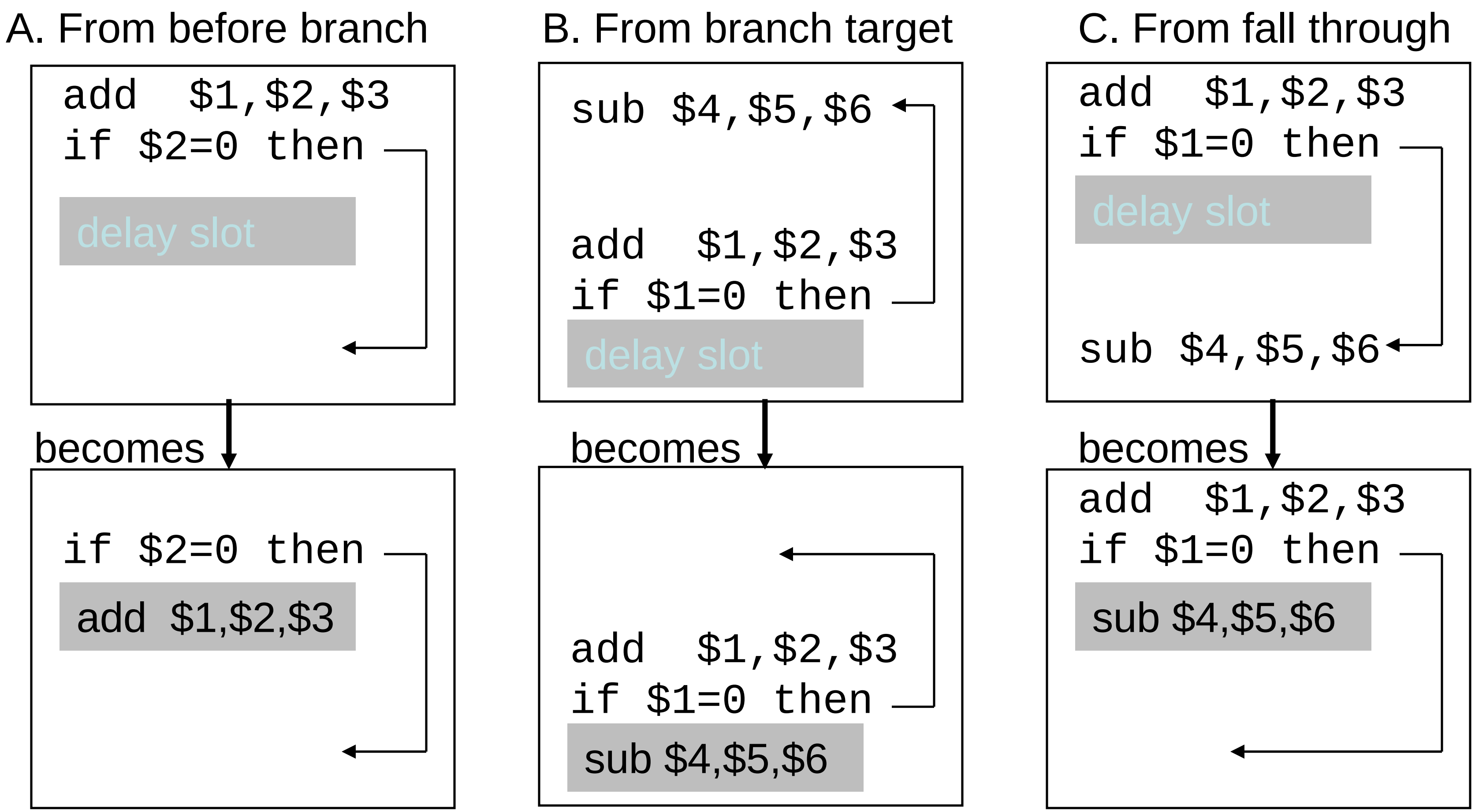
Sequential Successor n
branch target if taken

Ο MIPS έχει μια στάση για κάθε διακλάδωση. Δηλαδή η Διακλάδωση με καθυστέρηση δεν λαμβάνει χώρα μέχρι να εκτελεστεί η εντολή που ακολουθεί την διακλάδωση.

Σχισμή Καθυστέρησης Διακλάδωσης: Η σχισμή που ακολουθεί τη σχισμή της Διακλάδωσης με καθυστέρηση [Branch Delay Slots]

Μέρος της αρχιτεκτονικής συνόλου εντολών MIPS

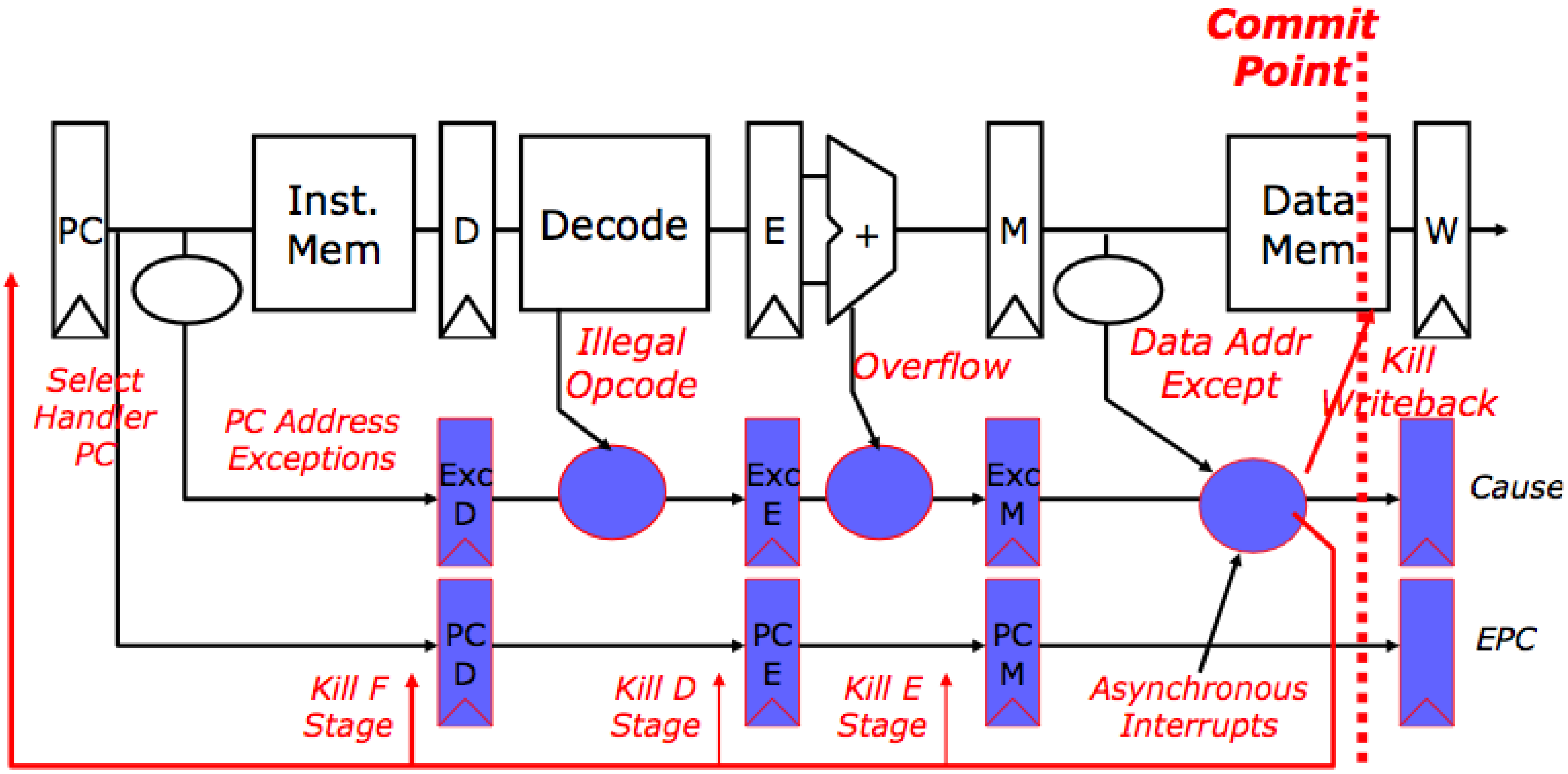
Scheduling Branch Delay Slots



In B and C, must be okay to execute `sub` when branch fails

- Αλλιώς πρέπει να ακυρώσεις την εντολή στο delay slot
- Ποτε ok not to cancel?

Precise Exceptions



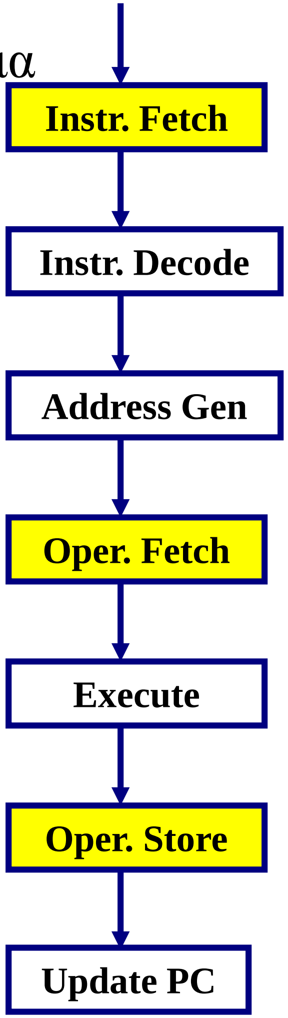
Πως χειριζόμαστε τις εξαιρέσεις και διακοπές (exceptions και interrupts)

- Η εντολή που προκαλεί την εξαίρεση την “σημειώνουμε” (flag) στον καταχωρητή διασωλήνωσης στο τέλος του σταδίου που βρίσκεται η εντολή
    - Οι διακοπές σημειώνονται στο προτελευταίο στάδιο
- Όταν η εντολή φτάσει στο “τελευταίο” στάδιο εξυπηρετούμαι το exception/interrupt
- Διασφαλίζεται πως εξυπηρετείτε το exception της πιο παλιάς εντολής
- Όλες οι προηγούμενες εντολές έχουν τελειώσει και καμία πιο μετά δεν έχει ξεκινήσει

Διασωλήνωσης Μονάδων (imbalanced stages – μη ισοζυγισμένα στάδια)

Υποθέστε αρχικτενική [mem]= [mem] op [mem] το pipeline με 7 στάδια

Σε κάθε κύκλο μηχανής τελείωνει η επεξεργασία μιας εντολής



- Ιδεάτα: Επιτάχυνση της τάξης του 7 σε σύγκριση με μη-διασωληνωμένη μηχανή.

Υποθέστε ότι η μνήμη χρειάζεται 4 φορές πιο αργή σε σχέση με τα στάδια χωρίς μνήμη.

- Το ρολόι του pipeline έχει περίοδο 4 φορές το χρόνο που χρειάζονται τα γρήγορα στάδια (κ) => 4κ

Με αυτό το ρυθμό παράγεται ένα αποτέλεσμα σε κάθε κύκλο.

- Η μη-διασωληνωμένη μηχανή χρειάζεται ρολόι με περίοδο

- 3 x 4 + 4 x 1 = 16κ
- ***Επιτάχυνση της τάξης του 4*** Ερώτηση: Πως μπορούμε να βελτιώσουμε αυτή την διασωλήνωση

Διασωλήνωσης Μονάδων

Προσθέτουμε 3 τμήματα καθυστέρησης μετά από κάθε τμήμα μνήμης

Η νέα διασωλήνωση έχει 16 τμήματα
Ένα αποτέλεσμα παράγεται σε κάθε κύκλο.

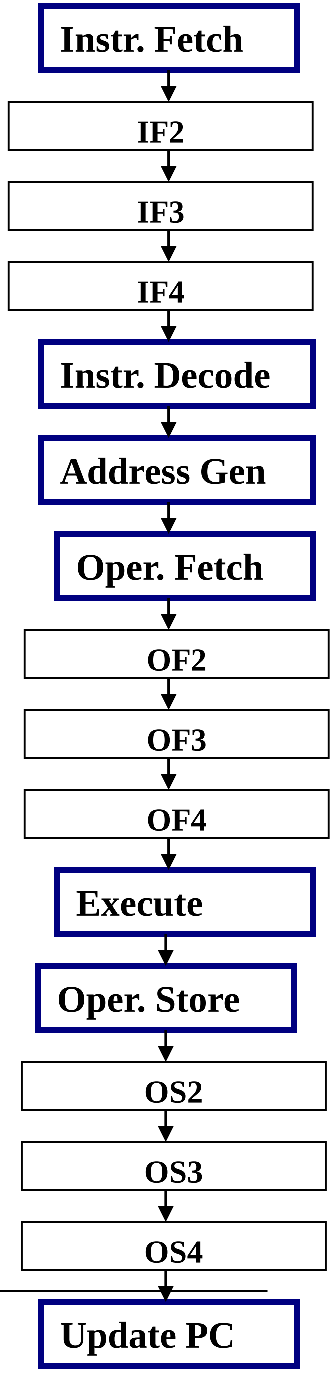
• *Επιτάχυνση της τάξης του 16*

• Ερώτηση: Είναι αυτό εφικτό/πρακτικό;

• Τι γίνεται με τις διακλαδώσεις;

• Η καινούργια διασωλήνωση χρειάζεται ταυτόχρονα 12 προσπελάσεις μνήμης

Σε αυτή την περίπτωση έχουμε λύση ένα πρόβλημα και δημιουργήσαμε άλλο



Χρονοπρογραμματισμός Εντολών [Instruction Scheduling]

Οι Μεταγλωττιστές επιδιώκουν να αποφεύγουν την παραγωγή κώδικα στον οποίο μια εντολή φόρτωσης ακολουθείται από μια ή περισσότερες εντολές που θα χρησιμοποιήσουν το αποτέλεσμα της φόρτωσης

Παράδειγμα: Γράψετε κώδικα που αποφεύγει στάσεις στη διασωλήνωση του MIPS (υποθέστε a,b,c,d,e διευθύνσεις μνήμης)

a = b + c

d = e - f

Unoptimized Code

```
lw Rb,  b
lw  Rc, c
add Ra, Rb, Rc
sw a , Ra
lw Re,  e
lw  Rf, f
sub Rd, Re, Rf
sw d, Rd
```

Optimized Code

```
lw Rb,  b
lw  Rc, c
lw Re,  e
add Ra, Rb, Rc
lw Rf,  f
sw a , Ra
sub Rd, Re, Rf
sw d, Rd
```

Χρονοπρογραμματισμός Εντολών [Instruction Scheduling]

Οι Μεταγλωττιστές επιδιώκουν να αποφεύγουν την παραγωγή κώδικα στον οποίο μια εντολή φόρτωσης ακολουθείται από μια ή περισσότερες εντολές που θα χρησιμοποιήσουν το αποτέλεσμα της φόρτωσης

Παράδειγμα: Γράψετε κώδικα που αποφεύγει στάσεις στη διασωλήνωση του MIPS (υποθέστε a,b,c,d,e διευθύνσεις μνήμης)

a = b + c

d = e - f

	Unoptimized Code
5	**lw Rb, b**
6	**lw Rc, c**
8	**add Ra, Rb, Rc**
9	**sw a , Ra**
10	**lw Re, e**
11	**lw Rf, f**
13	**sub Rd, Re, Rf**
14	**sw d, Rd**

	Optimized Code
5	**lw Rb, b**
6	**lw Rc, c**
7	**lw Re, e**
8	**add Ra, Rb, Rc**
9	**lw Rf, f**
10	**sw a , Ra**
11	**sub Rd, Re, Rf**
12	**sw d, Rd**