Αρχιτεκτονική Υπολογιστών

Κεφάλαιο #4 (γ)

Advanced Pipelining

Διονύσης Πνευματικάτος

pnevmati@cslab.ece.ntua.gr

5ο εξάμηνο ΣΗΜΜΥ – Ακαδημαϊκό Έτος: 2020-21

Τμήμα 3 (ΠΑΠΑΔ-Ω)

http://www.cslab.ece.ntua.gr/courses/comparch/

Recall: Compute CPI?

- Ξεκινάμε με το Base CPI
- Προσθέτουμε stalls:

$$CPI = CPI_{base} + CPI_{stall}$$

$$CPI_{stall} = STALL_{type-1} \times freq_{type-1} + STALL_{type-2} \times freq_{type-2}$$

° **Εαν:**

- **CPI_{base}=1**
- **$Freq_{branch}$=20%, $freq_{load}$=30%**
- **Διακλαδώσεις προκαλούν stall για 1 κύκλο**
- **1% από όλα τα loads προκαλούν stall 100 κύκλων**

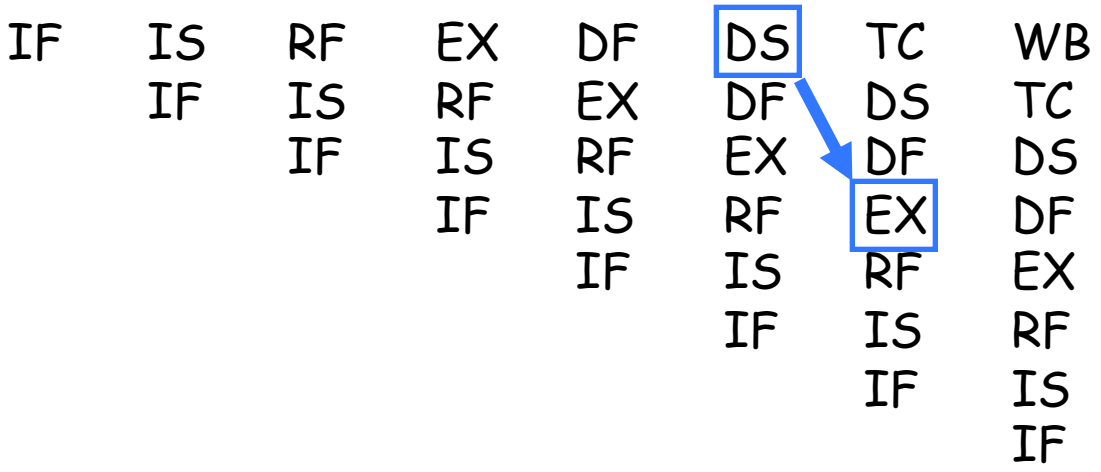
° **Τότε: CPI = 1 + (1×0.20)+(100 × 0.30×0.01)=1.5**

Case Study: MIPS R4000 (200 MHz @ 1991)

- 8 Stage Pipeline:
  - **IF**–first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access
  - **IS**–second half of access to instruction cache
  - **RF**–instruction decode and register fetch, hazard checking and also instruction cache hit detection
  - **EX**–execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation
  - **DF**–data fetch, first half of access to data cache
  - **DS**–second half of access to data cache
  - **TC**–tag check, determine whether the data cache access hit
  - **WB**–write back for loads and register-register operations
- 8 Stages: What is impact on Load delay? Branch delay? Why?

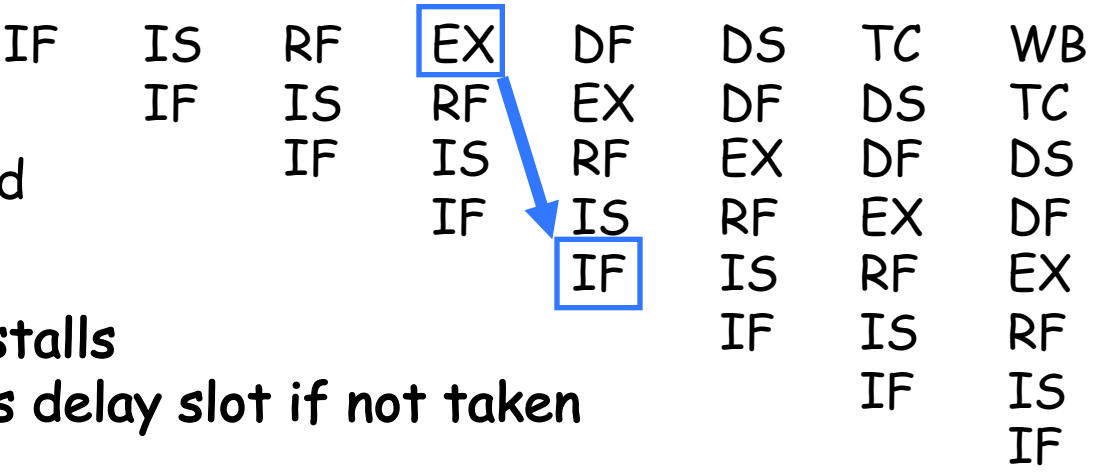
Case Study: MIPS R4000

TWO Cycle Load Latency



THREE Cycle Branch Latency

(conditions evaluated during EX phase)

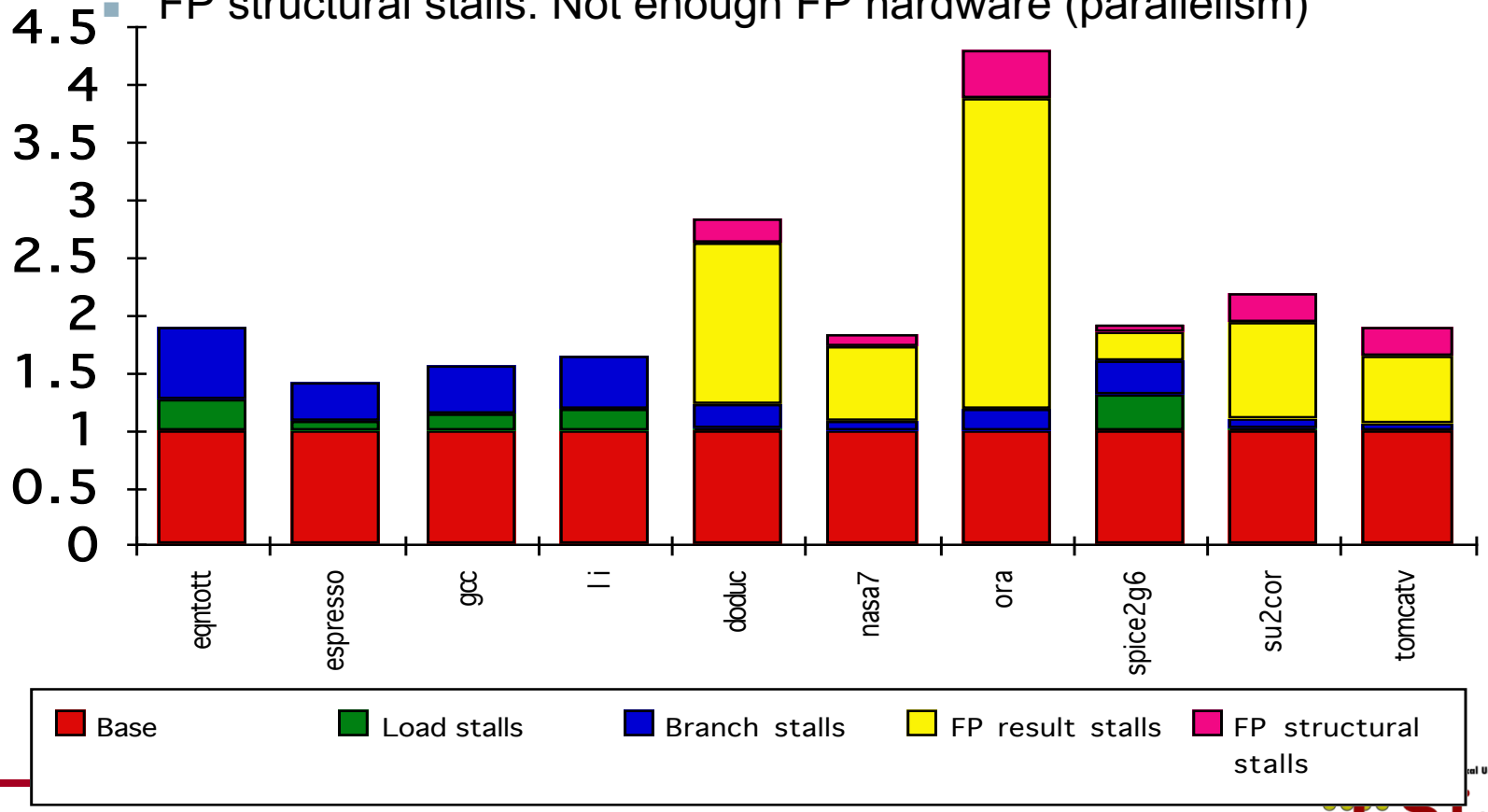


Delay slot plus two stalls

Branch likely cancels delay slot if not taken

R4000 Performance

- CPI > 1:
  - Load stalls (1 or 2 clock cycles)
  - Branch stalls (2 cycles + unfilled slots)
  - FP result stalls: RAW data hazard (latency)
  - FP structural stalls: Not enough FP hardware (parallelism)



Παραλληλία επιπέδου εντολής

- Instruction-Level Parallelism (ILP)
- Διοχέτευση: παράλληλη εκτέλεση πολλών εντολών
- Για αύξηση του ILP
  - Βαθύτερη διοχέτευση
    - Λιγότερη δουλειά ανά στάδιο ⇒ μικρότερος κύκλος ρολογιού
  - Πολλαπλή εκκίνηση (multiple issue)
    - Επανάληψη σταδίων διοχέτευσης ⇒ πολλές διοχετεύσεις
    - Εκκίνηση πολλών εντολών ανά κύκλο ρολογιού
    - CPI < 1 => χρήση του Instructions Per Cycle (IPC)
    - Π.χ., 4GHz πολλαπλή εκκίνηση 4 δρόμων (4-way multiple-issue)
      - 16 BIPS, μέγιστο CPI = 0.25, μέγιστο IPC = 4
    - Αλλά οι εξαρτήσεις το μειώνουν στην πράξη

Πολλαπλή εκκίνηση

- Στατική πολλαπλή εκκίνηση
  - Ο μεταγλωττιστής ομαδοποιεί τις εντολές που θα ξεκινήσουν μαζί
  - Τις συσκευάζει σε «υποδοχές εκκίνησης» (“issue slots”)
  - Ο μεταγλωττιστής ανιχνεύει και αποφεύγει τους κινδύνους
- Δυναμική πολλαπλή εκκίνηση
  - Η CPU εξετάζει το ρεύμα των εντολών και επιλέγει εντολές για εκκίνηση σε κάθε κύκλο
  - Ο μεταγλωττιστής μπορεί να βοηθήσει με αναδιάταξη των εντολών
  - Η CPU επιλύει τους κινδύνους με προηγμένες τεχνικές κατά το χρόνο εκτέλεσης

Στατική πολλαπλή εκκίνηση

- Ο μεταγλωττιστής ομαδοποιεί τις εντολές σε «πακέτα εκκίνησης» (“issue packets”)
  - Ομάδα εντολών που μπορούν να ξεκινήσουν σε έναν κύκλο
  - Καθορίζεται από τους πόρους της διοχέτευσης που απαιτούνται
- Σκεφθείτε ένα πακέτο εκκίνησης ως μια πολύ μεγάλη εντολή
  - Προσδιορίζει πολλές ταυτόχρονες λειτουργίες
  - $\Rightarrow$ Πολύ μεγάλη λέξη εντολής (Very Long Instruction Word – VLIW)

Χρονοπρογ/σμός στατικής

- Ο μεταγλωττιστής πρέπει να αφαιρέσει κάποιους ή όλους τους κινδύνους
  - Αναδιάταξη εντολών σε πακέτα εκκίνησης
  - Όχι εξαρτήσεις μέσα σε ένα πακέτο
  - Πιθανόν κάποιες εξαρτήσεις μεταξύ πακέτων
    - Διαφορές μεταξύ αρχιτεκτονικών – ο μεταγλωττιστής πρέπει να το γνωρίζει!
  - Συμπλήρωση με εντολές nop αν χρειάζεται

Μια εύκολη λύση: {1 int + 1 fp instruction}

- Επιτρέπεται: 1 anything + 1 FP
  - I-Fetch 64-bits/cycle: {Int αριστερά, FP δεξιά}
  - Εκδίδει την δεύτερη εντολή μετά την έκδοση της πρώτης
  - >2 πόρτες στην FP RF λόγω {FP load, FP op}
  - Load delay ενός κύκλου => 3 εντολές!!!

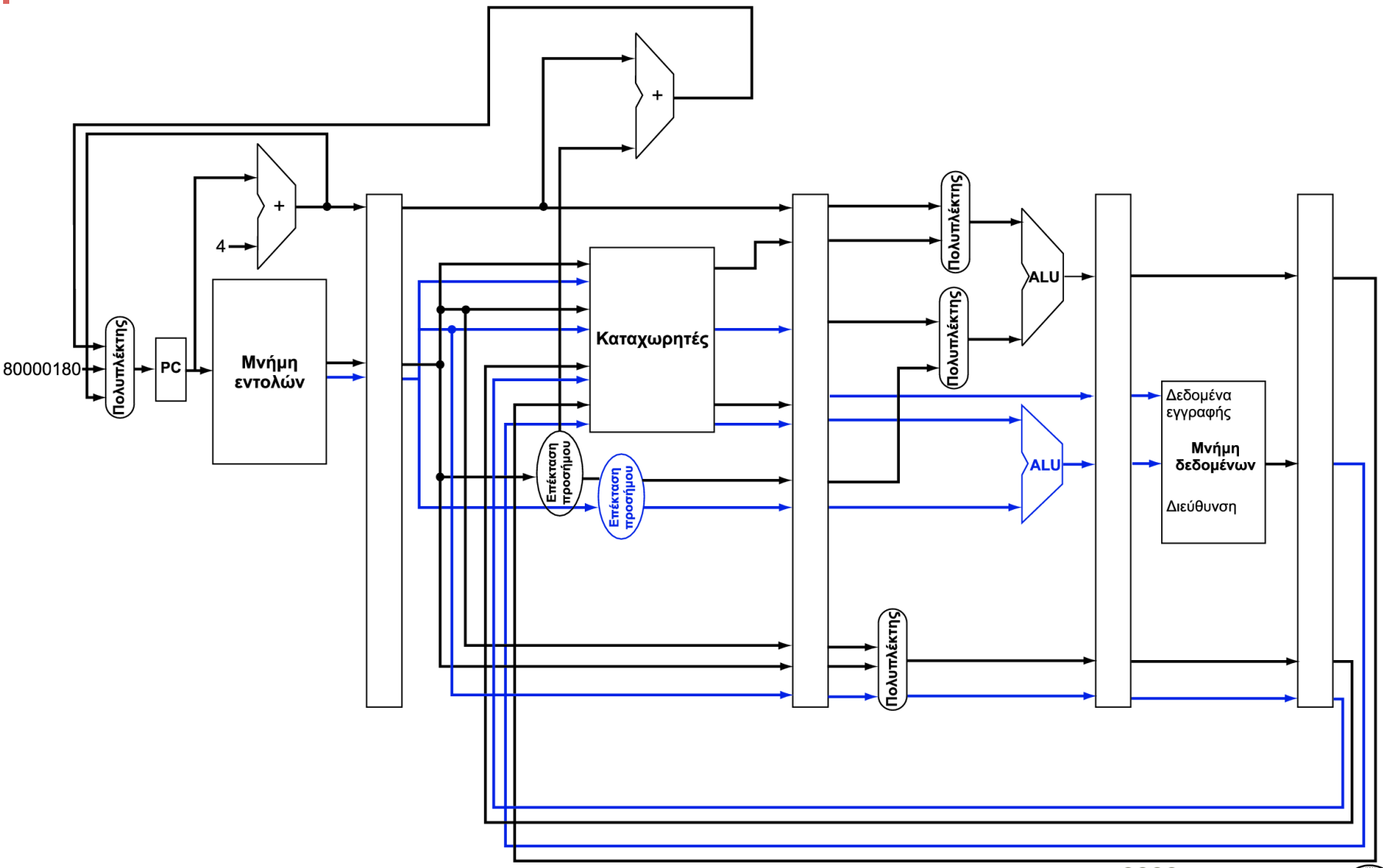
Type	*PipeStages*				
Int. instruction	IF	ID	EX	MEM	WB
FP instruction	IF	ID	EX	MEM	WB
Int. instruction		IF	ID	EX	MEM WB
FP instruction		IF	ID	EX	MEM WB
Int. instruction			IF	ID	EX MEM WB
FP instruction			IF	ID	EX MEM WB

MIPS με στατική διπλή εκκίνηση

- Πακέτα διπλής εκκίνησης
  - Μια εντολή ALU ή branch
  - Μια εντολή load ή store
  - Ευθυγραμμισμένες στα 64 bit
    - ALU/branch, μετά load/store
    - Συμπλήρωση μιας αχρησιμοποίητης εντολής με nop

Διεύθυνση	Τύπος εντολής	Στάδια διοχέτευσης						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

MIPS με στατική διπλή εκκίνηση



Κίνδυνοι στο MIPS με διπλή εκκίνηση

- Περισσότερες εντολές εκτελούνται παράλληλα
- Κίνδυνος δεδομένων EX
  - Η προώθηση απέφυγε τις καθυστερήσεις στην απλή εκκίνηση
  - Τώρα δεν μπορούμε να χρησιμοποιήσουμε το αποτέλεσμα της ALU σε μια load/store στο ίδιο πακέτο
    - ```
      add  $t0, $s0, $s1
      load $s2, 0($t0)
      ```
    - Χωρισμός σε δύο πακέτα, ουσιαστικά μία καθυστέρηση (stall)
- Κίνδυνος φόρτωσης-χρήσης
  - Και πάλι ένας κύκλος λανθάνοντος χρόνου χρήσης, αλλά τώρα δύο εντολές
- Ανάγκη πιο επιθετικού χρονοπρογραμματισμού
      ```

Παράδειγμα χρονοπρ/σμού

- Κώδικας: `while (p) {*p+=x; p--}`
- Χρονοπρογραμματισμός σε MIPS διπλής εκκίνησης:

```
Loop: lw   $t0, 0($s1)       # $t0=array element
      addu $t0, $t0, $s2     # add scalar in $s2
      sw   $t0, 0($s1)       # store result
      addi $s1, $s1,–4       # decrement pointer
      bne  $s1, $zero, Loop  # branch $s1!=0
```

	ALU/branch	Load/store	κύκλος
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1,–4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- IPC = 5/4 = 1.25 (σύγκριση με μέγιστο IPC = 2)

Ξετύλιγμα βρόχου

- Loop unrolling
- Επανάληψη του σώματος του βρόχου για να εμφανιστεί μεγαλύτερη παραλληλία
  - Μειώνει την επιβάρυνση ελέγχου του βρόχου
- Χρήση διαφορετικών καταχωρητών ανά επανάληψη
  - Ονομάζεται «μετονομασία καταχωρητών (“register renaming”)
  - Αποφυγή «αντεξαρτήσεων» (“anti-dependencies”) που μεταφέρονται στο βρόχο (loop-carried)
    - Store που ακολουθείται από load στον ίδιο καταχωρητή
    - Ονομάζεται επίσης και «εξάρτηση ονόματος» (“name dependence”)
      - Επαναχρησιμοποίηση ενός ονόματος καταχωρητή

Παράδειγμα ξετυλίγματος βρόχου

- Κώδικας «ξετυλιγμένος» 4 φορές (θεωρούμε ότι #στοιχείων είναι πολλαπλάσιο του 4):

```
While (p) {
   *p+=x;          // t0=*p; t0+=x; *p=t0;
   *(p+1) += x; // t1=*p; t1+=x; *p=t1;
   *(p+2) += x; // t2=*p; t2+=x; *p=t2;
   *(p+3) += x; // t3=*p; t3+=x; *p=t3;
   p = p-4; // μείωσε τον δείκτη κατά 4
}
```

Παράδειγμα ξετυλίγματος βρόχου

	ALU/branch	Load/store	Κύκλος
Loop:	addi $s1, $s1,-16	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t4, $s2	sw $t1, 12($s1)	6
	nop	sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

- IPC = 14/8 = 1.75
  - Πλησιέστερο στο 2, αλλά με κόστος σε καταχωρητές και μέγεθος κώδικα

Εικασία (speculation)

- «Μαντεψιά» – τι να κάνουμε με μια εντολή
  - Εκκίνηση λειτουργίας το συντομότερο
  - Έλεγχος αν η εικασία ήταν σωστή
    - Αν ναι, ολοκλήρωση της λειτουργίας
    - Αν όχι, επιστροφή (roll-back) και εκτέλεση του σωστού
- Συνηθισμένη στη στατική και τη δυναμική πολλαπλή εκκίνηση
- Παραδείγματα
  - Εικασία στο αποτέλεσμα διακλάδωσης
    - Επιστροφή αν η ληφθείσα διαδρομή είναι διαφορετική
  - Εικασία σε φόρτωση (load)
    - Επιστροφή αν η θέση μνήμης έχει αλλάξει τιμή

Εικασία μεταγλωττιστή/υλικού

- Ο μεταγλωττιστής μπορεί να αναδιατάξει τις εντολές
  - π.χ., μετακίνηση μιας φόρτωσης πριν από μια διακλάδωση
  - Μπορεί να περιλάβει εντολές «διόρθωσης» για να ανακάμψει από λάθος εικασίες
- Το υλικό μπορεί να δει πιο πέρα για εντολές προς εκτέλεση
  - Αποθηκεύει προσωρινά τα αποτελέσματα μέχρι να προσδιορίσει ότι χρειάζονται πραγματικά
  - Εκκενώνει τις προσωρινές μνήμες αν γίνει λάθος εικασία

Εικασία και εξαιρέσεις

- Τι γίνεται αν συμβεί εξαίρεση σε μια εντολή που εκτελείται με εικασία;
  - π.χ., φόρτωση με εικασία πριν τον έλεγχο κενού δείκτη (null-pointer check)
- Στατική εικασία
  - Μπορεί να προσθέσει υποστήριξη από την αρχιτεκτονική συνόλου εντολών για εξαιρέσεις που έχουν μετατεθεί (deferring exceptions)
- Δυναμική εικασία
  - Μπορεί να αποθηκεύει προσωρινά τις εξαιρέσεις μέχρι την ολοκλήρωση της εντολής (που μπορεί να μη συμβεί)

Δυναμική πολλαπλή εκκίνηση

- Υπερβαθμωτοί (“superscalar”) επεξεργαστές
- Η CPU αποφασίζει αν θα ξεκινήσουν 0, 1, 2, … εντολές σε κάθε κύκλο
  - Αποφυγή κινδύνων δομής και δεδομένων
- Αποφεύγει την ανάγκη για χρονοπρογραμματισμό από το μεταγλωττιστή
  - Παρόλο που αυτός μπορεί ακόμη να βοηθήσει
  - Η CPU εγγυάται τη σημασιολογία του κώδικα

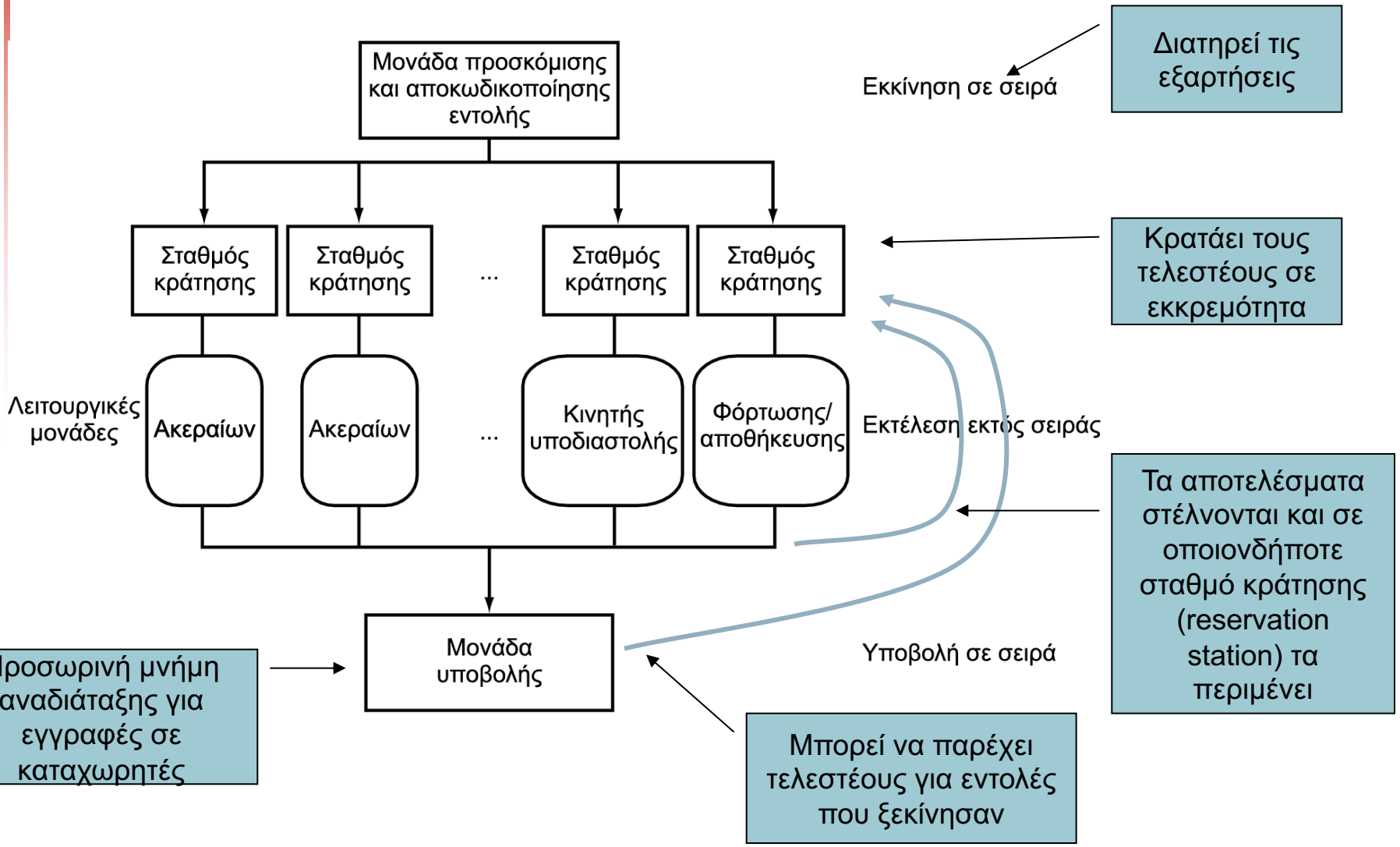
Χρονοπρογ/σμός δυναμικής

- Η CPU μπορεί να εκτελεί εντολές εκτός σειράς (out of order) για αποφυγή καθυστερήσεων
  - Αλλά δέσμευση (commit) του αποτελέσματος σε καταχωρητές σε σειρά
- Παράδειγμα

```
lw    $t0, 20($s2)
addu  $t1, $t0, $t2
sub   $s4, $s4, $t3
slti  $t5, $s4, 20
```

  - Μπορεί να ξεκινήσει η sub ενώ η addu περιμένει τη lw

CPU με δυναμικό χρον/σμό



Μετονομασία καταχωρητών

- Οι σταθμοί κράτησης (reservation stations) και η προσωρινή μνήμη αναδιάταξης (reorder buffer) στην πράξη παρέχουν μετονομασία καταχωρητών
- Κατά την εκκίνηση εντολής σε σταθμό κράτησης
  - Αν ο τελεστέος είναι διαθέσιμος στο αρχείο καταχωρητών ή την προσωρινή μνήμη αναδιάταξης
    - Αντιγράφεται στο σταθμό κράτησης
    - Δε χρειάζεται άλλο στον καταχωρητή, και μπορεί να αντικατασταθεί
  - Αν ο τελεστέος δεν είναι ακόμα διαθέσιμος
    - Θα παρασχεθεί στο σταθμό κράτησης από μια λειτουργική μονάδα
    - Μπορεί να μη χρειαστεί ενημέρωση καταχωρητή

Εικασία (speculation)

- Πρόβλεψη διακλάδωσης και συνέχιση της εκκίνησης
  - Δε γίνεται δέσμευση (commit) μέχρι να καθοριστεί το αποτέλεσμα της διακλάδωσης
- Εικασία φόρτωσης (load speculation)
  - Αποφυγή της καθυστέρησης της φόρτωσης και της αστοχίας κρυφής μνήμης (cache miss)
    - Πρόβλεψη της πραγματικής διεύθυνσης
    - Πρόβλεψη της τιμής φόρτωσης
    - Φόρτωση πριν την ολοκλήρωση εκκρεμουσών αποθηκεύσεων
    - Παράκαμψη/προώθηση τιμών αποθήκευσης προς τη μονάδα φόρτωσης
  - Δε γίνεται δέσμευση (commit) της φόρτωσης μέχρι να ξεκαθαριστεί η εικασία

Γιατί δυναμικός χρονοπ/σμός;

- Γιατί να μην αφήσουμε απλώς το μεταγλωττιστή να χρονοπρογραμματίσει τον κώδικα;
- Δεν είναι όλες οι καθυστερήσεις προβλέψιμες
  - π.χ., αστοχίες κρυφής μνήμης (cache misses)
- Δεν μπορεί να γίνει πάντα χρονοπρ/σμός γύρω από διακλαδώσεις
  - Το αποτέλεσμα της διακλάδωσης καθορίζεται δυναμικά
- Διαφορετικές υλοποιήσεις μιας αρχιτεκτονικής έχουν διαφορετικούς λανθάνοντες χρόνους και κινδύνους

Δουλεύει η πολλαπλή εκκίνηση;

ΓΕΝΙΚΗ εικόνα

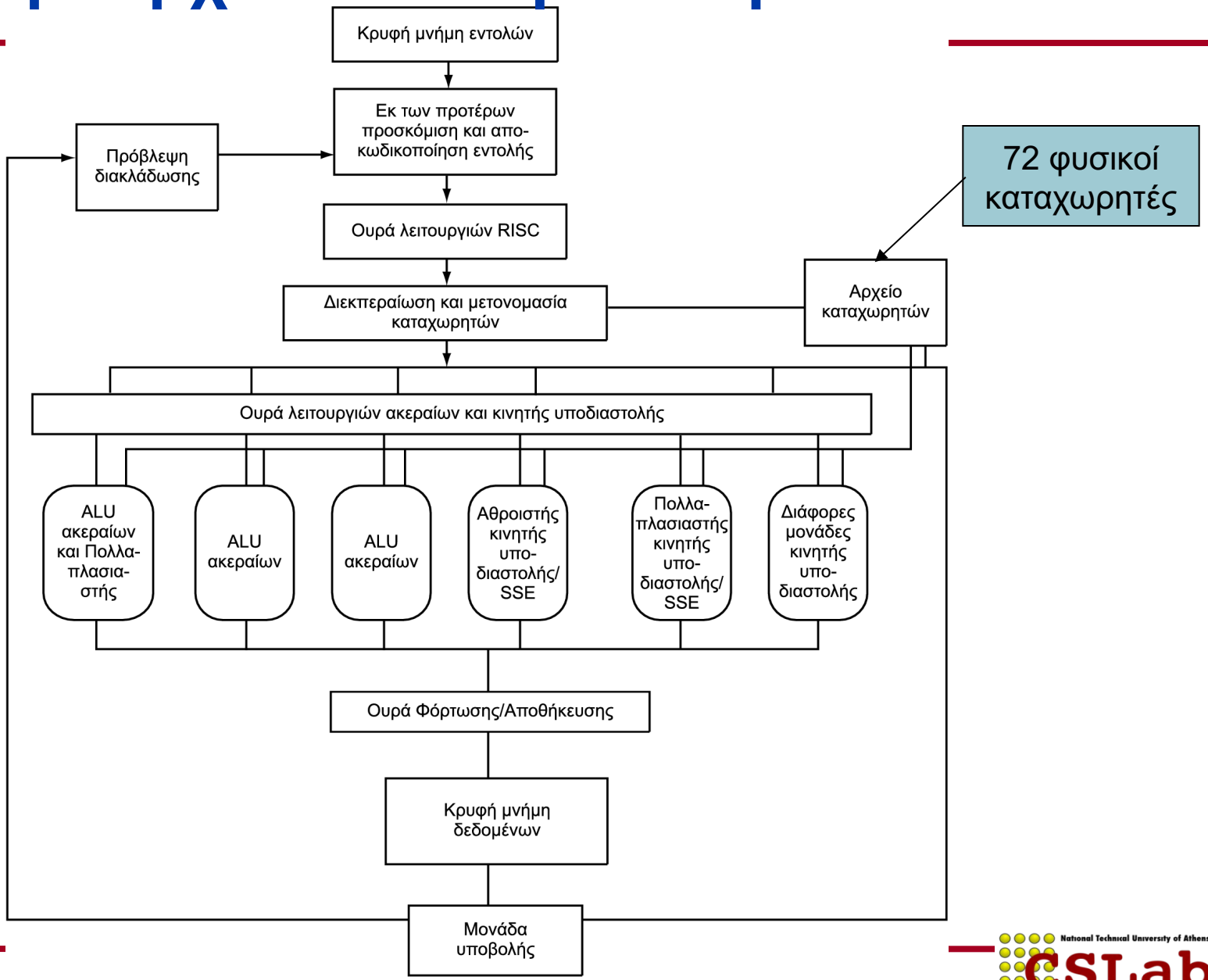
- Ναι, αλλά όχι όσο θα θέλαμε
- Τα προγράμματα έχουν πραγματικές εξαρτήσεις που περιορίζουν το ILP
- Μερικές εξαρτήσεις είναι δύσκολο να εξαλειφθούν
  - π.χ., ψευδωνυμία δείκτη (pointer aliasing)
- Κάποια παραλληλία είναι δύσκολο να εμφανιστεί
  - Περιορισμένο μέγεθος παραθύρου κατά την εκκίνηση εντολών
- Καθυστερήσεις μνήμης και περιορισμένο εύρος ζώνης
  - Δύσκολο να διατηρηθούν γεμάτες οι διοχετεύσεις
- Η εικασία μπορεί να βοηθήσει αν γίνει καλά

Αποδοτικότητα ισχύος

- Η πολυπλοκότητα του δυναμικού χρονοπρογραμματισμού και της εικασίας απαιτεί ηλεκτρική ισχύ
- Οι πολλοί απλούστεροι πυρήνες μπορεί να είναι καλύτεροι

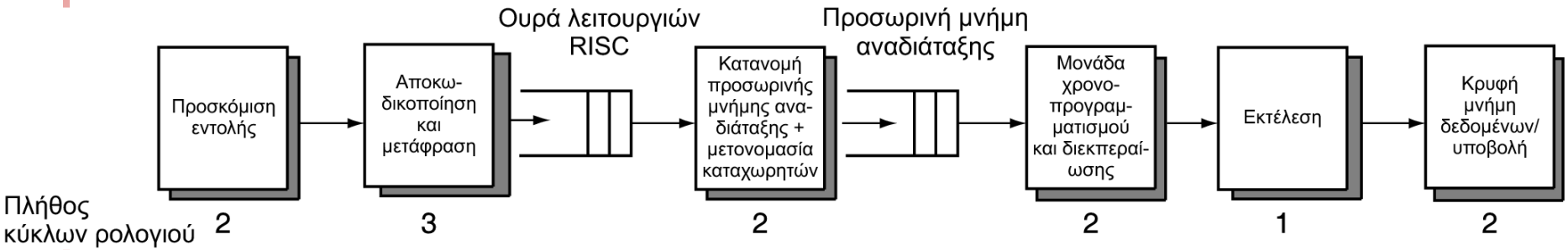
Μικροεπεξεργαστής	Έτος	Ρυθμός ρολογιού	Στάδια διοχέτευσης	Πλάτος εκκίνησης	Εκτός-σειράς/ Εικασία	Πυρήνες	Ισχύς
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

Μικροαρχιτεκτονική του Opteron X4



Διοχέτευση του Opteron X4

- Για ακέραιες λειτουργίες



  - Για κινητή υποδιαστολή είναι 5 στάδια μεγαλύτερη
  - Μέχρι 106 λειτουργίες RISC σε εξέλιξη
- Σημεία συμφόρησης
  - Πολύπλοκες εντολές με μεγάλες εξαρτήσεις
  - Λάθος προβλέψεις διακλάδωσης
  - Καθυστερήσεις προσπέλασης μνήμης

Πλάνες

- Η διοχέτευση είναι εύκολη (!)
  - Η βασική ιδέα είναι εύκολη
  - Ο διάβολος κρύβεται στις λεπτομέρειες
    - π.χ., ανίχνευση κινδύνων δεδομένων
- Η διοχέτευση είναι ανεξάρτητη από την τεχνολογία
  - Τότε γιατί δεν κάναμε πάντα διοχέτευση;
  - Τα περισσότερα τρανζίστορ κάνουν εφικτές τις πιο προηγμένες τεχνικές
  - Η σχεδίαση αρχιτεκτονικών συνόλου εντολών που σχετίζεται με τη διοχέτευση πρέπει να λαμβάνει υπόψη της τις τεχνολογικές τάσεις
    - π.χ., εντολές με κατηγορήματα

Παγίδες

- Η φτωχή σχεδίαση της αρχιτεκτονικής συνόλου εντολών μπορεί να κάνει δυσκολότερη τη διοχέτευση
  - π.χ., πολύπλοκα σύνολα εντολών (VAX, IA-32)
    - Σημαντική επιβάρυνση για να δουλέψει η διοχέτευση
    - Προσέγγιση του IA-32 με μικρολειτουργίες (micro-ops)
  - π.χ., πολύπλοκοι τρόποι διευθυνσιοδότησης
    - Παρενέργειες ενημέρωσης καταχωρητών, εμμεσότητα μνήμης
  - π.χ., καθυστερημένες διακλαδώσεις
    - Οι προηγμένες διοχετεύσεις έχουν μεγάλες υποδοχές καθυστέρησης

Συμπερασματικές παρατηρήσεις

- Η αρχιτεκτονική συνόλου εντολών επηρεάζει τη σχεδίαση της διαδρομής δεδομένων και της μονάδας ελέγχου
- Η διαδρομή δεδομένων και η μονάδα ελέγχου επηρεάζουν τη σχεδίαση της αρχιτεκτονικής συνόλου εντολών
- Η διοχέτευση βελτιώνει τη διεκπεραιωτική ικανότητα εντολών με τη χρήση παραλληλίας
  - Περισσότερες εντολές ολοκληρώνονται ανά δευτερόλεπτο
  - Ο λανθάνων χρόνος κάθε εντολής δε μειώνεται
- Κίνδυνοι: δομής, δεδομένων, ελέγχου
- Πολλαπλή εκκίνηση και δυναμικός χρονοπρογραμματισμός (παραλληλία επιπέδου εντολής – ILP)
  - Οι εξαρτήσεις περιορίζουν την επιτεύξιμη παραλληλία
  - Η πολυπλοκότητα οδηγεί στο τείχος της ισχύος (power wall)