Προηγμένες αρχιτεκτονικές υπολογιστών και προγραμματισμός παραλλήλων συστημάτων

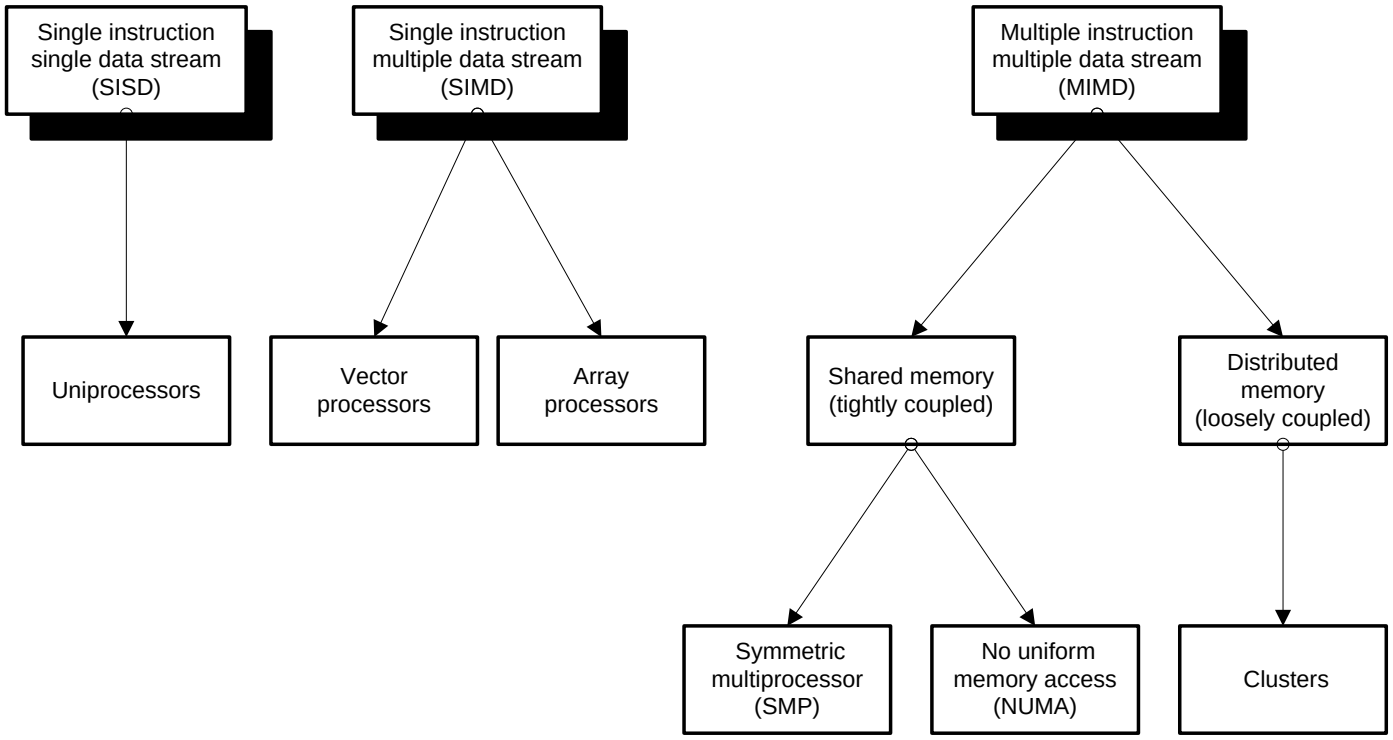
04. Τεχνολογίες Επεξεργαστών

Περιεχόμενα

- Επιλογές στη σχεδίαση επεξεργαστών.
- Επιλογές στη σχεδίαση του συνόλου των εντολών:
  - Αρχιτεκτονική εσωτερικής μνήμης
  - Κλήσεις διευθύνσεων μνήμης (memory addressing).
  - Τα είδη των εντολών.
- Κατηγορίες Επεξεργαστών:
  - Επεξεργαστές CISC
  - Επεξεργαστές RISC
  - Επεξεργαστές τύπου Superscalar
  - Η αρχιτεκτονική VLIW
  - Οι επεξεργαστές τύπου Vector

Στο παρελθόν…

- Επεξεργαστές τύπου scalar: Η αύξηση ταχύτητας έχει σχέση με το πλήθος των μονάδων και τις τεχνικές διασύνδεσης τους



Ορισμοί

- **Σφιχτά διασυνδεδεμένοι κόμβοι** (tightly coupled): Κόμβοι που επικοινωνούν με ταχύ δίκτυο και ανταλλάσσουν πολύ μεγάλο όγκο δεδομένων ανά δευτερόλεπτο.
- **Χαλαρά διασυνδεδεμένοι κόμβοι** (loosely coupled): Κόμβοι που επικοινωνούν με όχι αναγκαστικά ταχύ δίκτυο και ανταλλάσσουν σχετικά μικρότερο όγκο δεδομένων.
- **Μηχανή μη ομοιόμορφης προσπέλασης στη μνήμη** (Non-Uniform Memory Access – NUMA): παράλληλη μηχανή με κοινή κατανεμημένη μνήμη. Ο χρόνος προσπέλασης της μνήμης είναι μικρός αν αφορά στην τοπική μνήμη και μεγάλος αν αφορά απομακρυσμένη μνήμη.
- **Μηχανή ομοιόμορφης προσπέλασης στη μνήμη (UMA)** = Συμμετρικός πολυεπεξεργαστής (SMP). Ο χρόνος προσπέλασης είναι ίδιος για όλες τις κλήσεις στη μνήμη.

Επιλογές σχεδιασμού

- Επανασχεδιασμός της αρχιτεκτονικής του συνόλου των εντολών (instruction set).
- Αλλαγή τεχνολογίας με την οποία παράγονται τα σήματα ελέγχου (υλικό αντί από προγράμματα).
- Ελαχιστοποίηση των μεθόδων υπολογισμού της τελικής διεύθυνσης.
- Αύξηση του αριθμού των καταχωρητών.
- Προσπάθεια εκτέλεσης περισσότερων εντολών σε ένα κύκλο:
  - Pipelining.
  - Παραλληλισμός: Πολλαπλοί πυρήνες.

Μοντέρνες τεχνικές

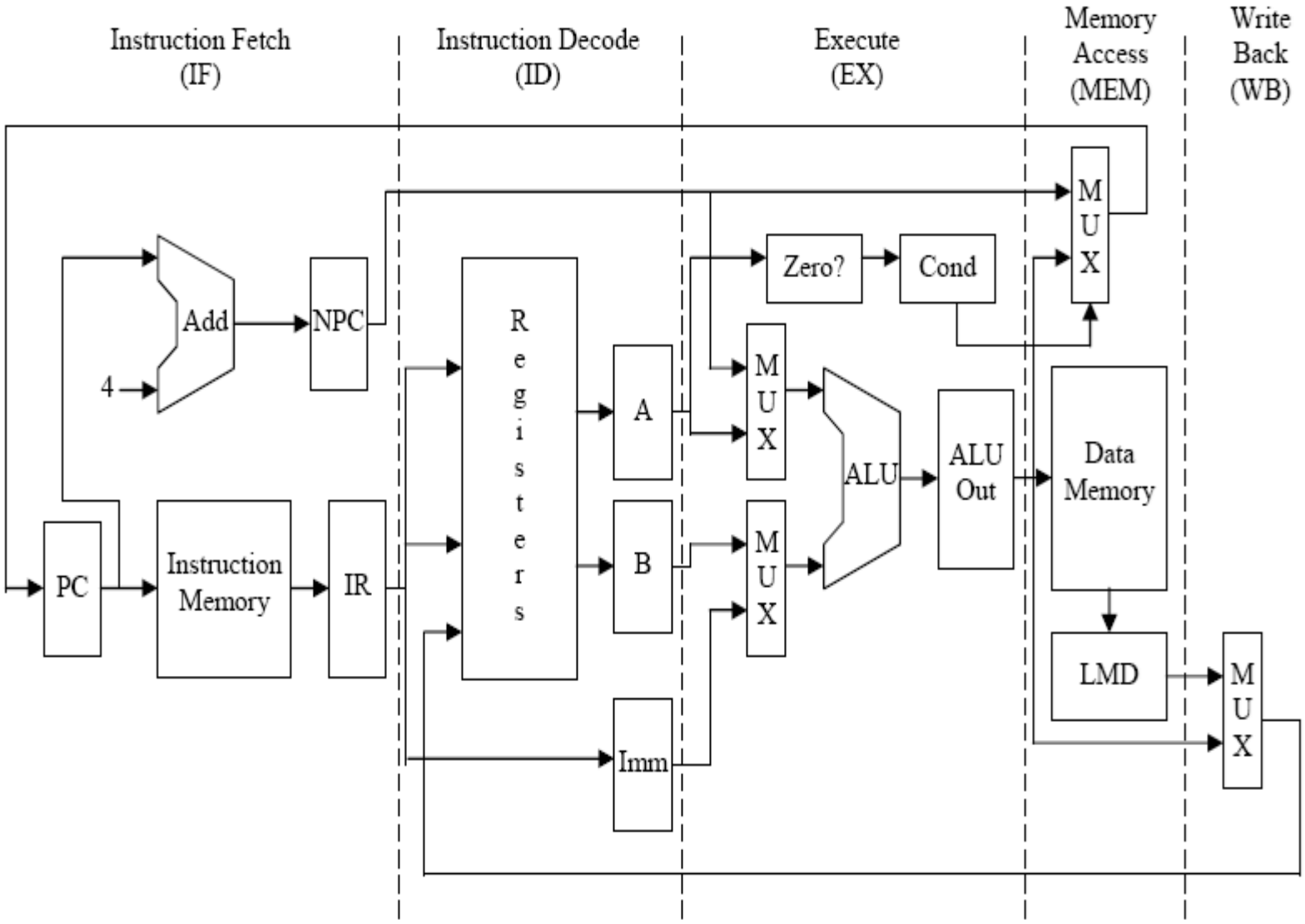
- Pipeline.
- Πρόβλεψη αλμάτων υπό συνθήκη (branch prediction).
  - Άλματα υπό συνθήκη = if-then-else. Στη γλώσσα μηχανής καλούνται conditional branches.
- Ανάλυση ροής δεδομένων (data flow analysis).
- Προληπτική εκτέλεση (speculative execution).
- Τεχνική Superscalar.
- Τεχνική (Very long Instruction Word – VLIW).

Pipeline

	Κύκλοι μηχανής								
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Εντολή 1	IF	ID	EX	MEM	WB				
Εντολή 2		IF	ID	EX	MEM	WB			
Εντολή 3			IF	ID	EX	MEM	WB		
Εντολή 4				IF	ID	EX	MEM	WB	
Εντολή 5					IF	ID	EX	MEM	WB

1. **IF** (Instruction Fetch): Φάση προσκόμισης εντολής από μνήμη.
2. **ID** (Instruction decode/register fetch): Φάση αποκωδικοποίησης / ανάγνωσης καταχωρητών.
3. **EX** (Execution) = Φάση εκτέλεσης.
4. **MEM** (Memory access): Φάση πρόσβασης στη μνήμη και ολοκλήρωση των αλμάτων.
5. **WB** (Write back): Φάση εγγραφής καταχωρητών / μνήμης.

Ένας εκπαιδευτικός επεξεργαστής: DLX



Πρόβλεψη άλματος (Branch prediction)

- Χρήσιμο σε άλματα υπό συνθήκη (if-then-else).
- Απόφαση αν το άλμα θα γίνει ή όχι μετά την φάση EXE της εντολής άλματος υπό συνθήκη.
- Η μη-πρόβλεψη (no branch prediction) ισοδυναμεί με την πρόβλεψη ότι το άλμα δεν θα γίνει, αφού κάνουμε προσκόμιση (φάση IF) της επόμενης εντολής.
- Η όλη αξία έγκειται στην ακρίβεια της πρόβλεψής μας.
- Ο branch predictor κρατάει στατιστικά για το συγκεκριμένο branch.
- Πολλά if είναι πολύ πιο συχνά true απ' ότι false. Πχ. στο τέλος ενός βρόχου Ν επαναλήψεων υπάρχει ένα άλμα υπό συνθήκη που εκτελείται Ν-1 φορές και μόνο 1 φορά δεν εκτελείται.

Χωρίς πρόβλεψη άλματος (No Branch prediction)

	Κύκλοι μηχανής								
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Εντολή 1	IF	ID	EXE	MEM	WB				
CONDITIONAL BRANCH → Εντολή N		IF	ID	EX	MEM	WB			
Εντολή 3			IF	ID	EX	MEM	WB		
. . . . .									
Εντολή N					IF	ID	EX	MEM	WB
Εντολή N+1									

Αν συνθήκη = false, όχι jump. Εκτελείται η εντολή 3 κανονικά

Αν συνθήκη = true, τότε jump

Αυτοί οι δύο κύκλοι χαμένοι

- Αν η συνθήκη είναι true υπάρχει απώλεια δύο κύκλων, εκτός αν το σύστημα προβλέψει και αποκτήσει την εντολή Ν στο κύκλο 3

Με πρόβλεψη άλματος (Branch prediction)

	Κύκλοι μηχανής								
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Εντολή 1	IF	ID	EXE	MEM	WB				
CONDITIONAL BRANCH → Εντολή N		IF	ID	EX	MEM	WB			
Εντολή 3			Η εντολή 3 δεν εκτελείται						
. . . . .									
Εντολή N			IF	ID	EX	MEM	WB		
Εντολή N+1									

Προβλέπουμε ότι συνθήκη = true οπότε εκτελούμε την εντολή *N*

- Αν προβλέψουμε σωστά δεν χάνεται κάνενας κύκλος. Αν προβλέψουμε λάθος χάνονται 2 κύκλοι όπως πριν.

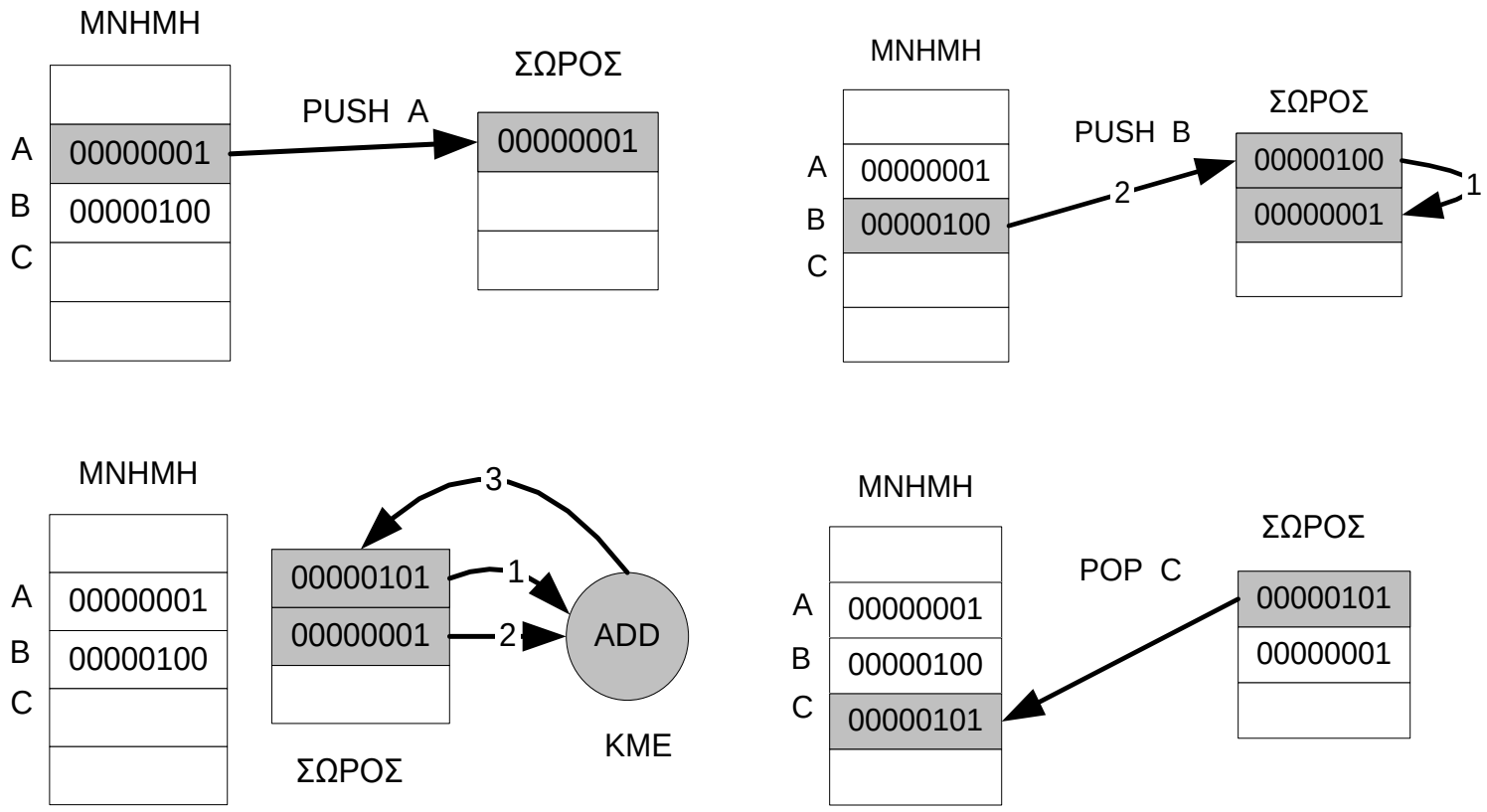
Επιλογές στη σχεδίαση των εντολών

- Μέθοδος χρήσης τις εσωτερικής μνήμης του επεξεργαστή (registers, stacks).
- Μέθοδος κλήσης μιας διεύθυνσης μνήμης (memory addressing).
- Τα είδη των εντολών.

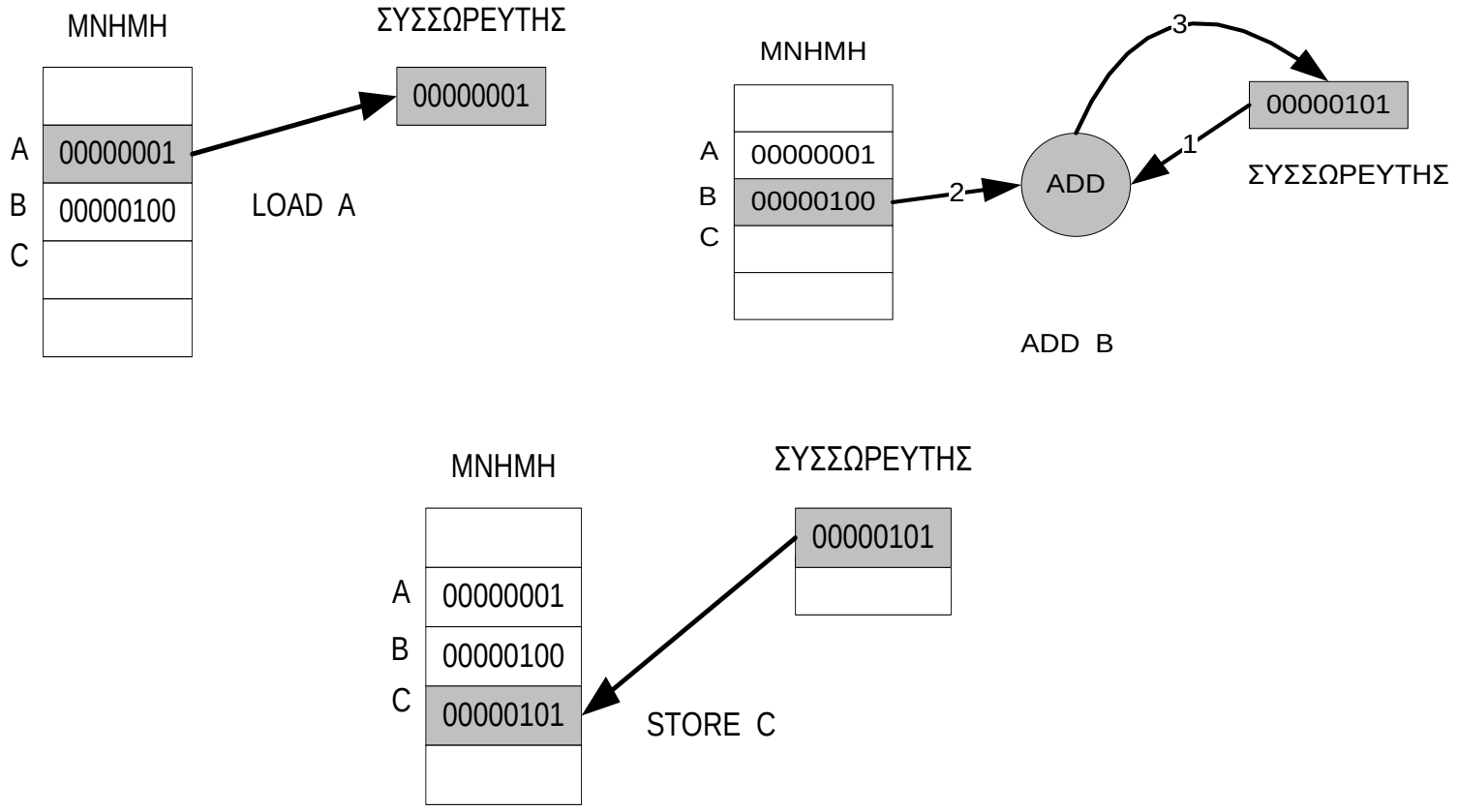
Αρχιτεκτονική εσωτερικής μνήμης

Σωρός	**Ένας Καταχωρητής (Συσσωρευτής)**	**Καταχωρητές (register-memory)**	**Καταχωρητές (register-register)**
PUSH A	LOAD A	LOAD R1, A	LOAD R1, A
PUSH B	ADD B	ADD R1, B	LOAD R2, B
ADD	STORE C	STORE C, R1	ADD R3, R2, R1
POP C			STORE C, R3

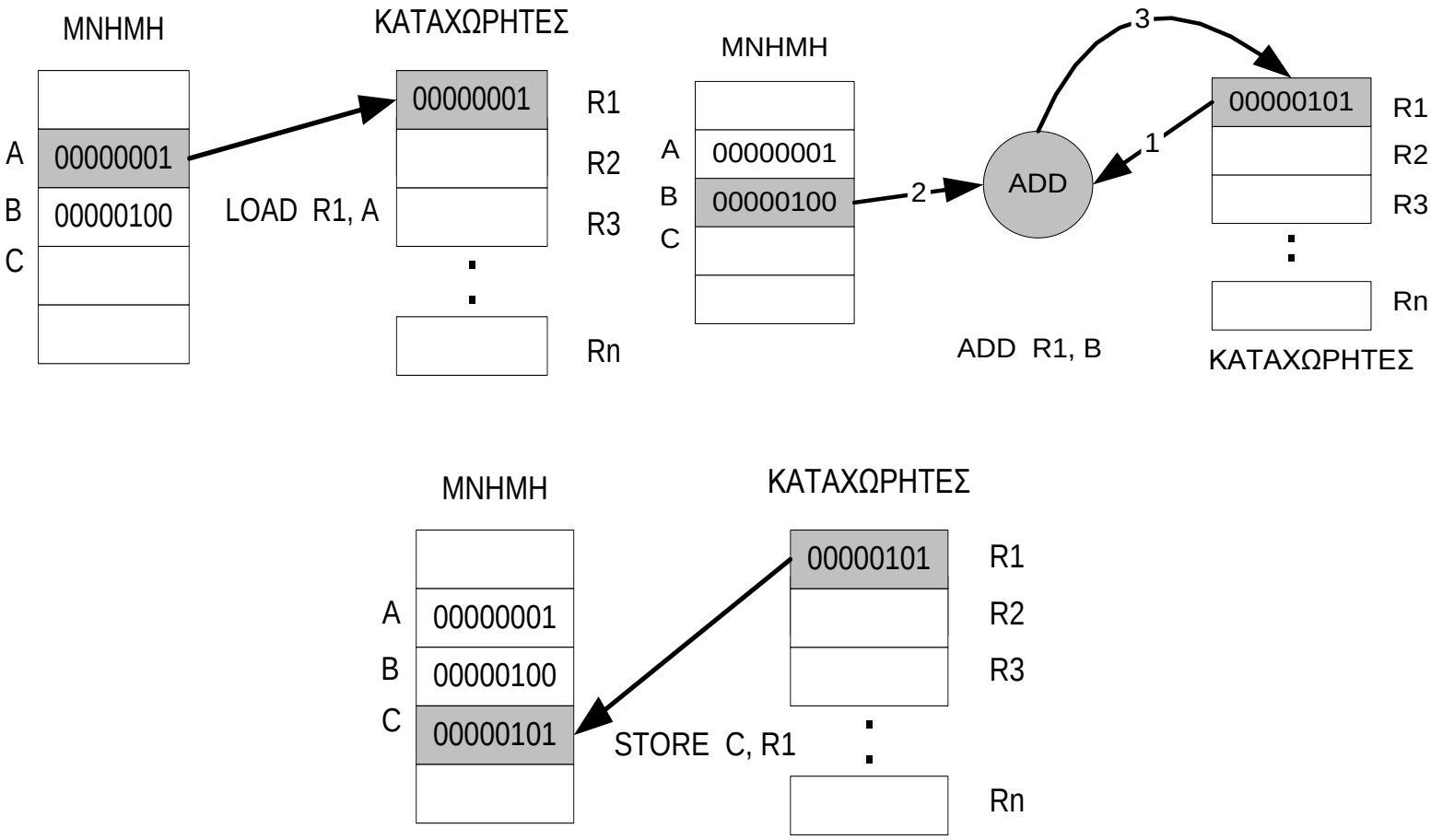
Αρχιτεκτονική σωρού



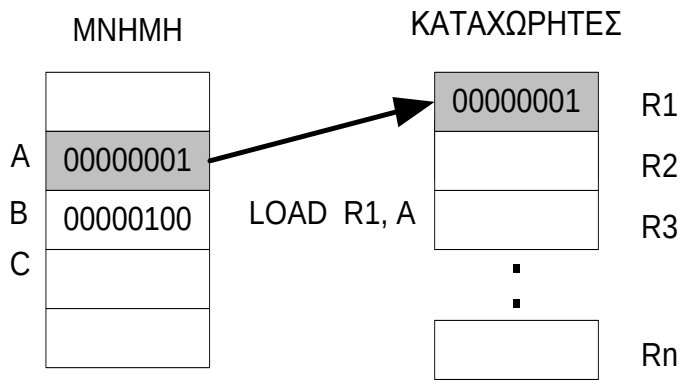
Αρχιτεκτονική συσσωρευτή

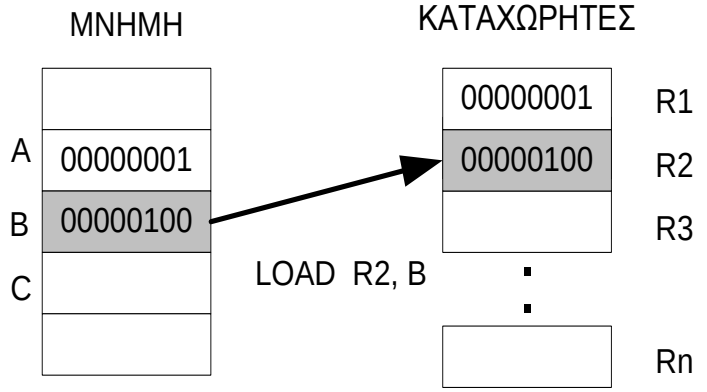


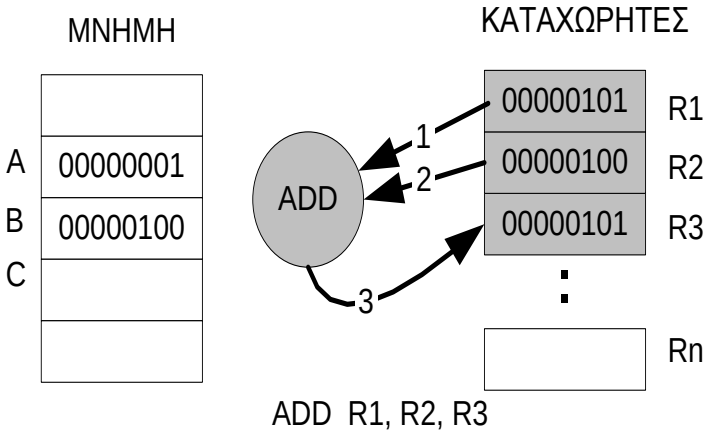
Αρχιτεκτονική register – memory

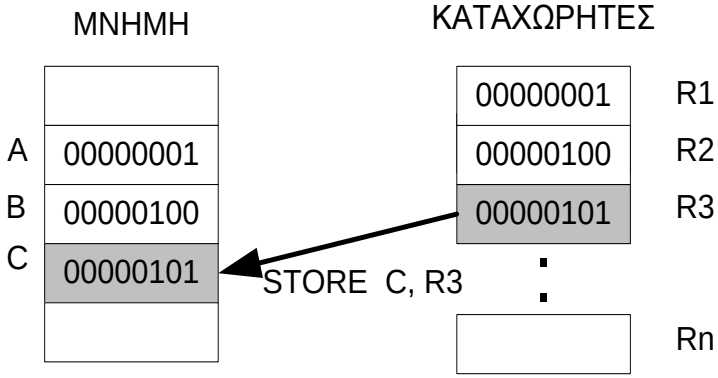


Αρχιτεκτονική load – store









Γιατί θέλουμε πολλούς καταχωρητές ;

- Είναι ταχύτεροι από τη μνήμη.
- Οι μεταφραστές μπορούν να τους χειριστούν καλυτέρα από τη μνήμη.
- Φύλαξη ενδιαμέσων αποτελεσμάτων.

Αναφορά στη μνήμη και πλήθος τελεστών

Αναφορά στη μνήμη	Μέγιστο πλήθος τελεστών	Τύπος	Πλήθος κύκλων/εντολή	Τεχνική Pipeline
0	3	Load-store	Ο ίδιος (εκτός από τις Load, store)	Εύκολη
1	2	Register-memory	Διαφορετικοί κύκλοι/εντολή χωρίς μεγάλες διαφορές	Δύσκολη
2 ή 3	2 ή 3	Memory-memory	Διαφορετικοί κύκλοι σχεδόν σε κάθε διαφορετική εντολή/εντολή	Καθόλου

Τελεστές ανά εντολή

Αριθμός κλήσεων μνήμης ανά εντολή	Μέγιστος αριθμός τελεστών ανά εντολή	Παραδείγματα επεξεργαστών
0	3	SPARC, MIPS, PowerPC, DEC-Alpha
1	2	Intel 80x86, Motorola 68000
2	2	VAX
3	3	VAX

Κλήσεις διευθύνσεων μνήμης (memory addressing)

Μέθοδος	**Παράδειγμα**	**Λειτουργία**	**Χρήση**
Register **Με καταχωρητή**	`Add R4,R3`	`R4 ← R4 + R3`	Η τιμή βρίσκεται σε καταχωρητή
Immediate **Άμεση**	`Add R4,#90`	`R4 ← R4 + 90`	Η τιμή είναι μια σταθερά
Displacement **Με μετατόπιση**	`Add R4,10(R1)`	`R4 ← R4 + M[10+R1]`	Η τιμή βρίσκεται στη θέση μνήμης με διεύθυνση 10(R1)
Register indirect **Έμμεση μέσω καταχωρητή**	`Add R4,(R1)`	`R4 ← R4 + M[R1]`	Η τιμή βρίσκεται στη θέση μνήμης με διεύθυνση R1
Indexed **Με δείκτη**	`Add R4,(R1+ R2)`	`R4 ← R4+ M[R1+R2]`	Η τιμή βρίσκεται μέσα σε μια θέση ενός πίνακα με αρχική διεύθυνση R1 και δείκτη R2

Κλήσεις διευθύνσεων μνήμης (memory addressing)

Direct ***Απόλυτη***	`Add R1,(1001)`	`R1←R1 + M[1001]`	Χρήσιμο για στατικές θέσεις μνήμης
Memory indirect ***Έμμεση μέσω μνήμης***	`Add R1,@(R3)`	`R1←R1 + M[M[R3]]`	Η τιμή βρίσκεται σε θέση μνήμης με διεύθυνση που βρίσκεται μέσα σε μια άλλη θέση μνήμης με διεύθυνση R3
Auto-increment ***Με αυτο-αύξηση***	`Add R1,(R2)+`	`R1 ← R1 + M[R2]` `R2 ← R2 + d`	Χρήσιμο για τη σάρωση πινάκων από την κορυφή προς το τέλος. R2: η αρχή του πίνακα *d*: η τιμή της αύξησης
Auto-decrement ***Με αυτο-μείωση***	`Add R1,-(R2)`	`R2 ← R2 – d` `R1 ← R1 + M[R2]`	Εκτελεί την αντιστροφή εργασία από την προηγούμενη μέθοδο
Scaled ***Κλιμακούμενη***	`Add R1, 100(R2)[R3]`	`R1 ← R1 + M[100*R2 + R3*d]`	Χρήσιμο για προσπέλαση 2D πινάκων. R2: γραμμή, R3: αρχή πίνακα, 100: μήκος γραμμής

Συχνότητα χρησιμοποίησης των τρόπων κλήσης

	Προγράμματα		
	TeX (επεξεργασία κειμένου)	**Spice** (εξομοιωτής ηλεκτρονικών κυκλωμάτων)	**Gcc** (μεταφραστής C)
Memory indirect	1%	6%	1%
Scaled	0%	16%	6%
Register indirect	24%	3%	11%
Immediate	43%	17%	39%
Displacement	32%	55%	40%

Κατηγορίες εντολών

Κατηγορία εντολής	**Παράδειγμα**
Αριθμητική / λογική	add, subtract, and, or, κλπ
Μεταφοράς δεδομένων	load, store, move, κλπ
Ελέγχου	branch, jump, return, call, trap
Συστήματος	κλήση λειτουργικού συστήματος, εικονική μνήμη, κλπ
Επεξεργασία Συμβολοσειρών	string move, string compare, string search, κλπ
Γραφικά	pixel operations

Συχνότητα χρήσης εντολών

Εντολή 80x86	**Ποσοστό επί συνόλου εντολών**
load	22%
conditional branch	20%
compare	16%
store	12%
add	8%
and	6%
sub	5%
move register-register	4%
call	1%
return	1%
Σύνολο	96%

Συχνότητες εντολών αλλαγής ροής

Άλματα χωρίς συνθήκη (jumps)	5%
Άλματα υπό συνθήκη (conditional jumps)	85%
Κλήσεις υπορουτινών (procedure calls)	5%
Επιστροφές υπορουτινών (procedure returns)	5%

Κατηγορίες Επεξεργαστών

- Επεξεργαστές CISC.
- Επεξεργαστές RISC.
- Υπερβαθμωτοί επεξεργαστές (τύπου Superscalar).
- Επεξεργαστές VLIW.
- Διανυσματικοί επεξεργαστές (τύπου Vector).

CISC vs. RISC

Αρχιτεκτονικά Χαρακτηριστικά	Υπολογιστές CISC	Υπολογιστές RISC
Μέγεθος συνόλου εντολών και κωδικοποίηση εντολών	Μεγάλο σύνολο εντολών (π.χ. 400 εντολές). Οι εντολές δεν έχουν το ίδιο μήκος το οποίο κυμαίνεται από 16 έως 64 bits.	Μικρό σύνολο εντολών (π.χ. 150 εντολές). Οι εντολές έχουν το ίδιο μήκος και βασίζονται στη χρήση καταχωρητών.
Τρόποι κλήσης μνήμης	12 έως 25.	3 έως 5.
Καταχωρητές γενικού σκοπού και μνήμη cache	8-24 καταχωρητές γενικού σκοπού. Είτε μια cache κοινή για εντολές και δεδομένα είτε δύο ξεχωριστές cache.	Πολλοί καταχωρητές γενικού σκοπού (32-256). Ξεχωριστές cache για δεδομένα και εντολές.
Αριθμός κύκλων μηχανής ανά εντολή	Μεταβλητός από εντολή σε εντολή (από 1 έως 20).	1 κύκλος/εντολή για τις περισσότερες εντολές και Μέσος όρος < 1.5 κύκλος/ εντολή.
Κύκλωμα ελέγχου	Μικροπρογραμματιζόμενος με χρήση ROM. Η μοντέρνα τάση ωστόσο χρησιμοποιεί καλωδιωμένη λογική.	Καλωδιωμένη λογική.

Επεξεργαστές superscalar

- Αύξηση του εύρους της κλασικής pipeline δίνοντας τη δυνατότητα να εκτελούνται περισσότερες από 1 εντολές ταυτόχρονα σε κάθε φάση.
- Απαιτούνται πολλές λειτουργικές μονάδες σε κάθε φάση. Π.χ. σε έναν επεξεργαστή 3πλό superscalar απαιτούνται 3 μονάδες IF, 3 μονάδες ID, 3 μονάδες EX, 3 μονάδες MEM και 3 μονάδες WB.
- Οι επεξεργαστές superscalar διαθέτουν:
  - Μονάδα απόκτησης εντολών με δυνατότητα απόκτησης πολλών εντολών ταυτόχρονα.
  - Μονάδα αποκωδικοποίησης εντολών με δυνατότητα ανίχνευσης ανεξαρτήτων μεταξύ τους εντολών.
  - Πολλαπλές μονάδες εκτέλεσης με δυνατότητα παράλληλης εκτέλεσης εντολών όταν αυτό είναι δυνατόν.

Superscalar Pipeline (1)

Εντολή 1	IF	ID	EX	MEM	WB				
2	IF	ID	EX	MEM	WB				
3		IF	ID	EX	MEM	WB			
4		IF	ID	EX	MEM	WB			
5			IF	ID	EX	MEM	WB		
6			IF	ID	EX	MEM	WB		
7				IF	ID	EX	MEM	WB	
8				IF	ID	EX	MEM	WB	
9					IF	ID	EX	MEM	WB
10					IF	ID	EX	MEM	WB

Superscalar Pipeline (2)

- Το Front-End της pipeline: οι φάσεις IF, ID.
- Το Back-End της pipeline: οι φάσεις EX, MEM, WB.

- Δύο τρόποι εκτέλεσης:
  - Στατικός τρόπος ή εκτέλεση εντολών με τη σειρά (in-order execution).
  - Δυναμικός τρόπος ή εκτέλεση εντολών εκτός σειράς (out-of-order execution).

Στατικό Superscalar (in-order execution)

- Οι εντολές εισάγονται στο front-end (IF-ID) με την σειρά που εμφανίζονται στο πρόγραμμα.

- Κατόπιν εισάγονται στο back-end (EX-MEM-WB) αφού έχει εξασφαλιστεί ότι υπάρχει διαθέσιμος χώρος στην pipeline και έχουν διευθετηθεί όλες οι εξαρτήσεις μεταξύ των δεδομένων.

Παράδειγμα in-order execution

- Έστω ότι έχουμε ένα 2-πλό superscalar επεξεργαστή. Οι εντολές με τη σειρά εμφάνισης στο πρόγραμμα είναι:

  E1: R1 ← R2 + R3 #Ακέραια πράξη
  E2: R4 ← R1 – R5 #Ακέραια πράξη
  E3: R7 ← R8 – R9 #Ακέραια πράξη
  E4: F0 ← F2 + F4 # Πράξη κινητής υποδιαστολής

Η E2 απαιτεί το R1 που παράγεται από την E1

- Η E1 στέλνεται στην pipeline.
- Η E2 στέλνεται στην pipeline αλλά για να εκτελεστεί απαιτεί το R1 που παράγεται από την εντολή E1. Άρα δρομολογείται μετά το WB της E1.
- Η E3 περιμένει την E2 διότι δεν υπάρχει ελεύθερος χώρος στην pipeline.
- Η E4 περιμένει αφού η E3 είναι μπλοκαρισμένη.

Δυναμικό superscalar (out-of-order execution)

- Οι εντολές εισάγονται στο front-end (IF-ID) με την σειρά που εμφανίζονται στο πρόγραμμα.
- Κατόπιν εισάγονται στο back-end (EX-MEM-WB) με οποιαδήποτε σειρά κρίνεται καλύτερη ακόμη κι αν οι προηγούμενες εντολές δεν έχουν εκτελεστεί.
- Ιδιαίτερα πολύπλοκη μέθοδος διότι απαιτεί μεγάλη προσοχή ώστε να μην διαταραχτεί η λογική του προγράμματος.

Παράδειγμα out-of-order execution

- Έστω ότι έχουμε ένα 2-πλό superscalar επεξεργαστή. Οι εντολές με τη σειρά εμφάνισης στο πρόγραμμα είναι:

  E1: R1 ← R2 + R3 #Ακέραια πράξη
  E2: R4 ← R1 – R5 #Ακέραια πράξη
  E3: R7 ← R8 – R9 #Ακέραια πράξη
  E4: F0 ← F2 + F4 # Πράξη κινητής υποδιαστολής

Η E2 απαιτεί το R1 που παράγεται από την E1

- Η E1 στέλνεται στην pipeline στον κύκλο 1.
- Η E3 στέλνεται στην pipeline στον κύκλο 1. Ανεξάρτητη της E1.
- Η E4 θα μπει στην pipeline στον κύκλο 2.
- Η E2 στέλνεται στην pipeline αργότερα (πχ. στον κύκλο 4) ώστε να προλάβει η E1 να παράξει το αποτέλεσμα R1. Έτσι μπορούμε να εκτελέσουμε χρήσιμες εντολές στους κύκλους 2 και 3.

Παραλληλισμός σε επίπεδο εντολών (1)

- Παράδειγμα (ψευδοκώδικας assembly).

```
E1:    add    r2 = r4,r5
E2:    store  [addr1] = r2
E3:    fmul   f5 = f2,f7
E4:    fmul   f2 = f2,f8
E5:    add    r4 = r12,r17
E6:    add    r6 = r12,r20
E7:    add    r7 = r4,r6
E8:    store  [addr2] = r6
```

r2, r4, r5, r6, r12, r17, r20	Integer registers
f2, f5, f7, f8	Floating point registers

Παραλληλισμός σε επίπεδο εντολών (2)

- Παράδειγμα (ψευδοκώδικας assembly).

```
E1:   add   r2 = r4,r5
E2:   store [addr1] = r2
E3:   fmul  f5 = f2,f7
E4:   fmul  f2 = f2,f8
E5:   add   r4 = r12,r17
E6:   add   r6 = r12,r20
E7:   add   r7 = r4,r6
E8:   store [addr2] = r6
```

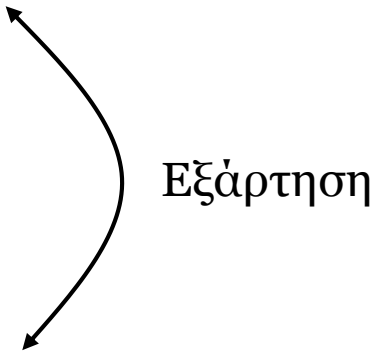
Εξάρτηση

r2, r4, r5, r6, r12, r17, r20	Integer registers
f2, f5, f7, f8	Floating point registers

Παραλληλισμός σε επίπεδο εντολών (3)

- Παράδειγμα (ψευδοκώδικας assembly).

```
E1:   add   r2 = r4,r5
E2:   store [addr1] = r2
E3:   fmul  f5 = f2,f7
E4:   fmul  f2 = f2,f8
E5:   add   r4 = r12,r17
E6:   add   r6 = r12,r20
E7:   add   r7 = r4,r6
E8:   store [addr2] = r6
```



r2, r4, r5, r6, r12, r17, r20	Integer registers
f2, f5, f7, f8	Floating point registers

Παραλληλισμός σε επίπεδο εντολών (4)

- Παράδειγμα (ψευδοκώδικας assembly).

```
E1:   add   r2 = r4,r5
E2:   store [addr1] = r2
E3:   fmul  f5 = f2,f7
E4:   fmul  f2 = f2,f8
E5:   add   r4 = r12,r17
E6:   add   r6 = r12,r20
E7:   add   r7 = r4,r6
E8:   store [addr2] = r6
```

Εξάρτηση

r2, r4, r5, r6, r12, r17, r20	Integer registers
f2, f5, f7, f8	Floating point registers

Παραλληλισμός σε επίπεδο εντολών (5)

- Παράδειγμα (ψευδοκώδικας assembly).

```
E1:   add   r2 = r4,r5
E2:   store [addr1] = r2
E3:   fmul  f5 = f2,f7
E4:   fmul  f2 = f2,f8
E5:   add   r4 = r12,r17
E6:   add   r6 = r12,r20
E7:   add   r7 = r4,r6
E8:   store [addr2] = r6
```

Εξάρτηση

r2, r4, r5, r6, r12, r17, r20	Integer registers
f2, f5, f7, f8	Floating point registers

Παραλληλισμός σε επίπεδο εντολών (6)

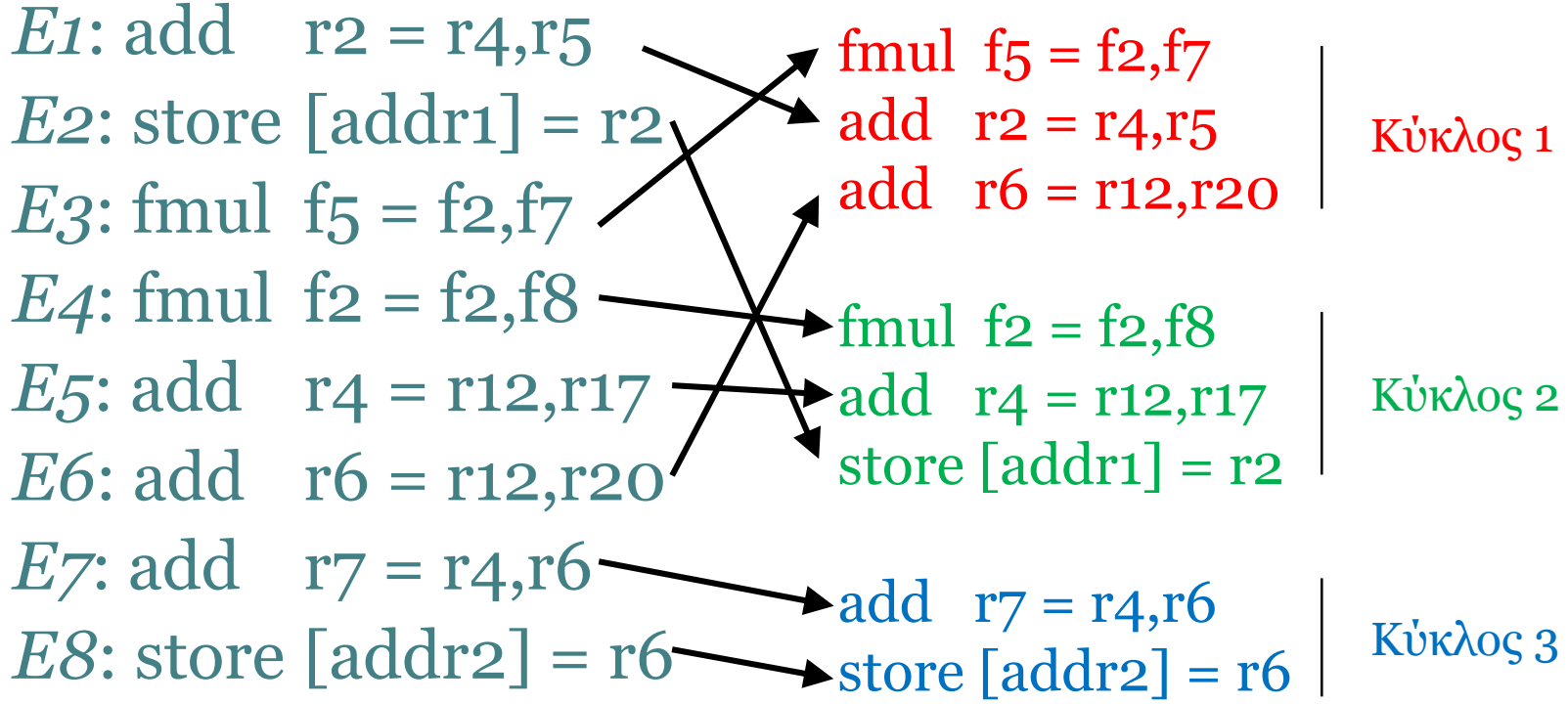
- Παράδειγμα (ψευδοκώδικας assembly).

```
E1:   add   r2 = r4,r5
E2:   store [addr1] = r2
E3:   fmul  f5 = f2,f7
E4:   fmul  f2 = f2,f8
E5:   add   r4 = r12,r17
E6:   add   r6 = r12,r20
E7:   add   r7 = r4,r6
E8:   store [addr2] = r6
```

r2, r4, r5, r6, r12, r17, r20	Integer registers
f2, f5, f7, f8	Floating point registers

Παραλληλισμός σε επίπεδο εντολών (7)

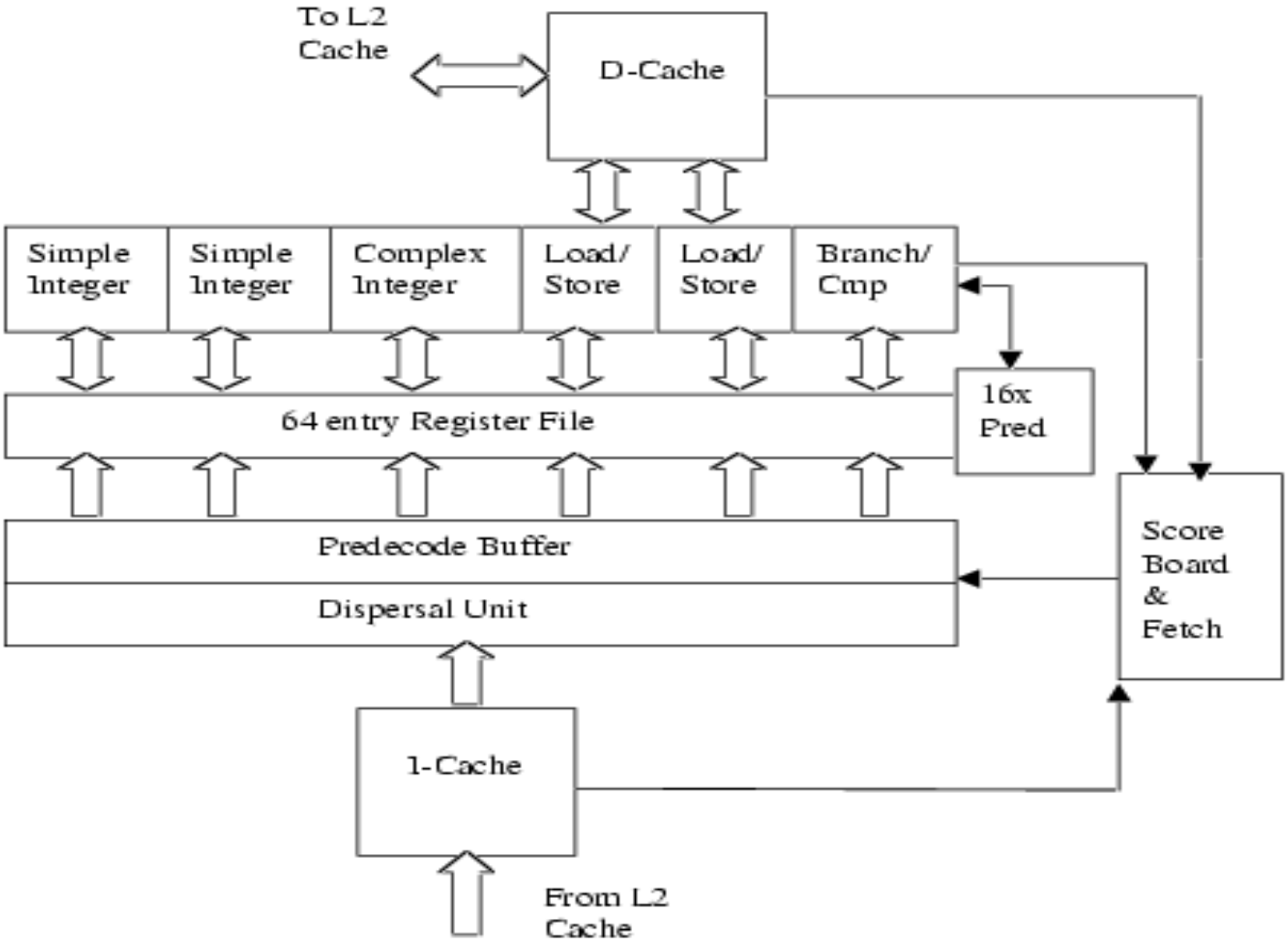
- Παράδειγμα (ψευδοκώδικας assembly).



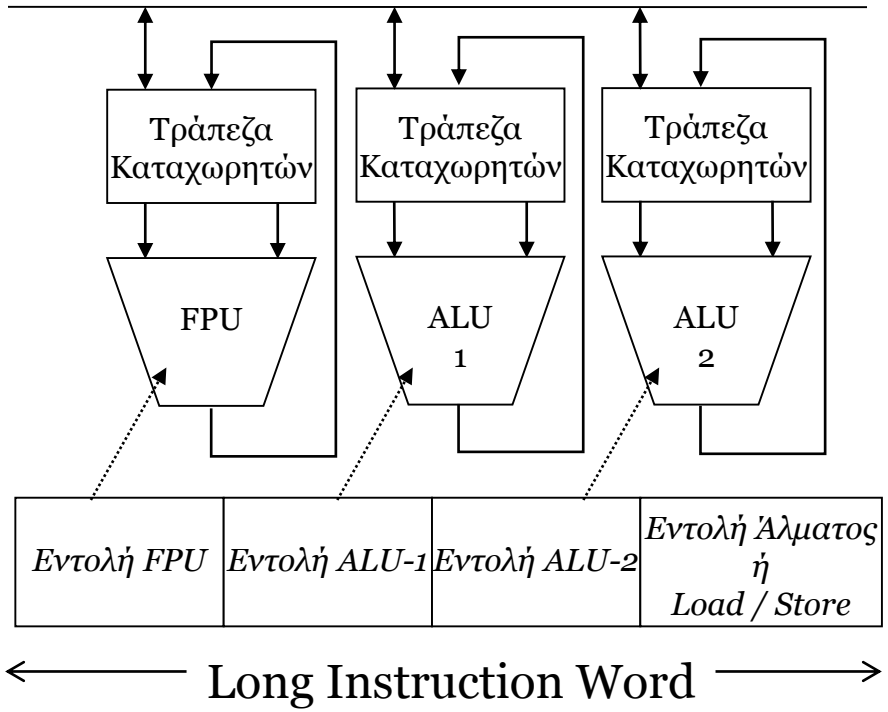
Superscalar: παράγοντες που επηρεάζουν την επίδοση

- 100% χρήση όλων των pipelines → πρέπει να υπάρχουν n εντολές που να μπορούν να εκτελούνται ταυτόχρονα → μεγάλος παραλληλισμός σε επίπεδο εντολών (Instruction Level Parallelism – ILP). Δεν εξαρτάται από τον σχεδιαστή. Τάση: πλήθος pipelines $n \leq 6$.
- n μονάδες εκτέλεσης (ALU ή FPU) → n^2 μονοπάτια ανάμεσά τους. Μεγάλο μήκος των καλωδιώσεων → μείωση συχνότητας του ρολογιού.
- Αύξηση του βάθους της pipeline → (θεωρητικά) ταχύτερες piplines αφού οι φάσεις είναι «λεπτότερες».
- Αλλά: εντολές αλμάτων υπό συνθήκη → χάσιμο περισσότερων κύκλων. Σχεδιαστική τάση είναι προς τη μείωση των φάσεων της pipeline. Pentium D 31 φάσεις Intel Core 14 φάσεις.

Επεξεργαστής VLIW



Τμήμα επεξεργαστή VLIW

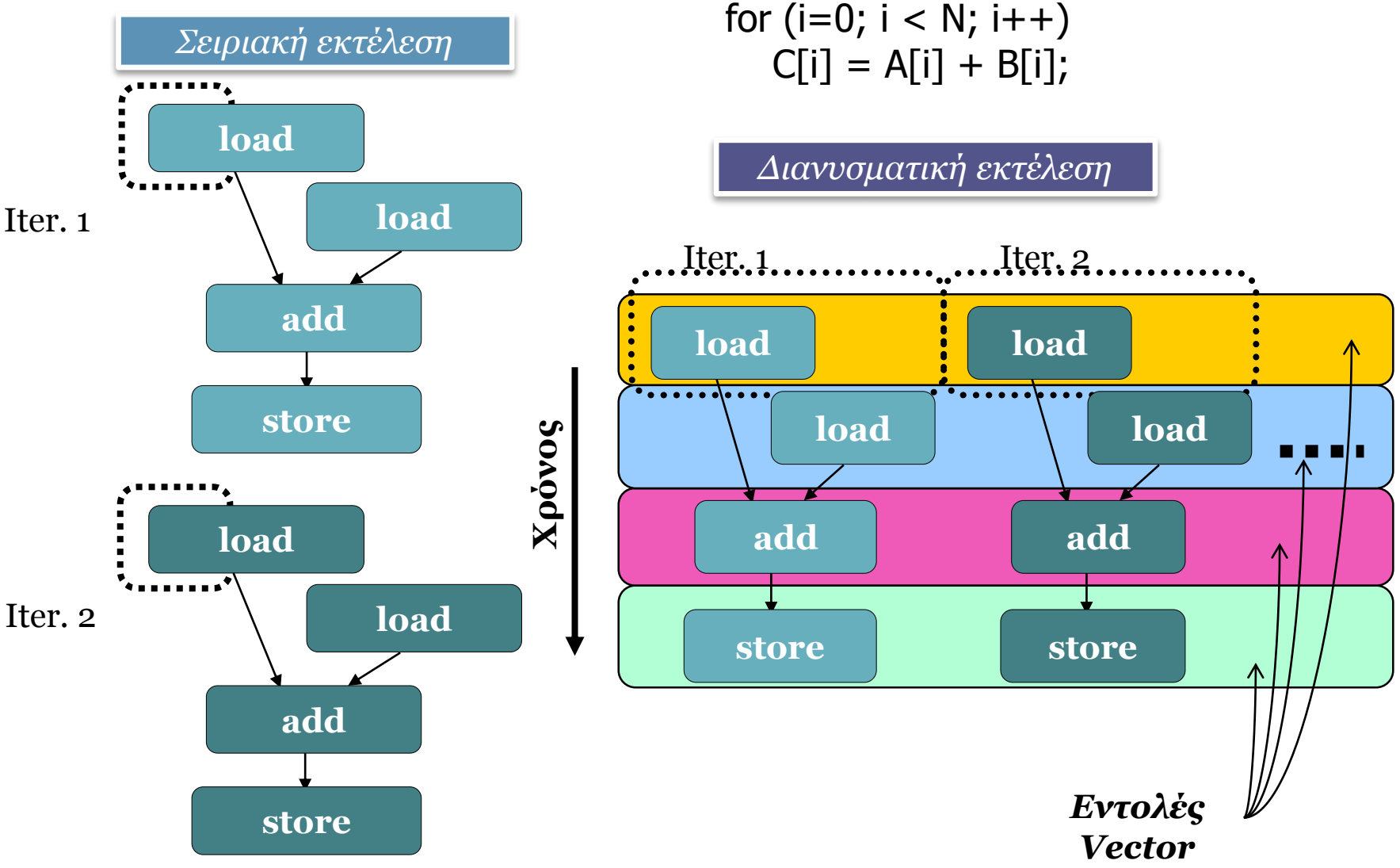


VLIW vs. Superscalar

Superscalar	VLIW
Πολλές εντολές σε ένα κύκλο μηχανής	Πολλές εντολές σε ένα κύκλο μηχανής
Υλικό με απαιτήσεις	Λογισμικό με απαιτήσεις
Η επιλογή για ποιες εντολές θα εκτελεστούν παράλληλα στον ίδιο κύκλο αποφασίζεται δυναμικά (κατά τη διάρκεια της εκτέλεσης) από τον επεξεργαστή	Η επιλογή για ποιες εντολές θα εκτελεστούν παράλληλα στον ίδιο κύκλο αποφασίζεται από τον μεταφραστή κατά τη διάρκεια της μετάφρασης

Αυτόματη διανυσματοποίηση κώδικα

```
for (i=0; i < N; i++)
    C[i] = A[i] + B[i];
```



Σειριακός κώδικας: scalar επεξεργαστής

Κώδικας υψηλού επιπέδου:

```
for (i=0; i<10; i++) {
        x[i] = x[i] + y[i];
}
```

Σειριακός κώδικας:

Εκτέλεσε 10 φορές:

Φόρτωσε από τη μνήμη R1 ← x[i]

Φόρτωσε από τη μνήμη R2 ← y[i]

Πρόσθεσε R3 ← R1 + R2

Σώσε το αποτέλεσμα x[i] ← R3

Τέλος βρόχου

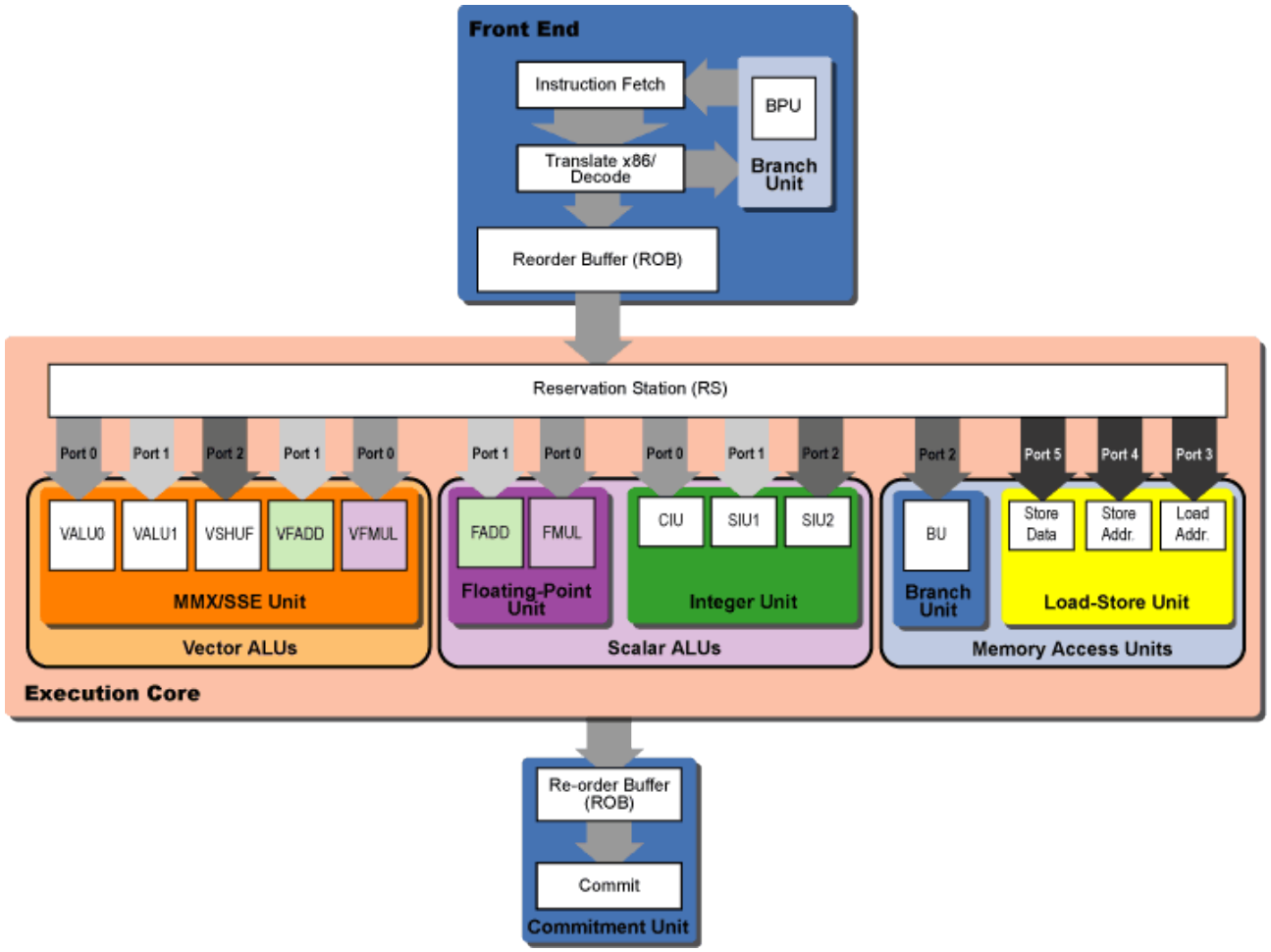
Διανυσματικός κώδικας: vector επεξεργαστής

Διανυσματικός κώδικας:

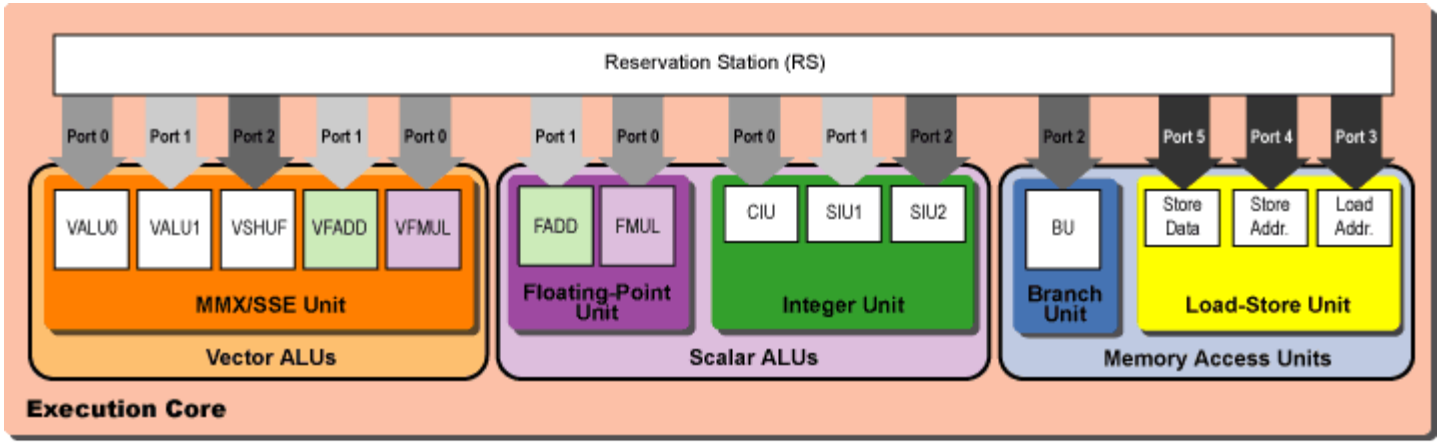
Φόρτωσε 10 αριθμούς R1 ← [x[0],x[1],…,x[9]]
Φόρτωσε 10 αριθμούς R2 ← [y[0],y[1],…,y[9]]
Πρόσθεσε R3 ← R1 + R2
Σώσε το αποτέλεσμα [x[0],x[1],…,x[9]] ← R3

- Διανυσματικοί καταχωρητές R1, R2, R3
- Οι εντολές που εκτελούν παράλληλα πρέπει να είναι όλες ίδιες, πχ 10 ταυτόχρονα ADD

Αρχιτεκτονική Core (Intel): out-of-order superscalar με δυνατότητες vector



Η καρδιά του Intel Core



VALU0	*VALU1*	VSHUF	VFADD	VFMUL
-VALU	*-VALU*	*-Vshuff*	-VFadd	-VFmul
-Vmul	*-Vshift*	*-Vrecip/ rsqrt*	-Vmov	-VFdiv
		-Vmov		*-Vsqrt*
				-Vmov

FADD	FMUL
-Fadd	-Fmul
	-Fdiv

CIU	SIU1	SIU2
-ALU	-ALU	-ALU
-mul		
-shift		