ΗΥ 232
Εισαγωγή στην Οργάνωση και στον Σχεδιασμό Υπολογιστών

Διάλεξη 18

Data-Level Parallelism
Linking & Loading

Νίκος Μπέλλας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

Παραλληλισμός σε εφαρμογές

- Πολλά σημαντικά υπολογιστικά προβλήματα έχουν έμφυτη την ιδιότητα του παραλληλισμού
- Παραλληλισμός πολλαπλών επιπέδων
- *Computer games:*
  - Γραφικά, ήχος, AI, μπορούν να γίνουν εντελώς παράλληλα μεταξύ τους
  - Επίσης, κάθε ένα από αυτά τα κομμάτια έχουν εσωτερικά παράλληλους υπολογισμούς
    - Κάθε Pixel στην οθόνη του υπολογιστή μπορεί να υπολογισθεί και να απεικονισθεί εντελώς ανεξάρτητα από τα υπόλοιπα
    - Η κίνηση αντικειμένων που δεν αλληλοεπιδρούν μπορεί να γίνει εντελώς παράλληλα
    - Stereo sound το ίδιο
- *Google queries:*
  - Κάθε google search μπορεί να γίνει εντελώς παράλληλα με οποιοδήποτε άλλο

Data Level Parallelism

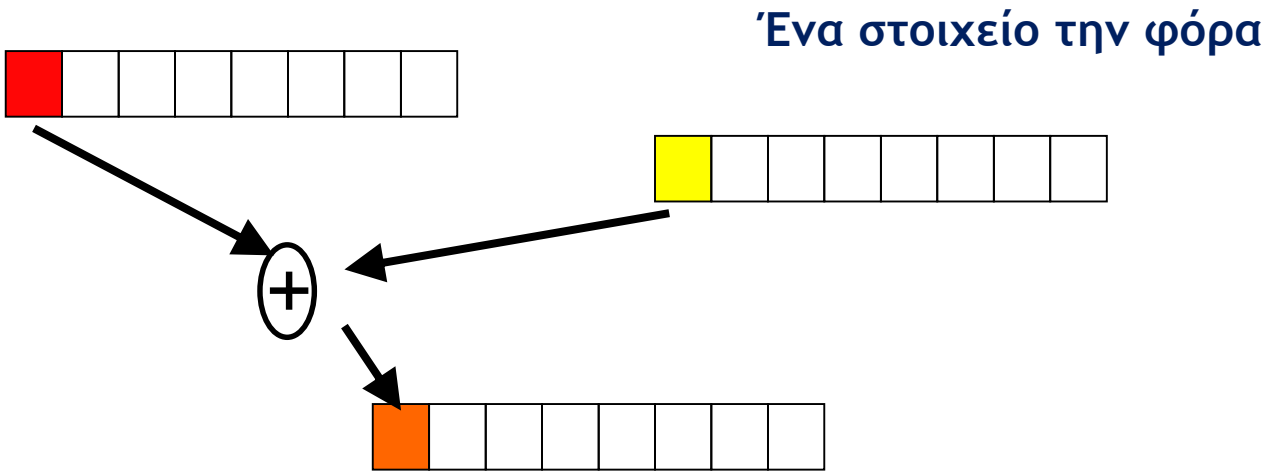
- Έστω ο παρακάτω κώδικας C που προσθέτει δύο πίνακες

```
void
array_add(int A[], int B[], int C[], int length) {
  int i;
  for (i = 0 ; i < length ; ++ i) {
    C[i] = A[i] + B[i];
  }
}
```

- Σε assembly MIPS

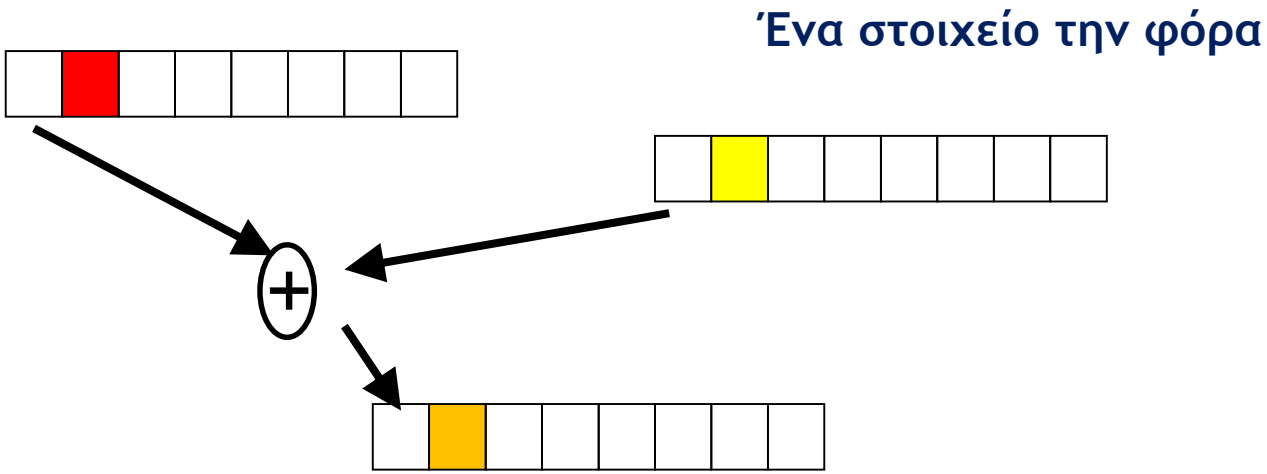
```
lw     $t0, 0($a0)
lw     $t1, 0($a1)
add    $t0, $t1, $t2
sw     $t2, 0($a2)
```

Data Level Parallelism




```
void
array_add(int A[], int B[], int C[], int length) {
  int i;
  for (i = 0 ; i < length ; ++ i) {
   C[i] = A[i] + B[i];
  }
}
```

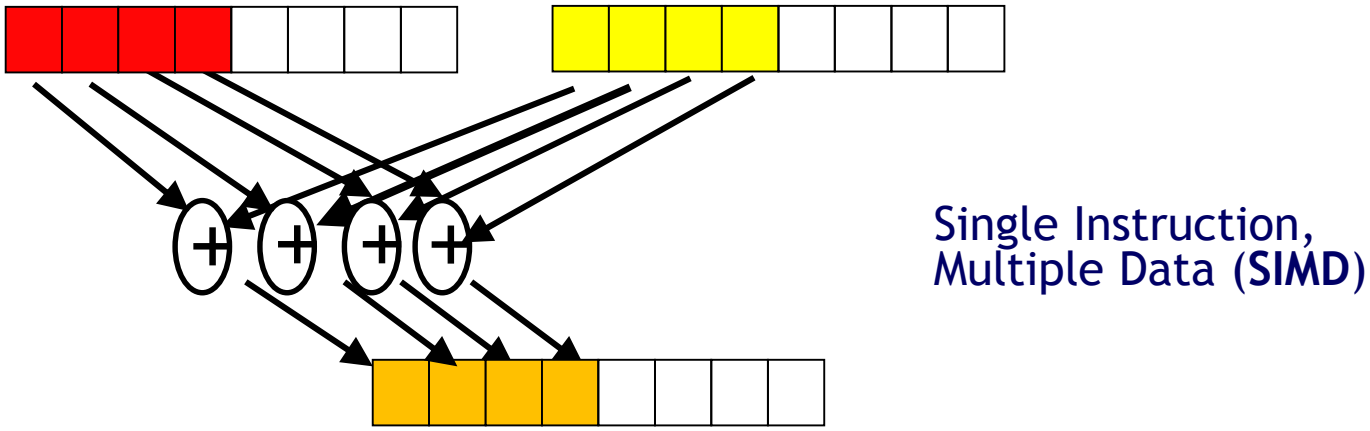
Data Level Parallelism




```
void
array_add(int A[], int B[], int C[], int length) {
  int i;
  for (i = 0 ; i < length ; ++ i) {
   C[i] = A[i] + B[i];
  }
}
```

Data Level Parallelism

Μία εντολή επενεργεί σε πολλαπλά δεδομένα

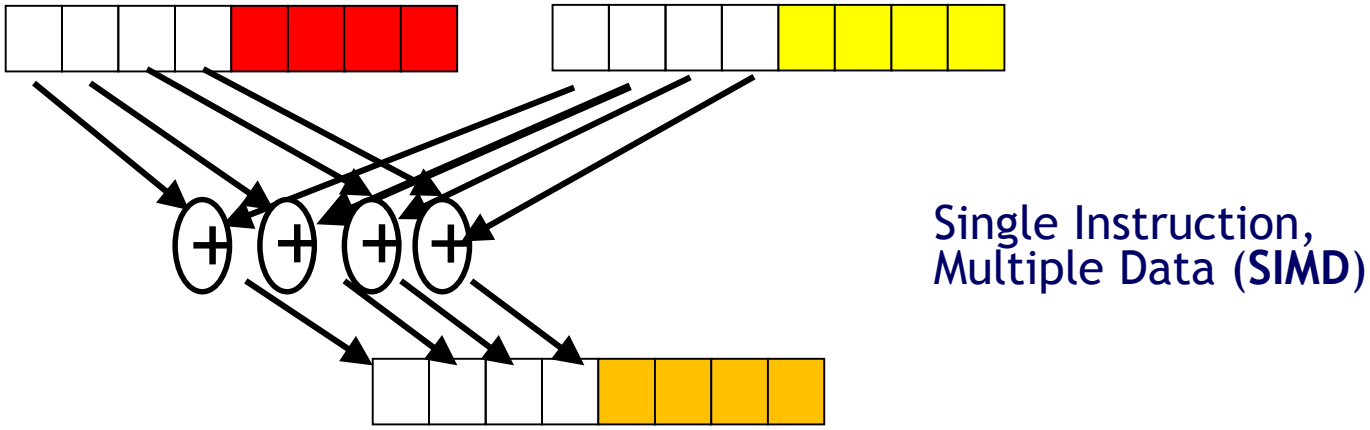


Single Instruction,
Multiple Data (**SIMD**)

```
void
array_add(int A[], int B[], int C[], int length) {
  int i;
  for (i = 0 ; i < length ; ++ i) {
   C[i] = A[i] + B[i];
  }
}
```

Data Level Parallelism

Μία εντολή επενεργεί σε πολλαπλά δεδομένα



Single Instruction,
Multiple Data (**SIMD**)

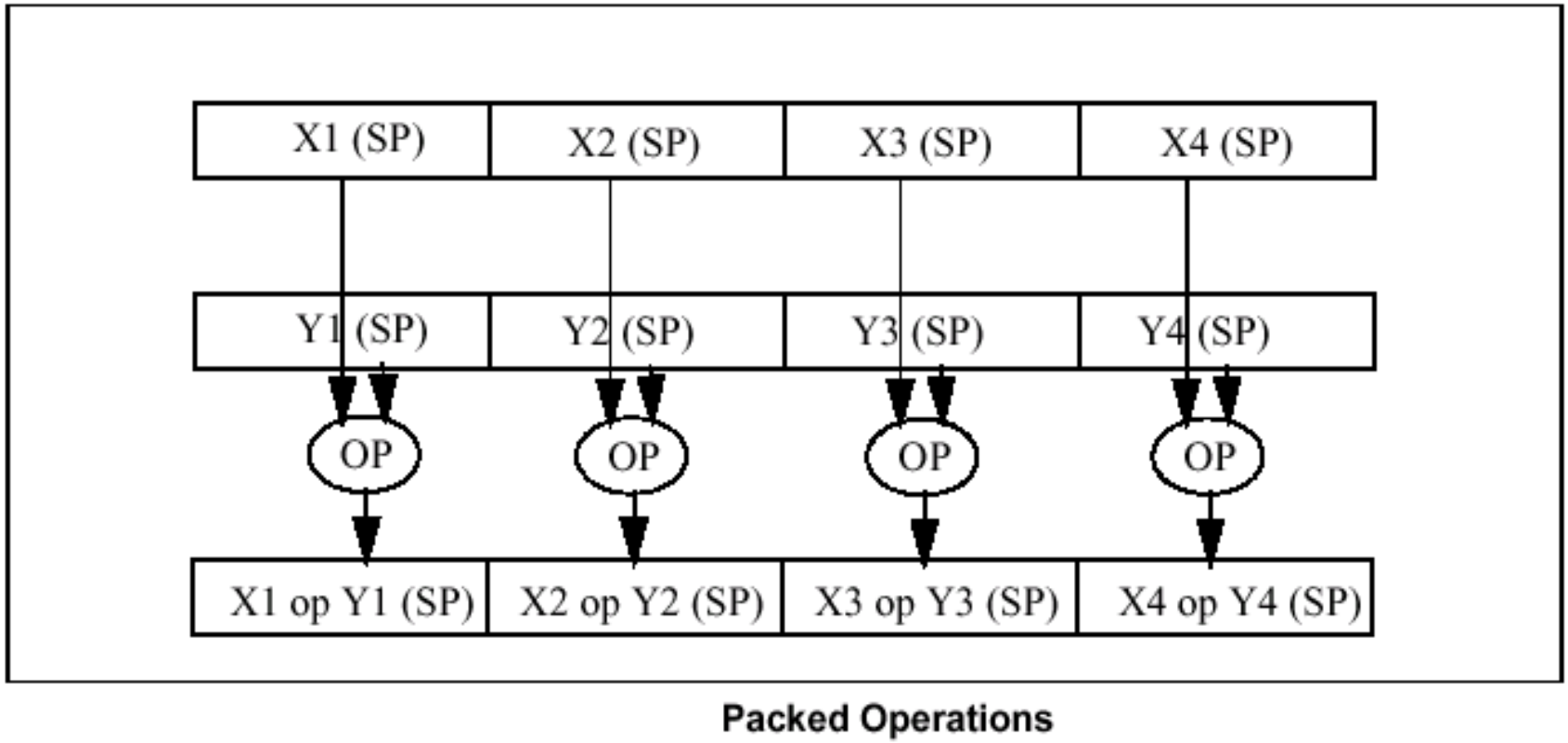
```
void
array_add(int A[], int B[], int C[], int length) {
  int i;
  for (i = 0 ; i < length ; ++ i) {
   C[i] = A[i] + B[i];
  }
}
```

Intel SSEx εντολές

- Αυτή η επέκταση στο x86 ISA προσέθεσε νέους 128-bit καταχωρητές (XMM0 – XMM7)
- Ο κάθε ένας μπορεί να αποθηκεύσει:
  - 4 single precision FP values (SSE) 4 * 32b
  - 2 double precision FP values (SSE2) 2 * 64b
  - 16 byte values (SSE2) 16 * 8b
  - 8 word values (SSE2) 8 * 16b
  - 4 double word values (SSE2) 4 * 32b
  - 1 128-bit integer value (SSE2) 1 * 128b

	4.0 (32 bits)	4.0 (32 bits)	3.5 (32 bits)	-2.0 (32 bits)
+	-1.5 (32 bits)	2.0 (32 bits)	1.7 (32 bits)	2.3 (32 bits)
	2.5 (32 bits)	6.0 (32 bits)	5.2 (32 bits)	0.3 (32 bits)

SSE εντολές



Packed Operations

Διανυσματικές εντολές (vector instructions) για πρόσθεση, αφαίρεση, πολλ/μό, διαίρεση, τετραγωνική ρίζα, min, max.

Ακέραιοι ή κινητής υποδιαστολής.

X86 SSE code

mov = data movement
dq = double-quad (128b)
a = aligned

%eax = A
%ebx = B
%ecx = C
%edx = i

A + 4*i

```
movdqa      (%eax,%edx,4), %xmm0      # load A[i] to A[i+3] to %xmm0
movdqa      (%ebx,%edx,4), %xmm1      # load B[i] to B[i+3] to %xmm1
paddd       %xmm0, %xmm1              # CCCC = AAAA + BBBB
movdqa      %xmm1, (%ecx,%edx,4)      # store C[i] to C[i+3]
addl        $4, %edx                  # i += 4
(loop control code)
```

p = packed

Παράδειγμα II

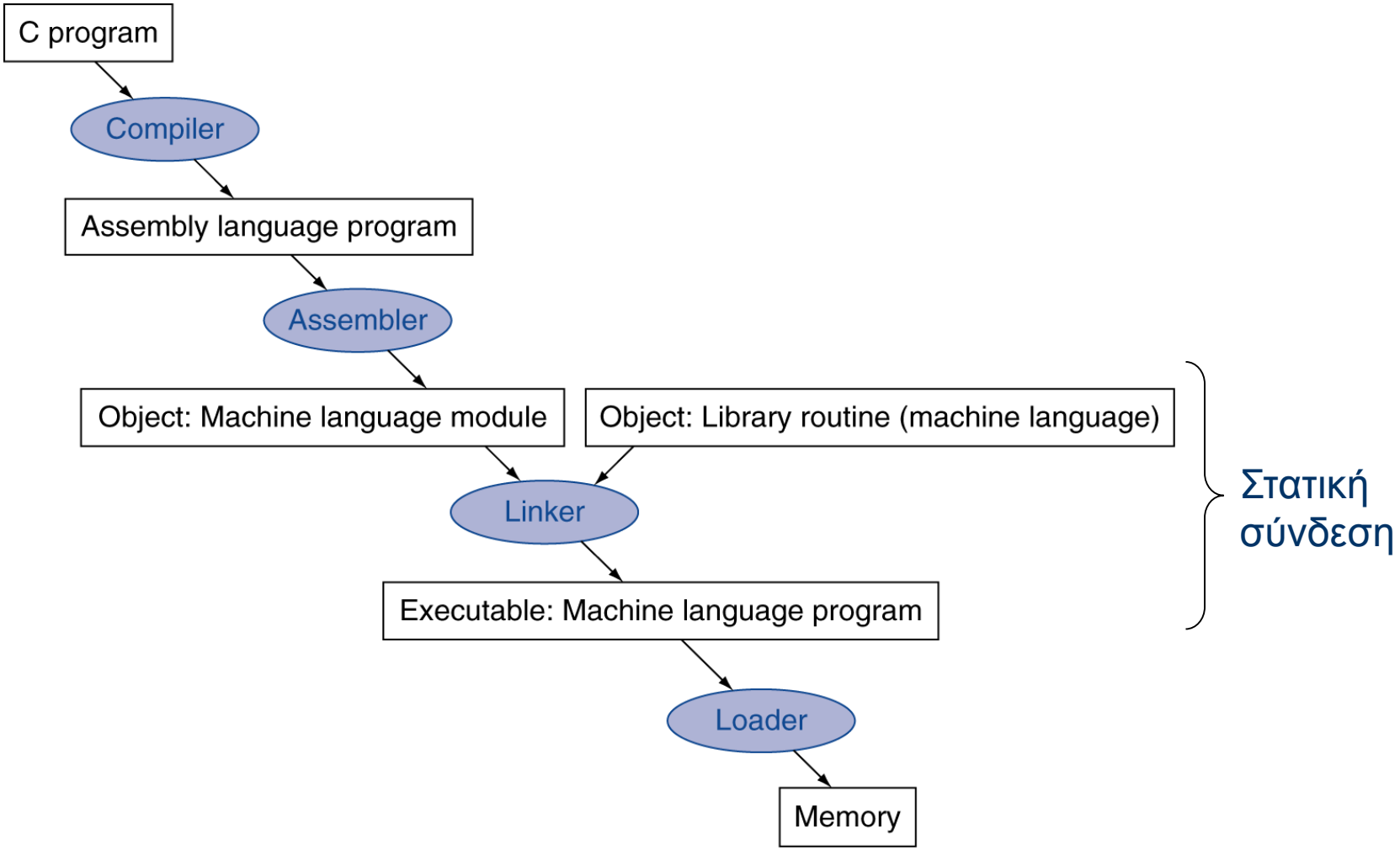
```
unsigned
sum_array(unsigned *array, int length) {
  int total = 0;
  for (int i = 0 ; i < length ; ++ i) {
      total += array[i];
  }
  return total;
}
```

Παράδειγμα II

```
unsigned
sum_array2(unsigned *array, int length) {
  unsigned total, i;
  unsigned temp[4] = {0, 0, 0, 0};
  for (i = 0 ; i < length & ~0x3 ; i += 4) {
    temp[0] += array[i];
    temp[1] += array[i+1];
    temp[2] += array[i+2];
    temp[3] += array[i+3];
  }
  total = temp[0] + temp[1] + temp[2] + temp[3];
  for ( ; i < length ; ++ i) {
    total += array[i];
  }
  return total;
}
```

Linking & Loading

Στάδια μετάφρασης ενός προγράμματος C



Εργασίες συμβολο-μεταφραστή (Assembler)

1. Μετάφραση ψευδοεντολών

- Οι περισσότερες εντολές του συμβολο-μεταφραστή έχουν 1-1 αντιστοιχία με εντολές της γλώσσας μηχανής.
- Ο συμβολο-μεταφραστής μετατρέπει τις ψευδοεντολές (pseudo-instructions) σε συνδιασμούς πραγματικών εντολών
- Οι ψευδοεντολές είναι:
  - Βοηθητικές εντολές οι οποίες αντικαθιστούν γνωστούς συνδυασμούς ή παραλλαγές εντολών της assembly που χρησιμοποιούνται για ειδικούς σκοπούς
  - 32-bit επεκτάσεις των σταθερών 16-bit στις εντολές I-type

```
move $t0, $t1     →  add $t0, $zero, $t1
blt $t0, $t1, L   →  slt $at, $t0, $t1
                     bne $at, $zero, L
– li $s0,0x0123ABCD →  lui $s0, 0x0123
                     ori $s0,$s0,0xABCD
```

  - $at (καταχωρητής 1): για χρήση απόκλειστικά από τον συμβολομεταφραστή

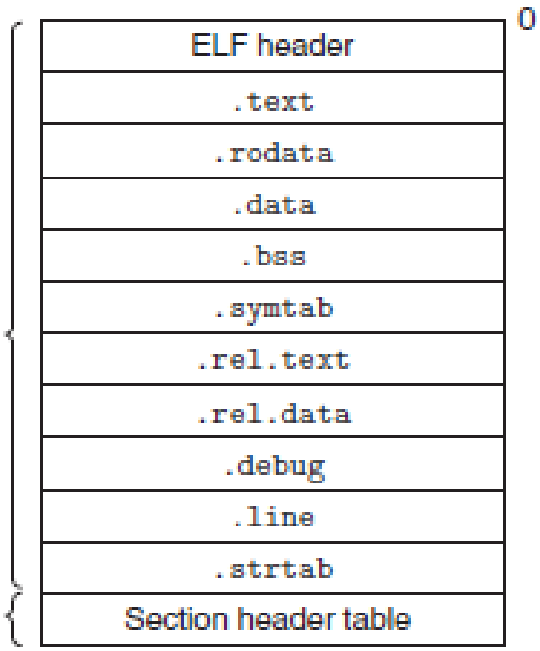
Εργασίες συμβολο-μεταφραστή

2. Παραγωγή αντικειμενικών υπομονάδων

- Ο μεταγλωττιστής ή ο συμβολομεταφραστής μετατρέπει το αρχικό πρόγραμμα σε ένα αντικειμενικό αρχείο (object file)
- Το αντικειμενικό αρχείο περιέχει
  - το πρόγραμμα μεταγλωττισμένο σε εντολές γλώσσας μηχανής
  - επιπλέον πληροφορίες και δεδομένα που χρειάζονται για την εκτέλεση του προγράμματος

Παραγωγή αντικειμενικής υπομονάδας

- Το αντικειμενικό αρχείο (*.o) περιέχει
  - **Επικεφαλίδα (Header):** περιγράφει το μέγεθος (σε bytes) και την θέση στην μνήμη των υπόλοιπων τμημάτων του αντικειμενικού αρχείου
  - **Τμήμα κειμένου (Text segment):** περιλαμβάνει τον κώδικα γλώσσας μηχανής
  - **Τμήμα στατικών δεδομένων (Static data segment):** Δεδομένα που διαρκούν καθ'όλη τη ζωή του προγράμματος (πχ. static variables). .data, .bss
  - **Πληροφορίες επανατοποθέτησης (Relocation info):** προσδιορίζουν εντολές και δεδομένα που εξαρτώνται από απόλυτες διευθύνσεις
  - **Πίνακας συμβόλων (Symbol table):** εξωτερικές αναφορές και συμβολικές αναφορές
  - **Πληροφορίες αποσφαλμάτωσης (Debug info):** για την αντιστοίχιση με τον πηγαίο κώδικα όταν κάνουμε debugging

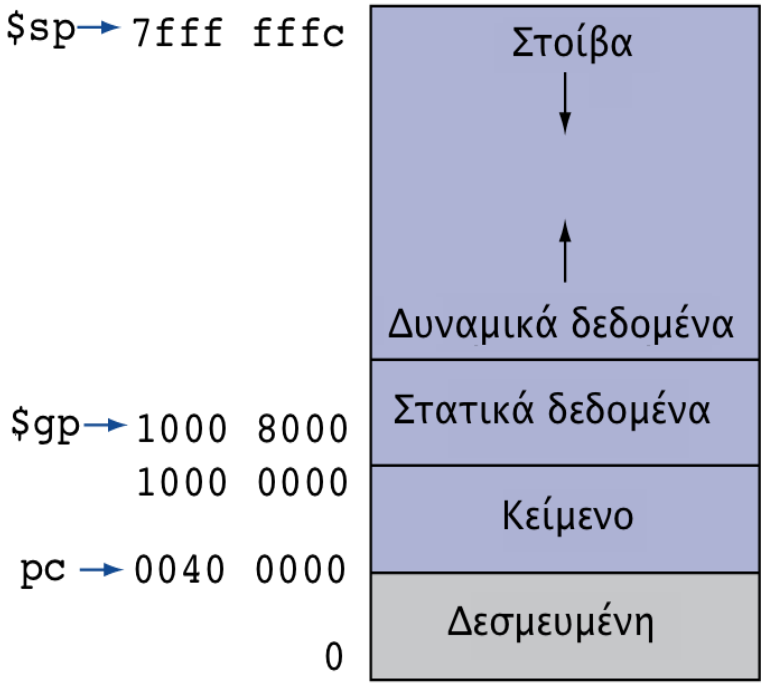


Εργασίες προγράμματος σύνδεσης (linker)

- Σύνδεση αντικειμενικών υπομονάδων και παραγωγή του εκτελέσιμου αρχείου (executable) σε 3 στάδια:
  1. Τοποθέτηση τμημάτων κώδικα και δεδομένων συμβολικά στη μνήμη
  2. Προσδιορισμός διευθύνσεων ετικετών εντολών και δεδομένων
  3. Επιδιόρθωση (patching) εσωτερικών κι εξωτερικών αναφορών (internal and external references)
- Το πρόγραμμα σύνδεσης παράγει ένα εκτελέσιμο αρχείο (executable file)
  - Binary file : σειρά από 0 και 1
  - Μπορεί να εκτελεσθεί από υπολογιστή
- Ας δούμε ένα παράδειγμα:

Επανάληψη: Κατανομή μνήμης

- Κείμενο (Text): κώδικας
- Στατικά δεδομένα (*static data segment*): καθολικές μεταβλητές
  - πχ., στατικές μεταβλητές C, πίνακες, συμβολοσειρές
  - $gp : επιτρέπει εύκολη πρόσβαση στο τμήμα
- Δυναμικά δεδομένα: σωρός (heap)
  - π.χ., malloc στη C, new στη Java
- Τοπικές ματαβλητές: Στοίβα (Stack)



Παράδειγμα σύνδεσης προγραμμάτων

```
AA.c:
    int X;

    A () {
      ....
      lw $a0, X
      call B;
      ....
    }
```

```
BB.c:
    int Y;

    void B () {
      ...
      sw $a1, Y
      call A;
      ....
    }
```

AA.o

BB.o

Object file header			
	Name	Procedure A	
	Text size	100_{hex}	
	Data size	20_{hex}	
Text segment	Address	Instruction	
	0	`lw $a0, 0($gp)`	
	4	`jal 0`	
	...	...	
Data segment	0	(X)	
	...	...	
Relocation information	Address	Instruction type	Dependency
	0	`lw`	X
	4	`jal`	B
Symbol table	Label	Address	
	X	—	
	B	—	

Object file header			
	Name	Procedure B	
	Text size	200_{hex}	
	Data size	30_{hex}	
Text segment	Address	Instruction	
	0	`sw $a1, 0($gp)`	
	4	`jal 0`	
	...	...	
Data segment	0	(Y)	
	...	...	
Relocation information	Address	Instruction type	Dependency
	0	`sw`	Y
	4	`jal`	A
Symbol table	Label	Address	
	Y	—	
	A	—	

- Θέλουμε να συνδέσουμε (link) τα δύο αντικειμενικά αρχεία (object files AA and BB)
- Οι εντολές φαίνονται εδώ σε συμβολική μορφή για να κατανοηθούν καλύτερα
- Στην πραγματικότητα, οι εντολές θα ήταν δυαδικοί αριθμοί

Παράδειγμα σύνδεσης προγραμμάτων

Executable file header		
	Text size	300_{hex}
	Data size	50_{hex}
Text segment	Address	Instruction
	$0040\ 0000_{hex}$	lw $a0, 8000_{hex}($gp)
	$0040\ 0004_{hex}$	jal 40 0100_{hex}
	…	…
	$0040\ 0100_{hex}$	sw $a1, 8020_{hex}($gp)
	$0040\ 0104_{hex}$	jal 40 0000_{hex}
	…	…
Data segment	Address	
	$1000\ 0000_{hex}$	(X)
	…	…
	$1000\ 0020_{hex}$	(Y)
	…	…

- Ο καταχωρητής $gp αρχικοποιείται πάντα με την τιμή 0x10008000

  0x10008000 + 0xFFFF8000 = 0x1000 0000

- Τελικό εκτελέσιμο αρχείο (executable file)

Φόρτωση προγράμματος (I)

- Αρχικά το εκτελέσιμο αρχείο βρίσκεται στον σκληρό δίσκο
- Ο φορτωτής (loader):
  1. Διαβάζει την επικεφαλίδα για να προσδιορίσει το μέγεθος των τμημάτων
  2. Δημιουργεί ένα χώρο διευθύνσεων στην κύρια μνήμη αρκετά μεγάλο για εντολές και δεδομένα
  3. Αντιγράφει εντολές και αρχικοποιημένα δεδομένα στη μνήμη (ο Loader είναι τμήμα του λειτουργικού συστήματος, Operating System)

Φόρτωση προγράμματος (II)

- Φόρτωση από το δίσκο στη μνήμη:
  4. Αντιγράφει τις παραμέτρους του προγράμματος που δίνει ο χρήστης στη στοίβα (*argc, argv)*
  5. Αρχικοποιεί τους καταχωρητές (συμπεριλαμβανομένων των $sp, $fp, $gp) και θέτει τον $sp στην πρώτη ελεύθερη θέση
  6. Μεταπηδά στη ρουτίνα εκκίνησης
     - Η ρουτίνα αντιγράφει τις παραμέτρους στους $a0, … και καλεί τη main
     - Όταν η main επιστρέφει, κάνει κλήση συστήματος για έξοδο (exit syscall)