Οργάνωση Υπολογιστών

Επιμέλεια:

Γεώργιος Θεοδωρίδης, Επίκουρος Καθηγητής

Ανδρέας Εμερετλής, Υποψήφιος Διδάκτορας

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

Άδειες Χρήσης

Ανάπτυξη

Το παρόν εκπαιδευτικό υλικό αναπτύχθηκε στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

ΚΕΦΑΛΑΙΟ 2

Η ΓΛΩΣΣΑ ΤΟΥ ΥΠΟΛΟΓΙΣΤΗ

ΑΣΚΗΣΗ 1

Υποθέστε ότι οι μεταβλητές f, g, h ανατίθενται στους καταχωρητές $s0, $s1, $s2, αντίστοιχα. Υποθέστε επίσης ότι η διεύθυνση βάσης των πινάκων A και B βρίσκεται στους καταχωρητές $s6, και $s7, αντίστοιχα.

A. `f = -g + h + B[2];`

B. `f = A[B[g]+4];`

Για τις παραπάνω εντολές της C, ποιος είναι ο αντίστοιχος κώδικας συμβολικής γλώσσας του MIPS;

ΛΥΣΗ

A.

```
I1.   lw    $s0, 8($s7)       //$s0 = Mem [$s7 +4] δηλ., f = B[2]
I2.   add   $s0, $s0, $s2     //$s0 = $s0+$s2 δηλ., f = f + h = B[2]+h
I3.   sub   $s0, $s0, $s1     //$s0 = $s0-$s1 δηλ., f=f-g= B[2]+h-g
```

Στην 1η εντολή εκτελείται η πράξη f = B[2]. Ο πίνακας B περιέχει τα στοιχεία B[0], B[1], B[2], κλπ. Επομένως, το στοιχείο B[2] είναι το 3ο στοιχείο του πίνακα. άρα, μέσω της εντολής lw πρέπει να κάνουμε προσπέλαση στο 3ο στοιχείο του πίνακα. Εφόσον η διεύθυνση βάσης του πίνακα B βρίσκεται στον καταχωρητή $s7 τότε η διεύθυνση του B[2] είναι ίση με Διεύθυνση βάσης του B + 8 δηλαδή, $s7 +8. Θυμηθείτε ότι μία λέξη είναι 4 bytes (32 bits). Άρα, οι διευθύνσεις διαδοχικών στοιχείων ενός πίνακα έχουν απόσταση μεταξύ τους ίση με 4. Αναλυτικότερα, η οργάνωση του πίνακα B στη μνήμη έχει ως εξής

Διεύθυνση	**Στοιχείο**
Διεύθυνση βάσης +0	B[0]
Διεύθυνση βάσης +4	B[1]
Διεύθυνση βάσης +8	B[2]
….	….

Η λειτουργία των επόμενων εντολών είναι προφανής και εξηγείται στα σχόλια αυτών.

Β.

Στην περίπτωση αυτή πρέπει:

i) να προσπελάσουμε τον πίνακα Β και να διαβάσουμε το στοιχείο (δεδομένο) Β[g]

ii) να εκτελέσουμε την πράξη Β[g]+4

iii) να διαβάσουμε το στοιχείο Α[Β[g]+4]

Σύμφωνα με την εξήγηση του προηγούμενου ερωτήματος, η διεύθυνση του στοιχείου Β[g] είναι ίση με Διεύθυνση_βάσης_Β + 4g με g= 0, 1, 2, …

Το ίδιο επίσης ισχύει και για την προσπέλαση του πίνακα Α.

Όσον αφορά το πολλαπλασιασμό x4, αυτός υλοποιείται με αριστερή ολίσθηση δύο θέσεων. Επομένως, έχουμε:

```
I1.   sll   $t1, $s1, 2       //$t1=$s1<<2 δηλ., $t1=4g
I2.   add   $t1, $t1, $s7     //$t1 = 4g + διεύθυνση_βάσης_Β δηλ., ο $t1=
                              διευθ. του B[g]
I3.   lw    $t2, 0($t1)       //$t2 = Mem[0+($t1)] = B[g]
I4.   addi  $t2, $t2, 4       //$t2 = B[g]+4
I5.   sll   $t2, $t2, 2       //Παρόμοια με την I1, δηλ., $t2 = 4(B[g]+4)
I6.   add   $t2, $t2, $s6     //Παρόμοια με την I2, $t2= διευθ. του A[B[g]+4]
I7.   lw    $s0, 0($t2)       //$s0 = f= Mem[0+($t2)] = A[B[g]+4]
```

ΑΣΚΗΣΗ 2

Στον επόμενο πίνακα δίνονται δύο εντολές και η περιγραφή τους, οι οποίες δεν περιλαμβάνονται στο σύνολο εντολών του MIPS. Για κάθε περίπτωση βρείτε μία ακολουθία εντολών του MIPS που εκτελούν την αντίστοιχη λειτουργία

A.	`abs $t2, $t1`	`# R[rd] = |R[rt]|`
B.	`sgt $t1, $t2, $t3`	`# R[rd] = (R[rs] > R[rt]) ? 1 : 0`

ΛΥΣΗ

Η 1η περίπτωση αφορά τον υπολογισμό της απόλυτης τιμής. Επίσης, η 2η περίπτωση αφορά τη λειτουργία set-greater-than δηλαδή, ο καταχωρητής προορισμού R[rd] είναι ίσος με R[rd] = 1 αν R[rs] > R[rt]. Είναι προφανές ότι η εντολή αυτή είναι τύπου R και στη συγκεκριμένη περίπτωση έχουμε: R[rd] = $t1, R[rs] = $t2 και R[rt] = $t3.

Δύο δυνατές υλοποιήσεις με εντολές MIPS για τις λειτουργίες αυτές είναι οι ακόλουθες:

A.

```
abs:
sub   $t2, $zero, $t1    //$t2 = 0 - $t1, $t2 = -$t1
slt   $t3, $t1, $zero    //αν $t1<0 τότε $t3=1; αλλιώς $t3=0 δηλ., $t1≥0
bne   $t3, $zero, DONE   //αν $t3≠0 (δηλ, $t3=1 άρα $t1<0) άλμα στο DONE
add   $t2, $zero, $t1    //$t1≥0, $t2=$t1+0=$t1
DONE
```

B.

```
sgt:
slt   $t1, $t3, $t2      //αν $t3<$t2 τότε $t1=1, αλλιώς $t1=0
```

ΑΣΚΗΣΗ 3

Παρακάτω δίνονται δύο βρόχοι στη συμβολική γλώσσα του MIPS. Αν υποθέσουμε ότι η αρχική τιμή των καταχωρητών $t1 και $ s1 είναι 10_{10} και 0 αντίστοιχα, ποια είναι η τιμή του καταχωρητή $s1 μετά την εκτέλεση των δύο βρόχων; Για καθέναν από τους παραπάνω βρόχους, γράψτε σε συμβολική γλώσσα C την ισοδύναμη ρουτίνα. Υποθέστε ότι οι καταχωρητές $s1, $t1, και $t2 είναι οι ακέραιοι B, i, και temp, αντίστοιχα.

A.
```
LOOP:   slt   $t2, $0, $t1
        bne   $t2, $0, ELSE
        j     DONE
ELSE:   addi  $s1, $s1, 2
        subi  $t1, $t1, 1
        j     LOOP
DONE:
```

B.
```
LOOP:   addi  $t2, $0, 10
LOOP2:  addi  $s1, $s1, 2
        subi  $t2, $t2, 1
        bne   $t2, $0, LOOP2
        subi  $t1, $t1, 1
        bne   $t1, $0, LOOP
DONE:
```

ΛΥΣΗ

A. Αναλύοντας τον κώδικα έχουμε:

```
LOOP:   slt   $t2, $0, $t1      //if 0<$t1 then $t2=1; else $t2=0
        bne   $t2, $0, ELSE     //if $t2≠0 άρα 0<$t1 then jump to ELSE
        j     DONE              //jump to DONE
ELSE:   addi  $s1, $s1, 2       //$s1=$s1+2
        subi  $t1, $t1, 1       //$t1=$t1-1
        j     LOOP              //jump to LOOP
DONE:
```

Με βάση την παραπάνω ανάλυση του κώδικα συμπεραίνουμε τα ακόλουθα. Αρχικά γίνεται ένας έλεγχος αν ο $t1 είναι θετικός ($t1>0). Αν ο $t1>0 τότε η εντολή bne στέλνει την εκτέλεση του προγράμματος στο ELSE. Αλλιώς η επόμενη εντολή, j DONE, στέλνει την εκτέλεση στον τερματισμό. Όταν η εκτέλεση του κώδικα έρθει στο ELSE, τότε γίνονται οι πράξεις $s1 = $s1 + 2 και $t1 = $t1 - 1. Στη συνέχεια με την εντολή j LOOP, γίνεται άλμα στο LOOP και ο βρόχος επαναλαμβάνεται. Επομένως, ο αντίστοιχος ψευδό-κώδικας είναι:

```
i=10      // από την εκφώνηση $st1=10
do{
    B+=2;
    i=i-1;
while (i>0); }
```

Δεδομένου ότι η αρχική του i = $t1 = 10, ο βρόχος θα εκτελεστεί 10 φορές. Επίσης, σε κάθε εκτέλεση γίνεται η πράξη $s1 = $s1 + 2 (B=B+2) και καθώς η αρχική τιμή του $s1 είναι $s1 = 0, μετά το τέλος των 10 επαναλήψεων η τιμή του $s1 θα είναι $s1 = 20 δηλαδή, B=20.

Β. Αναλύοντας τον κώδικα έχουμε:

```
LOOP:   addi  $t2, $0, 10       //$t2=$0+10=0+10=10
LOOP2:  addi  $s1, $s1, 2       //$s2=$s2+2
        subi  $t2, $t2, 1       //$t2=$t2-1
        bne   $t2, $0, LOOP2    //if $t2≠0 jump to LOOP2
        subi  $t1, $t1, 1       //$t1=$t1-1
        bne   $t1, $0, LOOP     //if $t1≠0 jump to LOOP
DONE:
```

Με βάση την παραπάνω ανάλυση του κώδικα συμπεραίνουμε τα ακόλουθα. Αρχικά ο καταχωρητής $t2 που αντιστοιχεί στη μεταβλητή temp αρχικοποιείται στην τιμή 10 δηλαδή, temp=10. Στη συνέχεια εκτελείται ο εσωτερικός βρόχος LOOP2, όπου γίνονται οι πράξεις $s1=$s1+2 και $t2=$t2-1 δηλαδή, B=B+2 και temp= temp-1. Στο τέλος του βρόχου γίνεται ο έλεγχος αν temp=0. Όσο ισχύει temp≠0, επαναλαμβάνεται ο βρόχος LOOP2. Όταν temp= 0, τότε γίνεται η πράξη $t1=$t1-1 δηλαδή, i=i-1 και όσο i≠0 επαναλαμβάνεται ο εξωτερικός βρόχος LOOP. Έτσι, με βάση τα παραπάνω α αντίστοιχος ψευδό-κώδικας C δίνεται παρακάτω.

```
i=10      // από την εκφώνηση $ st1=10
do{
    temp 10
          do {
          B+=2;
          temp=temp-1;}
          while (temp>0)
    i=i-1;
while (i>0); }
```

Από την εκφώνηση έχουμε ότι i=$t2=10. Με βάση την ανάλυση του κώδικα ο εξωτερικός βρόχος εκτελείται 10 φορές, ενώ ο εσωτερικός βρόχος εκτελείται 100 φορές (10 φορές για κάθε επανάληψη του εξωτερικού βρόχου). Εφόσον, σε κάθε επανάληψη του εσωτερικού βρόχου γίνεται η πράξη B=B+2 με αρχική τιμή του B=0, η τελική τιμή είναι B=200.

ΑΣΚΗΣΗ 4

Θεωρείστε τους δύο κώδικες C που δίνονται παρακάτω

Περίπτωση 1	**Περίπτωση 2**

`for (i=0; i<8; i++)`
`a += b;`
	`while (a < 10) {`
	`B[a] = b + a;`
	`a += 1;}`

A. Για τις παραπάνω περιπτώσεις, μεταφράστε τους κώδικες της C σε κώδικα συμβολικής γλώσσας του MIPS. Υποθέστε ότι οι τιμές των μεταβλητών a, b, i, j βρίσκονται στους καταχωρητές $s0, $s1, $t0, $t1, αντίστοιχα. Επίσης, υποθέστε ότι ο καταχωρητής $s2 περιέχει τη διεύθυνση βάσης του πίνακα B.

B. Πόσες εντολές χρειάζονται για την υλοποίηση του κώδικα C; Αν οι μεταβλητές a και b έχουν αρχικές τιμές 10 και 1, αντίστοιχα, ενώ όλα τα στοιχεία του B είναι αρχικά 0, πόσες εντολές MIPS εκτελούνται;

ΛΥΣΗ

A.

Περίπτωση 1

Για την περίπτωση αυτή πρέπει να υλοποιηθούν τα ακόλουθα. Αρχικά πρέπει ο καταχωρητής $t0, που αντιστοιχεί στη μεταβλητή i, να πάρει την τιμή 0. Στη συνέχεια να γίνει ένας έλεγχος αν i < 8 . Όσο ισχύει i < 8, πρέπει να εκτελούνται οι πράξεις i = i + 1 και a = a + b. Έτσι, με βάση τα παραπάνω, μία υλοποίηση σε συμβολική γλώσσα MIPS είναι η ακόλουθη

```
       addi  $t0, $zero, 0        //i=$t0=$0+0=0
       beq   $zero, $zero, TEST   //jump to TEST. Μία ενναλακτική υλοπ. της JUMP
LOOP:  add  $s0, $s0, $s1         //$s0=$s0+$s1 δηλ. a=a+b
       addi  $t0, $t0, 1          //$t0=$t0+1 δηλ. i=i+1
TEST   slti  $t2, $t0, 8          //if $t0<8(δηλ. i<8) then $t2=1; else $t2=0
       bne   $t2, $zero, LOOP     // if $t2≠0 jump to LOOP
DONE
```

Περίπτωση 2

Στην περίπτωση αυτή πρέπει να υλοποιηθούν οι ακόλουθες λειτουργίες. Πρώτον, πρέπει να γίνεται έλεγχος αν a < 10. Όσο ισχύει a < 10, πρέπει να γίνουν οι πράξεις B[a] = a + b και a = a + 1, θεωρώντας ότι η αρχική τιμή του a είναι ίση με 0.

Για την πράξη B [a] = a + b πρέπει να γίνει η πρόσθεση a + b και να βρεθεί η διεύθυνση του στοιχείου B [a] όπου θα αποθηκευθεί το αποτέλεσμα της πρόσθεσης. Η πρόσθεση υλοποιείται με την εντολή add χρησιμοποιώντας ως καταχωρητές πηγής τους $s0 και $s1, στους οποίους βρίσκονται σύμφωνα με την εκφώνηση τις τιμές των a και b. Για τον υπολογισμό της διεύθυνσης B [a] πρέπει να λάβουμε υπόψη την οργάνωση του πίνακα D στη μνήμη. Η οργάνωση αυτή έχει ως εξής:

Διεύθυνση	**Στοιχείο**
Διεύθυνση βάσης +0	B [0]
Διεύθυνση βάσης +4	B [1]
Διεύθυνση βάσης +8	B [2]
….	….

Επομένως, η διεύθυνση του κάθε στοιχείου D[a] υπολογίζεται με τον ακόλουθο τρόπο: *Διεύθυνση_D[a] = Διεύθυνση_βάσης_*B*+ απόσταση_στοιχείου_D[a]_από_τη_βάση*.

Δεδομένου ότι οι λέξεις έχουν μήκος 4 bytes, τότε ο υπολογισμός μεταφράζεται σε *Διεύθυνση_D[a] = Διεύθυνση_βάσης_*B*+ 4*a*.

Συνεπώς, με βάση τα παραπάνω έχουμε τον ακόλουθο κώδικα MIPS

```
LOOP:  slti  $t2, $s0, 10        //if $s0<10 then $t2=1; else $t2=0
       beq   $t2, $zero, DONE    //if $t2=0 δηλ. $s0≥10 jump to DONE; else
                                  continue
       add   $t3, $s0, $s1       //$t3=$s0+$s1 δηλ. $t3 = a + b
       sll   $t2, $s0, 2         //$t2=4*$s0 δηλ. $t2 =4*a
       add   $t2, $s2, $t2       //$t2=$s2+$t2 δηλ. $t2=Διευθ._βάσης_B+4*a
       sw    $t3, 0($t2)         //αποθήκευση αποτελέσματος
       addi  $s0, $s0, 1         //$s0=$0+1 δηλ. a=a+1
       j     LOOP                //jump to LOOP
DONE
```

Β.

Περίπτωση 1

Στην περίπτωση αυτή απαιτούνται 6 εντολές MIPS για την υλοποίηση του C κώδικα. Όσον αφορά το πλήθος των εκτελούμενων εντολών αυτές είναι 44. Συγκεκριμένα, ο βρόχος αποτελείται από 4 εντολές και εκτελείται 10 φορές. Επομένως, εκτελούνται 4x10=40 εντολές. Επίσης, στην αρχή εκτελούνται οι δυο πρώτες και οι δύο τελευταίες εντολές. Άρα, σύνολο 44 εκτελούμενες εντολές.

Περίπτωση 2

Στην περίπτωση αυτή απαιτούνται 8 εντολές MIPS για την υλοποίηση του C κώδικα. Δεδομένου ότι a=10 εκτελούνται μόνο οι δύο πρώτες εντολές.

ΑΣΚΗΣΗ 5

Για τον παρακάτω κώδικα C γράψτε ένα ισοδύναμο πρόγραμμα στη συμβολική γλώσσα του MIPS. Θεωρείστε ότι η πρώτη συνάρτηση (compare) είναι αυτή που εκτελείται πρώτη. Επίσης, υποθέστε ότι οι τιμές των μεταβλητών a, και b βρίσκονται στους καταχωρητές $s1, και $s2 αντίστοιχα, ενώ το αποτέλεσμα επιστρέφεται στον καταχωρητή $v0.

```
int compare(int a, int b) {
    if (sub(a, b) >= 0)
        return 1;
    else
        return 0;
}
int sub (int a, int b) {
    return a-b;}
```

ΛΥΣΗ

Στην περίπτωση αυτή όπου έχουμε κλήση μίας συνάρτησης θα πρέπει να χρησιμοποιηθούν οι εντολές jal και jr ra. Όταν καλείται μία συνάρτηση, η εντολή jal αποθηκεύει στον καταχωρητή ra (return address) τη διεύθυνση επιστροφής του καλούντος προγράμματος και εκτελεί άλμα στη διεύθυνση όπου βρίσκονται οι εντολές της συνάρτησης. Η διεύθυνση επιστροφής του καλούντος προγράμματος είναι η διεύθυνση της επόμενης εντολής αυτού. Επίσης, η εντολή jr ra δίνει τιμή στον καταχωρητή PC την τιμή του καταχωρητή ra δηλαδή, PC=ra.

Με βάση τα παραπάνω, μια υλοποίησε σε γλώσσα MIPS είναι η ακόλουθη.

```
I100.jal   compare        //Κλήση της συνάρτησης compare. Άρα PC=διευθ._της_I1 και ra= διευθ._της_I101 (επόμενη εντολή)
I101. …….

compare
I1.  addi  $sp, $sp, -4   //$sp=$sp-4 δηλ., ο δείτης του σωρού δείχνει μία θέση παρακάτω (δημιουργεί χώρο για μία αποθήκευση)
I2.  sw    ra, 0($sp)     //Αποθήκευση στην κορυφαία θέση του stack (στη θέση που δημιουργήθηκε προηγουμένως) της διεύθυνσης
```

```
                                  επιστροφής του κυρίου προγράμματος. Δηλάδή,
                                  κορυφαία_θέση_του_stack=ra=διευθ._της_I101
I3.  jal   sub                    //Κλήση της συνάρτησης sub. Ο PC=Διυθ._της_I13 (1η
                                  εντολή της sub). Επίσης, ο ra γίνεται ίσος με τη
                                  διεύθυνση της επόμενης εντολής δηλ. ra=Διυθ._της_I4
I4.  addi  $t2, $zero, 1          //$t2=0+1=1
I5.  beq   $v0, $zero, exit       //αν $v0=0 άλμα στο exit; αλλιώς συνέχισε
I6.  slt   $t3, $zero, $v0        //αν $v0>0 τότε $t3=1; αλλιώς $t3=0
I7.  bne   $t3, $zero, exit       //αν $t3≠0 τότε άλμα στο exit; αλλιώς συνέχισε
I8.  addi  $t2, $zero, $zero      //Επομένως $v0<0. Θέσε $t2=0+0=0
exit
I9.  add   $v0, $t2, $zero        //$v0=$t2. Ο $v0 γίνεται ίσος με 0 ή 1 ανάλογα με
                                  το αν $v0<0 ή $v0≥0, όπως ελέγχθηκε στις I4-I8.
I10. lw    ra, 0($sp)             //Ο καταχωρητής ra γίνεται ίσος με το κορυφαίο
                                  στοιχειο του stack. Από τη I2, ισχύει ότι
                                  κορυφαία_θέση_του_stack=διευθ._της_I101. Επομένως,
                                  ra=διευθ._της_I101
I11. addi  $sp, $sp, 4            //$sp=$sp+4. Διόρθωση του stack pointer
I12. jr    ra                     //PC=ra δηλαδή, επιστροφή στο κύριο πρόγραμμα
sub
I13. sub   $v0, $s0, $s1          //$v0=$s0-$s1 δηλ. $v0=a-b
I14. jr    ra                     //PC=ra. Άλμα στη διεύθυνση που η τιμή της
                                  βρίσκεται στον καταχωρητή ra. Λόγω της I3,
                                  ra=Διυθ._της_I4. Άρα, άλμα στη I4
```

Στην παραπάνω υλοποίηση θεωρούμε ότι το κύριο πρόγραμμα κάλεσε στην εντολή I100 τη συνάρτηση compare. Επομένως, η διεύθυνση επιστροφής είναι η διεύθυνση της επόμενης εντολής I101. Επίσης, χρησιμοποιούμε τον καταχωρητή $t2 σαν προσωρινή μεταβλητή στην οποία δίνουμε τις τιμές 0, 1. Αρχικά στην εντολή I4 δίνουμε την τιμή $t2=1 και ελέγχουμε στις εντολές I5-I7 αν a-b≥0. Αν η συνθήκη ισχύει, τότε η εκτέλεση του κώδικα πάει στην ετικέτα (label) exit. Αλλιώς, γίνεται $t2=0 και ακολουθούν οι επόμενες εντολές (label exit). Τέλος, η ετικέτα exit περιλαμβάνει της εντολές για τον καθορισμό της τιμής επιστροφής της συνάρτησης compare και τη διόρθωση του stack pointer.

Σημείωμα Αναφοράς

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Πατρών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.