Πανεπιστήμιο Θεσσαλίας - Τμήμα Πληροφορικής

Εισαγωγή στους Η/Υ

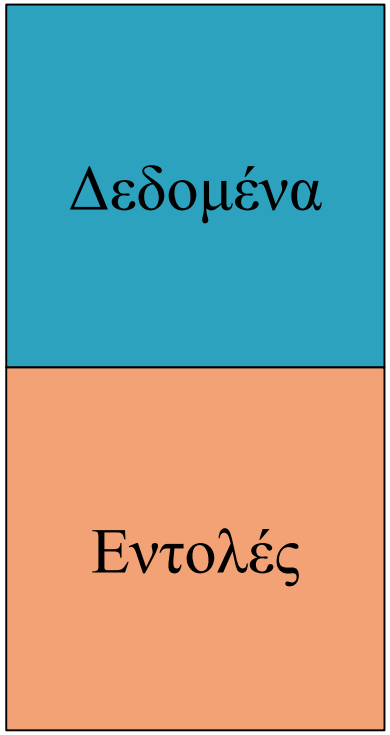
Γιώργος Δημητρίου

Μάθημα 3 και 4: Προγραμματισμός MIPS

Προγραμματισμός σε Συμβολική Γλώσσα

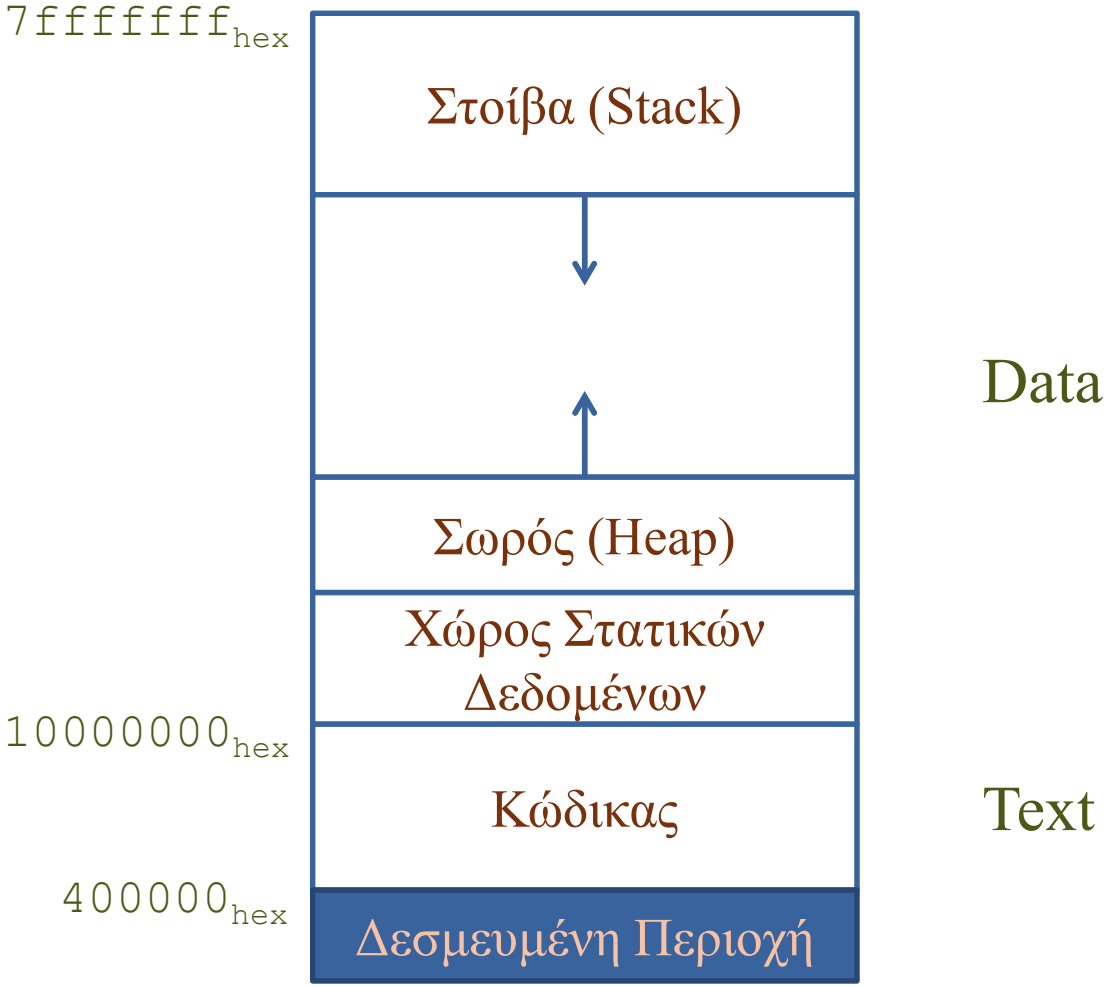
- Η συμβολική γλώσσα:
  - δεν έχει τύπους,
  - δεν έχει δηλώσεις μεταβλητών,
  - δεν έχει δομές ελέγχου,
  - δεν έχει εντολές βρόχων,
  - δεν έχει σύνθετες εντολές
- Τότε γιατί τη μαθαίνουμε;
  - Πλησιέστερα στο υλικό σημαίνει καλύτερος κώδικας;

Μοντέλο Εκτέλεσης Κώδικα Συμβολικής Γλώσσας



- Επίπεδη κατανομή δεδομένων
  - στην ουσία ετικέτες και όχι μεταβλητές
  - αρχικές τιμές
- Εντολές
  - σειριακός κώδικας με διακλαδώσεις
  - πιθανά υποπρογράμματα

Οργάνωση Μνήμης MIPS



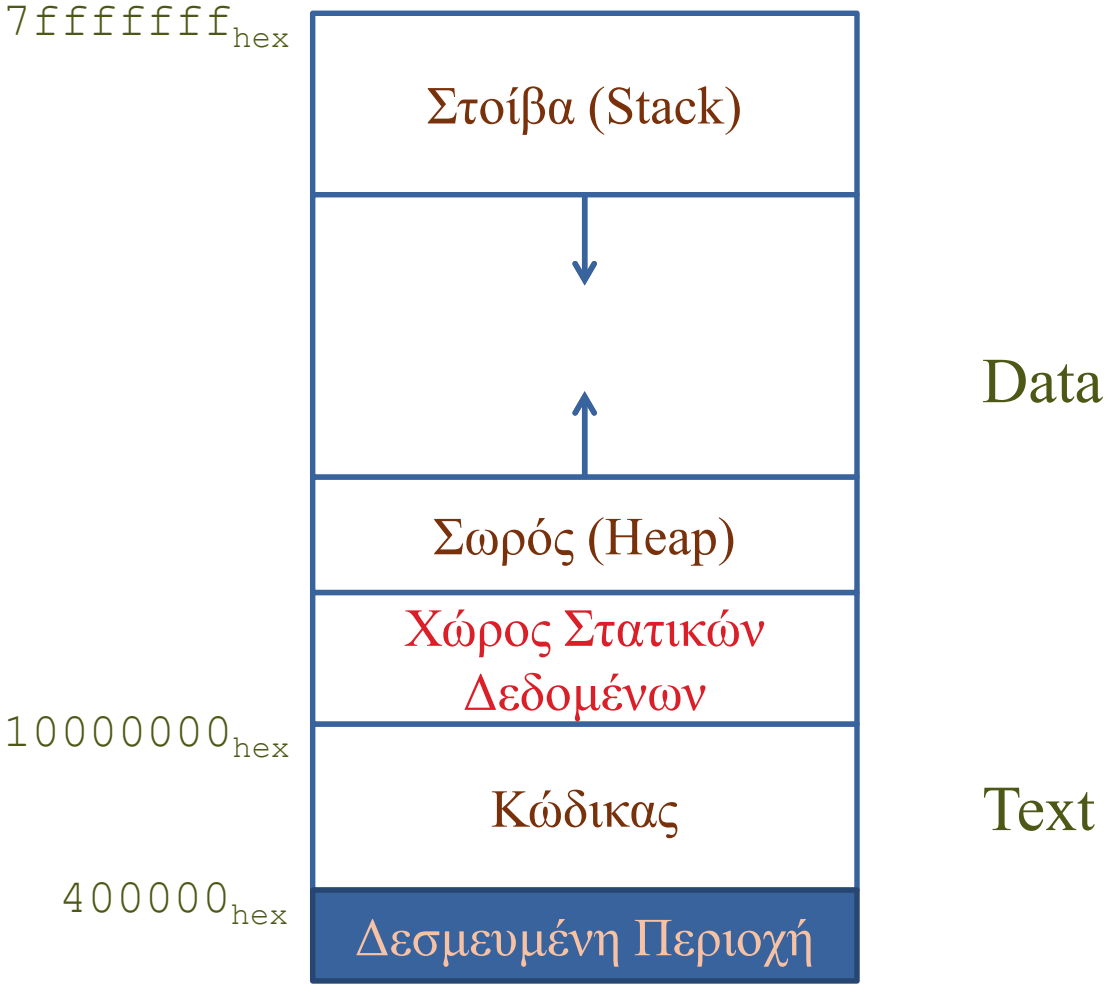
Διευθυνσιοδότηση Μνήμης

- Όλες οι θέσεις μνήμης προσπελαύνονται με:
  - Σχετική διευθυνσιοδότηση (μετατοπίσεις από μια διεύθυνση βάσης) για όλες τις εντολές φόρτωσης και αποθήκευσης
  - Κατ' ευθείαν διευθυνσιοδότηση (απόλυτες διευθύνσεις – συνήθως σε μορφή ετικέτας) για άλματα

Ετικέτες

- Σε κώδικα ή σε δεδομένα
- Απόλυτες διευθύνσεις μνήμης
  - Μέσα στο χώρο διευθύνσεων του προγράμματος
  - Βιβλιοθήκες
- Μπορούν να αναφέρονται πριν τον ορισμό τους
  - Περάσματα μετάφρασης
- Καθολικές ετικέτες (globl)

Οργάνωση Μνήμης MIPS



Καθολικές και Στατικές Μεταβλητές

- Ορίζονται ως διευθύνσεις μνήμης
  - Είτε ως ετικέτες (απόλυτες διευθύνσεις)
  - Είτε με κάποια σχετική διευθυνσιοδότηση, δηλαδή με μετατόπιση σε σχέση με κάποια στατικά ορισμένη αρχική θέση στο χώρο δεδομένων του προγράμματος
- Αρχικοποίηση καθολικών και στατικών μεταβλητών

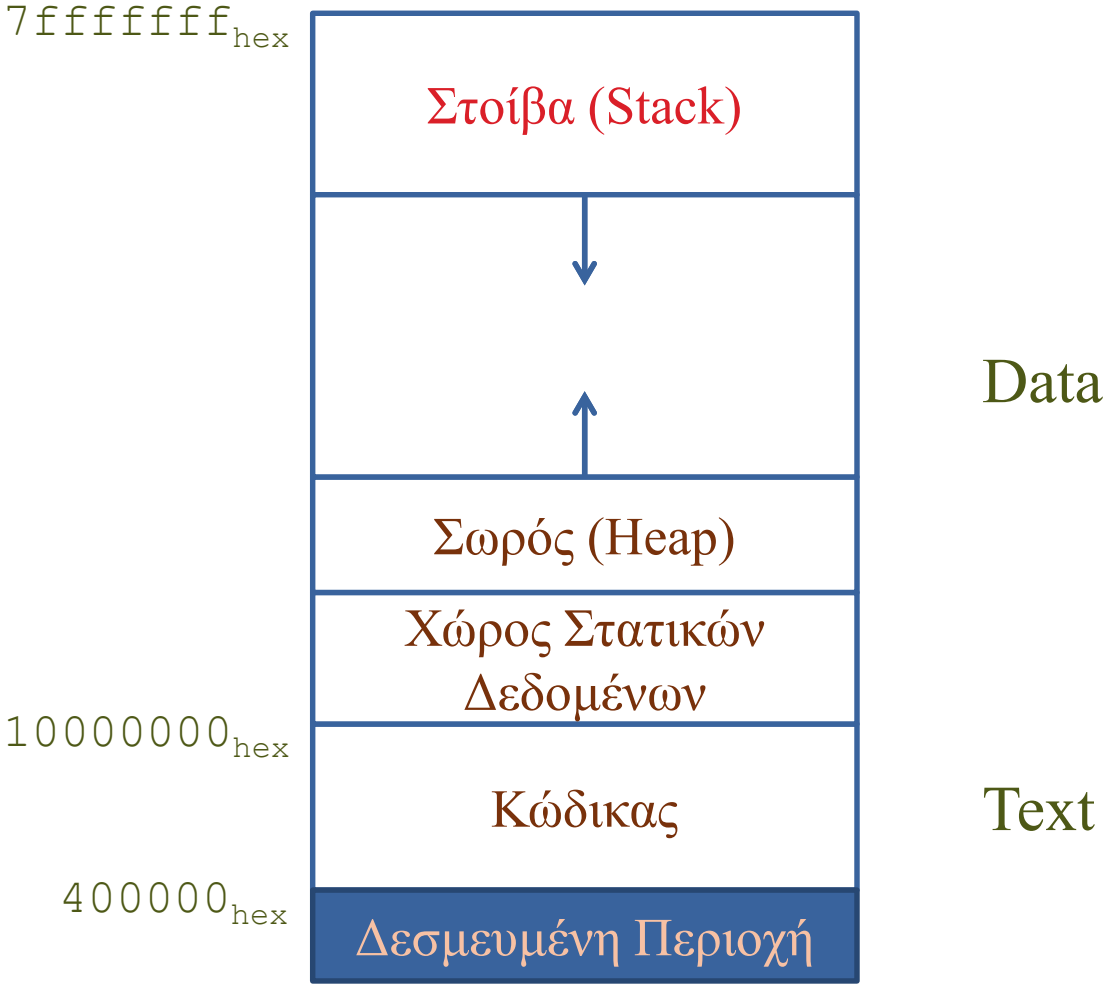
Σταθερές

- Σταθερές που δεν είναι ενσωματωμένες σε εντολές πρέπει να αποθηκεύονται στη μνήμη
  - Ποιες σταθερές ενσωματώνονται σε εντολές;
  - Πώς αποθηκεύουμε σταθερές στη μνήμη;
  - Πώς χρησιμοποιούμε αυτές τις σταθερές;

Τοπικές μη Στατικές Μεταβλητές

- Δε μπορούν να ορίζονται στατικά, γιατί δεν έχουν σταθερή διεύθυνση μνήμης
  - Αυτό συμβαίνει επειδή οι συναρτήσεις στις οποίες είναι ορισμένες καλούνται από διάφορα σημεία του προγράμματος, πιθανά αναδρομικά
  - Άρα δε μπορούν να οριστούν με ετικέτες
  - Ούτε σχετικά με κάποια στατική διεύθυνση του χώρου δεδομένων

Οργάνωση Μνήμης MIPS



Συναρτήσεις και Στοίβα

- Οι κλήσεις συναρτήσεων ακολουθούν δομή στοίβας
- Άρα μπορούμε να δεσμεύουμε τους τοπικούς χώρους δεδομένων σε κάποια στοίβα στη μνήμη
- Έτσι οι διευθύνσεις τοπικών μη στατικών μεταβλητών είναι σχετικές με την αρχή του αντίστοιχου χώρου στη στοίβα
- Αρκεί να παρακολουθούμε την εξέλιξη της στοίβας

Συμβάσεις Καταχωρητών

- Για καλύτερη διαχείριση της μνήμης από κάποιον κώδικα που περιέχει κλήσεις συναρτήσεων, ακολουθούμε κάποιες συμβάσεις στη χρήση των ΚΓΣ:
  - Σύνδεση και επιστροφή
  - Πέρασμα παραμέτρων
  - Τιμή αποτελέσματος
  - Διαχείριση στοίβας
  - Διατήρηση τιμών καταχωρητών

Σύνδεση και Επιστροφή

- Σε κάθε κλήση με την εντολή jal (ή την εντολή jalr), η διεύθυνση της εντολής που ακολουθεί την κλήση αποθηκεύεται αυτόματα στον $ra
- Άρα η επιστροφή από συνάρτηση υλοποιείται ως έμμεσο άλμα:

```
jr $ra
```

- Αποθήκευση του $ra σε προκαθορισμένη θέση στη στοίβα για φωλιασμένες κλήσεις ή άλλη χρήση του $ra

Πέρασμα Παραμέτρων

- Τις παραμέτρους μιας συνάρτησης τις περνάμε σε αυτήν μέσω των καταχωρητών $a0-$a3
  - Και μέσω της στοίβας για περισσότερες παραμέτρους
- Και πάλι αποθηκεύουμε τις τιμές των παραμέτρων σε προκαθορισμένες θέσεις στη στοίβα για επόμενη χρήση των αντίστοιχων καταχωρητών

Τιμή Αποτελέσματος

- Μια συνάρτηση επιστρέφει την τιμή α-πότελέσματός της μέσα από τον κατά-χωρητή $v0
  - Συμπληρωματική επιστροφή μέσω του $v1

Διαχείριση Στοίβας

- Για να δεσμεύουμε χώρο στη στοίβα χρησιμοποιούμε τον ΚΓΣ $sp, ο οποίος:
  - αρχικοποιείται σε κάποια μεγάλη διεύθυνση,
  - μειώνεται κατά το μέγεθος του χώρου που θέλουμε για να κατεβάσουμε τη στοίβα
  - αυξάνεται κατά το ίδιο μέγεθος για να επαναφέρουμε τη στοίβα στην προηγούμενη κατάστασή της

Διατήρηση Τιμών ΚΓΣ

- Κατά σύμβαση, οι καταχωρητές $t0-$t9 χρησιμοποιούνται ελεύθερα για αποθήκευση προσωρινών μεταβλητών
- Όμως οι καταχωρητές $s0-$s7 χρησιμοποιούνται μόνο αφού αποθηκεύσουμε την τιμή τους στη στοίβα, και αφού ολοκληρώσουμε τη χρήση τους, τους ξαναφορτώνουμε

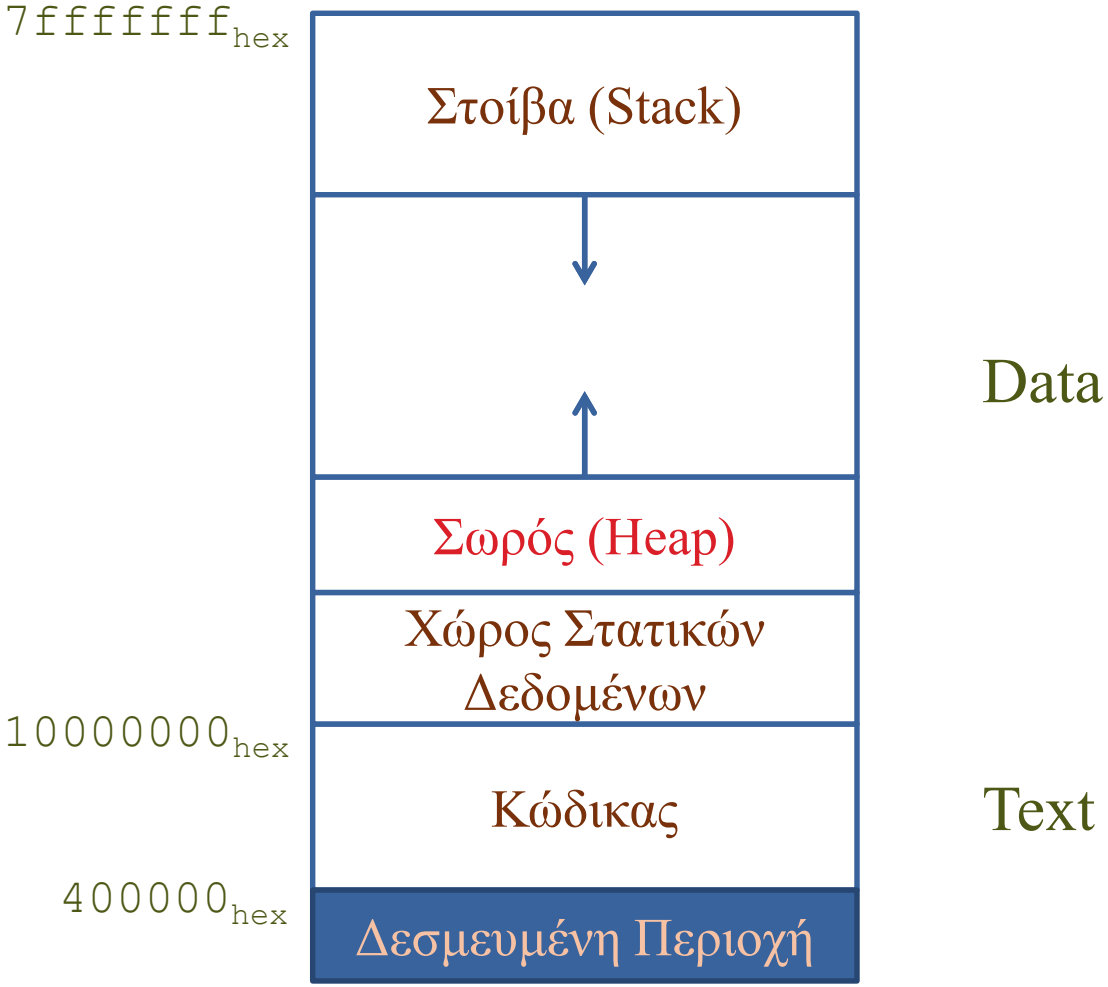
Κλήσεις Συναρτήσεων

- Τα βήματα που ακολουθούμε σε μια κλήση συνάρτησης (μέσα από την καλούσα):
  - Προετοιμασία παραμέτρων
    - Οι τέσσερεις πρώτες αντιγράφονται στους ΚΓΣ $a0-$a3
    - Οι υπόλοιπες αποθηκεύονται στη στοίβα
  - Αποθήκευση τιμών ΚΓΣ (εκτός των $s0-$s7) που θα χρειαστούμε όταν επιστρέψουμε
  - Άλμα με σύνδεση στη συνάρτηση
  - Αντιγραφή της τιμής αποτελέσματος σε ασφαλή αποθήκευση
  - Επαναφορά των τιμών ΚΓΣ που αποθηκεύσαμε

Κλήσεις Συναρτήσεων

- Τα βήματα που ακολουθούμε σε μια κλήση συνάρτησης (μέσα από την καλούμενη):
  - Δέσμευση χώρου στη στοίβα με μείωση του $sp
  - Εκτέλεση του κώδικα της συνάρτησης
    - με αποθήκευση όποιων από τους $ra, $a0-$a3, $s0-$s7 θέλουμε να χρησιμοποιήσουμε
    - και μετέπειτα επαναφορά όποιων από τους $ra, $s0-$s7 χρησιμοποιήσαμε
  - Μεταφορά της τιμής αποτελέσματος στον $v0
  - Επαναφορά της στοίβας με αύξηση του $sp
  - Επιστροφή με έμμεσο άλμα μέσω του $ra
- Υπερχείλιση στοίβας

Οργάνωση Μνήμης MIPS

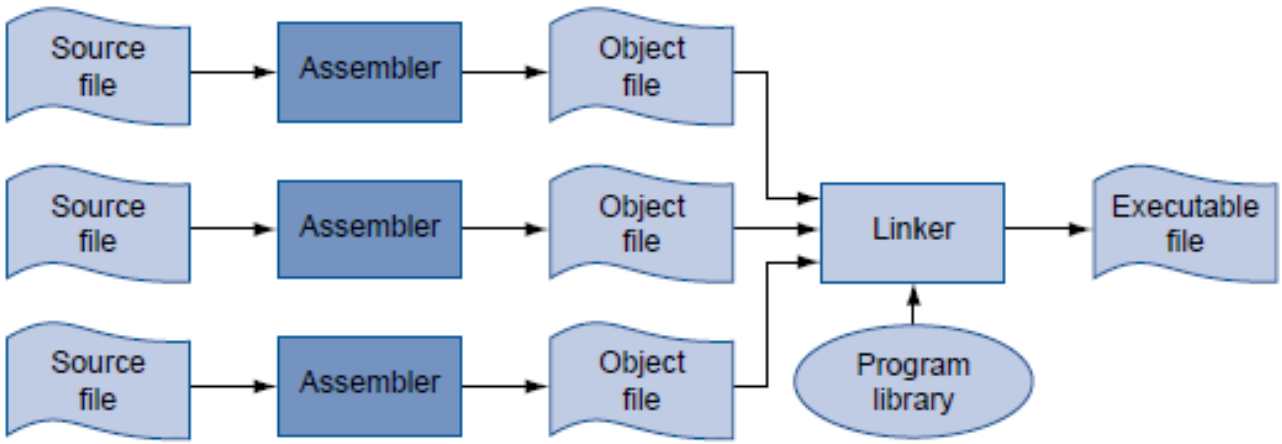


Δυναμική Μνήμη

- Ο σωρός όπου γίνεται δυναμική δε-σμευση μνήμης μέσα από συναρτήσεις όπως η malloc() κλπ
- Κάθε εντολή δέσμευσης ανεβάζει το σωρό προς τη στοίβα
- Υπερχείλιση σωρού

Μετάφραση Κώδικα Συμβολικής Γλώσσας

- Αναγκαία εργαλεία: Συμβολομεταφραστής (Assembler), Συνδέτης (Linker), Βιβλιοθήκες



Παραδείγματα Κώδικα MIPS

- Ας δούμε κάποια παραδείγματα, ξεκινώντας από κώδικα C και καταλήγοντας σε κώδικα MIPS
- Γενικά δίνουμε κώδικα MIPS ανεξάρτητο από την πλατφόρμα προσομοίωσης που χρησιμοποιούμε, ώστε να μοιάζει περισσότερο με τον πραγματικό κώδικα που παράγει ένας μεταγλωττιστής

Παραδείγματα Κώδικα MIPS

- Προσοχή στη δημιουργία του χώρου δεδομένων
  - Καθολικές και στατικές μεταβλητές δεσμεύονται με τα ονόματά τους
  - Σταθερές συμβολοσειρές δεσμεύονται με ονόματα για να μπορούν να αναφέρονται
  - Τοπικές μεταβλητές μπαίνουν στη στοίβα και δε δεσμεύονται ονομαστικά
  - Θεωρούμε ότι η στοίβα είναι δεσμευμένη και ο $sp κατάλληλα αρχικοποιημένος

Παραδείγματα Κώδικα MIPS

- Όσο αφορά τις εντολές:
  - Γενικά προτιμάμε πραγματικές εντολές, εκτός από τις πιο συνηθισμένες li, la και move
  - Για είσοδο/έξοδο διατηρούμε τις συναρτήσεις του αρχικού κώδικα C
  - Για τερματισμό χρησιμοποιούμε την εντολή break

Παράδειγμα Απλού Κώδικα

- Έστω το απλό πρόγραμμα C:

```
#include <stdio.h>
int a, b=26, c=24; //variables
int main() {
   a = b + c;       //statement
   printf("%d",a); //output
}
```

Παράδειγμα Απλού Κώδικα

- Σε συμβολική γλώσσα MIPS:

```
        .data
b:      .word   26              #space and value for b
c:      .word   24              #space and value for c
a:      .space  4               #space for a
pr:     .asciiz "%d"            #printf format string
        .text
        .globl  main
main:   la      $t0,b           #address of b
        lw      $t1,0($t0)      #load value of b from memory
        la      $t0,c           #address of c
        lw      $t2,0($t0)      #load value of c from memory
        add     $t1,$t1,$t2     #add
        la      $t0,a           #address of a
        sw      $t1,0($t0)      #store result to a in memory
        la      $a0,pr          #load first argument ("%d")
        lw      $a1,0($t0)      #load second argument (a)
        jal     printf          #call function printf()
        break                   #terminate program
```

Παράδειγμα Απλού Κώδικα

- Παρατηρήσεις:
  - Η αρχικοποίηση των καθολικών μεταβλητών b και c γίνεται με απ' ευθείας τοποθέτηση της τιμής τους στο χώρο δεδομένων
  - Παρόλο που η στοίβα δε χρησιμοποιείται άμεσα από τον κώδικα της main(), έχει δεσμευτεί κανονικά από το σύστημα, ώστε να μπορεί να χρησιμοποιηθεί από τη συνάρτηση printf()

Παράδειγμα Σύνθετου Κώδικα

- Πρόγραμμα C:

```
#include <stdio.h>
int main() {
    int i;
    int sum = 0;
    for (i=0; i<=100; i++)
        sum = sum + i*i;
    printf("The sum from 0 to 100 is %d\n",sum);
}
```

Παράδειγμα Σύνθετου Κώδικα

- Συμβολική γλώσσα MIPS:

```
        .text
        .globl  main
main:   sw      $zero,12($sp)   #initialize sum to 0
        sw      $zero,8($sp)    #initialize i to 0
loop:   lw      $t6,8($sp)      #load i
        mul     $t7,$t6,$t6     #calculate i*i
        lw      $t8,12($sp)     #load sum
        add     $t9,$t8,$t7     #add result of i*i
        sw      $t9,12($sp)     #store sum
        addi    $t0,$t6,1       #increase i
        sw      $t0,8($sp)      #store i
        ble     $t0,100,loop    #check end of loop
        la      $a0,str         #first argument
        lw      $a1,12($sp)     #second argument
        jal     printf          #call function printf
        break                   #terminate program
        .data
str:    .asciiz "The sum from 0 to 100 is %d\n"
```

Παράδειγμα Σύνθετου Κώδικα

- Παρατηρήσεις:
  - Οι δύο πρώτες θέσεις στοίβας ($sp+0 και $sp+4) είναι δεσμευμένες για διεύθυνση επανόδου και αποτέλεσμα, αλλά εδώ δε χρησιμοποιούνται
  - Για λόγους συντομίας χρησιμοποιήθηκαν και οι ψευδοεντολές mul και ble, οι οποίες μπορούν εύκολα να αντικατασταθούν από τις αντίστοιχες πραγματικές εντολές

Βελτίωση Σύνθετου Κώδικα

- Μείωση πλήθους lw και sw:

```
        .text
        .globl  main
main:   move    $t8,$zero       #initialize sum to 0
        move    $t6,$zero       #initialize i to 0
loop:   mul     $t7,$t6,$t6     #calculate i*i
        add     $t8,$t8,$t7     #add result of i*i
        addi    $t6,$t6,1       #increase i
        ble     $t6,100,loop    #check end of loop
        la      $a0,str         #first argument
        move    $a1,$t8         #second argument
        jal     printf          #call function printf
        break                   #terminate program
        .data
str:    .asciiz "The sum from 0 to 100 is %d\n"
```

Βελτίωση Σύνθετου Κώδικα

- Αντικατάσταση πολ/σμού με προσθέσεις:

```
        .text
        .globl  main
main:   move    $t8,$zero       #initialize sum to 0
        move    $t6,zero        #initialize i to 0
        move    $t7,$zero       #initialize square of i to 0
loop:   add     $t0,$t6,$t6     #calculate i+i
        addi    $t0,$t0,1       #calculate i+i+1
        add     $t7,$t7,$t0     #calculate next square of i
        add     $t8,$t8,$t7     #add result of i*i
        addi    $t6,$t6,1       #increase i
        ble     $t6,100,loop    #check end of loop
        la      $a0,str         #first argument
        move    $a1,$t8         #second argument
        jal     printf          #call function printf
        break                   #terminate program
        .data
str:    .asciiz "The sum from 0 to 100 is %d\n"
```

Βελτίωση Σύνθετου Κώδικα

- Παρατηρήσεις:
  - Στην πρώτη βελτίωση υποθέτουμε ότι οι εντολές προσπέλασης μνήμης (lw και sw) είναι πιθανό να οδηγήσουν σε καθυστερήσεις στην εκτέλεση του κώδικα
  - Στη δεύτερη βελτίωση υποθέτουμε ότι ο πολλαπλασιασμός κοστίζει σε χρόνο περισσότερο από τρεις απλές εντολές

Πιο Πολύπλοκο Παράδειγμα

- Κώδικας C

```
#include <stdio.h>
int main(int argc, char* argv[]) {
   printf("The factorial of 10 is %d\n",fact(10));
}
int fact(int n) {
   if (n < 1)
      return 1;
   else
      return n*fact(n-1);
}
```

Πιο Πολύπλοκο Παράδειγμα

- Συμβολική γλώσσα (συνάρτηση main())

```
        .text
        .globl main
main:   addi    $sp,$sp,-16     #push 16-byte stack frame
        sw      $ra,0($sp)      #save return address
        li      $a0,10          #first argument
        jal     fact            #call fact()
        la      $a0,LC          #first argument
        move    $a1,$v0         #second argument
        jal     printf          #call printf()
        lw      $ra,0($sp)      #restore return address
        addi    $sp,$sp,16      #pop stack frame
        jr      $ra             #return
        .data
LC:     .asciiz       "The factorial of 10 is %d\n"
```

Πιο Πολύπλοκο Παράδειγμα

- Συμβολική γλώσσα (συνάρτηση fact())

```
        .text
        .globl fact
fact:   addi    $sp,$sp,-12     #push 12-byte stack frame
        sw      $ra,0($sp)      #save return address
        sw      $a0,8($sp)      #save argument n
        bgtz    $a0,L2          #branch to L2 if n>0
        li      $v0,1           #move 1 to $v0
        j       L1              #jump to L1
L2:     addi    $a0,$a0,-1      #set argument to n-1
        jal     fact            #recursive call to fact()
        lw      $t0,8($sp)      #restore argument n
        mul     $v0,$t0,$v0     #calculate n*fact(n-1)
L1:     lw      $ra,0($sp)      #restore return address
        addi    $sp,$sp,12      #pop stack frame
        jr      $ra             #return to caller
```

Πιο Πολύπλοκο Παράδειγμα

- Παρατηρήσεις:
  - Σε αντίθεση με τα προηγούμενα παραδείγματα, εδώ κάνουμε άμεσα χρήση της στοίβας, με αύξηση προς τα κάτω
  - Για την αναδρομή χρησιμοποιήθηκε κανονικά η πρώτη θέση στοίβας της συνάρτησης fact(), αλλά όχι η δεύτερη, μια που αντί αυτής χρησιμοποιήθηκε ο καταχωρητής $v0 για το αποτέλεσμά της