ΗΡΥ 211– Ψηφιακοί Υπολογιστές

Συμβολικές Εντολές MIPS, Ετικέτες, Οδηγίες Συμβολομεταφραστή

Γιάννης Παπαευσταθίου

Ψευδοεντολές Assembly MIPS

- Ψευδοεντολές είναι εντολές που μπορούν να συνθεθούν εύκολα με λίγες (1-2 συνήθως) πραγματικές εντολές. Οι ψευδοεντολές απλοποιούν την ζωή του προγραματιστή ή κάνουν το πρόγραμμα πιό εκφραστικό και ευκολονόητο.
- Παραδείγματα:
  - Ψευδοεντολή move, αντιγράφει τα ένα καταχωρητή σε ένα άλλο. Υλοποιείται από τον συμβολομεταφραστή με μία addu rd, rs, 0
  - Ψευδοεντολή li (load immediate), φόρτώνει ένα καταχωρητή με μια σταθερή μέχρι 32 bits: li $2, Immediate. Εάν η σταθερή χωράει σε 16 bits, η li μεταφράζεται από τον συμβολομεταφραστή σε ori. Εάν η σταθερή χρειάζεται πάνω από 16 bits, η li μεταφράζεται σε δύο εντολές: μια lui για τα περισσότερο σημαντικά 16 bits και μια ori για τα λιγότερο σημαντικά 16 bits.
- Όταν ο συμβολομεταφραστής χρειάζεται ένα προσωρινό καταχωρητή για να κρατήσει ενδιάμεσες τιμές για μιά ψευδοεντολή, χρησιμοποιεί τον καταχωρητή $1 ($at = Assembler Temporary)

Ψευδοεντολές: move, li (load immediate)

- move rd, rs Αντιγραφή καταχωρητή
  - Σύνταξη: move $16, $23
  - Functionality: RF[rd] = RF[rs]
  - Υλοποιείται με μια addui rd, rs, 0
- li rd, Constant Φόρτωση σταθερής
  - Σύνταξη: li rd, 1000000
  - Functionality: RF[rd] = Constant
  - Υλοποιείται με μια ori rd, Constant_{16} όταν η σταθερή χωράει σε 16 bits, ή εάν η σταθερή χρειάζεται παραπάνω από 16 bits με την ακολουθία (*παρατηρήστε την χρήση του $1*):

    lui $1, $\text{MostSignificant}_{16}(\text{Constant}_{32})$

    ori rd, $1, $\text{LeastSignificant}_{16}(\text{Constant}_{32})$

Ψευδοεντολές: la (Load Address)

- la rd, Constant(rs) Φόρτωση διεύθυνσης
  - Σύνταξη: la $11, 1000000($8)
  - Functionality: RF[rd] = Constant+RF[rs]
  - Υλοποιείται με μια addi rd, rs, Constant_{16} όταν η σταθερή χωράει σε 16 bits, ή εάν η σταθερή χρειάζεται παραπάνω από 16 bits με την ακολουθία:

    li $1, Constant_{32} # Συμβολική Εντολή!

    add rd , rs, $1
- la rd, Label Φόρτωση συμβολικής διεύθυνσης
  - Σύνταξη: la $11, MyArray
  - Functionality: RF[rd] = LabelValue
  - Υλοποιείται με τον ίδιο τρόπο όπως και η li rd, Constant_{32}

Ψευδοεντολές: li

- li Constant Φόρτωση σταθερής
  - Σύνταξη: li 1000000
  - Functionality
    - RF[rd] = Constant
  - Υλοποιείται με μια ori rd, $Constant_{16}$ όταν η σταθερή χωράει σε 16 bits, ή εάν η σταθερή χρειάζεται παραπάνω από 16 bits με την ακολουθία:

    lui rd, $MostSignificant_{16}(Constant_{32})$

    ori rd, rd, $LeastSignificant_{16}(Constant_{32})$

Ψευδοεντολές Διακλάδωσης

- Ο συμβολομεταφραστής για να κάνει τα προγράματα πιο εκφραστικά, πιο μικρά και να διευκολύνει τον προγραμματιστή προσφέρει μια πληθώρα ψευδοεντολών διακλάδωσης οι οποίες συντίθενται χρησιμοποιόντας τις πραγματικές εντολές διακλάδωσης και άλλες εντολές όπως η set-less-than (slt).
- b label Διακλάδωση χωρίς συνθήκη
  - Σύνταξη: b loop1
  - Functionality: PC = AddressOf(label)
  - Υλοποιείται με μια beq $0, $0, label

Ψευδοεντολές Διακλάδωσης

- bge rs, rt, label Διακλάδωση μεγαλύτερο ή ίσο μεταξύ καταχωρητών
  - Σύνταξη: bge $8, $7, loop1
  - Functionality:

    nextPC = PC + $SignExtend_{32}(\text{Offset}_{16} << 2)$

    if ((RF[rs] >= RF[rt]) {

      PC = nextPC

    } else /* condition is false, just execute next instruction: */

      PC = PC + 4
  - Υλοποιείται με την ακολουθία:

      slt $1, rs, rt

      beq $1, $0, label

Ψευδοεντολές Διακλάδωσης

- Με ανάλογο τρόπο ο συμβολομεταφραστής συνθέτει και τις ακόλουθες ψευδοεντολές:
  - Bgt (branch greater than), ble (branch less equal), blt (branch less than) οι οποίες συγκρίνουν δύο προσημασμένους καταχωρητές.
  - Bgeu (branch greater or equal unsigned), Bgtu (branch greater than unsigned), ble (branch less or equal unsigned), blt (branch less than unsigned) οι οποίες συγκρίνουν δύο μή-προσημασμένους καταχωρητές.

Ψευδοεντολές Διακλάδωσης

- Ο συμβολομεταφραστής όμως επιτρέπει και συγκρίσεις με σταθερές σε ψευδοεντολές διακλάδωσης:
  - Παράδειγμα: ble $8, 100, loop1
  - Εάν η σταθερή χωράει σε 16 bits η ψευδοεντολή αυτή υλοποιείται με την ακολουθία:

    slti $1, rs, Constant_{16}

    bne $1, $0, label
  - Αλλιώς η μεγάλη σταθερή φορτώνεται στον καταχωρητή $1 και ακολουθεί η σύγκριση καταχωρητών.
- Ομοίως ο συμβολομεταφραστής επιτρέπει την χρήση σταθερής σε όλες τις εντολές και ψευδοεντολές διακλάδωσης

Assembler Directives (Οδηγίες)

- Δίνει οδηγίες στον συμβολομεταφραστή για το πώς να ερμηνεύσει το κώδικα του χρήστη, αλλά ***δεν*** αντιστοιχούν σε εντολές του επεξεργαστή.

  Οι οδηγίες στον συμβολομεταφραστή MIPS ξεκινούν με μια τελεία «.». Για παράδειγμα, μια από τις κυριότερες χρήσεις οδηγιών είναι η τοποθέτηση δεδομένων στην μνήμη.

  Στους παρακάτω ορισμούς, ό,τι βρίσκεται σε αγκύλες («[» και «]») είναι προαιρετικό και μπορεί να παραλειφθεί

.text [addr] Ακολουθούν εντολές. Εάν έχει δοθεί η προαιρετική διεύθυνση, οι εντολές αρχιζουν να τοποθετούνται στην μνήμη από την διεύθυνση αυτή.

.data [addr] Ακολουθούν δεδομένα. Εάν έχει δοθεί η προαιρετική διεύθυνση, τα δεδομένα αρχιζουν να τοποθετούνται στην μνήμη από την διεύθυνση αυτή

Περιοχές Εντολών και Δεδομένων

Ο συμβολομεταφραστής διακρίνει δύο περιοχές μνήμης, την περιοχή του προγράματος (που ονομάζεται text), και την περιοχή δεδομένων (που ονομάζεται data). Όταν μεταφράζει ένα πρόγραμμα, ο συμβολομεταφραστής τοποθετεί δεδομένα στην περιοχή δεδομένων, και εντολές στην περιοχή προγράμματος.

Γιατί όμως να διαχωρίζουμε τις εντολές από τα δεδομένα; Είναι λογικό να περιμένουμε οτι κατά την εκτέλεση ενός προγράμματος τα δεδομένα αλλάζουν, ενώ το πρόγραμμα παραμένει σταθερό.

Επειδή όμως ένα λανθασμένο πρόγραμμα μπορεί να γράψει σε λανθασμένες διευθύνσεις, είναι πιθανό να «πανωγράψει» θέσεις μνήμης που αποθηκεύουν εντολές, με αποτέλεσμα η συμπεριφορά του προγράμματος να είναι απρόβλεπτη!

Ο διαχωρισμός προγράμματος και δεδομένων επιτρέπει στο λειτουργικό σύστημα να προστατεύει από εγγραφές την περιοχή του προγράμματος, ενώ την ίδια στιγμή να επιτρέπει τις εγγραφές στα δεδομένα.

Οδηγίες .text and .data

Οι οδηγίες **.text** και **.data** μπορούν να εμφανιστούν πολλές φορές σε ένα πρόγραμμα. Εάν δεν δοθεί συγκεκριμένη διεύθυνση στις οδηγίες αυτές, τα διάφορα κομμάτια τοποθετούνται σε συνεχόμενες διευθύνσεις. Οι παρακάτω δύο ακολουθίες κώδικα είναι ισοδύναμες:

.text add $9, $8, $7 … **.data** s1: .ascii “string1” **.text** ori ... beq ... **.data** s2: .ascii “string2”	**.text** add $9, $8, $7 ... ori ... beq ... **.data** s1: .ascii “string1” s2: .ascii “string2”

Οδηγίες για Δέσμευση Μνήμης

.word w1 [,…, wn]	Δεσμεύει χώρο για n λέξεις όπου και αποθηκεύει τις τιμές w1, …, wn.
.half h1 [,…, hn]	Δεσμεύει χώρο για n ποσότητες 16-bit (short) όπου και αποθηκεύει τις τιμές h1, …, hn.
.byte b1 [,…, bn]	Δεσμεύει χώρο για n bytes όπου και αποθηκεύει τις τιμές b1, …, bn.
. ascii str	Δεσμεύει χώρο για strlen(str) bytes όπου και αποθηκεύει την κωδικοποιημένη σε Ascii συμβολοσειρά .
.asciiz str	Όπως και η .ascii αλλά με ένα μηδέν μετά τον τελευταίο χαρακτήρα (για γλώσσες όπως «C»)
.space n	Δεσμεύει χώρο για n bytes (χωρίς αρχικοποίηση)
.align n	Ευθυγραμμίζει την διεύθυνση του επόμενου δεδομένου σε πολ/σιο του 2^n. (.align 2 => διεύθυνση πολ/σιο του 4).

.data examples

```
.data
prompt:     .asciiz  “Hello World\n”
msg:        .asciiz  “The answer is”
x:           .space 4
y:           .word   4
str:        .space1 00
```

Καθολικές Ετικέτες

- Η οδηγία .globl χρησιμοποιείται όταν μια ετικέτα χρειάζεται να είναι καθολική, δηλαδή να είναι ορατή και σε άλλα αρχεία. Τα μεγάλα προγράμματα γράφονται σε πολλά αρχεία για πολλούς λόγους:
  - Δομημένος προγραματισμός, που ζητάει ετερογενή θέματα να βρισκονται ξεχωριστά
  - Επαναχρησιμοποίηση κώδικα (συναρτήσεων, κ.α.) από άλλα προγράμματα
  - Χρήση βιβλιοθηκών (είτε του χρήστη/προγραμματιστή, είτε του συστήματος, π.χ. printf)
  - Διατήρηση του μεγέθους του κώδικα κάθε αρχείου περιορισμένου ώστε ο προγραμματιστής να μην «χάνεται»

Πίνακας Συμβόλων

- Ο συμβολομεταφραστής διατηρεί ένα πίνακα συμβόλων, όπου αποθηκέυει πληροφορίες για όλες τις ετικέτες που συναντάει.
  - Ο πίνακας έχει από δύο πεδία ανα καταχώρηση: Όνομα ετικέτας και διεύθυνση στην οποία αντιστοιχεί.
  - Ο πίνακας χωρίζεται σε δύο κομμάτια: τοπικές και καθολικές ετικέτες. Οι τοπικές χρησιμοποιούνται μόνο από τον συμβολομεταφραστή για εσωτερική χρήση (παραγωγή κώδικα). Οι καθολικές είναι ορατές και σε άλλα αρχεία ώστε να μπορεί να γίνει η «διασύνδεση» των διάφορων κομματιών του κώδικα.
  - Στον spim μπορείτε να τυπώσετε τον πίνακα καθολικών συμβόλων (menu Simulator->Display Symbol Table)

Παράδειγμα

```
#sample example 'add two numbers'

.text           #       text    section
.globl  main    #       call    main by SPIM

main:   la      $t0,    value   #load   address 'value' into $t0

   lw   $t1,    0($t0)  #       load    word 0(value) into $t1
   lw   $t2,    4($t0)  #       load    word 4(value) into $t2
   add $t3, $t1, $t2    #       add two numbers into $t3
   sw $t3, 8($t0)       #       store word $t3 into 8($t0)

.data
value:  .word 10, 20, 0         ##data section data for addition
```

Συμβολικά Ονόματα Καταχωρητών

Οι καταχωρητές του MIPS έχουν και συμβολικά ονόματα που υποδηλώνουν την χρήση τους στις συμβάσεις χρήσης που προτείνει η MIPS και τις οποίες υιοθετούν ***όλοι*** οι μεταφραστές και συμβολομεταφραστές. Τα ονόματα αυτά είναι:

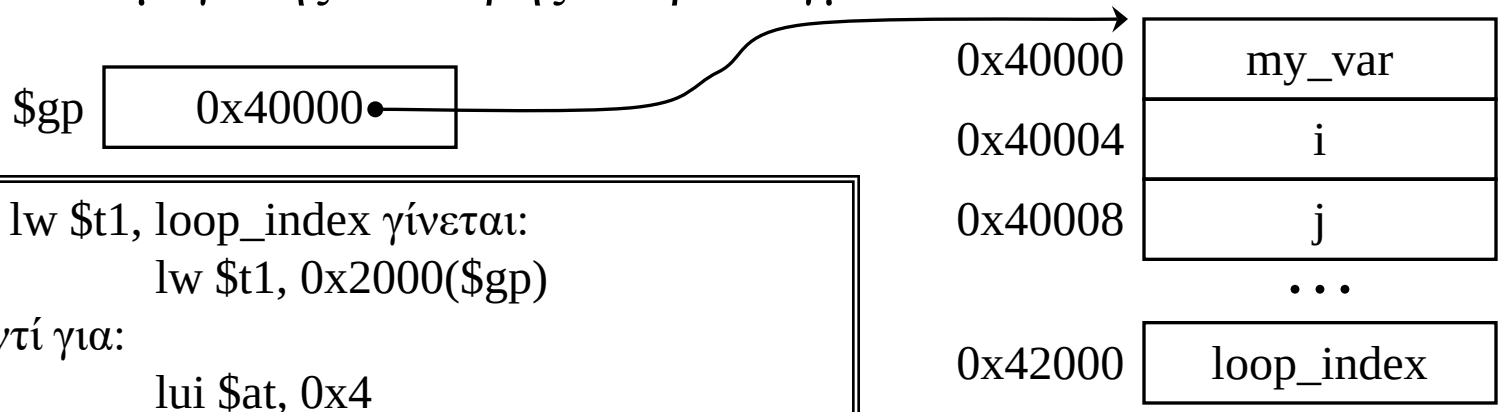
$zero	$0	zero
$at	$1	assembler temporary register
$v0, $v1	$2, $3	expression evaluation and function result
$a0..$a3	$4..$7	procedure arguments
$t0..$t7	$8..$15	temporary
$s0..$s7	$16..$23	saved temporaries
$t8, $t9	$24, $25	temporary
$k0, $k1	$26, $27	used by OS Kernel
$gp	$28	global pointer
$sp	$29	stack pointer
$fp	$30	frame pointer
$ra	$31	return address (from subroutine call)

Συμβάσεις Χρήσης Καταχωρητών

- Οι συμβάσεις χρήσης καταχωρητών απαιτούν τα εξής:
  - Ο καταχωρητής $at χρησιμοποιείται από τον συμβολομεταφραστή για την σύνθεση ψευδοεντολών.
  - Οι καταχωρητές $v0, $v1 χρησιμοποιούνται για την επιστροφή τιμών από συναρτήσεις (functions)
  - Οι καταχωρητές $a0..$a3 χρησιμοποιούνται για το πέρασμα παραμέτρων σε διαδικασίες και συναρτήσεις (procedures & functions)
  - Οι καταχωρητές $sp και $fp χρησιμοποιούνται ώς stack pointer και frame pointer αντίστοιχα.
  - Ο καταχωρητής $ra (return address) χρησιμοποιείται για την αποθήκευση της διεύθυνσης επιστροφής από υπορουτίνα

Συμβάσεις Χρήσης Καταχωρητών

- Ο καταχωρητής $gp (global pointer) χρησιμοποιείται για την γρήγορη φόρτωση καθολικών δεδομένων.
  - Αρχικοποιείται ώστε να δείχνει στη περιοχή των καθολικών δεδομένων, και η φόρτωση μιάς λέξης γίνεται με μία εντολή lw με offset από τον $gp αντί να χρειαστεί η φόρτωση μιας μεγάλης σταθερής. Παράδειγμα:



Η lw $t1, loop_index γίνεται:

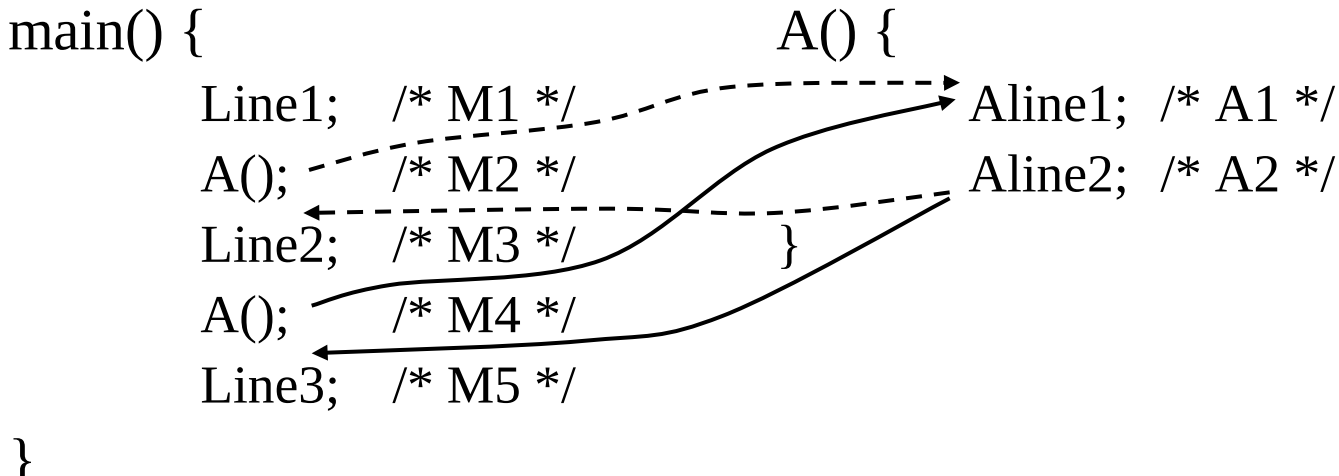
```
lw $t1, 0x2000($gp)
```

αντί για:

```
lui $at, 0x4
lw, $t1, 0x2000($at)
```

Κλήσεις Διαδικασιών/Συναρτήσεων

- Η μόνη διαφορά διαδικασίας και συνάρτησης είναι οτι οι συναρτήσεις επιστρέφουν μια τιμή στο σημείο που τις κάλεσε, ενώ οι διαδικασίες όχι. Σε επίπεδο assembly λοιπόν, η γενική δομή λοιπόν διαδικασιών και συναρτήσεων είναι παρόμοια.
- Παράδειγμα:



Παράδειγμα Κλήσης Υπορουτίνας

- Η σωστή σειρά εκτέλεσης των «εντολών» της C είναι η εξής:

```
main Line1          /* M1 */
A Line 1            /* A1 */
A Line 2            /* A2 */
main Line2          /* M2 */
A Line 1            /* A1 */
A Line 2            /* A2 */
main Line3          /* M5 */
```

- Παρατηρήστε ότι στην κλήση της διαδικασίας Α η διεύθυνση της επόμενης εντολής είναι η ίδια για τις δύο κλήσεις (η πρώτη εντολή της Α, η γραμμή /* A1 */). Η διεύθυνση επιστροφής όμως από την διαδικασίας Α αλλάζει, ανάλογα με την κλήση! Για την πρώτη κλήση η επόμενη εντολή είναι η main Line2 /* M2 */, ενώ για την δεύτερη κλήση η επόμενη εντολή είναι η main Line3 /* M5 */.

Παράδειγμα σε Assembly

- Η κλήση διαδικασίας/συνάρτησης γίνεται με μια ***jal*** (jump and link) και η επιστροφή με μια ***jr*** (jump register)

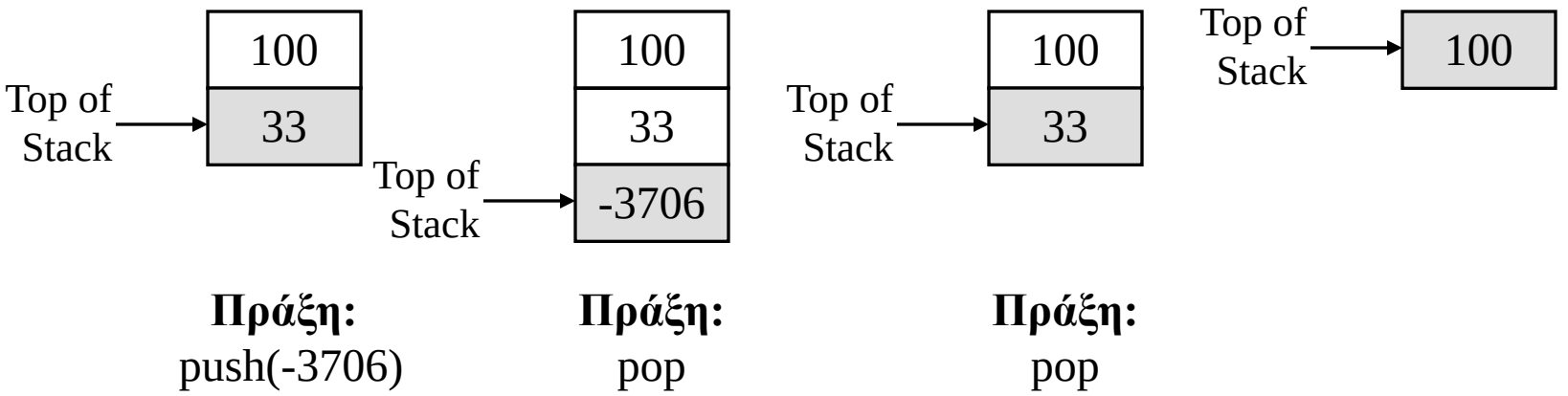
```
main:
    li    $4, 1000          # M1
    jal   A                 # M2 A(), διεύθυνση επιστροφής (M3) -> $31
    li    $4, 2000          # M3
    jal   A                 # M4 A(), διεύθυνση επιστροφής (M5) -> $31
    ori   $5, $2, $9        # M5
…
A:
    addi  $11, $12, $4      # A1
    lw    $2, 0($11)        # A2
    ori   $2, $2, $13       # A3
    jr    $31               # Επιστροφή στην επόμενη εντολή από την κλήση
```

Πολλαπλά Επίπεδα Κλήσης Διαδικασιών

- Τι γίνεται όταν έχουμε πολλαπλά επίπεδα κλήσης
  - Η διαδικασία A καλεί την B
    - Η διαδικασία B καλεί την C
      - διαδικασία C καλεί την D, κλπ.
- Τι γίνεται όταν έχουμε αναδρομική κλήση;
  - Η A_0 καλεί την A_1, η οποία καλεί την A_2, ... μέχρι A_ν.
- Πολλαπλές διευθύνσεις επιστροφής:
  - Μια για την επιστροφή από την D, μια για την επιστροφή από την C, μια για την επιστροφή από την B, κλπ.
  - Ομοίως και για την αναδρομή ($A_\nu \rightarrow A_{\nu-1}$ ... $A_0 \rightarrow$ main)
- Πώς «θυμάμαι» όλες τις διευθύνσεις επιστροφής?
  - Χρήση στοίβας (stack)!

Στοίβα

- Δομή δεδομένων που συμπεριφέρεται με τρόπο LIFO: Last-In, First-Out. Παράδειγμα χρήσης στοίβας: στοίβα δίσκων σε εστιατόρια.
  - Πράξεις: ***Push*** (εισαγωγή στοιχείου στην στοίβα), ***Pop*** (αφαίρεση στοιχείου από την στοίβα)
  - Μόνο η «κορυφή» (top) της στοίβας είναι προσβάσιμη



Υλοποίηση Στοίβας σε Assembly

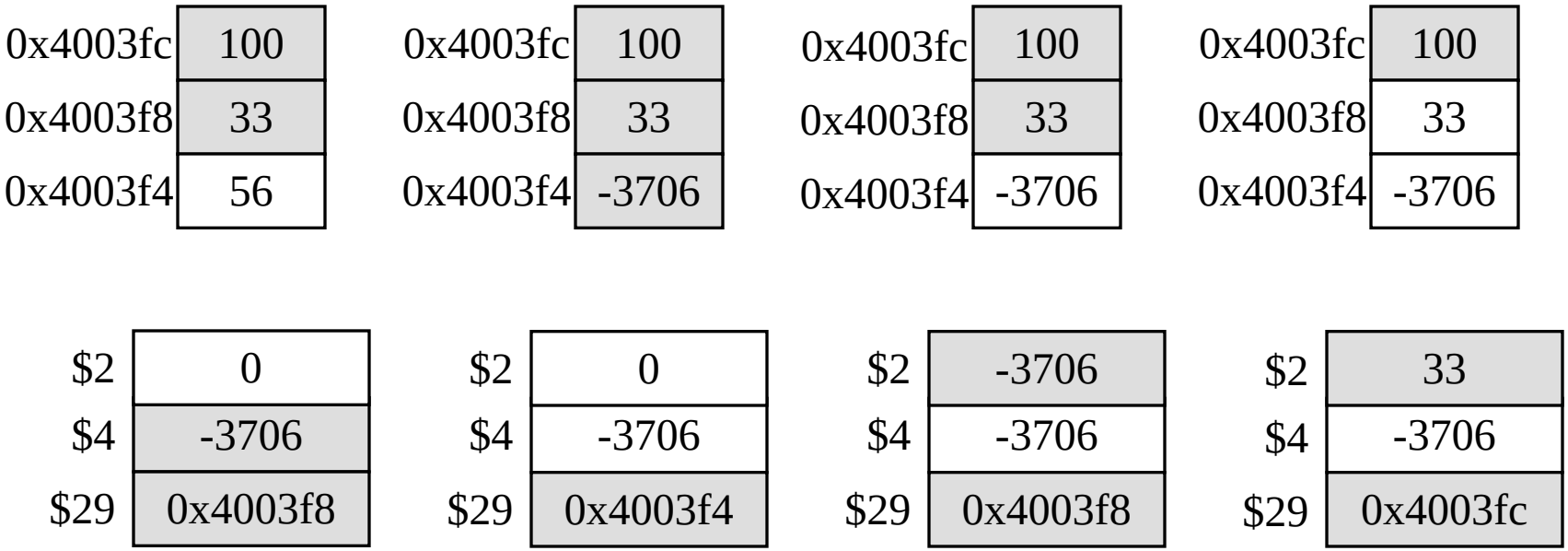
- Χρησιμοποιούμε ένα δείκτη στοίβας (stack pointer) $29
- Παραδοσιακά (και στον MIPS) οι στοίβες ξεκινούν από μεγάλες και τα στοιχεία προσθέτονται προς μικρότερες διευθύνσεις
- Ο $sp δείχνει στην *τελευταία γεμάτη* θέση της στοίβας
- Push(x) (θεωρούμε οτι η τιμή X βρίσκεται στον $4)

```
addi $29, $29, -4      # δημιουργία κενής θέσης στην στοίβα
sw   $4, 0($29)        # γέμισμα της θέσης με το X
```

- Pop() (θεωρούμε ότι η τιμή X επιστρέφεται στον $2)

```
lw   $2, 0($29)        # επιστροφή «κορυφής» στοίβας
addi $29, $29, 4       # αφαίρεση της  κενής θέσης
```

Παράδειγμα Στοίβας



Πράξη:
push(-3706)

Πράξη:
pop
$2 ← -3706

Πράξη:
Pop
$2 ← 33

Συμβάσεις Κλήσης Υπορουτινών

- Παράμετροι συνάρτησης/υπορουτίνας
  - Scalar τιμές (ακέραιοι, bytes, χαρακτήρες) στους καταχωρητές $a0-$a3.
  - Οι μή-scalar τιμές (συμβολοσειρές, πίνακες, structures, κ.λ.π), όπως και οι υπόλοιπες παράμετροι αν είναι πάνω από 4, περνιούνται στην στοίβα
- Τιμή επιστροφής συνάρτησης:
  - Στους καταχωρητές $v0, $v1 (οι γλώσσες προγραμματισμού συνήθως ορίζουν μόνο μία τιμή => $v0)

Συμβάσεις Χρήσης Καταχωρητών

- Οι καταχωρητές $t0..$t9 ονομάζονται «προσωρινοί» (temporary) και ενδύκνειται για την αποθήκευση προσωρινών τιμών οι οποίες δεν απαιτείται να διατηρηθούν και μετά από μια κλήση διαδικασίας ή συνάρτησης. Εάν η διαδικασία που καλείται χρησιμοποιήσει αυτούς τους καταχωρητές, οι παλαιές τους τιμές χάνονται.
- Οι καταχωρητές $s0..$s7 ονομάζονται saved και ενδείκνυται να χρησιμοποιούνται για αποθήκευση τιμών που διατηρούνται για περισσότερο χρόνο και μπορούν να διατηρούν την τιμή τους και μετά από μία κλήση διαδικασίας ή συνάρτησης.
- Οι καταχωρητές $t0..$t9 είναι τύπου «caller-save», ενώ οι $s0..$s7 είναι τύπου «callee-save».

Συμβάσεις Κλήσης Υπορουτινών #2

- Οι συναρτήσεις πρέπει να διατηρήσουν αναλλοίωτες τις τιμές των καταχωρητών τύπου callee-save ($s0-$s7).
  - Εάν μεταβάλλουν κάποιους καταχωρητές τύπου callee-save, αυτοί πρέπει να σωθούν στην στοίβα κατά την έναρξη της συνάρτησης (στον «πρόλογο»), και να επαναφερθούν πριν την επιστροφή (στον «επίλογο»), ώστε μετά την επιστροφή, οι καταχωρητές αυτοί να έχουν τις παλιές τιμές τους.
- Ακόμα, εάν μια συνάρτηση Α χρησιμοποιεί ένα καταχωρητή τύπου caller-save για να διατηρήσει μια τιμή πέρα από μια κλήση υπορουτίνας Β, τότε πρέπει να τον σώσει στην στοίβα πρίν την κλήση της Β και να τον επαναφέρει πριν τον ξαναχρησιμοποιήσει.
  - Αυτό γιατί η υπορουτίνα Β σύμφωνα με τις συμβάσεις χρήσης καταχωρητών του MIPS, έχει δικαίωμα να πανωγράψει τους καταχωρητές τύπου caller-save.
- Παρατήρηση: caller-save καταχωρητές εκτός των $t0-$t9 είναι ***και*** οι $v0, $v1, $a0-$a3, $ra (πανωγράφεται από την jal),

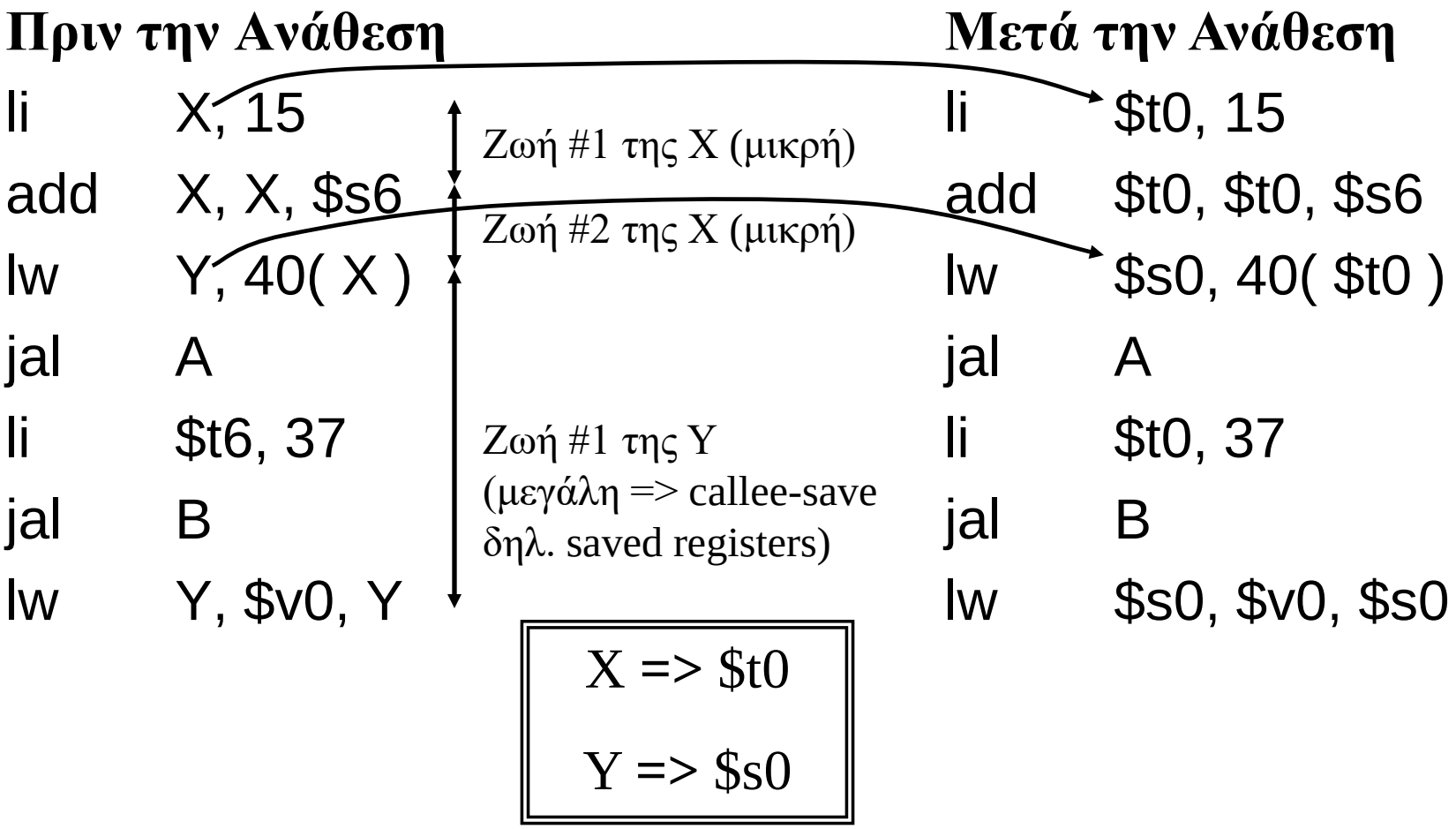
Διαδικασία Κλήσης Υπορουτίνας

- Η υπορουτίνα Α που καλεί (caller):
  - Σώζει στην στοίβα όσους καταχωρητές «caller-save» έχει χρησιμοποιήσει και χρειάζεται την τιμή τους μετά την κλήση
  - Περνάει τις παραμέτρους στους καταχωρητές $a0-$a3 ή/και στην στοίβα (ανάλογα με το πλήθος και το είδος των παραμέτρων)
  - Jal function
  - Επαναφέρει τους καταχωρητές «caller-save» πού είχε σώσει
- Η υπορουτίνα Β που καλείται αποτελείται από τρία κομμάτια:
  - **Πρόλογος:** κομμάτι κώδικα που κάνει διαδικαστικούς υπολογισμούς. Σώζει στην στοίβα όσους καταχωρητές «callee-save» θα χρησιμοποιηθούν στην υπορουτίνα αυτή.
  - **Κυρίως Σώμα:** ο κώδικας της συνάρτησης/υπορουτίνας
  - **Επίλογος:** Επαναφέρει τους καταχωρητές «callee-save» πού είχε σώσει
  - Ο καταχωρήτής $ra είναι τύπου caller-save, οπότε σώζεται στο πρόλογο και επαναφέρεται στον επίλογο μόνο αν η Β καλεί κάποια συνάρτηση

Πως διαλέγω ποιόν καταχωρητή να χρησιμοποιήσω για τις μεταβλητές μου;

- Χωρίζουμε δύο είδη μεταβλητών: με *σύντομη* ζωή, και με *μακρόχρονη* ζωή. Η ζωή μιάς μεταβλητής ορίζεται ως η απόσταση από την εγγραφή στην μεταβλητή αυτή, μέχρι την τελευταία ανάγνωση αυτής της τιμής. Αν η τιμή δεν διαβάζεται (πριν πανωγραφτεί), η μεταβλητή είναι «νεκρή» και δεν χρειάζεται να καταναλώνει καταχωρητή.
- Σε μια συνάρτηση όμως μας ενδιαφέρει κυρίως αν η «ζωή» της μεταβλητής ***επεκτείνεται και πέρα από κλήση υπορουτίνας.***
  - Αν **ναι**, τότε ή ζωή της μεταβλητής είναι **«μεγάλη»**, και προσπαθούμε να την βάλουμε σε καταχωρητή τύπου callee-save.
  - Αν **όχι**, τότε ή ζωή της μεταβλητής είναι **«μικρή»**, και προσπαθούμε να την βάλουμε σε καταχωρητή τύπου caller-save.
- Ετσι, αν έχω πολλές συναρτήσεις με πολλές μεταβλητές μικρής ζωής που δεν επεκτείνονται πέρα από κλήση υπορουτίνας, τις τοποθετώ όλες (όσες χωράνε) σε καταχωρητές τύπου caller-save. Αφού η ζωές των μεταβλητών δεν εκτείνονται μετά από τις κλήσεις υπορουτινών, δεν χρειάζεται να σωθούν στην στοίβα!

Παράδειγμα Ανάθεσης Μεταβλητών σε Καταχωρητές



Αναδρομική συνάρτηση factorial() #1

```
factorial:
#  Prologos: apothikeysh dieythynshs epistrofhs, kai tou orismatos
#  to opoio xreiazetai kai meta thn anadromikh klhsh ths factorial
        addui $sp, $sp, -8
        sw $ra, 4($sp) # save return address
        sw $a0, 0($sp) # save argument (n)
# Telos prologou

        bgtz $a0, greater_than_zero
# If we reach here, the argument is zero, and fact(0) = 1.
        li $v0, 1
        j epilogue
```

Αναδρομική συνάρτηση factorial() #2

```
greater_than_zero:
	sub $a0, $a0, 1 # call factorial(n-1)
	jal factorial
	# factorial(n-1) is in $v0

	lw $v1, 0($sp) # restore n from stack
	mul $v0, $v0, $v1 # multiply n * factorial(n-1)

	# we are done, $v0 has the correct value
epilogue:
	lw $ra, 4($sp) # Epanafora ths dieythynshs epistrofhs
	addiu $sp, $sp, 8 # Epanafora tou $sp sthn timh prin thn
klhsh
	jr $ra
```

SPIM System Calls

- Ο Συμβολομεταφραστής και προσομοιωτής SPIM προσφέρει ένα μικρό σύνολο από ευκολίες με την μορφή «κλήσεων συστήματος» (system calls).
- Spim system calls: εκτύπωση/ανάγνωση συμβολοσειρών, ακεραίων, και χαρακτήρων στην οθόνη/από το πληκτρολόγιο, δυναμική δέσμευση μνήμης (malloc) και τερματισμός προγράμματος.
- Χρήση:
  - Κωδικός κλήσης συστήματος στον καταχωρητή $v0 ($2)
  - Παράμετρος (αν υπάρχει), (π.χ. ο ακέραιος ή η συβολοσειρά που πρέπει να τυπωθεί) στον καταχωρητή $a0 ($4)
  - Εντολή syscall
  - Πιθανό αποτέλεσμα στον καταχωρητή $v0

Κωδικοί Κλήσεων Συστήματος SPIM

Ο κωδικός των κλήσεων συστήματος που μπαίνει στον καταχωρητή $v0 μπορεί να πάρει τις εξής τιμές:

1. εκτύπωση ακεραίου
2. εκτύπωση αριθμού κινητής υποδοαστολής (float)
3. εκτύπωση κινητής υποδοαστολής (double)
4. εκτύπωση συμβολοσειράς
5. ανάγνωση ακεραίου
6. ανάγνωση αριθμού κινητής υποδοαστολής (float)
7. ανάγνωση κινητής υποδοαστολής (double)
8. ανάγνωση συμβολοσειράς
9. δέσμευση μνήμης
10. τερματισμός προγράμματος

Παραδείγματα SPIM System Calls

- Παράδειγμα: εκτύπωση συμβολοσειράς

```
li    $v0, 4          # print string
la    $a0, str        # str is the strings label
syscall
```

- Παράδειγμα: εκτύπωση ακέραιου

```
li    $v0, 1          # print integer
la    $a0, 1500       # integer to print
syscall
```

- Παράδειγμα: ανάγνωση ακέραιου από πληκτρολόγιο

```
li    $v0, 5          # read integer
syscall
move $a0, $v0         # read value is in $a0
```

Μονοδιάστατοι Πίνακες

Υπολογισμός διεύθυνσης:

Δηλώνεται ένας πίνακας array[100]; Η αρχική διεύθυνση του πίνακα (δηλαδή η διεύθυνση του πρώτου στοιχείου) βρίσκεται στην διεύθυνση I.A. (initial address). Ποιά είναι η διεύθυνση του στοιχείου array[i];

Απάντηση: address = I.A. + (i * sizeof(array element))

Για ένα πίνακα ακεραίων, η διεύθυνση του array[I] είναι:

address = I.A. + i * 4

Για ένα πίνακα από χαρακτήρες (char) είναι:

address = I.A. + i * 1

Διδιάστατοι Πίνακες

Υπολογισμός διεύθυνσης:

Δηλώνεται ένας πίνακας array[nrows,ncolumns]; Η αρχική διεύθυνση του πίνακα (δηλαδή η διεύθυνση του στοιχείου array[0,0]) είναι Ι.Α. (initial address). Ποιά είναι η διεύθυνση του στοιχείου array[j,i];

Απάντηση:

address = I.A. + (j * ncolumns * sizeof(array element))
+ (i * sizeof(array element))

Ο πρώτος όρος (j * …) αντιστοιχεί στο μέγεθος σε bytes μιάς ολόκληρης γραμμής του πίνακα, ενώ ο δεύτερος (i * …) είναι ίδιος με την περίπτωση μονοδιάστατου πίνακα.

Για τον πίνακα int array[50, 100], η διεύθυνση του array[j, i] είναι:

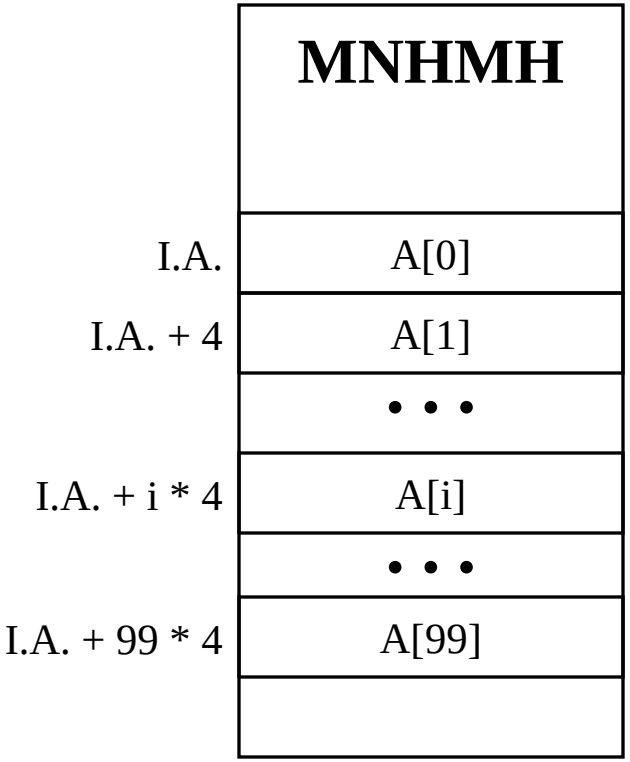
address = I.A. + j * 100 * 4 + i * 4

Για ένα πίνακα από χαρακτήρες (char) με τις ίδιες διαστάσεις:

address = I.A. + j * 100 * 1 + i * 1

Πίνακες και Αποθήκευση στην Μνήμη

Μονοδιάστατος Πίνακας
int array[100]



Διδιάστατος Πίνακας
int array[50,100]