

ΗΥ 232  
Οργάνωση και  
Σχεδίαση Υπολογιστών

Διάλεξη 4  
Συναρτήσεις-Διαδικασίες

Νίκος Μπέλλας  
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

# Διαδικασίες (procedures)

Γνωστές και σαν υπορουτίνες (subroutines)

- Τι πρέπει να προσέξουμε στις διαδικασίες (ή συναρτήσεις)
  1. Η εμφάνιση των μεταβλητών που ορίζονται μέσα στις διαδικασίες (automatic variables) περιορίζεται στο σώμα των διαδικασιών αυτών
  2. Οι μεταβλητές αυτές (automatic variables), “κρύβουν” τον ορισμό των καθολικών μεταβλητών (global variables) με τις οποίες έχουν το ίδιο όνομα.

```
int x;
void func(int a) {
    int x;
    int y;
    ....
}
```

Καθολική μεταβλητή: δεν μπορεί όμως να προσπελασθεί μέσα στην func() επειδή έχει το όνομα x.

Παράμετρος διαδικασίας

Αυτόματες μεταβλητές: έγκυρες μόνο μέσα στην func()

Οργάνωση και Σχεδίαση Υπολογιστών (HY232)

# Διαδικασίες (procedures)

Γνωστές και σαν υπορουτίνες (subroutines)

- Η διαδικασία γράφεται σαν ένα ανεξάρτητο πρόγραμμα που ορίζει και επεξεργάζεται τις δικές της - εσωτερικές - μεταβλητές ή δεδομένα, κάτι που συνεπάγεται ότι:
  - Δεσμεύει δικό της τμήμα δεδομένων στον αντίστοιχο χώρο της μνήμης
  - Έχει τη δυνατότητα να χρησιμοποιήσει τους ίδιους καταχωρητές με το κυρίως πρόγραμμα για την επεξεργασία των δεδομένων της
- Η χρήση κοινών καταχωρητών σημαίνει ότι εκείνοι που χρειάζονται από το κυρίως πρόγραμμα θα πρέπει να αποθηκευτούν σε κάποιο χώρο μνήμης κατά την κλήση της διαδικασίας, και αργότερα να ανακτηθούν από το χώρο αυτό με την επιστροφή από τη διαδικασία
  - Η δομή μνήμης που χρησιμοποιείται για την αποθήκευση και ανάκτηση των περιεχομένων καταχωρητών είναι η *στοίβα (stack)*

# Υποστήριξη διαδικασιών (procedures) στον MIPS

- Στον MIPS, ένα πρόγραμμα πρέπει να ακολουθήσει τα παρακάτω βήματα όταν καλεί μια διαδικασία (*caller calls a callee*) :
  1. Ο καλών (caller) τοποθετεί τις παράμετρους της διεργασίας (callee) κάπου που να μπορεί τις ανακτήσει η διεργασία αυτή.
    - Καταχωρητές ορίσματος \$a0-\$a3 (argument registers) (\$4-\$7)
    - Για περισσότερα από 4 ορίσματα, ο καλών(caller) χρησιμοποιεί την μνήμη στοίβας (stack)
  2. Ο καλών μεταβιβάζει τον έλεγχο στη διαδικασία
    - Χρησιμοποιώντας μια εντολή αλλαγής ροής χωρίς συνθήκη (εντολή άλματος και σύνδεσης, *jal*)

# Υποστήριξη διαδικασιών(procedures) στον MIPS

3. Η διαδικασία (callee) καταλαμβάνει μνήμη στην στοίβα (stack) που θα χρειαστεί κατά την εκτέλεση της.
  - Αυτόματες μεταβλητές (automatic variables) της διαδικασίας (callee) τοποθετούνται στην στοίβα.
  - Επίσης, όσοι καταχωρητές διαγράφονται μέσα στην διαδικασία, ενώ η παλιά τους τιμή χρειάζεται από τον καλούντα αποθηκεύονται στην στοίβα από τον καλούντα.
4. Η διαδικασία εκτελείται
5. Η διαδικασία τοποθετεί το αποτέλεσμα σε καταχωρητή (για να το προσπελάσει η καλούσα διαδικασία)
  - Καταχωρητές τιμής \$v0-\$v1 (value registers) (\$2-\$3)
6. Η διαδικασία επιστρέφει στο σημείο κλήσης
  - Χρησιμοποιώντας μια εντολή αλλαγής ροής χωρίς συνθήκη (jump register, *jr*)

# Χρήση καταχωρητών

- Κατά την εκτέλεση μιας διαδικασίας, πολλοί από τους καταχωρητές του MIPS, έχουν ειδική σημασία:
- $\$a0 - \$a3$ : καταχωρητές παραμέτρων ( $\$4 - \$7$ )
- $\$v0, \$v1$ : καταχωρητές τιμής (ή αποτελεσμάτων) ( $\$2 - \$3$ )
- $\$ra$ : καταχωρητής διεύθυνσης επιστροφής ( $\$31$ )
- $\$t0 - \$t9$ : προσωρινοί καταχωρητές ( $\$8-\$15, \$24-\$25$ )
  - Μπορεί να διαγραφούν από τον καλούμενο (callee)
- $\$s0 - \$s7$ : αποθηκευμένοι καταχωρητές ( $\$16- \$23$ )
  - Ο καλούμενος πρέπει να τους αποθηκεύει στην αρχή και να τους επαναφέρει στο τέλος της κλήσης

# Χρήση καταχωρητών

- \$gr: καθολικός δείκτης (global pointer) στατικών δεδομένων (\$28)
- \$sp: δείκτης στοίβας (stack pointer) (\$29)
- \$fp: δείκτης πλαισίου (30)
- pc: μετρητής προγράμματος (program counter)

# Εντολές κλήσης διαδικασιών

- Κλήση διαδικασίας:  
Εντολή άλματος και σύνδεσης (jump and link, *jal, jalr*)

– `jal L1` →  $\$ra = PC+4 ; PC = L1$

`0x1000: jal L1`

`0x1004 . . . .`

- Η διεύθυνση της επόμενης εντολής τοποθετείται στον  $\$ra$  ( $\$ra = 0x1004$ )
- Μεταπήδηση στη διεύθυνση προορισμού `L1`



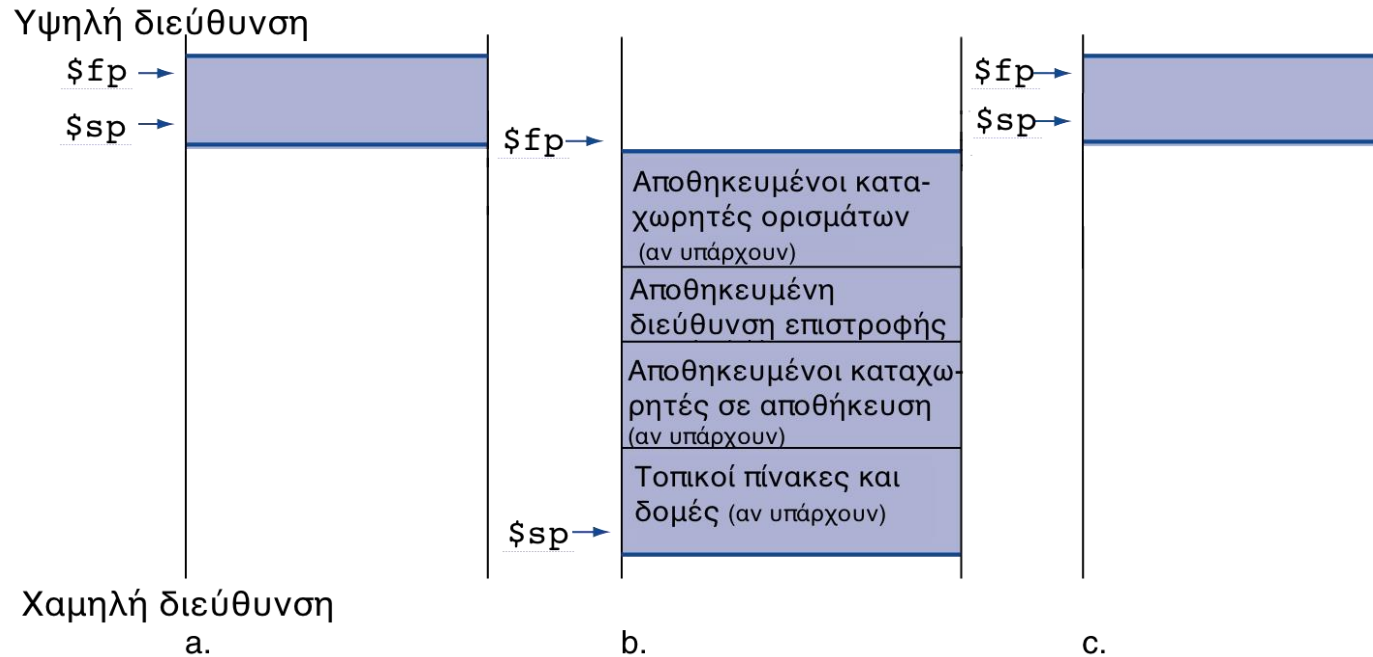
# Εντολές κλήσης διαδικασιών

- Επιστροφή διαδικασίας:  
Εντολή jump register

`jr $ra` →  $PC = \$ra$

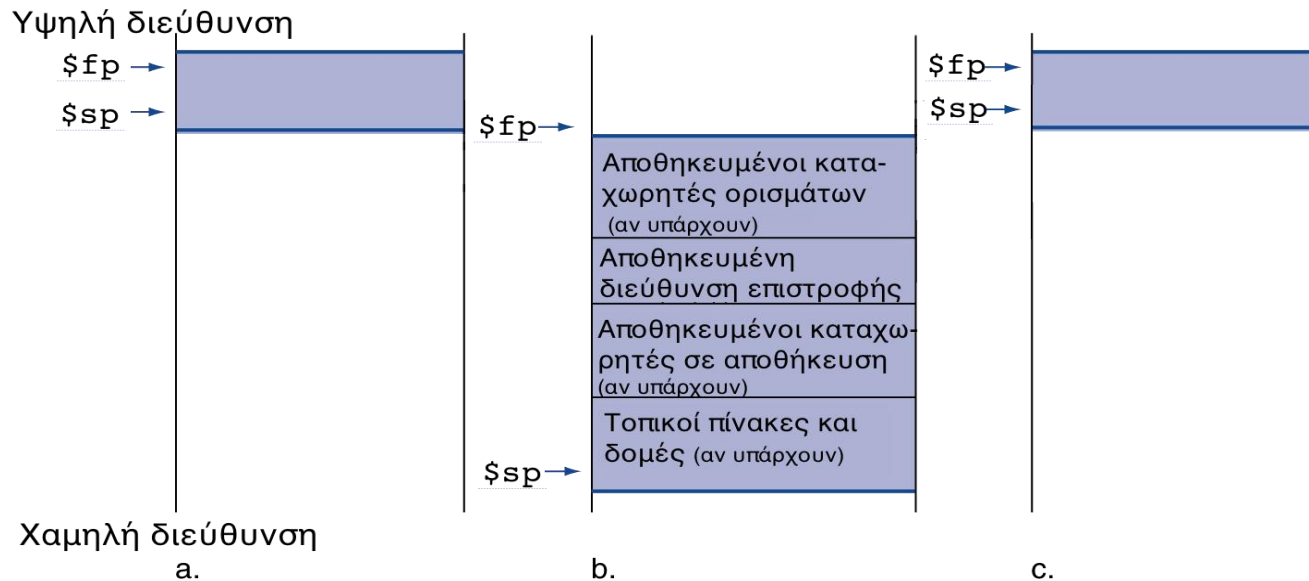
- Αντιγράφει το `$ra` (31) στο μετρητή προγράμματος ( $pc = \$ra$ )
- Μπορεί επίσης να χρησιμοποιηθεί για υπολογισμένα άλματα
  - π.χ. για εντολές case/switch που υλοποιούνται με πίνακα αλμάτων

# Στοιίβα (Stack)



- Η στοίβα είναι προσπελάσιμη ως Last In First Out (LIFO)
- Η στοίβα του MIPS «μεγαλώνει» από τις υψηλότερες προς τις χαμηλότερες διευθύνσεις.
- Ένας δείκτης στοίβας (stack pointer) χρησιμοποιείται για να δείξει το τελευταίο έγκυρο (valid) στοιχείο στην στοίβα
  - Στον MIPS, ο δείκτης στοίβας είναι ο  $\$sp$  ( $\$29$ )

# Στοίβα (Stack)



- Κατά την κλήση μιάς διαδικασίας η στοίβα μπορεί να περιέχει τα παρακάτω πεδία (από υψηλότερες προς χαμηλότερες διευθύνσεις):
  - τους επιπλέον καταχωρητές ορισμάτων στην περίπτωση που η διαδικασία έχει πάνω από 4 ορίσματα. Ο καλών (caller) πρέπει να τους τοποθετήσει στην στοίβα
  - Την διεύθυνση επιστροφής από την διαδικασία (δηλ. τον  $\$ra$ ). Αυτό χρειάζεται μόνο εάν η διαδικασία δεν είναι φύλλο, δηλ. θα καλέσει μια άλλη διαδικασία.
  - Καταχωρητές που θα διαγραφούν στην διαδικασία αυτή, αλλά η παλιά τους τιμή χρειάζεται στον καλούντα (caller)
  - Τοπικές μεταβλητές, πίνακες, λίστες, (automatic variables only)

# Σύμβαση διαφύλαξης καταχωρητών

- Οι γενικής χρήσης καταχωρητές του MIPS ( $\$s0$ - $\$s7$  και  $\$t0$ - $\$t9$ ) έχουν διαφορετική αντιμετώπιση ως προς τη διαφύλαξή τους στη στοίβα και την ανάκτησή τους από αυτή κατά την κλήση μιας διαδικασίας
  - Οι καταχωρητές  $\$s0$ - $\$s7$  (saved registers) θεωρούνται *ex orisμού* ότι χρειάζονται στο κυρίως πρόγραμμα και μετά το πέρας της διαδικασίας  
=> η διαδικασία είναι εκείνη που θα πρέπει να σώσει στη στοίβα αρχικά, και να ανακτήσει στο τέλος της όσους από τους καταχωρητές αυτούς *μεταβάλλει* (καταχωρητές τύπου *callee-save*)
  - Οι καταχωρητές  $\$t0$ - $\$t9$  (temporary registers) θεωρούνται *ex orisμού* ότι δεν χρειάζονται στο κυρίως πρόγραμμα μετά από την κλήση και το πέρας της διαδικασίας  
=> το *κυρίως πρόγραμμα* θα πρέπει να αναλάβει την πρωτοβουλία να σώσει στη στοίβα πριν από την κλήση, και να ανακτήσει κατά την επιστροφή όσους από τους καταχωρητές αυτούς *χρειάζεται* (καταχωρητές τύπου *caller-save*)
- Ο καταχωρητής  $\$ra$ , και πιθανόν οι  $\$a0$ - $\$a3$ , πρέπει να αποθηκεύονται στην στοίβα εάν η διαδικασία καλεί κάποια άλλη διαδικασία

# Κατανομή μνήμης

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

`$sp` → 7fff fffc

`$gr` → 1000 8000  
          1000 0000

`pc` → 0040 0000

0



# Παράδειγμα: Διαδικασία-φύλλο (Leaf procedure)

```
int leaf_ex (int g, int h, int i, int j) {  
    int f;  
    f = (g + h) - (i + j);  
    return f;  
}
```

## – Διαδικασίες που δεν καλούν άλλες

- Ο caller πριν καλέσει την *leaf\_ex* θέτει:  $\$a0=g$ ,  $\$a1=h$ ,  $\$a2=i$ ,  $\$a3=j$
- Ο  $\$s0$  χρειάζεται στον caller και άρα πρέπει να σωθεί στην στοίβα από τον callee
- Αποτέλεσμα γράφεται στον  $\$v0$

# Διαδικασίες-φύλλα

```
int leaf_ex (int g, int h, int i, int j) {  
    int f;  
    f = (g + h) - (i + j);  
    return f;  
}
```

- κώδικας MIPS:

leaf_ex:			
addi	\$sp,	\$sp,	-4
sw	\$s0,	0(\$sp)	
add	\$t0,	\$a0,	\$a1
add	\$t1,	\$a2,	\$a3
sub	\$s0,	\$t0,	\$t1
move	\$v0,	\$s0	
lw	\$s0,	0(\$sp)	
addi	\$sp,	\$sp,	4
jr	\$ra		

Αποθήκευση \$s0  
στη στοίβα

Σώμα διαδικασίας

Αποτέλεσμα

Επαναφορά \$s0

Επιστροφή

# Παράδειγμα strcpy

- Κώδικας C

```
void strcpy (char x[], char y[])  
{  
    int i;  
    i = 0;  
    while ((x[i]=y[i])!='\0')  
        i += 1;  
}
```

- Διευθύνσεις των x, y στους \$a0, \$a1
- Το i στον \$s0



# Παράδειγμα (assembly)

- Κώδικας MIPS:

strcpy:		
addi	\$sp, \$sp, -4	# ρύθμιση στοίβας για 1 τιμή
sw	\$s0, 0(\$sp)	# αποθήκευση \$s0
li	\$s0, \$zero	# i = 0
L1:	add \$t1, \$s0, \$a1	# διευθ. του y[i] στον \$t1
lb	\$t2, 0(\$t1)	# \$t2 = y[i]
add	\$t3, \$s0, \$a0	# διευθ. του x[i] στον \$t3
sb	\$t2, 0(\$t3)	# x[i] = y[i]
beq	\$t2, \$zero, L2	# εξοδος αν y[i] == 0
addi	\$s0, \$s0, 1	# i = i + 1
j	L1	# επόμενη επανάληψη
L2:	lw \$s0, 0(\$sp)	# επαναφορά \$s0
addi	\$sp, \$sp, 4	# αφαίρ. τιμής από στοίβα
jr	\$ra	# και επιστροφή

# Ένθετες διαδικασίες

- Διαδικασίες που καλούν άλλες διαδικασίες
  - Ή και τον εαυτόν τους (recursive procedures)
- Ο καλών πρέπει να αποθηκεύσει στη στοίβα:
  - Τη διεύθυνση επιστροφής (return address)
  - Όσες προσωρινές τιμές και παραμέτρους θα χρειαστούν μετά το τέλος της κλήσης.
- Μετά την ολοκλήρωση της κλήσης, ο καλών επαναφέρει από τη στοίβα τις αποθηκευμένες τιμές

# Bubble Sort

**Input:** Πίνακας από  $N$  στοιχεία

A[0]	A[1]	A[2]	A[3]	A[4]
3	4	10	5	3

**Output:** Ταξινομημένος πίνακας από  $N$  στοιχεία

A[0]	A[1]	A[2]	A[3]	A[4]
3	3	4	5	10

**Βασική ιδέα:**

Τοποθέτησε το εκάστοτε μικρότερο στοιχείο στην αρχή του πίνακα

# Bubble Sort

A[0]	A[1]	A[2]	A[3]	A[4]
3	4	10	5	3

A[0]	A[1]	A[2]	A[3]	A[4]
3	4	10	3	5

A[0]	A[1]	A[2]	A[3]	A[4]
3	4	3	10	5

A[0]	A[1]	A[2]	A[3]	A[4]
3	3	4	10	5

A[0]	A[1]	A[2]	A[3]	A[4]
3	3	4	10	5

A[0]	A[1]	A[2]	A[3]	A[4]
3	3	4	10	5

A[0]	A[1]	A[2]	A[3]	A[4]
3	3	4	5	10

A[0]	A[1]	A[2]	A[3]	A[4]
3	3	4	5	10

A[0]	A[1]	A[2]	A[3]	A[4]
3	3	4	5	10

# Παράδειγμα : Bubble Sort

Ο παρακάτω αλγόριθμος , τοποθετεί το μικρότερο στοιχείο του υπο-πίνακα κάθε φορά στην αρχή.

```
void sort(int v[], int n) {
    int i, j;
    for (i = 0; i < n; i++) {
        for (j = n-2; j >= i && v[j] > v[j+1]; j--) {
            swap(v, j);
        }
    }
}
```

– \$a0 = &v[0], \$a1 = n

# Η διαδικασία *swap*

```
void swap(int v[],int k) {  
    int temp;  
  
    temp = v[k];  
    v[k] = v[k+1];  
    v[k+1] = temp;  
}
```

```
swap: sll $t1, $a1, 2    # $t1 = k * 4  
      add $t1, $a0, $t1 # $t1 = v+(k*4)  
                          # (διεύθυνση του v[k])  
      lw $t0, 0($t1)    # $t0 (temp) = v[k]  
      lw $t2, 4($t1)    # $t2 = v[k+1]  
      sw $t2, 0($t1)    # v[k] = $t2 (v[k+1])  
      sw $t0, 4($t1)    # v[k+1] = $t0 (temp)  
      jr $ra           # επιστροφή στον καλώντα
```

# Το σώμα της sort

\$a0 = &v[0], \$a1 = n, \$s0=i, \$s1=j

```
void sort(int v[], int n) {
    int i, j;
    for (i = 0; i < n; i++) {
        for (j = n-2; j >= i && v[j] > v[j+1]; j--)
            swap(v, j);
    }
}
```

```

move $s2, $a0      # αποθ. τον $a0 στον $s2
move $s3, $a1      # αποθ. $a1 στον $s3
move $s0, $zero    # i = 0
for1tst:
bge $s0, $s3, exit1 # go to exit1 if $s0 ≥ $s3 (i ≥ n)
addi $s1, $a1, -2  # j = n-2
for2tst:
blt $s1, $s0, exit2 # go to exit2 if $s1 < $s0 (j < i)
sll $t1, $s1, 2     # $t1 = j * 4
add $t2, $s2, $t1   # $t2 = v + (j * 4)
lw $t3, 0($t2)      # $t3 = v[j]
lw $t4, 4($t2)      # $t4 = v[j + 1]
bge $t4, $t3, exit2 # go to exit2 if $t4 ≥ $t3
move $a0, $s2       # 1η παράμ. της swap είναι v
move $a1, $s1       # 2η παράμ. της swap είναι j
jal swap           # κλήση διαδικασίας swap
addi $s1, $s1, -1  # j -= 1
j for2tst         # προς έλεγχο εσωτερικού βρόχου
exit2: addi $s0, $s0, 1 # i += 1
j for1tst         # jump to test of outer loop
```

Move  
params

Outer loop

Inner loop

Pass  
params  
& call

Inner loop

Outer loop

# Ολόκληρη η διαδικασία

sort:	addi \$sp,\$sp, -20	# κάνε χώρο στη στοίβα για 5 κατ/τές
	sw \$ra, 16(\$sp)	# αποθ. \$ra στη στοίβα
	sw \$s3,12(\$sp)	# αποθ. \$s3 στη στοίβα
	sw \$s2, 8(\$sp)	# αποθ. \$s2 στη στοίβα
	sw \$s1, 4(\$sp)	# αποθ. \$s1 στη στοίβα
	sw \$s0, 0(\$sp)	# αποθ. \$s0 στη στοίβα
	...	# σώμα διαδικασίας
	...	
exit1:	lw \$s0, 0(\$sp)	# επαναφορά \$s0 από στοίβα
	lw \$s1, 4(\$sp)	# επαναφορά \$s1 από στοίβα
	lw \$s2, 8(\$sp)	# επαναφορά \$s2 από στοίβα
	lw \$s3,12(\$sp)	# επαναφορά \$s3 από στοίβα
	lw \$ra,16(\$sp)	# επαναφορά \$ra από στοίβα
	addi \$sp,\$sp, 20	# επαναφορά stack pointer
	jr \$ra	# επιστροφή στον καλώντα