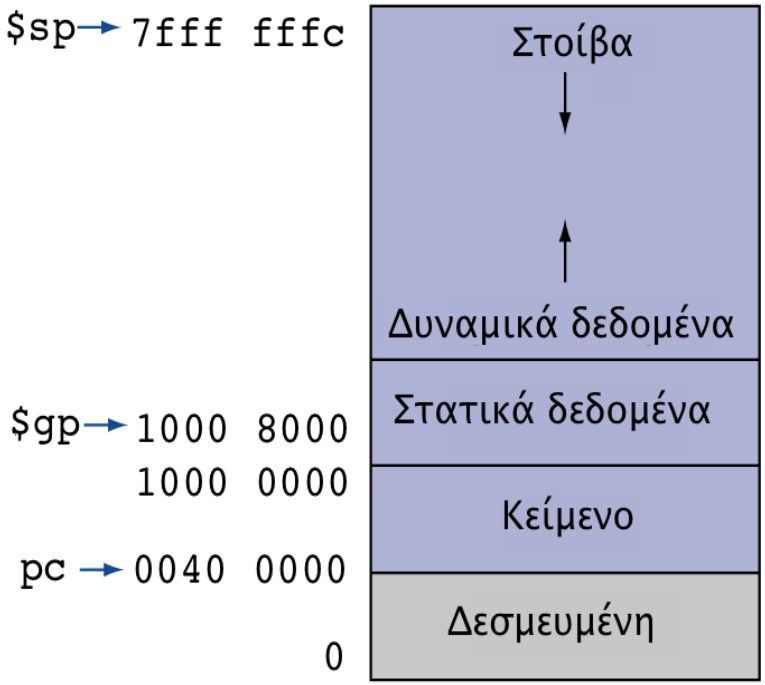
ΗΥ 134
Εισαγωγή στην Οργάνωση και στον Σχεδιασμό Υπολογιστών Ι

Διάλεξη 5
Διαδικασίες ΙΙ

Νίκος Μπέλλας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

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: 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
    add  $s0, $zero, $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 Ν στοιχεία

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

Output: Ταξινομημένος πίνακας από Ν 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	3	4	10	5

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	3	4	5 ↔	10

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	5	10

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	10	5

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

```
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, $s0=i, $s1=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
```

Πίνακες ή Δείκτες?

- Η διευθυνσιοδότηση με αριθμοδείκτες απαιτεί:
  - Πολ/σμό του αριθμοδείκτη με το μέγεθος των στοιχείων
  - Πρόσθεση στη διεύθυνση της αρχής του πίνακα.
- Οι δείκτες αντιστοιχούν απευθείας σε διευθύνσεις στη μνήμη
  - Αποφεύγουν την πολυπλοκότητα της χρήσης αριθμοδεικτών

Παράδειγμα

Συνάρτηση καθαρισμού (*clear*) μιάς περιοχής μνήμης

```
clear1(int array[], int size) {
  int i;
  for (i = 0; i < size; i += 1)
    array[i] = 0;
}
```

```
clear2(int *array, int size) {
  int *p;
  for (p = &array[0]; p < &array[size];
       p = p + 1)
    *p = 0;
}
```

```
      move $t0,$zero   # i = 0
loop1: sll $t1,$t0,2   # $t1 = i * 4
      add $t2,$a0,$t1  # $t2 =
                       #   &array[i]
      sw $zero, 0($t2) # array[i] = 0
      addi $t0,$t0,1   # i = i + 1
      slt $t3,$t0,$a1  # $t3 =
                       #   (i < size)
      bne $t3,$zero,loop1 # if (…)
                          # goto loop1
```

```
      move $t0,$a0     # p = & array[0]
      sll $t1,$a1,2    # $t1 = size * 4
      add $t2,$a0,$t1  # $t2 =
                       #   &array[size]
loop2: sw $zero,0($t0) # Memory[p] = 0
      addi $t0,$t0,4   # p = p + 4
      slt $t3,$t0,$t2  # $t3 =
                       #(p<&array[size])
      bne $t3,$zero,loop2 # if (…)
                          # goto loop2
```

Αριθμός εντολών που εκτελούνται, *clear1*: 6N+1, *clear2*: 4N+3

Count Number of Zeros

Να γραφεί πρόγραμμα σε MIPS assembly το να επιστρέφει τον αριθμό των 1 στην δυαδική αναπαράσταση ενός απρόσημου ακέραιου αριθμού.
Για παράδειγμα, εάν η είσοδος στο πρόγραμμα είναι ο δεκαδικός αριθμός 201, πού έχει δυαδική αναπαράσταση $(11001001)_2$, το πρόγραμμα σας θα πρέπει να τυπώνει το 4.
Η είσοδος θα πρέπει να γίνεται από την κονσόλα και ο αριθμός να τυπώνεται στην κονσόλα.
Το πρόγραμμα σας δεν θα πρέπει να χρησιμοποιεί επιπλέον μνήμη.

Count Number of Zeros

```
.text
 .globl main

 main:
   li $v0, 5    # read integer number
   syscall
   move $a0, $v0
   jal count_ones     # call the function

   move $a0, $v0   # print output
   li $v0, 1
   syscall

   li $v0, 10   # and exit
   syscall

#
```

```
# function to count number of ones of
# input number N (in $a0)
count_ones:
        addi $sp, $sp, -4   # stack logistics
        sw $s0, 0($sp)
        move $s0, $a0
        li $v0, 0
L1:
        beqz $s0, Exit
        andi $t1, $s0, 0x1     # check the last bit
        beqz $t1, L2            # if 0, jump tou L2
        addi $v0, $v0, 1     # add 1 to the counter
L2:   srl  $s0, $s0, 1        # right shift by 1
        j     L1

Exit:                          # stack logistics
        lw $s0, 0($sp)
        addi $sp, $sp, 4
        jr $ra
```

Byte swapping

Έστω ότι ο καταχωρητής $t0 περιέχει αρχικά μια 32-bit τιμή 0XAABBCCDD. Να γράψετε κώδικα MIPS assembly που να αντιστρέφει την τιμή του $t0 σε 0XDDCCBBAA.
Ο κώδικας δεν θα πρέπει να περιέχει πάνω από 15 εντολές assembly και δεν θα πρέπει να χρησιμοποιήσετε την καθόλου την μνήμη στο πρόγραμμα σας (δηλ. εντολές τύπου *lw* και *sw*).

Byte swapping

```
# We need to re-arrange the bytes in the word.

li  $t1, 0
andi $t2, $t0, 0xFF000000   # $t0 has AA BB CC DD
srl  $t1, $t2, 24           # and final result
andi $t2, $t0, 0x00FF0000
srl  $t2, $t2, 8
or  $t1, $t2, $t1
andi $t2, $t0, 0x0000FF00
sll  $t2, $t2, 8
or  $t1, $t2, $t1
andi $t2, $t0, 0x000000FF
sll  $t2, $t2, 24
or  $t0, $t2, $t1
```

Reverse A String

Η συνάρτηση *void reverse(char *s)*
αντιστρέφει την σειρά των χαρακτήρων του null-terminated string s. Για παράδειγμα, εάν πριν την κλήση της συνάρτησης, s="abcde" , τότε μετά την κλήση της συνάρτησης, s="edcba".

Reverse A String

```
reverse:
            addi $sp, $sp, -12
            sw $ra, 8($sp)
            sw $s0, 4($sp)
            sw $s1, 0($sp)
             move $s0, $a0
            move $s1, $a0
# compute length of string in $v0
             jal strlen
             addi $t0, $v0, -1
       # $s1 points at the end of the string
             add  $s1, $s1, $t0
L3:          bge  $s0, $s1, Exit
             lb  $t0, 0($s0)      #swap
             lb  $t1, 0($s1)
             sb  $t0, 0($s1)
             sb  $t1, 0($s0)
             addi $s0, $s0, 1
             addi $s1, $s1, -1
             j L3
```

```
Exit:
                          move $v0, $a0
                          lw $ra, 8($sp)
                          lw $s0, 4($sp)
                          lw $s1, 0($sp)
                          addi  $sp, $sp, 12
                          jr  $ra

# Compute the length of a null-terminated
string (the null char is not included)
strlen:
                          addi $sp, $sp, -4
                          sw $s1, 0($sp)
                          move $s1, $a0
                          move $t0, $zero
L1:                       lb    $t1, 0($s1)
                          beq   $t1, $zero, L2
                          addi   $s1, $s1, 1
                          addi   $t0, $t0, 1
                          j     L1
L2:                       move  $v0, $t0
                          lw    $s1, 0($sp)
                          addi   $sp, $sp, 4
                          jr     $ra
```