Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Βήματα στην εκτέλεση μιας διαδικασίας (procedure)

1. Τοποθέτηση παραμέτρων
2. Μεταβίβαση ελέγχου στη διαδικασία
3. Λήψη πόρων αποθήκευσης
4. Εκτέλεση επιθυμητής εργασίας
5. Τοποθέτηση αποτελέσματος σε θέση προσβάσιμη από καλούν πρόγραμμα (caller)
6. Επιστροφή ελέγχου στο σημείο εκκίνησης

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Κλήση διεργασιών: Σύμβαση κατανομής καταχωρητών

- $a0-$a3: τέσσερις καταχωρητές ορίσματος (argument regs)
- $v0-$v1: δύο καταχωρητές τιμής (value regs)
- $ra: καταχωρητής διεύθυνσης επιστροφής (return address reg)

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Άλμα και σύνδεση (jump and link)

PC: Μετρητής προγράμματος (program counter)

Κρατάει τη διεύθυνση της εντολής που εκτελείται

```
jal ΔιεύθυνσηΔιαδικασίας
```

$ra ← PC+4

PC ← ΔιεύθυνσηΔιαδικασίας

Για να επιστρέχουμε καλούμε

```
jr $ra
```

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Άλμα και σύνδεση (jump and link) - Σύνοψη

1. Ο caller τοποθετεί τιμές παραμέτρων στους `$a0-$a3`
2. Καλεί `jal X` για να μεταπηδήσει στη διαδικασία X (callee)
3. Εκτελεί υπολογισμούς
4. Τοποθετεί αποτελέσματα στους `$v0 - $v1`
5. Επιστρέφει με `jr $ra`

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

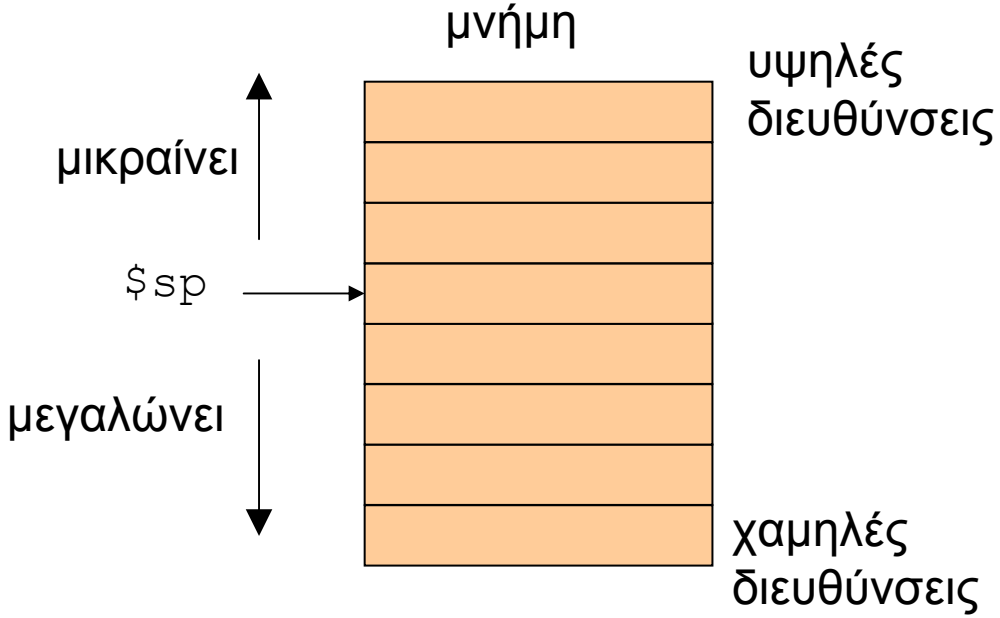
Χρήση πολλών καταχωρητών σε διαδικασίες;

Τι γίνεται αν έχουμε >4 ορίσματα ή/και >2 αποτελέσματα;

Χρησιμοποιούμε στοίβα (stack)

Last-In-First-Out

push, pop



Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Παράδειγμα

```
int leaf_example(int g,
    int h, int i, int j)
{
    int f;

    f = (g+h) - (i+j);
    return f;
}
```

g, h, i, j αντιστοιχίζονται στους `$a0, $a1, $a2, $a3`

f αντιστ. `$s0`

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Παράδειγμα

```
int leaf_example(int g,
    int h, int i, int
    j)
{
    int f;

    f = (g+h) - (i+j);
    return f;
}
```

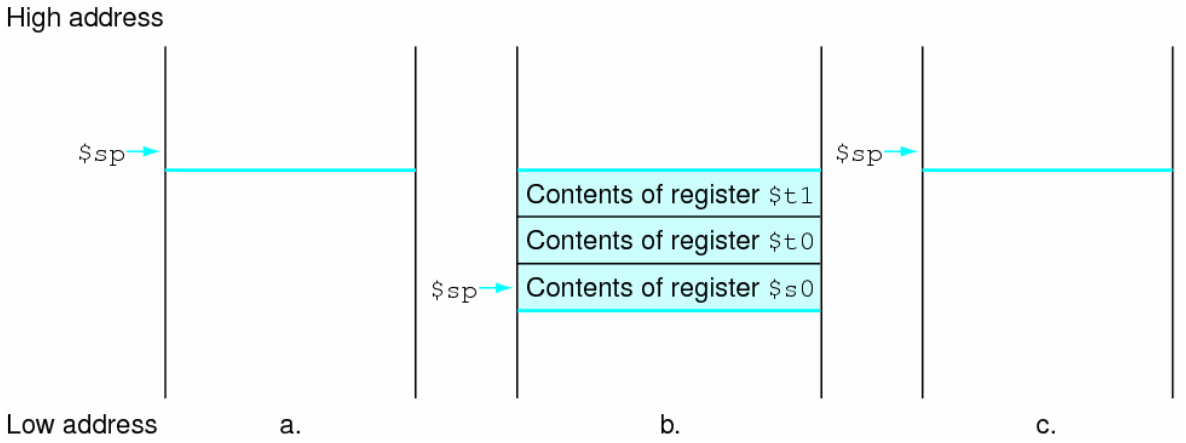
Σκονάκι

g:$a0, h:$a1, i:$a2,
j:$a3, f:$s0

```
leaf_example:
 addi $sp,$sp,-12
 sw $t1, 8($sp) #σώζουμε $t1
 sw $t0, 4($sp) #σώζουμε $t0
 sw $s0, 0($sp) #σώζουμε $s0
 add $t0, $a0, $a1
 add $t1, $a2, $a3  # f=(g+h)-(i+j);
 sub $s0, $t0, $t1
 add $v0, $s0, $zero
 lw $s0, 0($sp) #επαν. $s0
 lw $t0, 4($sp) #επαν. $t0
 lw $t1, 8($sp) #επαν. $t1
 addi $sp, $sp, 12
 jr $ra # πίσω στον caller
```

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

```
leaf_example:
 addi $sp,$sp,-12
 sw $t1, 8($sp) #σώζουμε $t1
 sw $t0, 4($sp) #σώζουμε $t0
 sw $s0, 0($sp) #σώζουμε $s0
 add $t0, $a0, $a1
 add $t1, $a2, $a3 }# f=(g+h)-(i+j);
 sub $s0, $t0, $t1
 add $v0, $s0, $zero
 lw $s0, 0($sp) #επαν. $s0
 lw $t0, 4($sp) #επαν. $t0
 lw $t1, 8($sp) #επαν. $t1
 addi $sp, $sp, 12
 jr $ra # πίσω στον caller
```



Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

```
leaf_example:
 addi $sp,$sp,-12
 sw $t1, 8($sp) #σώζουμε $t1
 sw $t0, 4($sp) #σώζουμε $t0
 sw $s0, 0($sp) #σώζουμε $s0
 add $t0, $a0, $a1
 add $t1, $a2, $a3
 sub $s0, $t0, $t1
 add $v0, $s0, $zero
 lw $s0, 0($sp) #επαν. $s0
 lw $t0, 4($sp) #επαν. $t0
 lw $t1, 8($sp) #επαν. $t1
 addi $sp, $sp, 12
 jr $ra # πίσω στον caller
```

Σύμβαση: ΔΕ ΣΩΖΟΥΜΕ `$t0-$t9`

Έτσι έχουμε:

```
leaf_example:
 addi $sp,$sp,-4
 sw $s0, 0($sp) #σώζουμε $s0
 add $t0, $a0, $a1
 add $t1, $a2, $a3
 sub $s0, $t0, $t1
 add $v0, $s0, $zero
 lw $s0, 0($sp) #επαν. $s0
 addi $sp, $sp, 4
 jr $ra # πίσω στον caller
```

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Ένθετες διαδικασίες

leaf procedures (διαδικασίες φύλλα): δεν καλούν άλλες διαδικασίες

Δεν είναι όλες οι διαδικασίες, διαδικασίες φύλλα (καλά θα ήταν ☺)

Πολλές διαδικασίες καλούν άλλες διαδικασίες, ακόμα και τον εαυτό τους!

π.χ.

```
int foo(int a){
    ...
    f=bar(a*2);
    ...
}
```

ή

```
int foo(int a){
    ...
    f=foo(a-1);
    ...
}
```

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Αναδρομική διαδικασία παραγοντικού

```
int fact (int n){
   if(n<1) return(1);
      else return(n*fact(n-1));
}
```

```
fact:
 addi $sp,$sp,-8
 sw $ra,4($sp) #διεύθ. επιστροφής
 sw $a0,0($sp) #όρισμα n
 slti $t0,$a0,1
 beq $t0,$zero,L1
 addi $v0,$zero,1
 addi $sp,$sp,8
 jr $ra
L1:
 addi $a0,$a0,-1
 jal fact
 lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

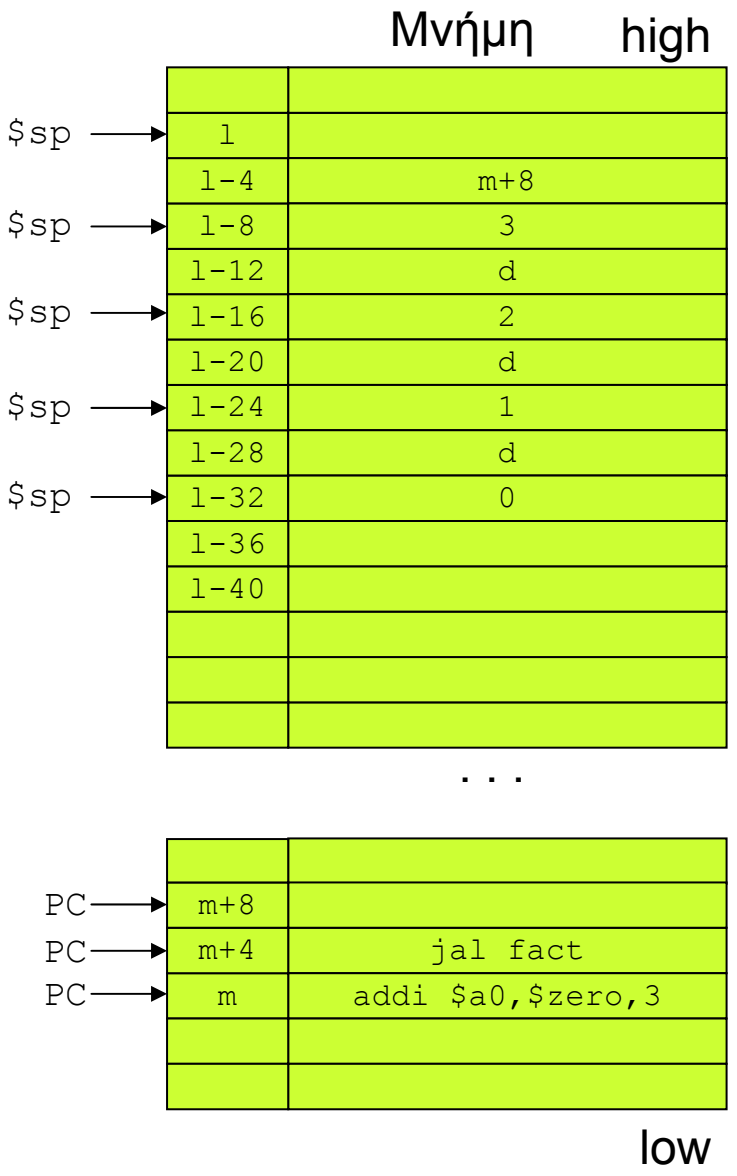
Εκτέλεση:

Έστω ότι καλούμε `fact(3)` και περιμένουμε να μας επιστρέψει το αποτέλεσμα στον καταχωρητή `$v0`

```
fact:
  addi $sp,$sp,-8 ← PC
  sw $ra,4($sp) #διεύθ. επιστροφής ← PC
  sw $a0,0($sp) #όρισμα n ← PC
  slti $t0,$a0,1 ← PC
  beq $t0,$zero,L1 ← PC
  addi $v0,$zero,1 ← PC
  addi $sp,$sp,8 ← PC
  jr $ra ← PC
L1:
  addi $a0,$a0,-1 ← PC
  jal fact ← PC
d: lw $a0,0($sp) ← PC
  lw $ra,4($sp) ← PC
  addi $sp,$sp,8 ← PC
  mul $v0,$a0,$v0 ← PC
  jr $ra ← PC
```

finish

```
PC:  m+8 PC
$sp: l-8 4
$ra: m+8
$v0: 6
$a0: 3
$t0: 0
```



Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Αναδρομική διαδικασία παραγοντικού

```
int fact (int n){
   if(n<1) return(1);
      else return(n*fact(n-1));
}
```

Εκτός ύλης!!!!

```
#include <stdio.h>
int __attribute__((regparm(3)))fact(register
int n);
int main(void){
   printf("%d\n", fact(10));}

-----------------------------------------------
#gcc -c fact.s
#gcc caller.c fact.o -o exec
#./exec
```

x86 Assembly

```
1.    .globl fact
2.    fact:
3.      pushl   %ebx
4.      movl    %eax,
      %ebx
5.      decl    %eax
6.      jz      .L4
7.      call    fact
8.      imull   %ebx,
      %eax
9.    .L3:
10.     popl    %ebx
11.     ret
12.   .L4:
13.     movl    $1, %eax
14.     jmp .L3
```

Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Πληροφορίες που διατηρούνται/δε διατηρούνται κατά τη κλήση μιας διαδικασίας

Διατηρούνται	Δε διατηρούνται
Αποθηκευμένοι (saved) καταχωρητές: `$s0 - $s7`	Προσωρινοί καταχωρητές: `$t0 - $t9`
Καταχωρητής δείκτη στοίβας (stack pointer): `$sp`	Καταχωρητές ορίσματος: `$a0 - $a3`
Καταχωρητής διεύθυνσης επιστροφής (return address): `$ra`	Καταχωρητές τιμής επιστροφής (return value): `$v0 - $v1`
Στοίβα επάνω (higher addresses) από το δείκτη στοίβας	Στοίβα κάτω (lower addresses) από το δείκτη στοίβας

Η επικοινωνία με τους ανθρώπους

Οι υπολογιστές «καταβροχθίζουν» αριθμούς ☺

Επεξεργασία κειμένου και άλλες εφαρμογές όμως θέλουν χαρακτήρες (chars).

Κώδικας ASCII (American Standard Code for Information Interchange)

Ένας χαρακτήρας = 1 byte = 8 bits

Οι λειτουργίες σε chars είναι πολύ συχνές- ο MIPS παρέχει εντολές για μεταφορά bytes.

```
lb $t0, 0($s1) # ανάγνωση ενός byte – load byte
sb $t1, 0($s2) # αποθήκευση ενός byte – store byte
```

Η `lb` φορτώνει ένα byte στα λιγότερο σημαντικά bits ενός καταχωρητή.

Η `sb` αποθηκεύει το λιγότερο σημαντικό byte στη μνήμη.

Η επικοινωνία με τους ανθρώπους

Αντιγραφή συμβολοσειράς (string)

```
void strcpy(char x[], char y[]){
  int i;
  i = 0;
  while((x[i] = y[i]) != '\0') /* αντιγραφή και έλεγχος
    του byte */
    i+=1;
}
```

Πώς είναι η assembly MIPS του παραπάνω κώδικα C;

Η επικοινωνία με τους ανθρώπους

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

Base address x: `$a0`

Base address y: `$a1`

`i` αντιστ. στον `$s0`

```
strcpy:
 addi $sp,$sp,-4 #χώρος για να
 sw $s0, 0($sp)  #σωθεί ο $s0
 add $s0,$zero,$zero # i←0
L1:
 add $t1,$s0,$a1 #$t1 ← i+x
 lb $t2,0($t1)   #$t2 ← M[i+x]
 add $t2,$s0,$a0 #$t2 ← i+y
 sb $t2,0($t3)   #$t2 → M[i+y]
 beq $t2,$zero,L2 #είναι $t2==0?
 addi $s0,$s0,1  # i += 1
 j L1
L2:
 lw $s0, 0($sp) #ανάκτησε τον $s0
 addi $sp,$sp,4 #διόρθωσε στοίβα
 jr $ra #πίσω στον καλούντα
```