MIPS Functions-Procedures

Νεκ. Κοζύρης - Κωστής Νίκας
nkoziris@cslab.ece.ntua.gr - knikas@cslab.ece.ntua.gr

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 επιστρέψουμε 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

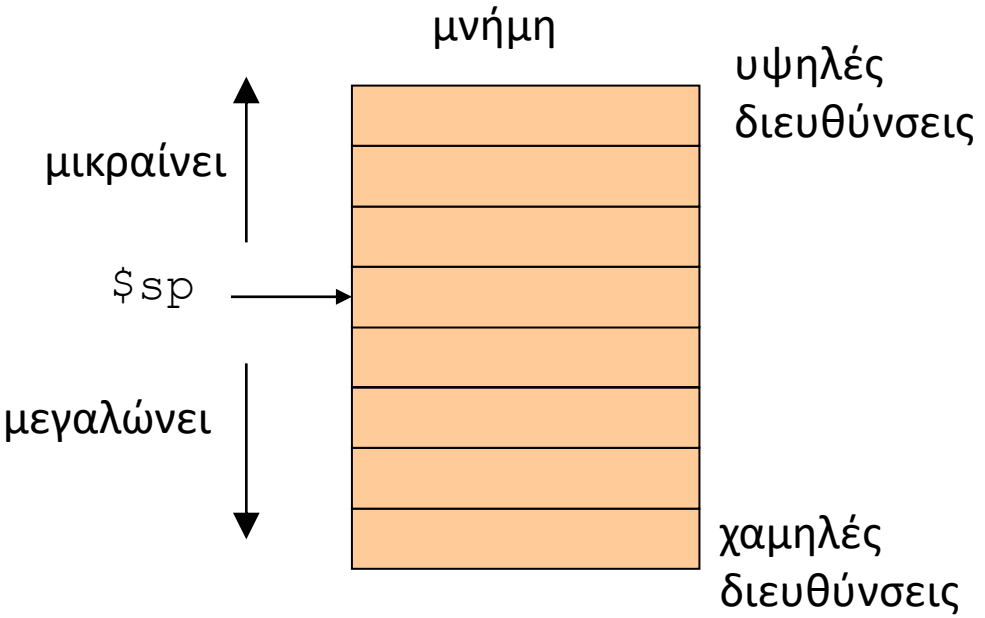
Χρήση πολλών καταχωρητών σε διαδικασίες;

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

Χρησιμοποιούμε στοίβα (stack)

Last-In-First-Out

push, pop



Παράδειγμα χρήσης καταχωρητών

Έστω οι παρακάτω δύο συναρτήσεις Α και Β:

```
int A(int x) {
    int y;

    y = B(x);
    y = y + x;
    return y;
}

int B(int arg) {
    return arg + 5;
}
```

Οι συμβάσεις χρήσης καταχωρητών του MIPS ορίζουν ότι το Χ στην Α (και στην Β) θα πρέπει να περαστεί στον $a0.

Επίσης ορίζουν ότι η τιμή επιστροφής θα πρέπει να μπει στον $v0.

Ακόμα η διεύθυνση επιστροφής της συνάρτησης βρίσκεται στον καταχωρητή $ra

MIPS assembly – 1η προσπάθεια

Στην Α, η παράμετρος εισόδου x βρίσκεται στον $a0, οπότε για να περαστεί στην Β βάλαμε την γραμμή που αντιγράφει το $a0 στο $a0, το οποίο προφανώς είναι περιττό

```
A:
    mov  $a0, $a0 // περιττό!
    jal  B
    add  $v0, $v0, $a0
    jr   $ra
```

Η Β είναι απλή: το arg υπάρχει ήδη στον καταχωρητή $a0, οπότε το αυξάνουμε κατά 5, και γράφουμε το αποτέλεσμα στον $v0

```
B:
    addi $v0, $a0, 5
    jr   $ra
```

Κατόπιν, καλούμε την Β η οποία επιστρέφει το αποτέλεσμα της στον $v0. Ακολούθως προσθέτουμε το αποτέλεσμα με το $a0 και βάζουμε το αποτέλεσμα στον $v0

Η κλήση Α(10) καλεί την Β(10) η οποία επιστρέφει το 15, η Α προσθέτει το 10 και επιστρέφει 25

Ο κώδικας είναι φαινομενικά σωστός

MIPS assembly – 2η προσπάθεια

Η Β άλλαξε λίγο: πρώτα γίνεται η αύξηση στον $a0 και μετά η αντιγραφή του αποτελέσματος στον $v0.

Ο κώδικας της Β είναι σωστός!

Αλλά υπάρχουν προβλήματα:

1) Η κλήση Α(10) καλεί την Β(10) η οποία επιστρέφει το 15, η Α προσθέτει το 15 (η Β άλλαξε το $a0!) και υπολογίζει το 30 το οποίο και επιστρέφει η Α.

```
A:
    mov   $a0, $a0 // περιττό!
    jal   B
    add   $v0, $v0, $a0
    jr    $ra

B:  addi  $a0, $a0, 5
    mov   $v0, $a0
    jr    $ra
```

2) Το ίδιο συμβαίνει και για τον $ra! Το $ra στην Α δείχνει π.χ. στην main από όπου κλήθηκε. Μετά την κλήση Β αλλάζει ώστε να δείχνει στην 3η εντολή της Α. Η Β επιστρέφει σωστά, αλλά το jr $ra της Α δεν επιστρέφει στην main αλλά στην ίδια την Α.

MIPS assembly – 3η προσπάθεια

Η Α δημιουργεί ένα αντίγραφο του $a0 σε ένα μπλε κουτί. Κατόπιν καλεί την Β η οποία αλλάζει την τιμή του $a0 και επιστρέφει το αποτέλεσμα της (15 στην περίπτωση της κλήσης Α(10)) στον $v0 και επιστρέφει στην Α.

Η Α επαναφέρει την παλιά τιμή του $a0 (10) από το μπλε κουτί, και τώρα η πρόσθεση είναι σωστή και υπολογίζει 15 + 10 = 25 το οποίο και γράφεται στον $v0.

Ο κώδικας της Α υπολογίζει πλέον το σωστό αποτέλεσμα ανεξάρτητα πως θα γραφτεί η Β!

Το SAVE γίνεται πριν την κλήση της συνάρτησης που «φοβόμαστε»

Το RESTORE γίνεται μετά την κλήση και πριν την χρήση του καταχωρητή που χρειαζόμαστε

```
A:  SAVE $a0, LOCATION
    jal  B
    RESTORE $a0, LOCATION
    add  $v0, $v0, $a0
    jr   $ra

B:  addi $a0, $a0, 5
    mov  $v0, $a0
    jr   $ra
```

MIPS assembly – 3η προσπάθεια

Το ίδιο πρέπει να γίνει και για τον $ra!!!

Σε ένα άλλο πορτοκαλί κουτί η Α δημιουργεί ένα αντίγραφο του $ra

Η Α καλεί όποια συνάρτηση χρειάζεται (την Β στο συγκεκριμένο παράδειγμα)

Η Α επαναφέρει την παλιά τιμή του $ra από το πορτοκαλί κουτί, και τώρα η διεύθυνση επιστροφής είναι σωστή και δείχνει σε όποιον κάλεσε την Α.

Τώρα πια ο κώδικας της Α λειτουργεί σωστά υπολογίζει το σωστό αποτέλεσμα ανεξάρτητα πως θα γραφτεί η Β και επιστρέφει σωστά σε όποιον την κάλεσε!

```
A:   SAVE $ra, LOC-ra
     SAVE $a0, LOCATION
     jal  B
     RESTORE $a0, LOCATION
     add  $v0, $v0, $a0
     RESTORE $ra, LOC-ra
     jr   $ra

B:   addi $a0, $a0, 5
     mov  $v0, $a0
     jr   $ra
```

MIPS assembly – 3η προσπάθεια: εξήγηση

Η Α δεν γνωρίζει αν η Β -όπως έχει γραφτεί- θα χρησιμοποιήσει ή όχι κάποιους καταχωρητές. Δεν υπάρχει καμμία εγγύηση για κάτι τέτοιο. Όπως και να γράφεί η Β είναι σωστή.

Συνεπώς, η Α θεωρεί ότι οτιδήποτε χρήσιμο για την ίδια (ο $a0) μπορεί να καταστραφεί (πανωγραφεί) από την Β.

Ο $ra είναι διαφορετική περίπτωση: ανεξάρτητα πως θα γραφεί η Β (ακόμα και για μια κενή Β), το $ra πανωγράφεται από την ίδια την Α όταν εκτελείται η εντολή jal Β!

```
A:  SAVE $ra, LOC-ra
    SAVE $a0, LOCATION
    jal  B
    RESTORE $a0, LOCATION
    add  $v0, $v0, $a0
    RESTORE $ra, LOC-ra
    jr   $ra

B:  addi $a0, $a0, 5
    mov  $v0, $a0
    jr   $ra
```

Γενικός Τρόπος γραφής κώδικα

Πρόλογος

```
A:  SAVE $ra, LOCATION-ra
```

Σώμα

```
    SAVE $a0, LOCATION
    jal B
    RESTORE $a0, LOCATION
    add $v0, $v0, $a0
```

Επίλογος

```
    RESTORE $ra, LOCATION-ra
    jr $ra
```

```
B:                                  Πρόλογος
    addi $a0, $a0, 5
    mov    $v0, $a0                 Σώμα
    jr $ra                          Επίλογος
```

Κάθε συνάρτηση έχει Πρόλογο, Σώμα, Επίλογο. Ο πρόλογος μπορεί να είναι κενός (όπως στην περίπτωση της B).

Γράφουμε *πρώτα* το Σώμα, και *μετά* γεμίζουμε Πρόλογο και Επίλογο (όταν ξέρουμε τι χρειαζόμαστε όπως θα δούμε παρακάτω).

Πρέπει να συζητήσουμε:

1. γιατί το Μπλέ και το Πορτοκαλί κουτάκι είναι διαφορετικά;
2. που βρίσκονται τα κουτάκια αυτά;
3. Υπάρχουν άλλες επιλογές για να γράψω κώδικα;

Συζήτηση

Δυο τύποι SAVE/RESTORE

- Αυτά που αφορούν πρόλογο/επίλογο (μόνο το $ra στο προηγούμενο παράδειγμα)
- Τα άλλα που αφορούν καταχωρητές και τιμές που χρησιμοποιώ στο σώμα συνάρτησης

Που βρίσκονται τα κουτάκια αυτά; Πως γίνεται το SAVE/RESTORE;

- Ο χώρος θα πρέπει να είναι σε ιδιωτικό για την κάθε συνάρτηση μέρος
- Σε περίπτωση αναδρομικής συνάρτησης θα πρέπει να έχω πολλές τέτοιες θέσεις (μία για κάθε ενεργοποίηση της συνάρτησης)
- Συνεπώς θα πρέπει να είναι στην στοίβα. Οπότε SAVE = Push, RESTORE = Pop

Στοίβα συστήματος στον MIPS

- Υλοποιείται με τον $sp (η στοίβα μεγαλώνει προς τα κάτω)
- Push($a0) => ακολουθία: addui $sp, $sp, -4 ; sw $a0, 0($sp)
- Pop($a0) => ακολουθία: lw $a0, 0($sp) ; addui $sp, $sp, 4

Ο τελικός κώδικας της Α

A:
Πρόλογος

```
addui   $sp, $sp, -8
sw      $ra, 0($sp) //push
```

Σώμα

```
sw      $a0, 4($sp) //push
jal     B
lw      $a0, 4($sp) //pop
add     $v0, $v0, $a0
```

Επίλογος

```
lw      $ra, 0($sp) //pop
addui   $sp, $sp, 8
jr      $ra
```

Χρειαζόμαστε 2 λέξεις στην στοίβα

Στο offset 0 από τον $sp θα αποθηκευτεί ο $ra και σε offset 4 ο $a0

Η πρώτη εντολή του Προλόγου (addui) δημιουργεί τον χώρο κουνώντας τον $sp προς τα κάτω (8 bytes = 2 λέξεις)

Κατόπιν μπορούμε να γράψουμε τον $ra υλοποιώντας το δεύτερο μισό της push

Ανάλογα αποθηκεύεται και ο $a0

Η επαναφορά του $a0 είναι απλά ένα lw

Η επαναφορά του $sp στην αρχική του θέση γίνεται στο τέλος του Επιλόγου αμέσως πριν το jr $ra.

Μια άλλη προσέγγιση;

Τι αν η B εγγυόταν ότι **δεν θα άλλαζε** κανένα καταχωρητή;

Η B σώζει και επαναφέρει τον $a0, οπότε η A *δεν βλέπει αλλαγή*!

Ομως και η A πρέπει να δώσει την ίδια εγγύηση σε όποιον την καλεί! Αλλά δεν χρησιμοποιεί άλλον καταχωρητή, οπότε είναι OK.

Η τεχνική αυτή λέγεται *callee-save*, διότι η καλούμενη συνάρτηση έχει την υποχρέωση να κάνει όλη την δουλειά.

Η προηγούμενη τεχνική λέγεται *caller-save* διότι η συνάρτηση που καλεί πρέπει να σώσει τις ενδιαφέρουσες τιμές

Τα πορτοκαλί τί είναι; (Callee-save!)

```
B:  addui   $sp, $sp, -4
    sw      $a0,  0($sp) //push

    addi $a0, $a0, 5
    mov     $v0, $a0

    lw      $a0, 0($sp) //pop
    addui   $sp, $sp, 4
    jr      $ra
```

```
A:  addui   $sp, $sp, -4
    sw      $ra, 0($sp) //push

    jal     B
    add     $v0, $v0, $a0

    lw      $ra, 0($sp) //pop
    addui   $sp, $sp, 4
    jr      $ra
```

Caller/Callee Save

Callee-Save

- Η κάθε συνάρτηση *εγγυάται ότι δεν θα αλλάξει κανένα καταχωρητή* (εκτός π.χ. του $v0)
- Πολύ Απλό:
  - Στον πρόλογο σώζω όλους τους καταχωρητές που χρησιμοποιώ
  - Στον επίλογο επαναφέρω την παλιά τιμή τους

Caller-Save

- Οι συναρτήσεις *δεν εγγυόνται τίποτα* σε αυτόν που τις καλεί
- Πλήρης ελευθερία χρήσης καταχωρητων
- Είναι πιο αποτελεσματικό (;)
- Πρέπει να σώζω μόνο ότι μου είναι χρήσιμο και επηρρεάζεται από κλήσεις συναρτήσεων => όχι όλους τους καταχωρητές.
  - Εντός του σώματος της συνάρτησης

Caller/Callee Save

Πότε συμφέρει το ένα και πότε το άλλο;

- Το να εγγυόμαι στους άλλους ότι δεν θα αλλάξω τίποτα είναι σημαντική ευθύνη
- Το να μην έχω εγγυήσεις από τους άλλους κάνει την ζωή μου πιο δύσκολη

Ιδανικά;

- Υπάρχουν περιπτώσεις που σε προσέγγιση caller-save δεν χρειάζεται να σώσω τους καταχωρητές που χρησιμοποιώ
- Όταν η «ζωή» της τιμής που βρίσκεται στον καταχωρητή δεν διασταυρώνεται με κλήση συνάρτησης:

```
addi   $t3, $a2, 4
xor    $a0, $t3, $a1
jal    B
…
addi   $t3, $v0, 5
```

Το καλύτερο και από τις δύο προσεγγίσεις;;;

Οι συμβάσεις του MIPS χωρίζουν τους καταχωρητές σε δύο ομάδες

- Callee-save ($s0-$s7)
- Caller-save ($t0-$t9)

Η κάθε συνάρτηση εγγυάται ότι ***δεν* θα αλλάξει** τους $s0-$s7

Η κάθε συνάρτηση **έχει το δικαίωμα να αλλάξει** τους υπόλοιπους καταχωρητές: $t0-$t7, $a0-$a3, $v0, $v1, $at, $ra

Απλή στρατηγική χρήσης:

- Για καταχωρητή που η ζωή του «διασταυρώνεται» με κλήση συνάρτησης ➔ χρήση $s_ (με save/restore σε πρόλογο/επίλογο)
- Για καταχωρητή που χρησιμοποιείται χωρίς να «διασταυρώνεται» με κλήση συνάρτησης ➔ χρήση $t_ (με save/restore σε πρόλογο/επίλογο)

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
  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
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

PC:

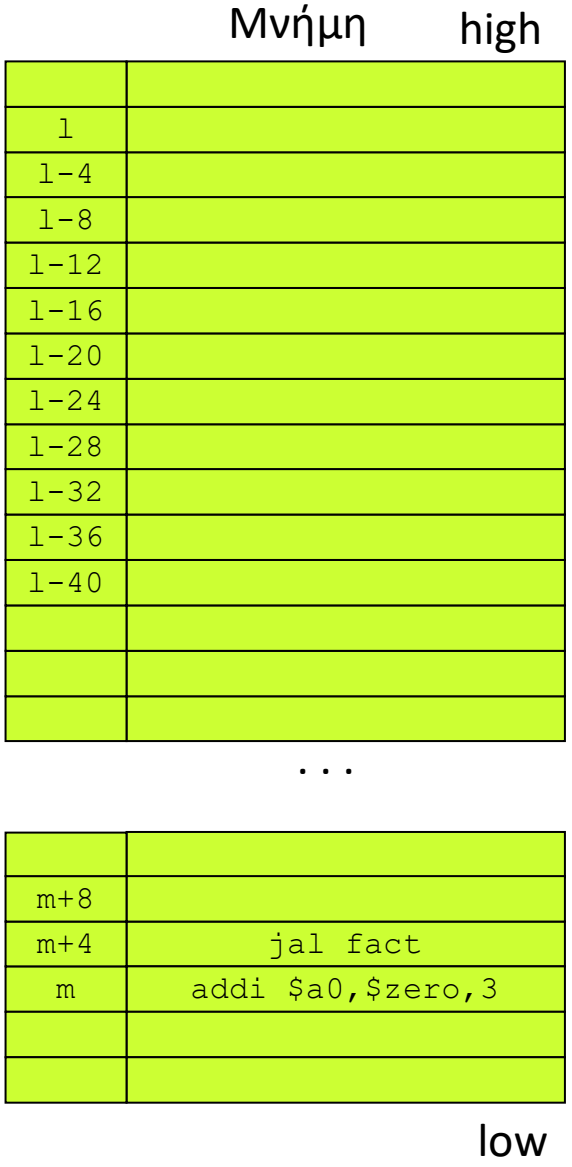
$sp:

$ra:

$v0:

$a0:

$t0:



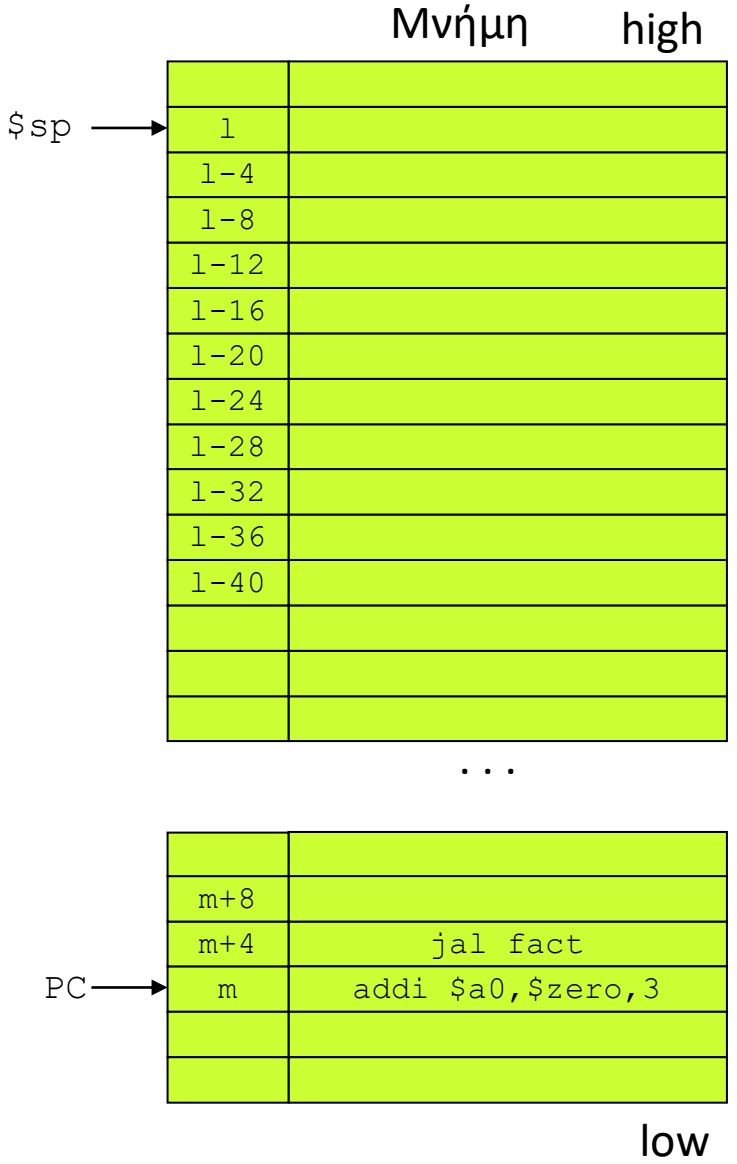
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  m

$sp: l

$ra:

$v0:

$a0:

$t0:
```



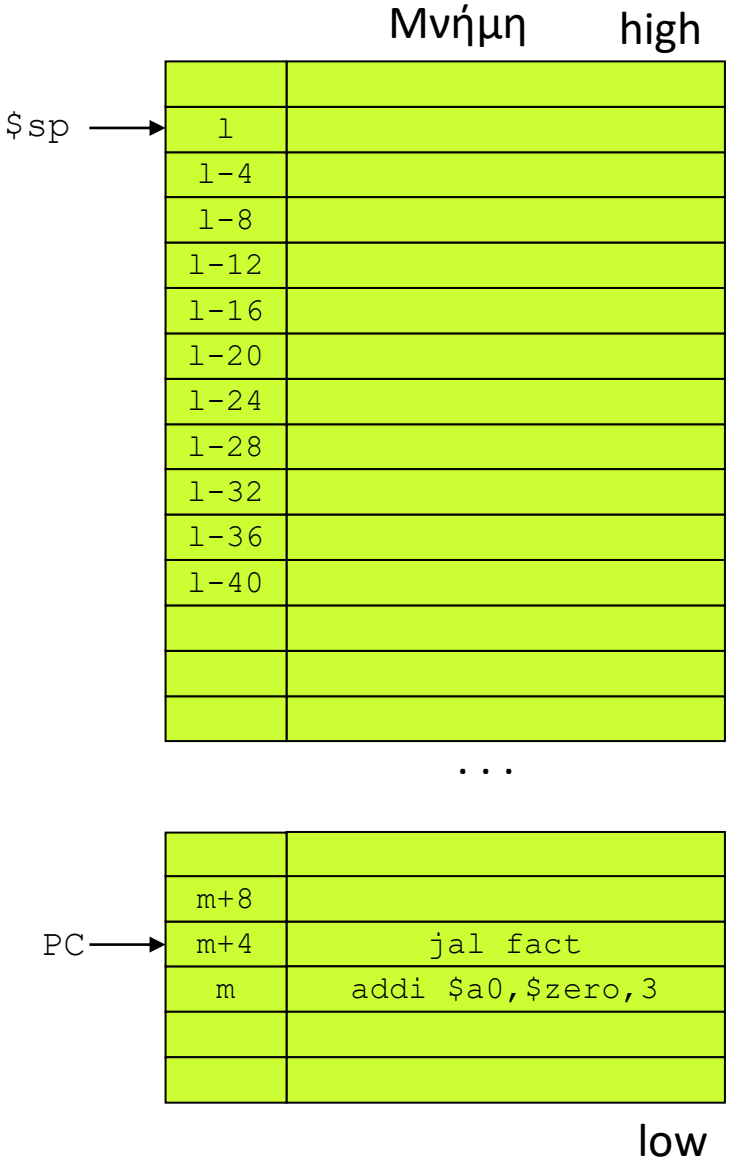
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  m+4

$sp: l

$ra:

$v0:

$a0: 3

$t0:
```



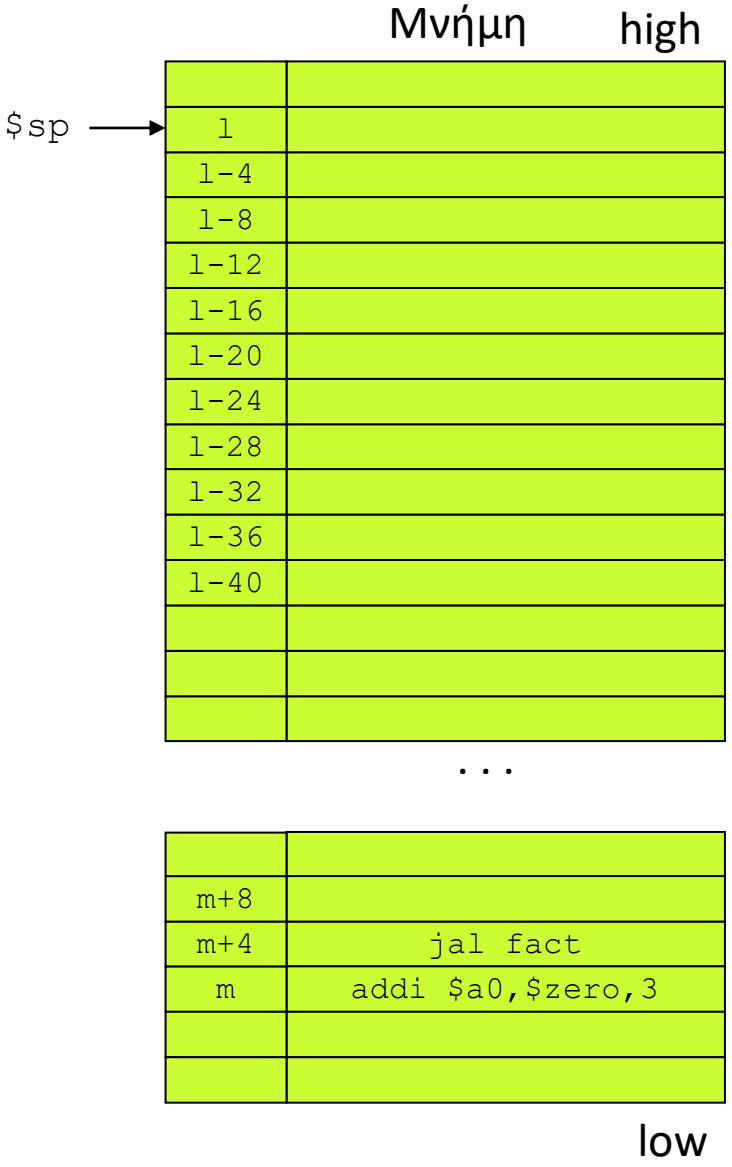
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

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
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC
$sp: l
$ra: m+8
$v0:
$a0: 3
$t0:
```



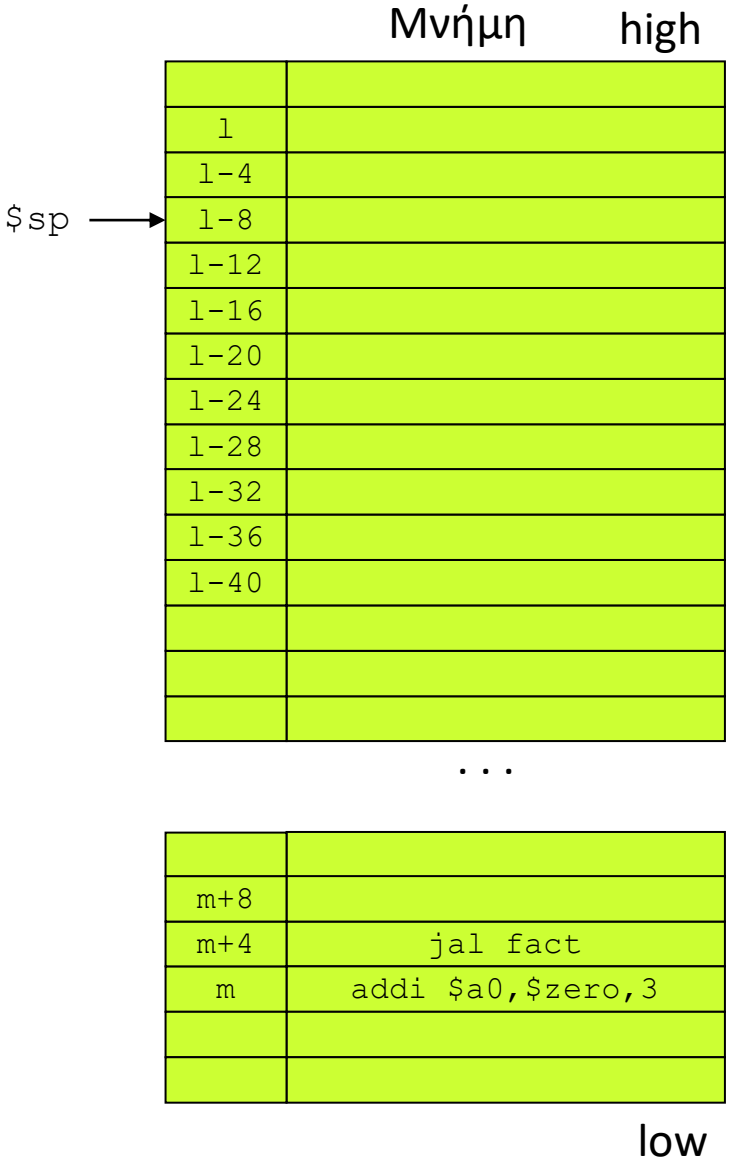
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

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
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 3

$t0:
```



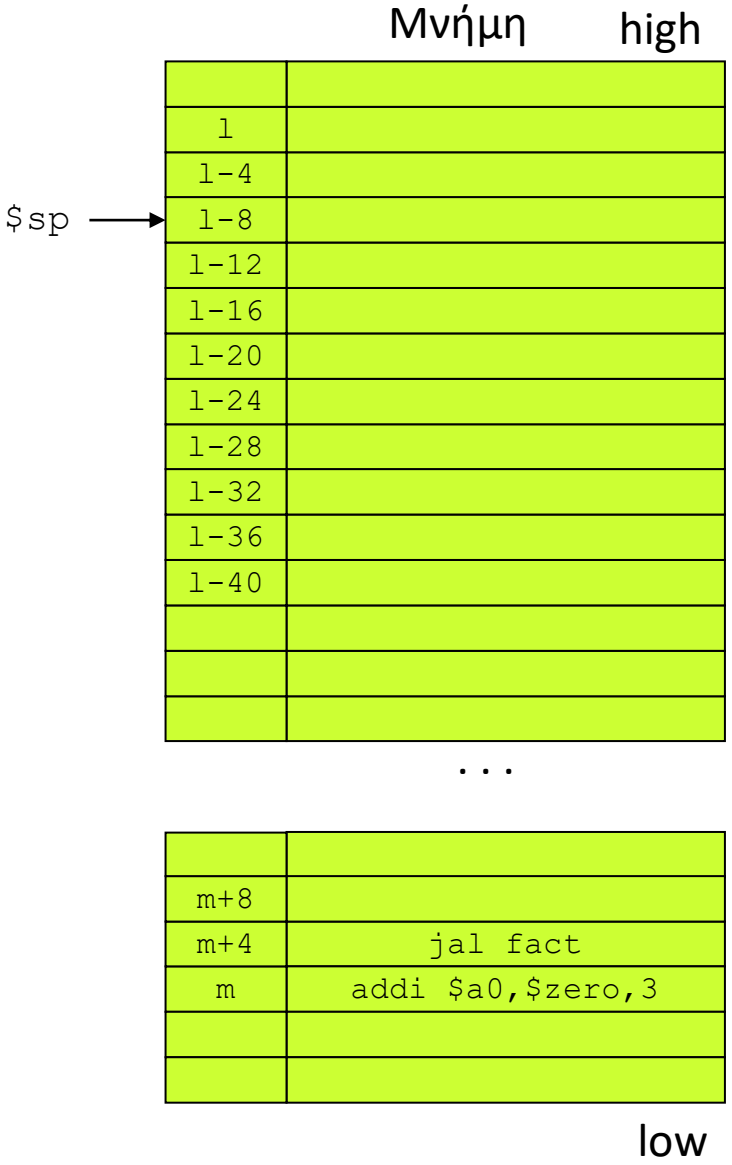
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 sw $ra,4($sp) #διεύθ. επιστροφής ← PC
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 3

$t0:
```



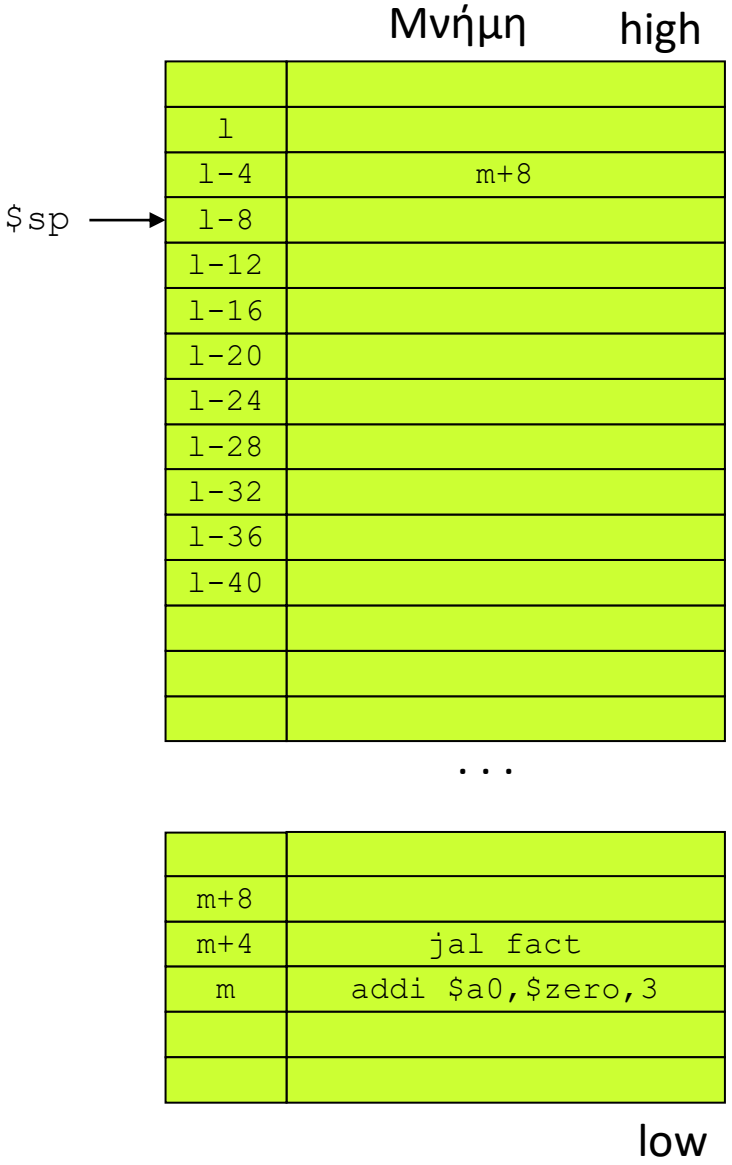
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 sw $ra,4($sp) #διεύθ. επιστροφής ← PC
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 3

$t0:
```



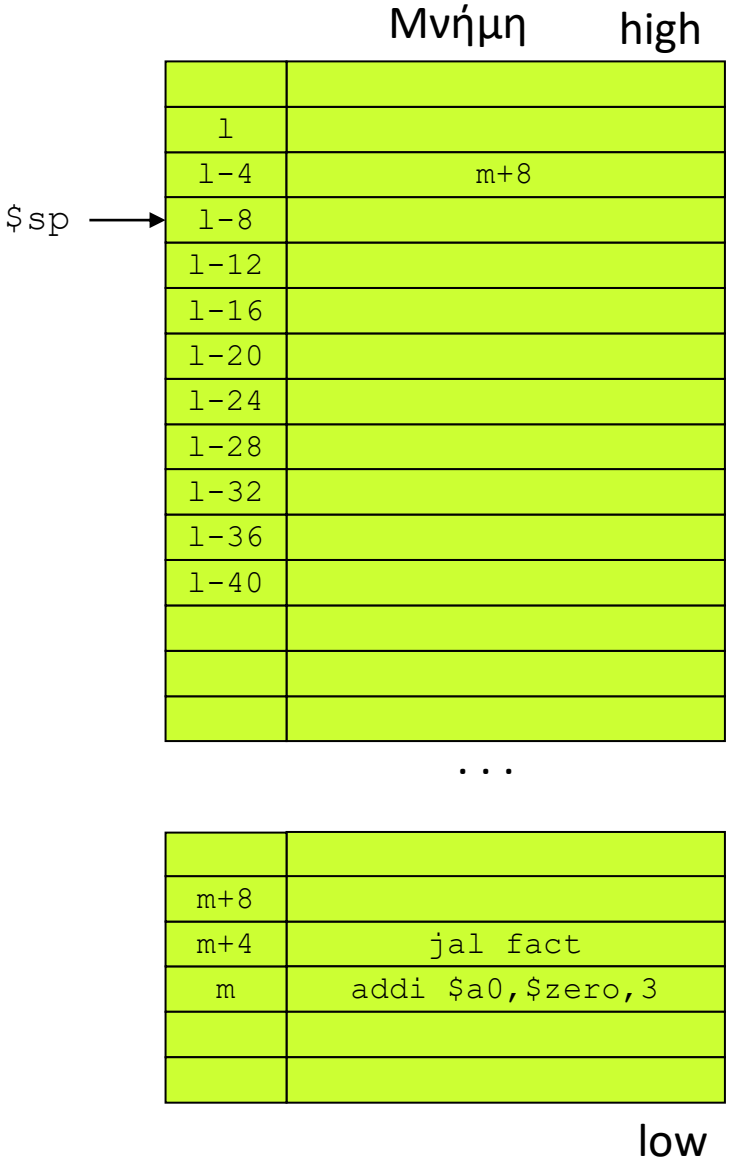
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 3

$t0:
```



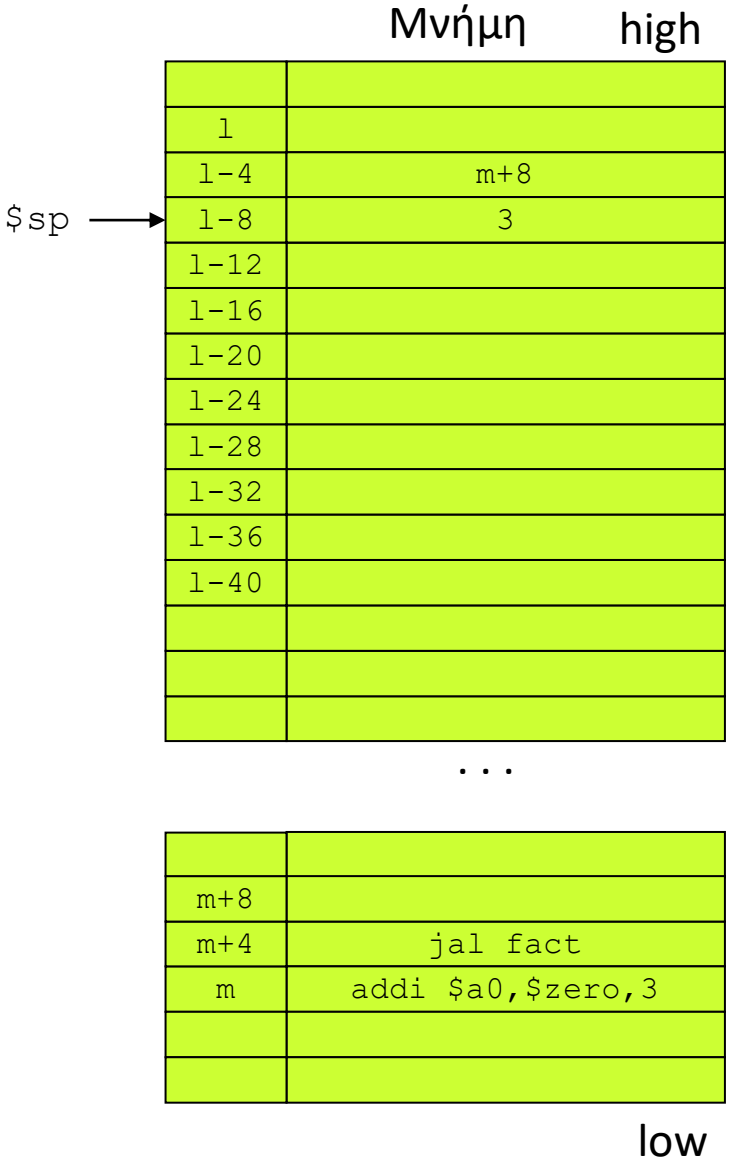
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 3

$t0:
```



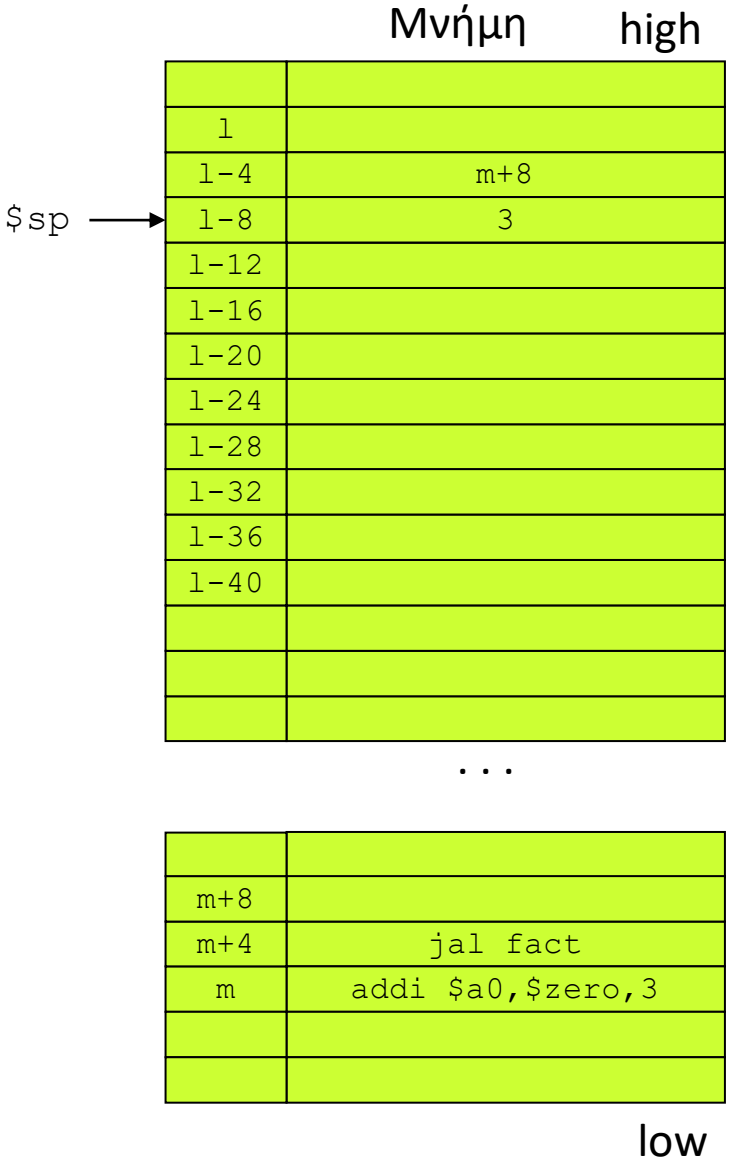
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 3

$t0:
```



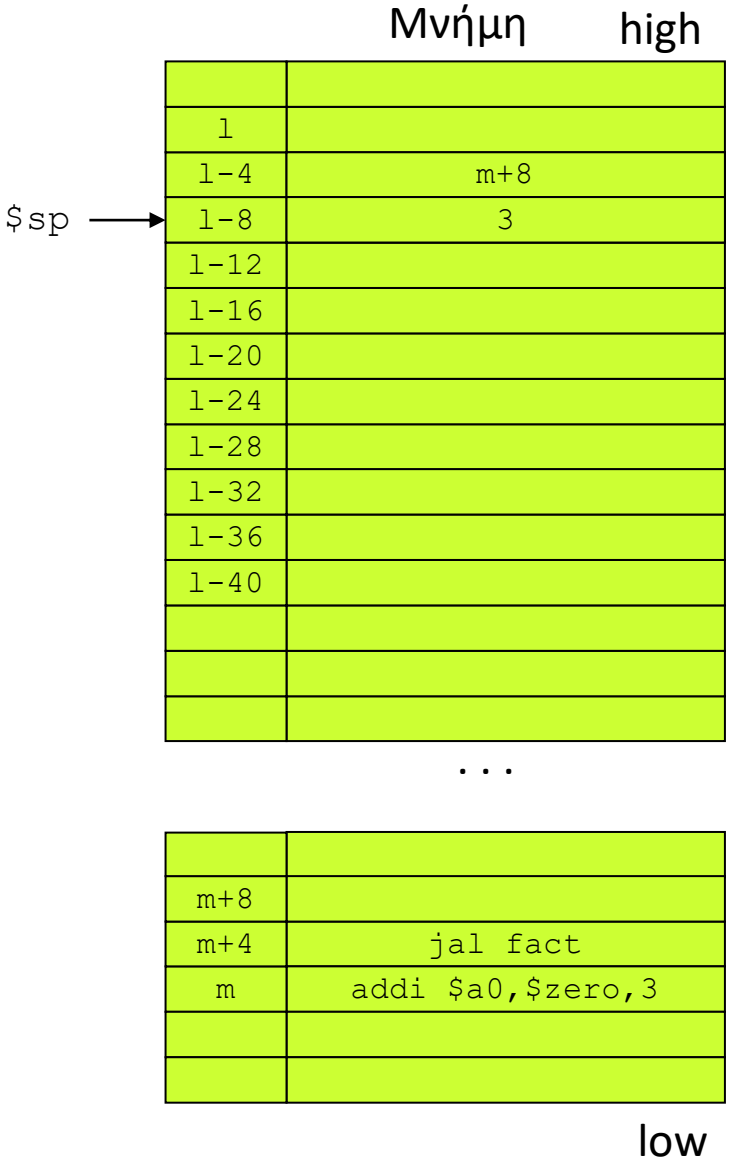
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 3

$t0: 0
```



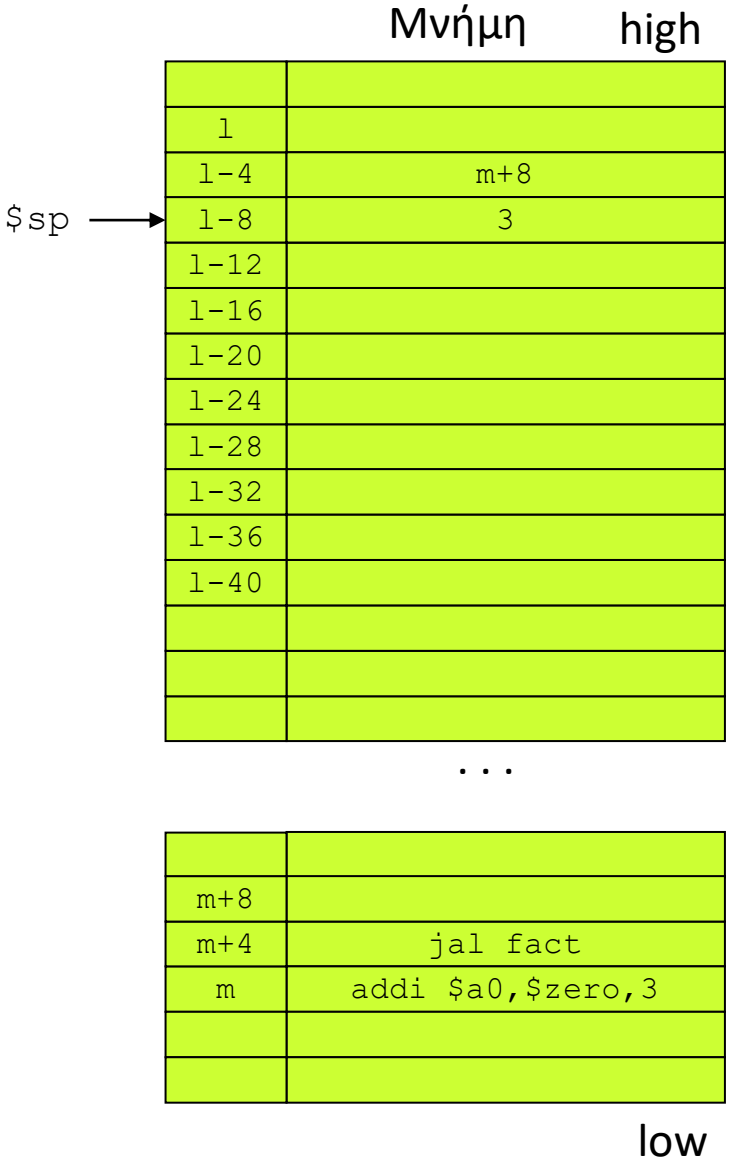
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 3

$t0: 0
```



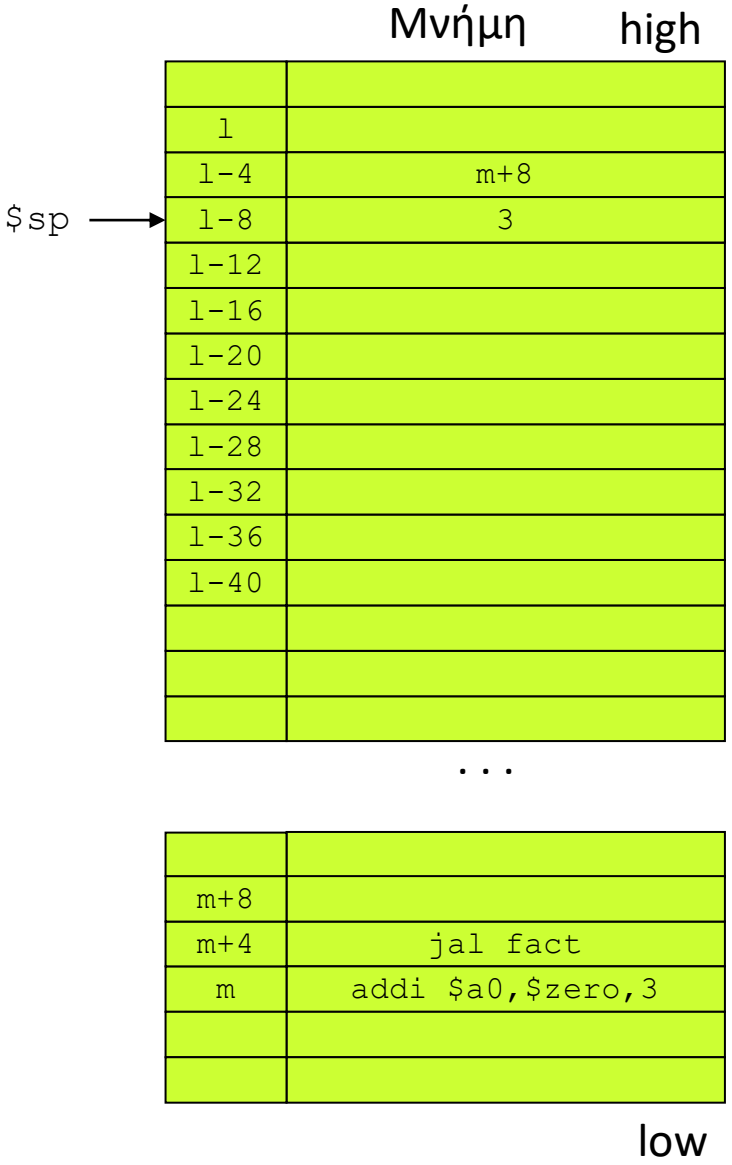
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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 ← PC
 jal fact
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 3

$t0: 0
```



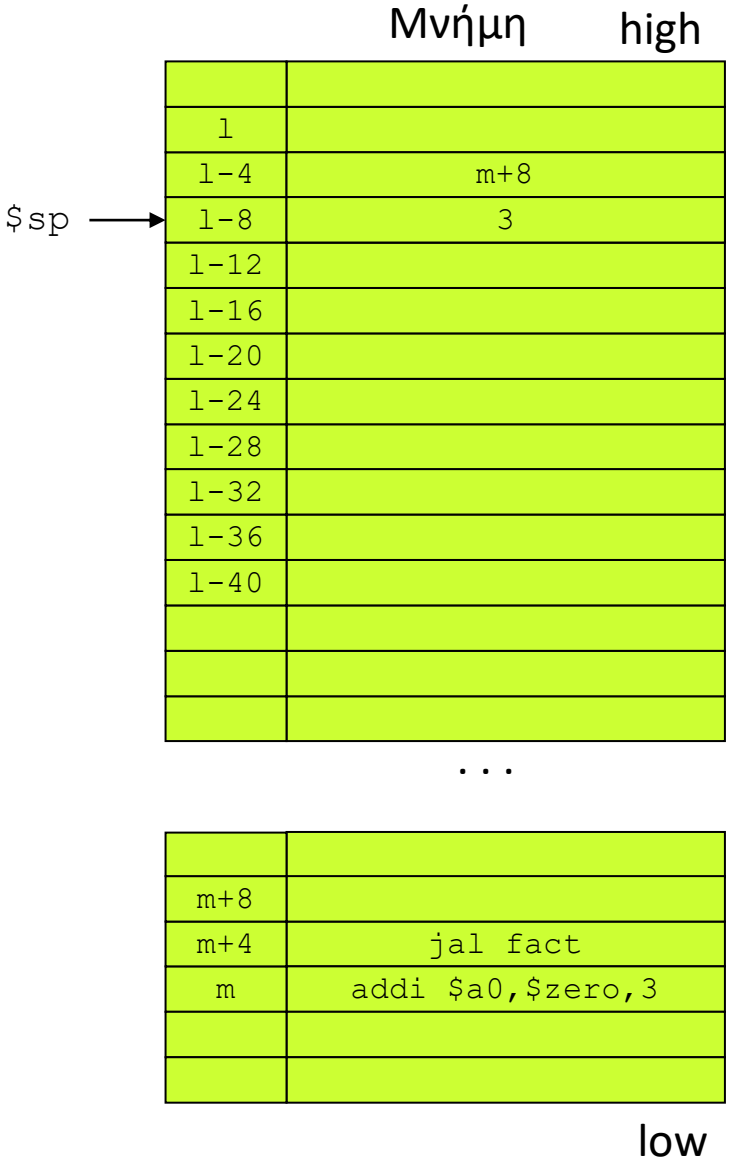
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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 ← PC
 jal fact
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 2

$t0: 0
```



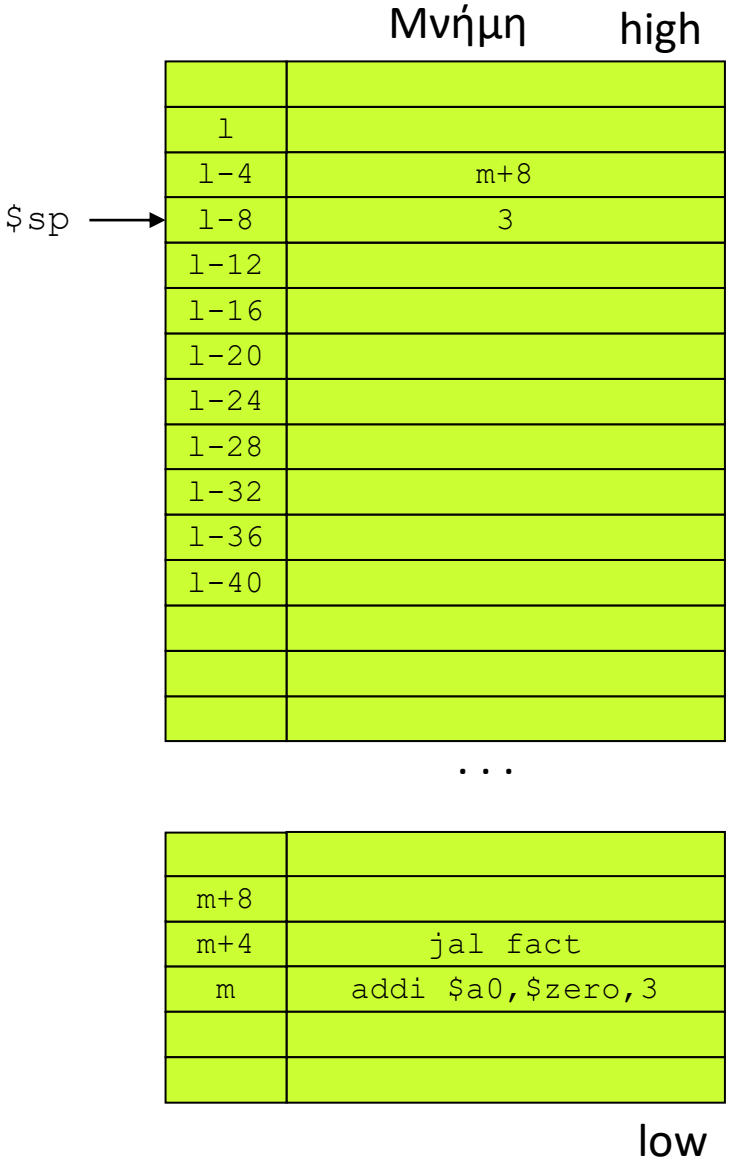
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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 ← PC
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0:

$a0: 2

$t0: 0
```



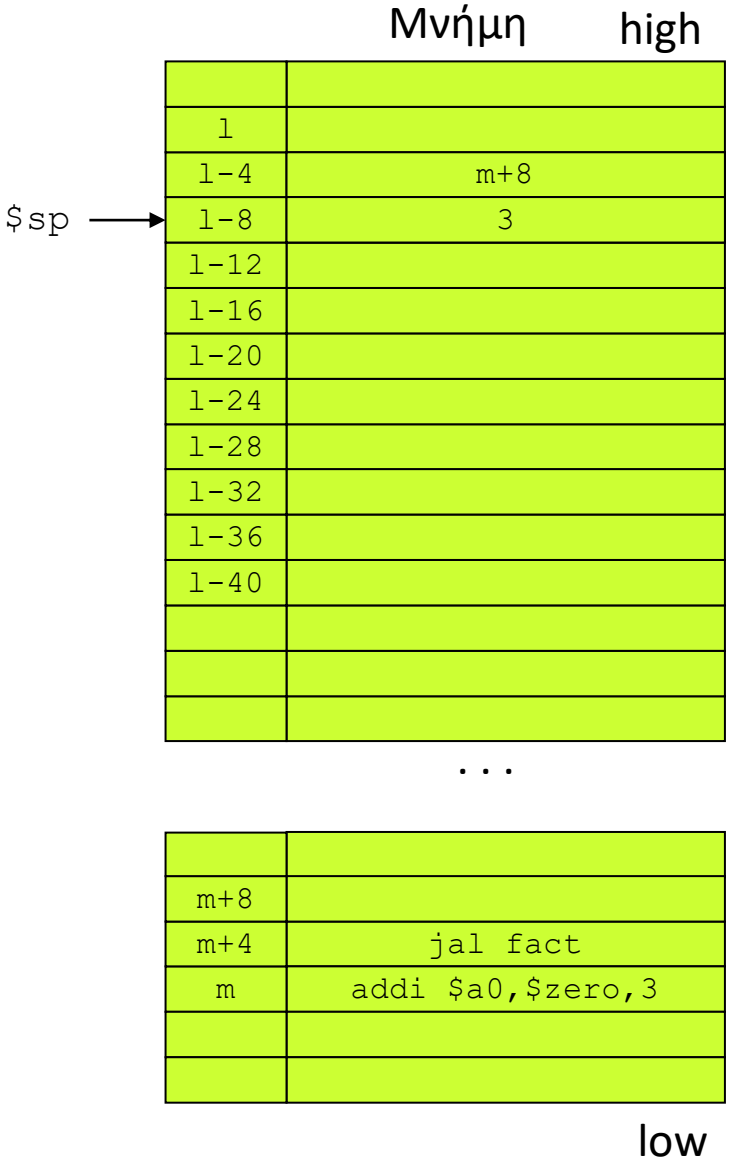
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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 ← PC
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: d

$v0:

$a0: 2

$t0: 0
```



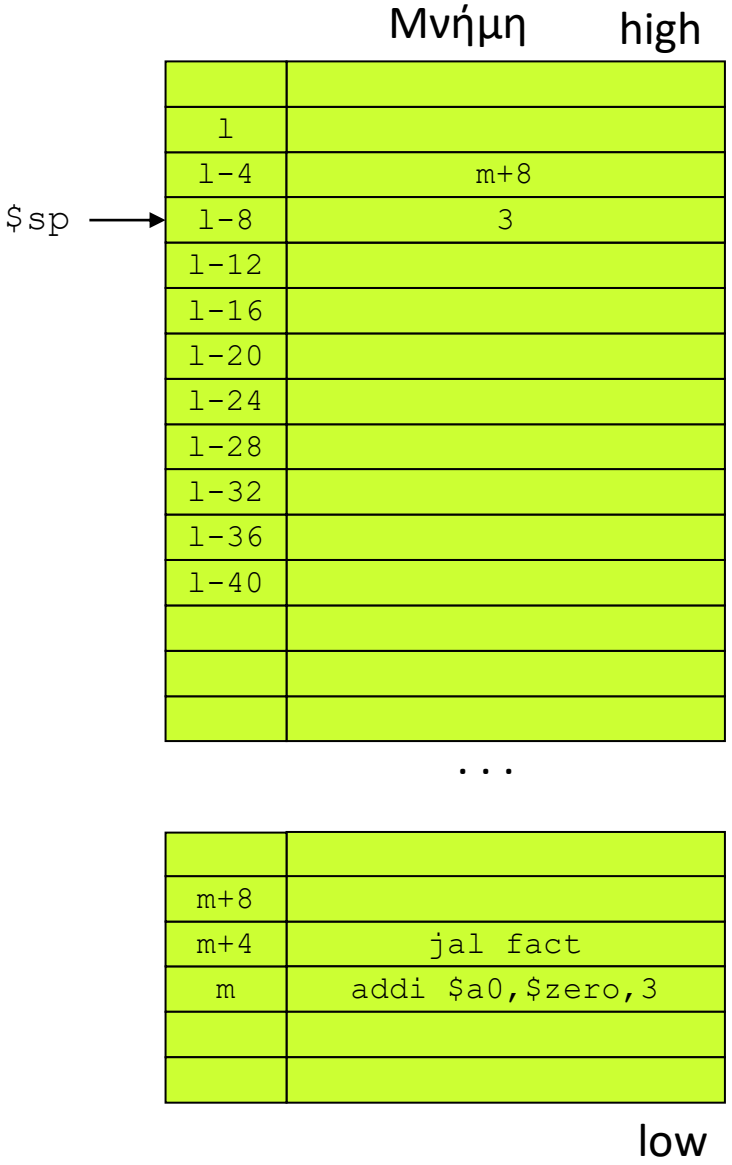
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

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
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: d

$v0:

$a0: 2

$t0: 0
```



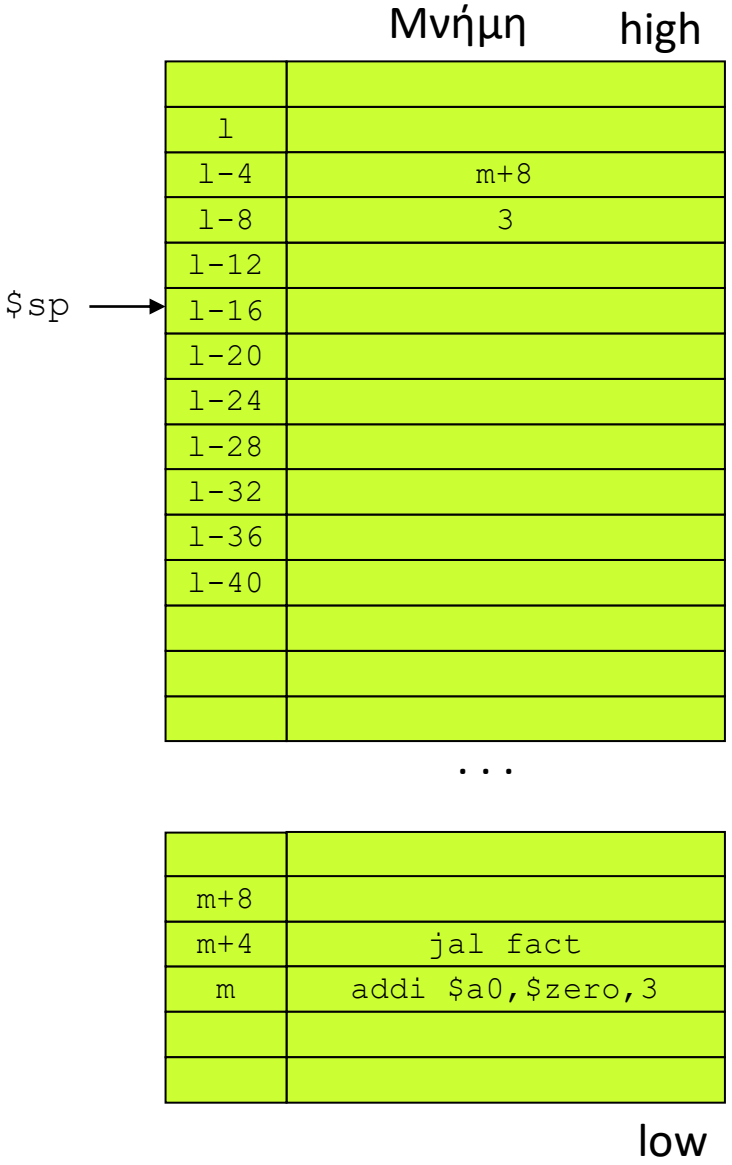
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

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
  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
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

```
PC:  βλ. PC
$sp: l-16
$ra: d
$v0:
$a0: 2
$t0: 0
```



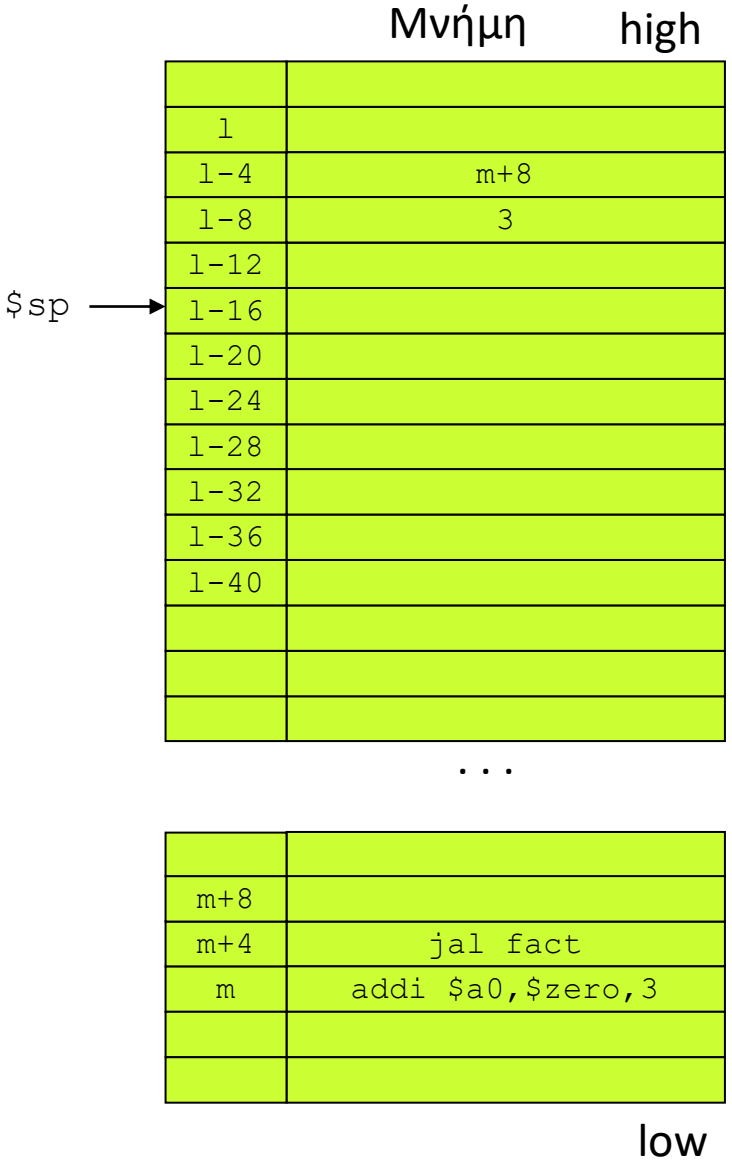
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 sw $ra,4($sp) #διεύθ. επιστροφής ← PC
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC
$sp: l-16
$ra: d
$v0:
$a0: 2
$t0: 0
```



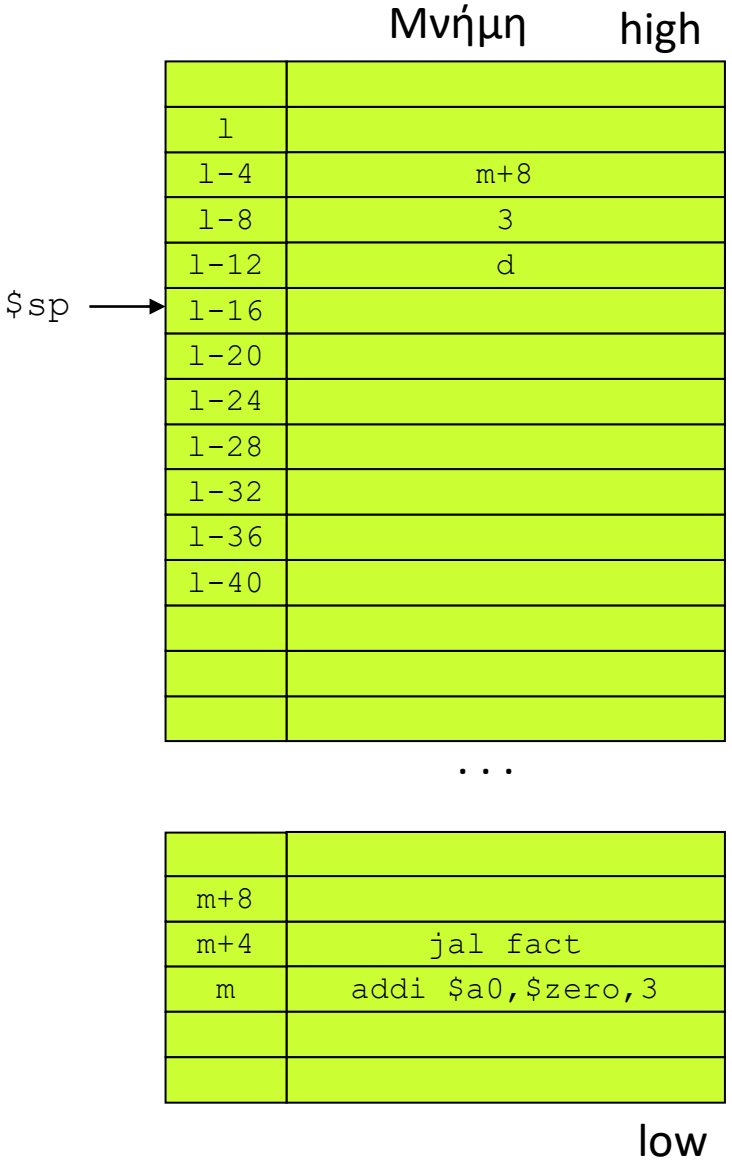
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 sw $ra,4($sp) #διεύθ. επιστροφής ← PC
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC
$sp: l-16
$ra: d
$v0:
$a0: 2
$t0: 0
```



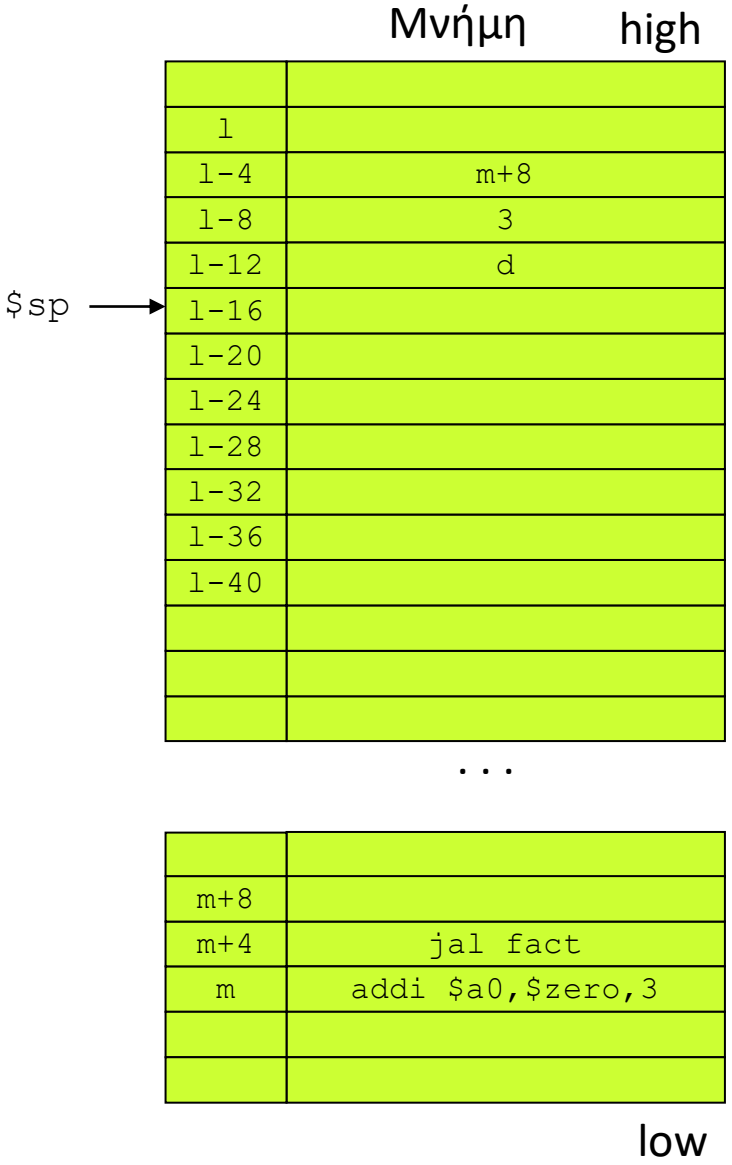
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0:

$a0: 2

$t0: 0
```



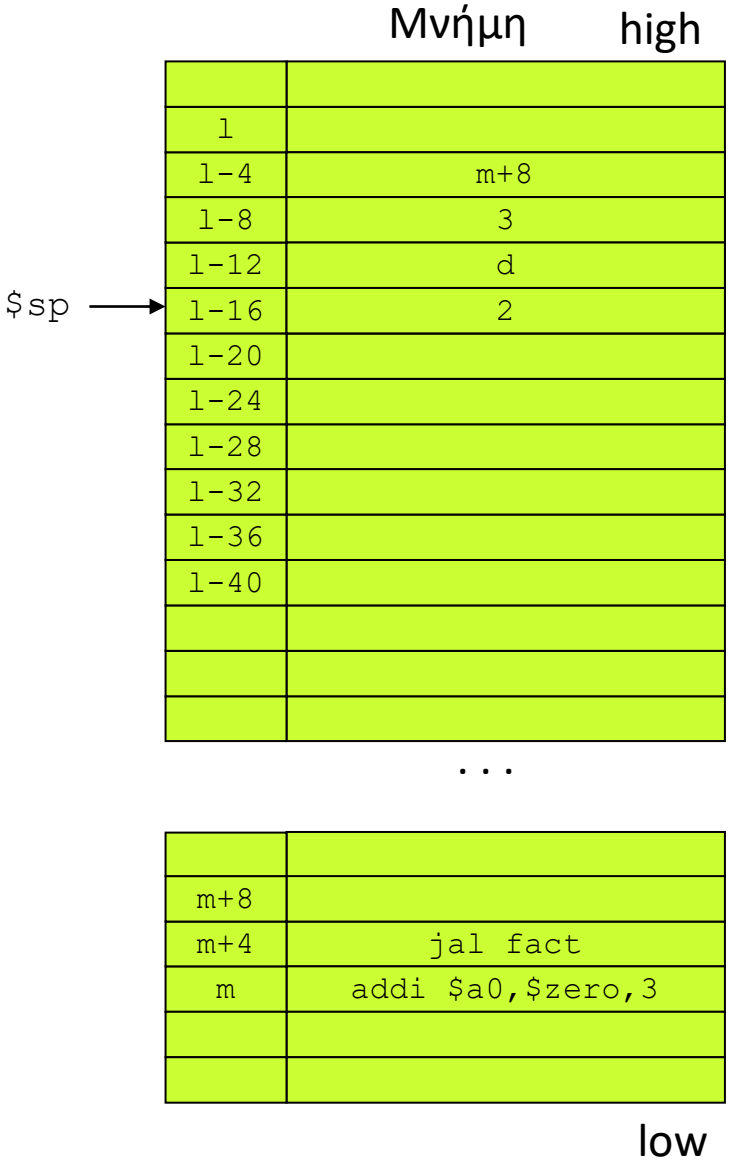
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0:

$a0: 2

$t0: 0
```



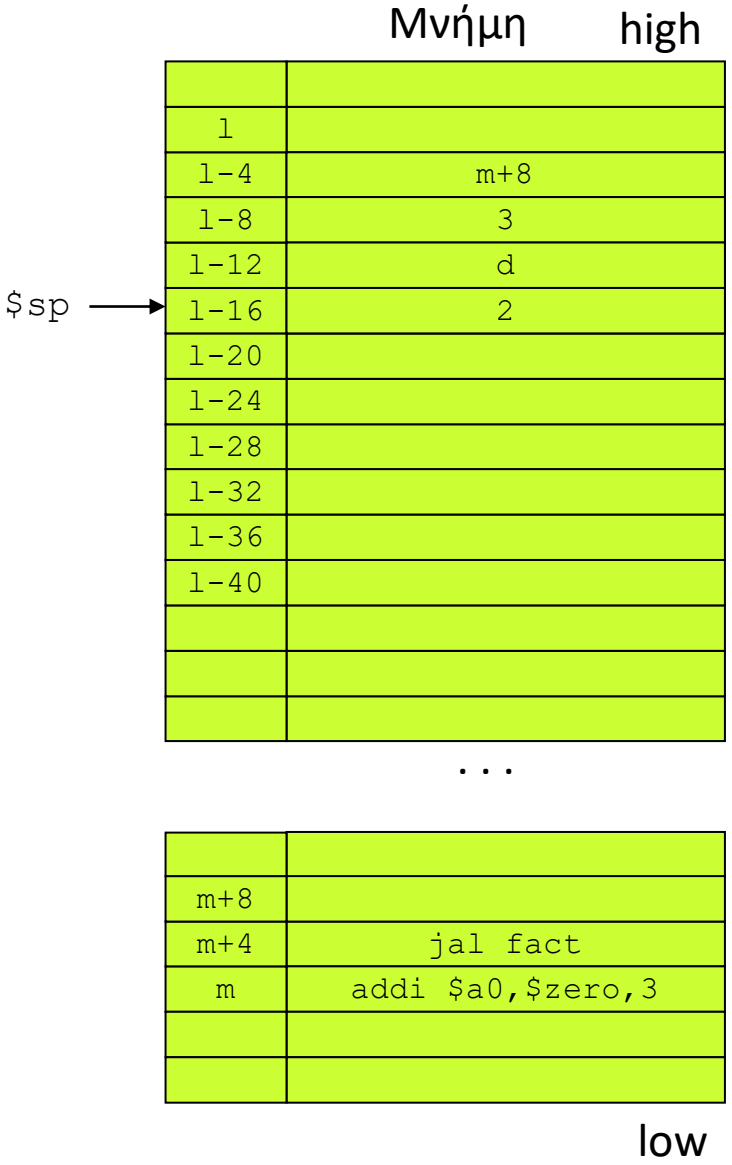
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC
$sp: l-16
$ra: d
$v0:
$a0: 2
$t0: 0
```



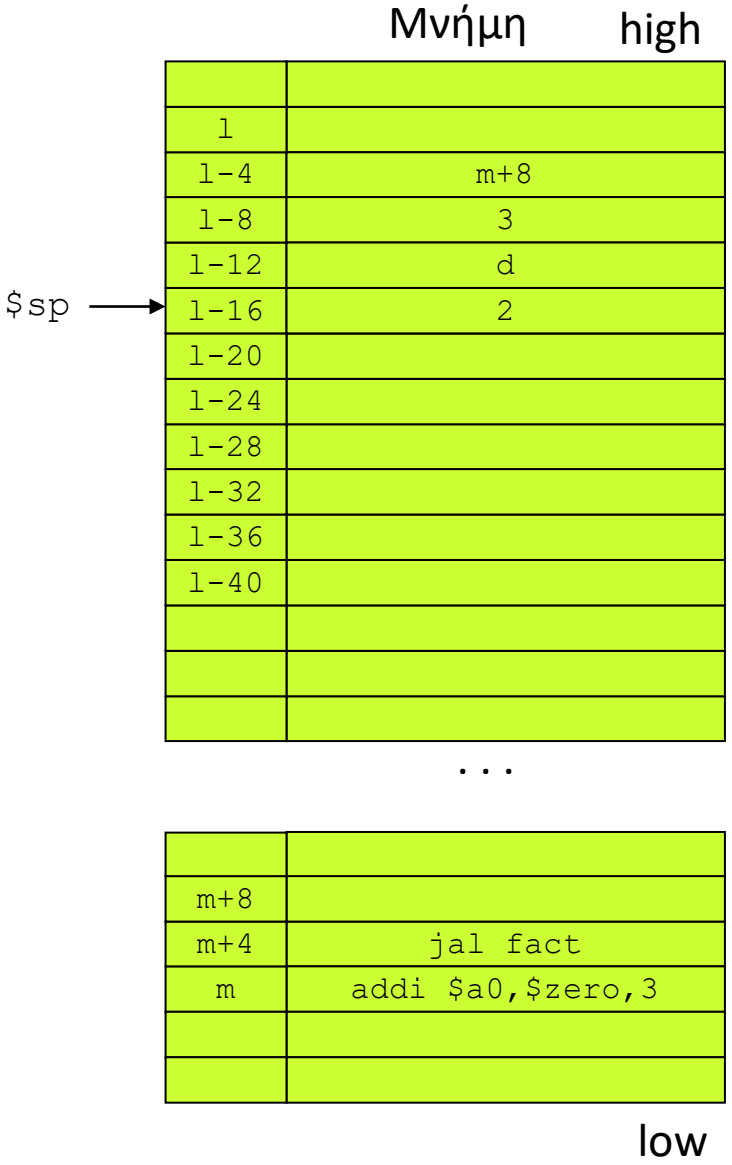
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC
$sp: l-16
$ra: d
$v0:
$a0: 2
$t0: 0
```



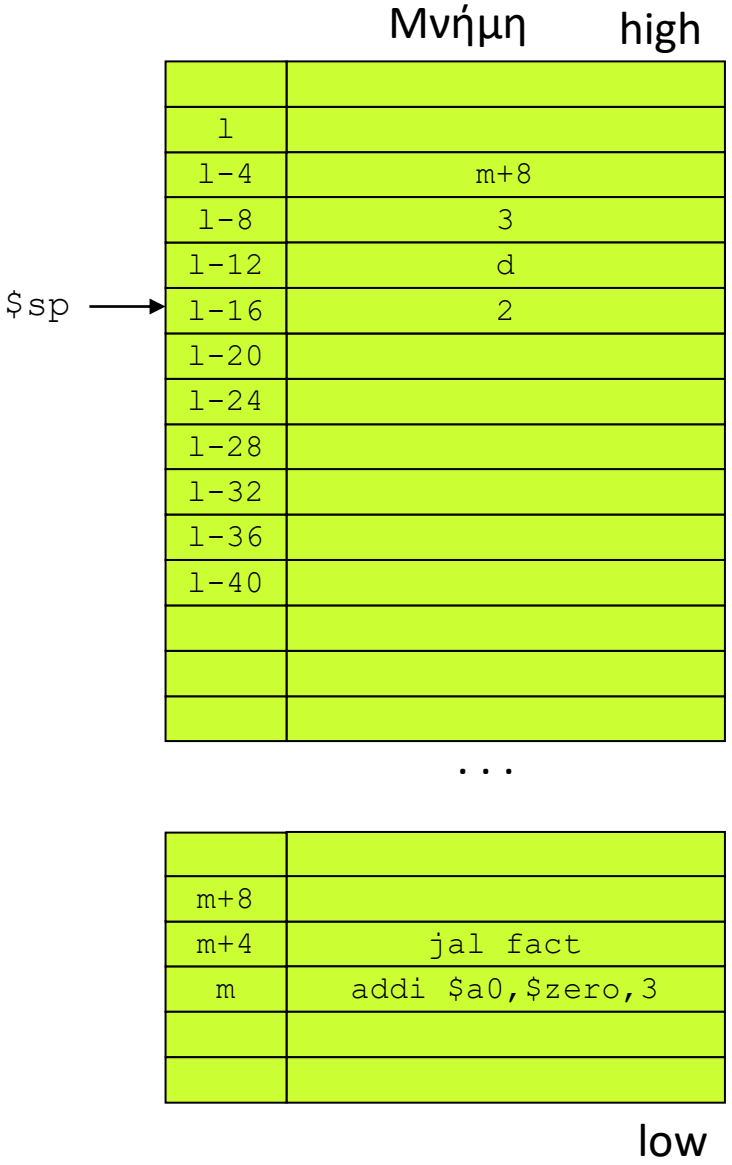
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC
$sp: l-16
$ra: d
$v0:
$a0: 2
$t0: 0
```



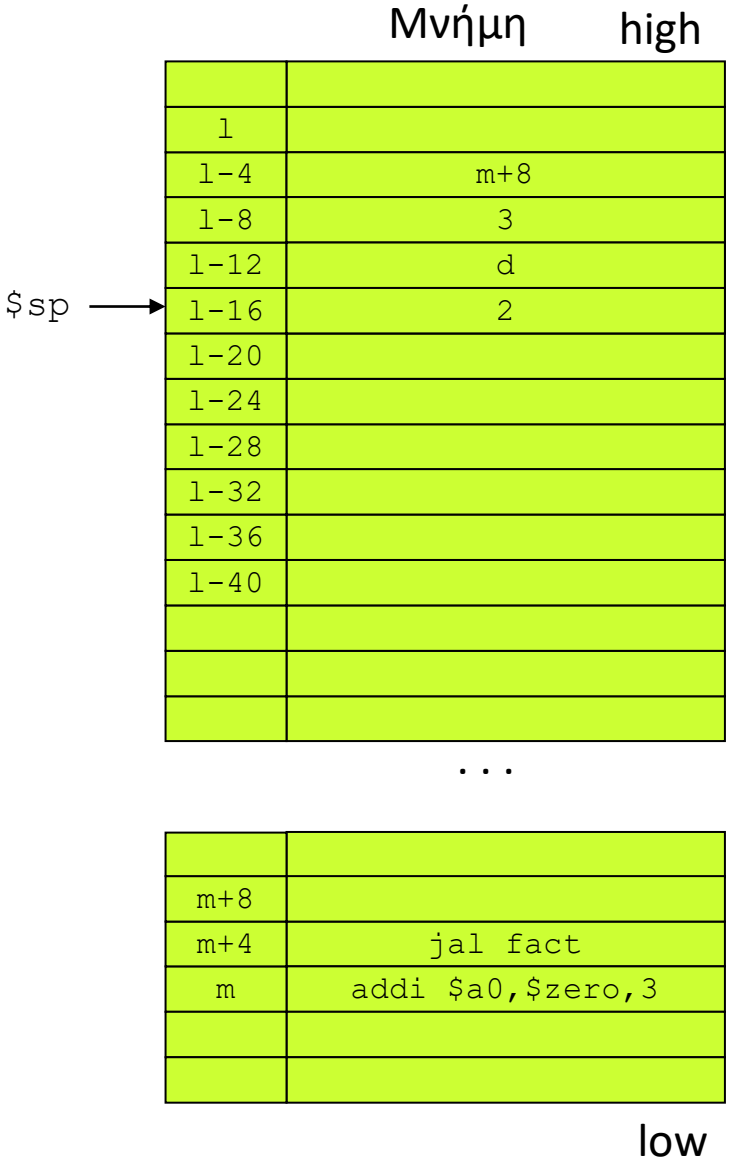
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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 ← PC
 jal fact
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0:

$a0: 2

$t0: 0
```



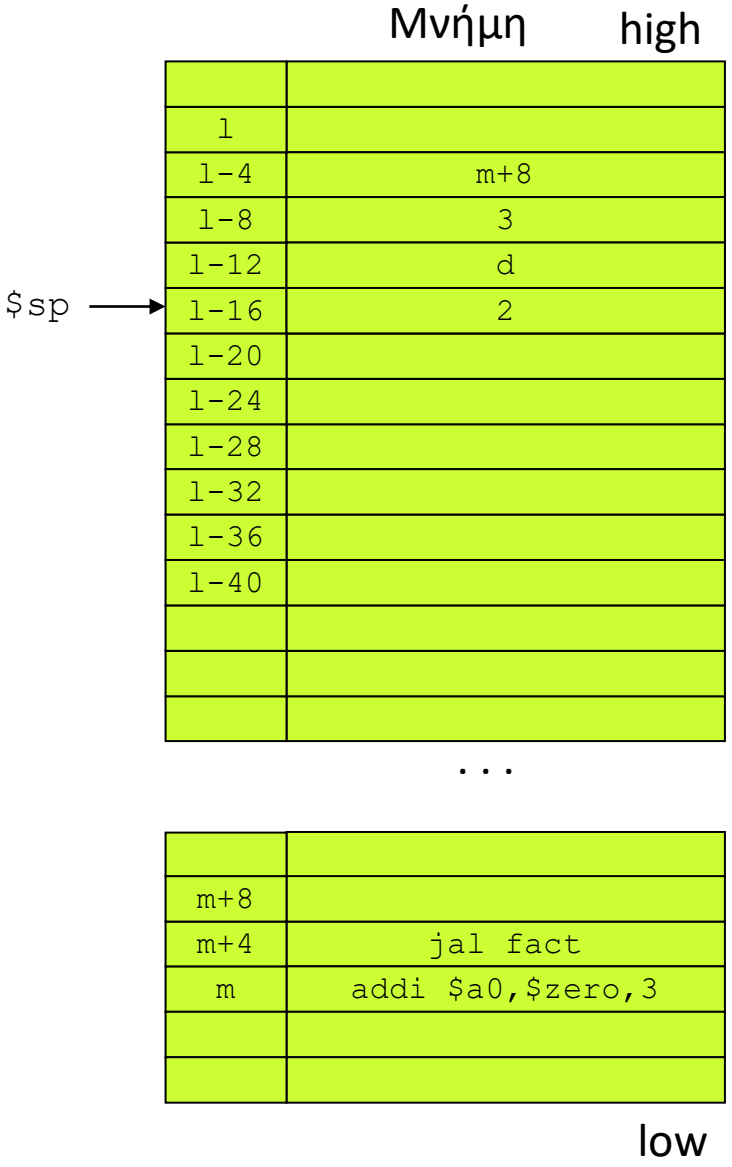
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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 ← PC
 jal fact
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0:

$a0: 1

$t0: 0
```



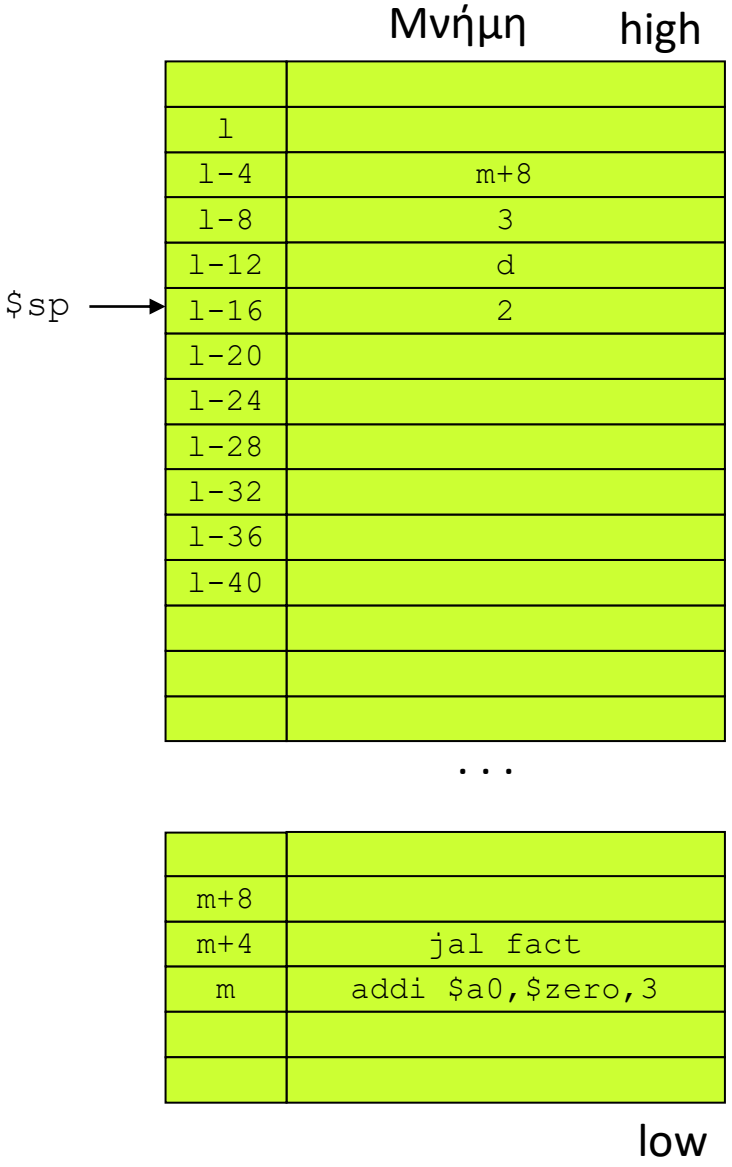
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
  addi $sp,$sp,-8
  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 ← PC
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

```
PC:  βλ. PC
$sp: l-16
$ra: d
$v0:
$a0: 1
$t0: 0
```



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
  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 ← PC
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

PC: βλ. PC

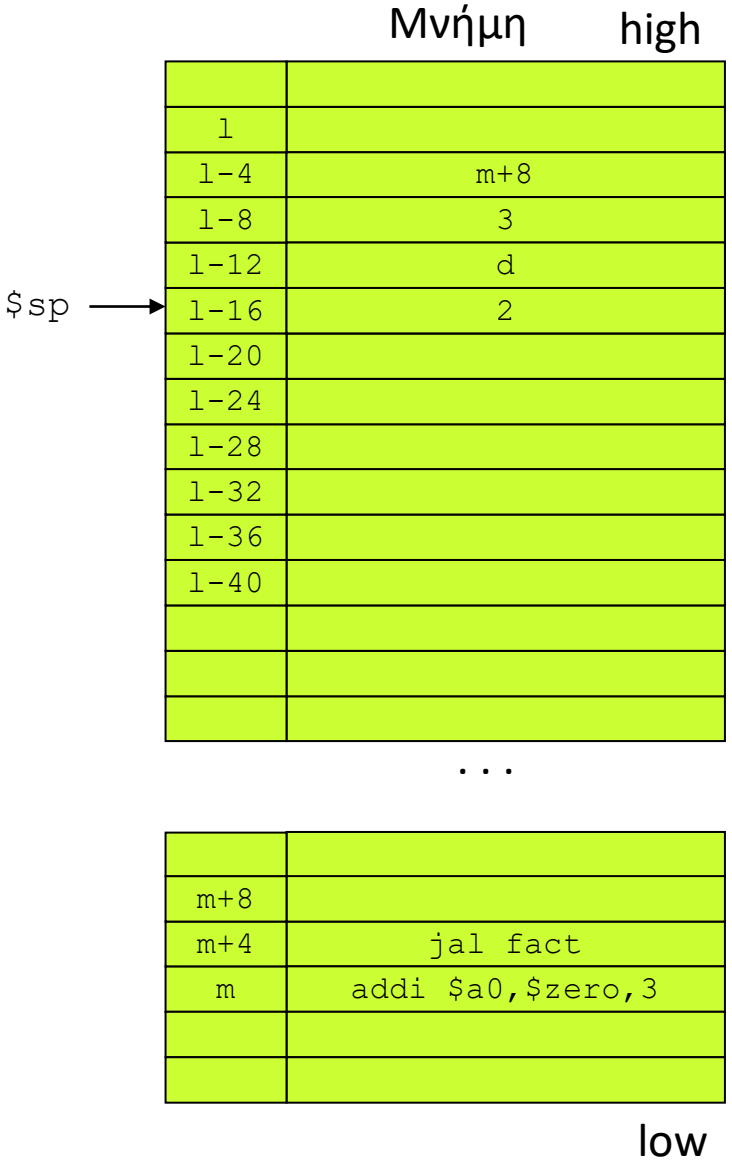
$sp: l-16

$ra: d

$v0:

$a0: 1

$t0: 0



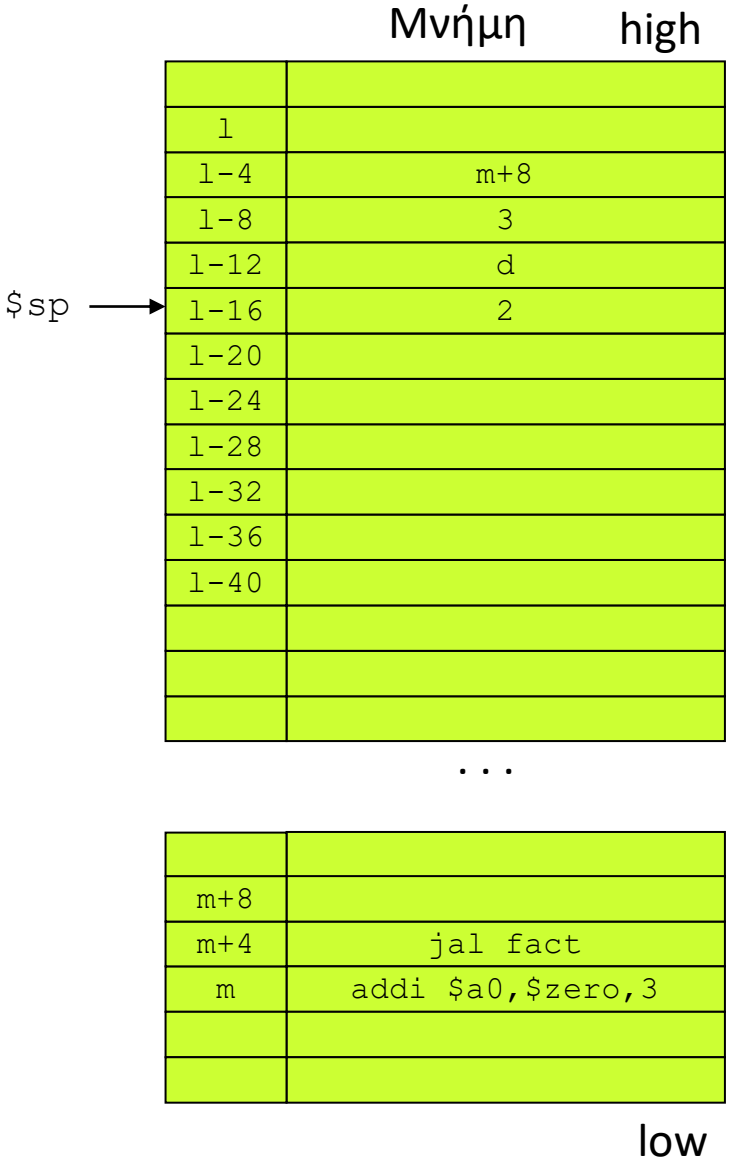
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

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
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC
$sp: l-16
$ra: d
$v0:
$a0: 1
$t0: 0
```



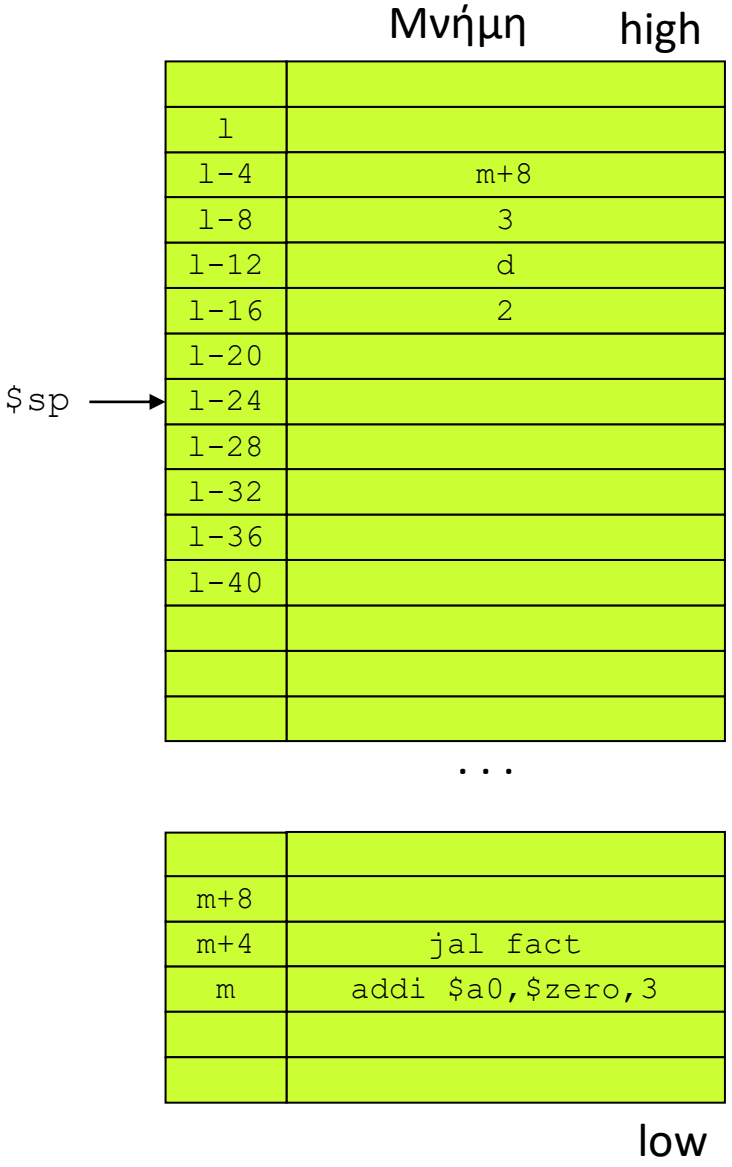
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

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
  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
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

```
PC:  βλ. PC
$sp: l-24
$ra: d
$v0:
$a0: 1
$t0: 0
```



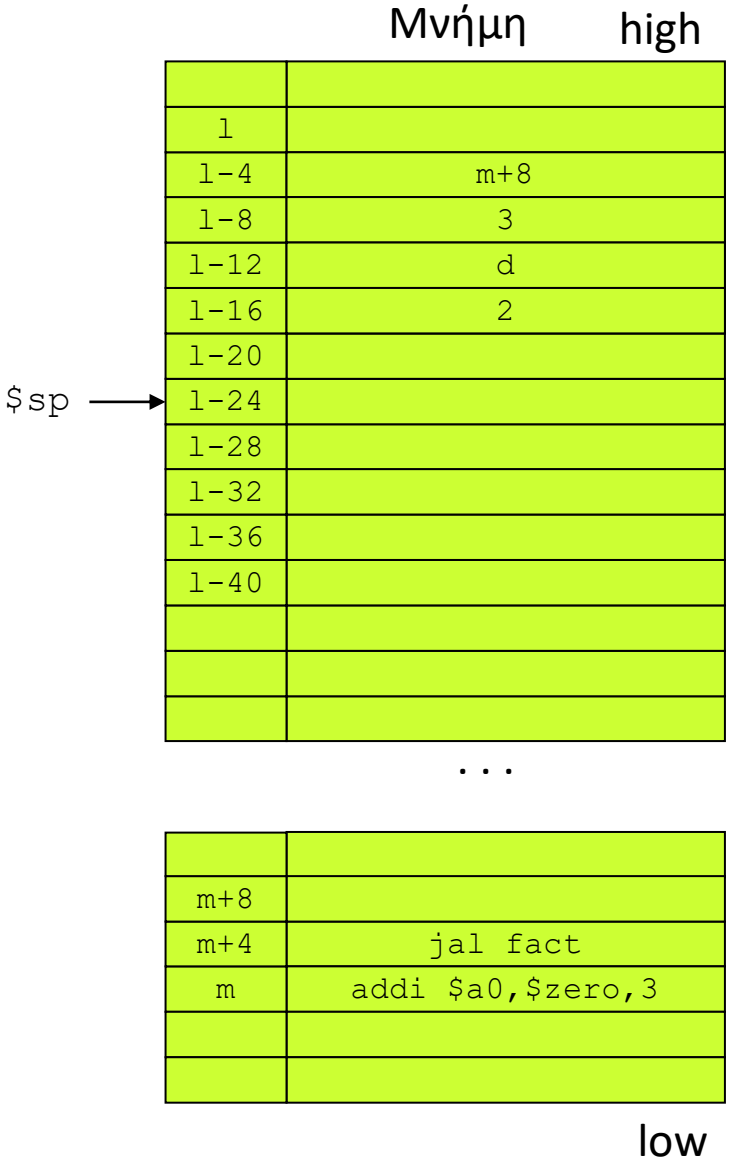
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 sw $ra,4($sp) #διεύθ. επιστροφής ← PC
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC
$sp: l-24
$ra: d
$v0:
$a0: 1
$t0: 0
```



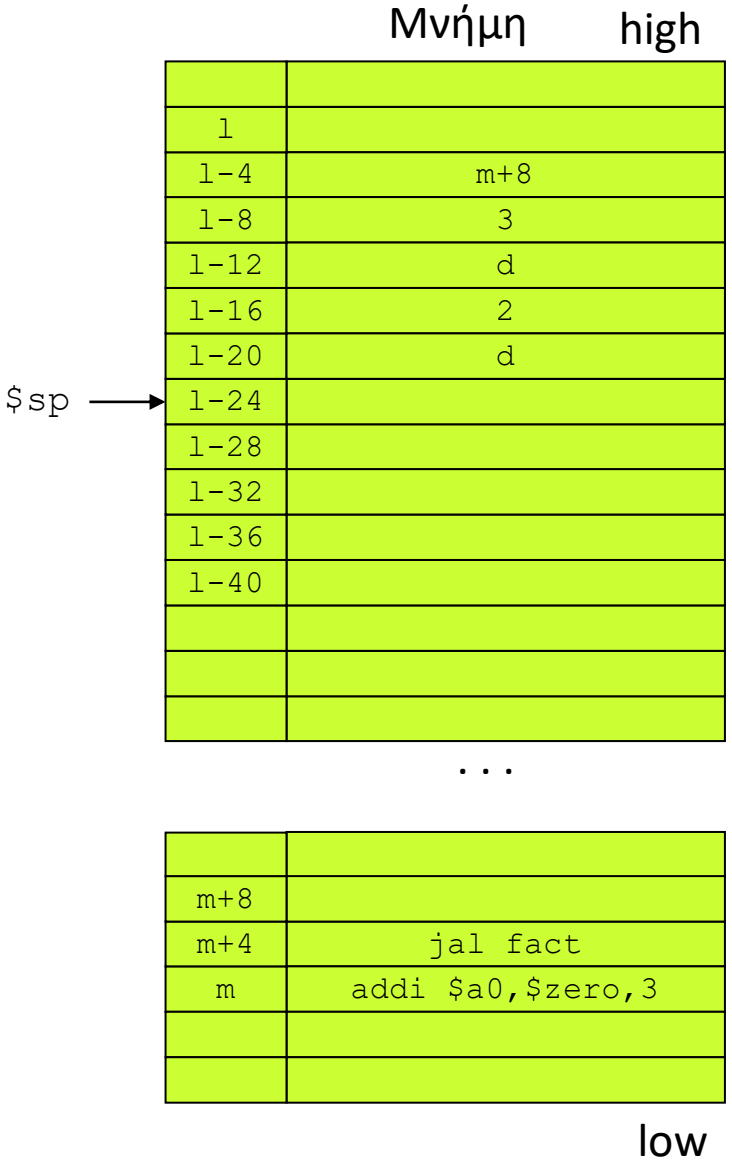
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 sw $ra,4($sp) #διεύθ. επιστροφής ← PC
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0:

$a0: 1

$t0: 0
```



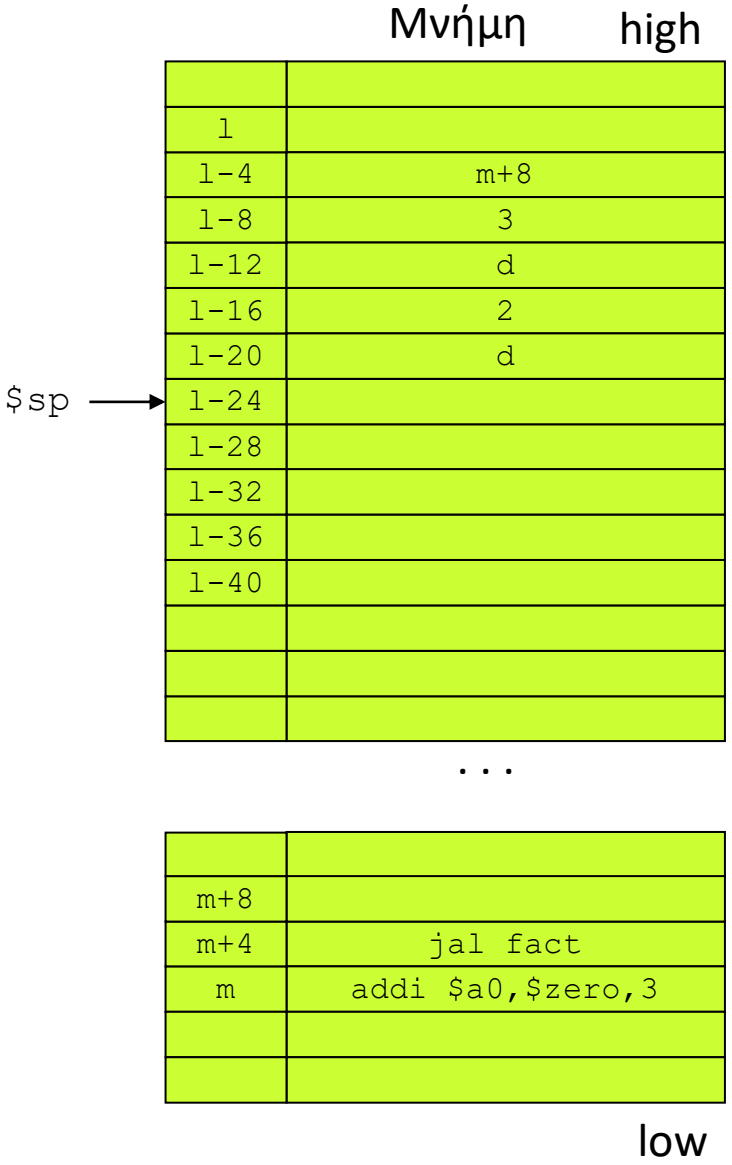
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0:

$a0: 1

$t0: 0
```



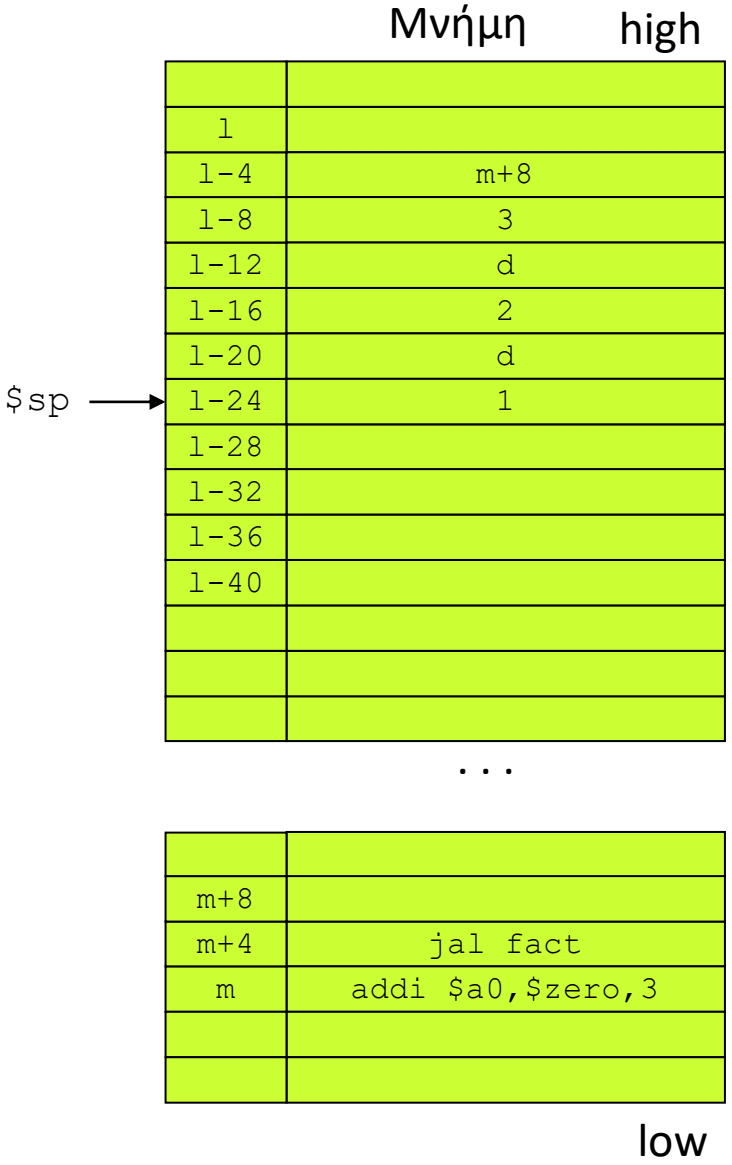
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0:

$a0: 1

$t0: 0
```



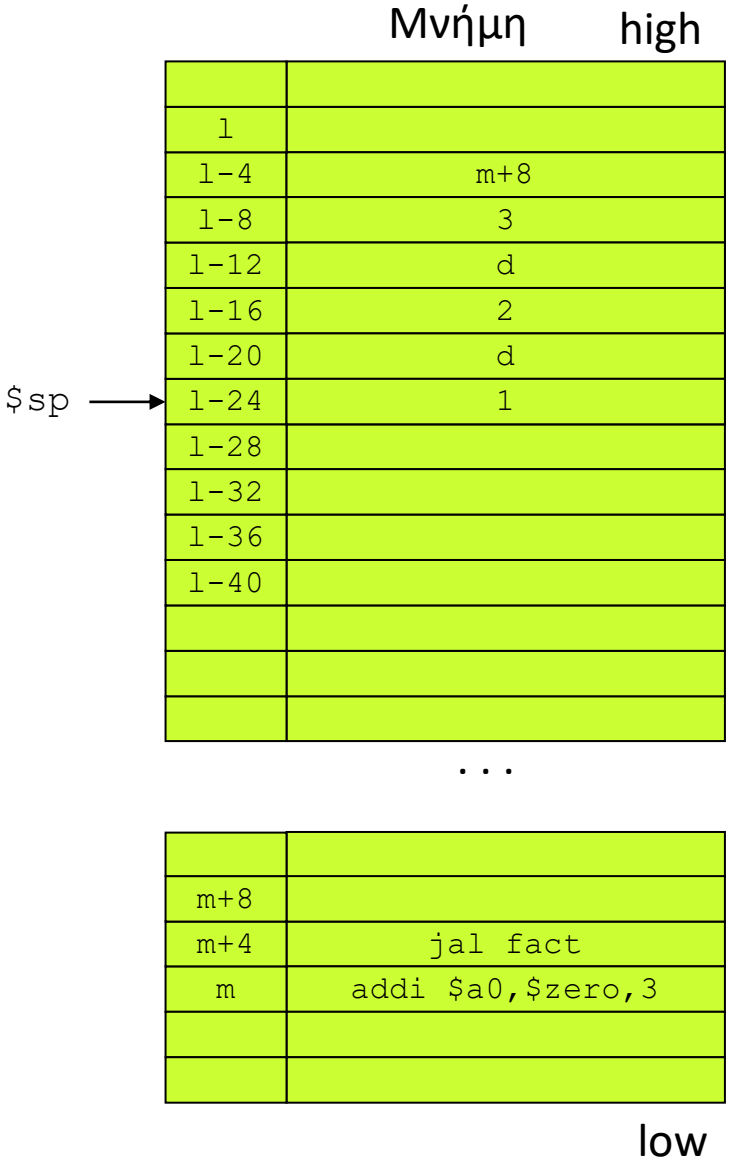
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0:

$a0: 1

$t0: 0
```



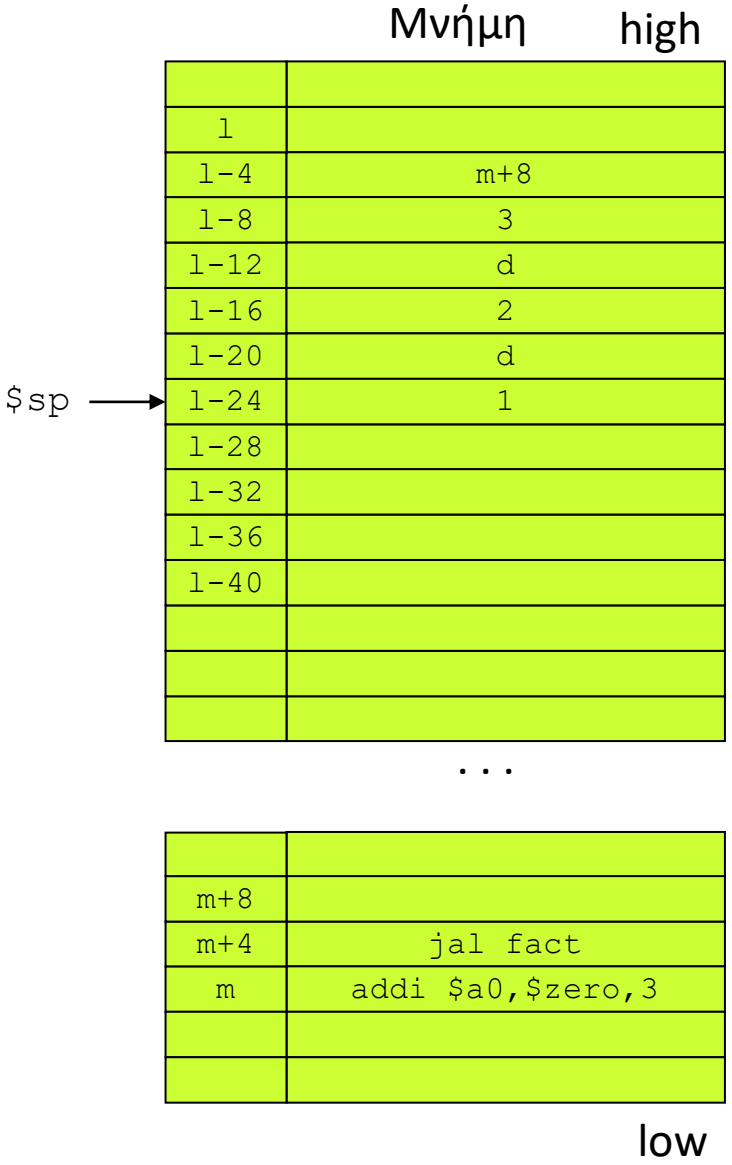
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0:

$a0: 1

$t0: 0
```



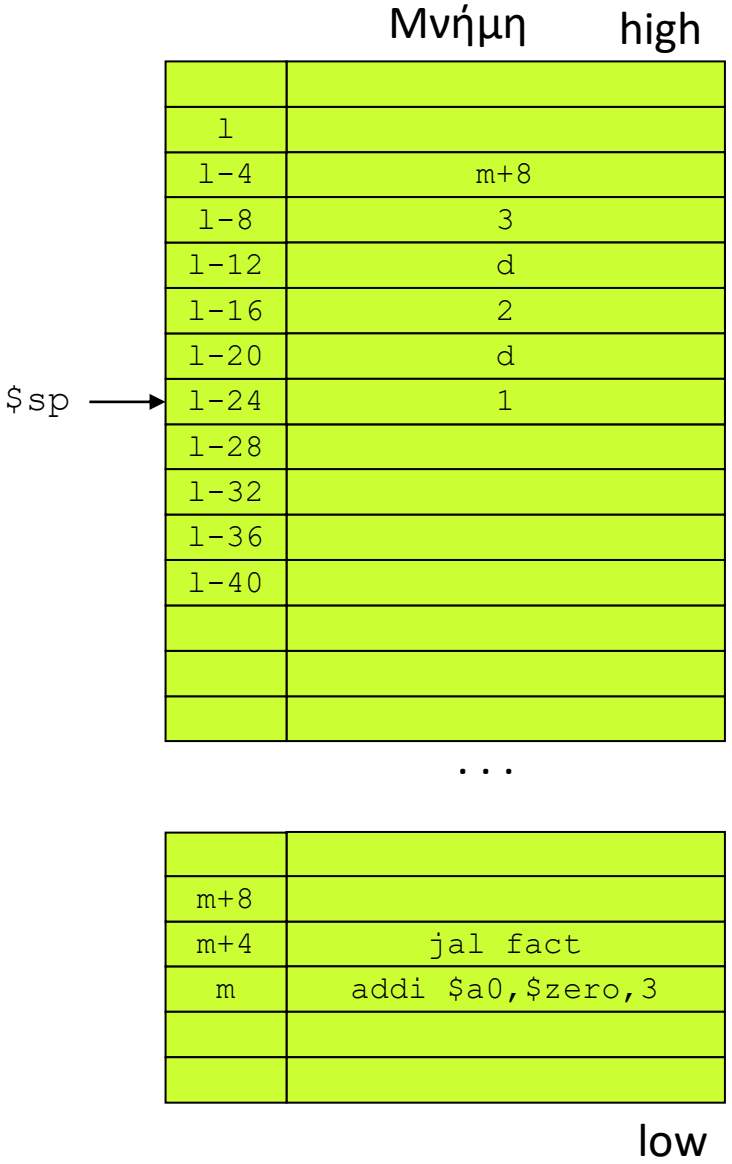
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC
$sp: l-24
$ra: d
$v0:
$a0: 1
$t0: 0
```



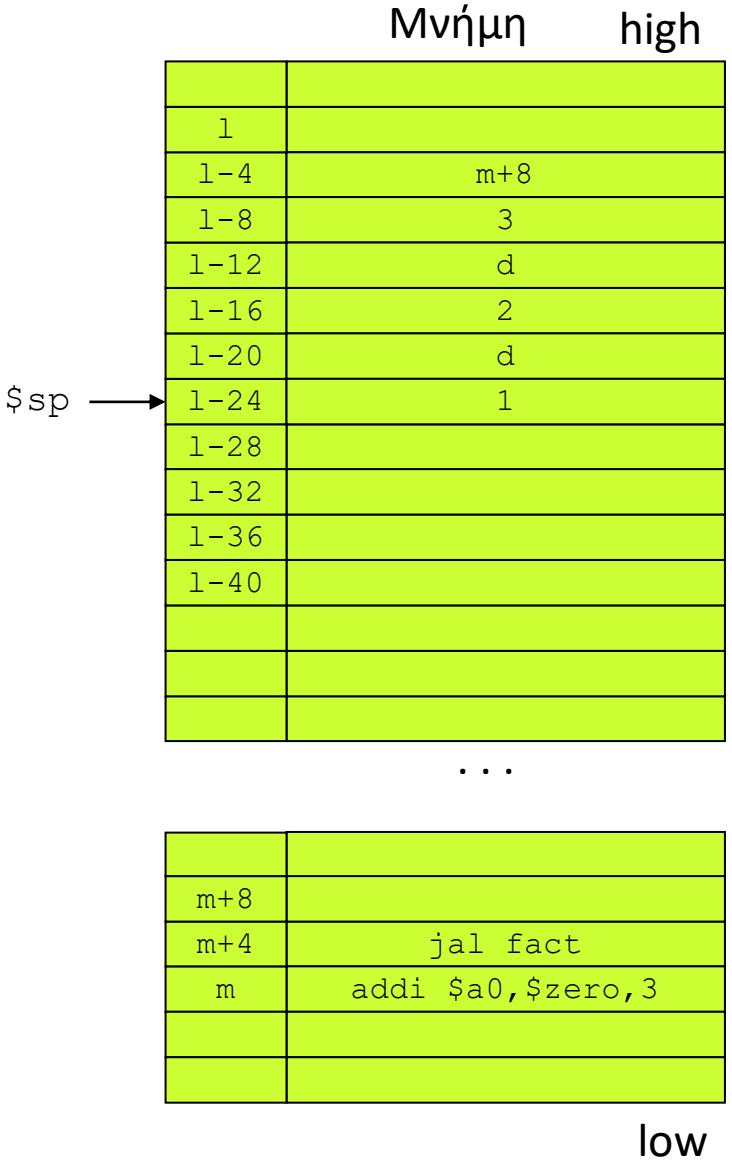
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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 ← PC
 jal fact
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0:

$a0: 1

$t0: 0
```



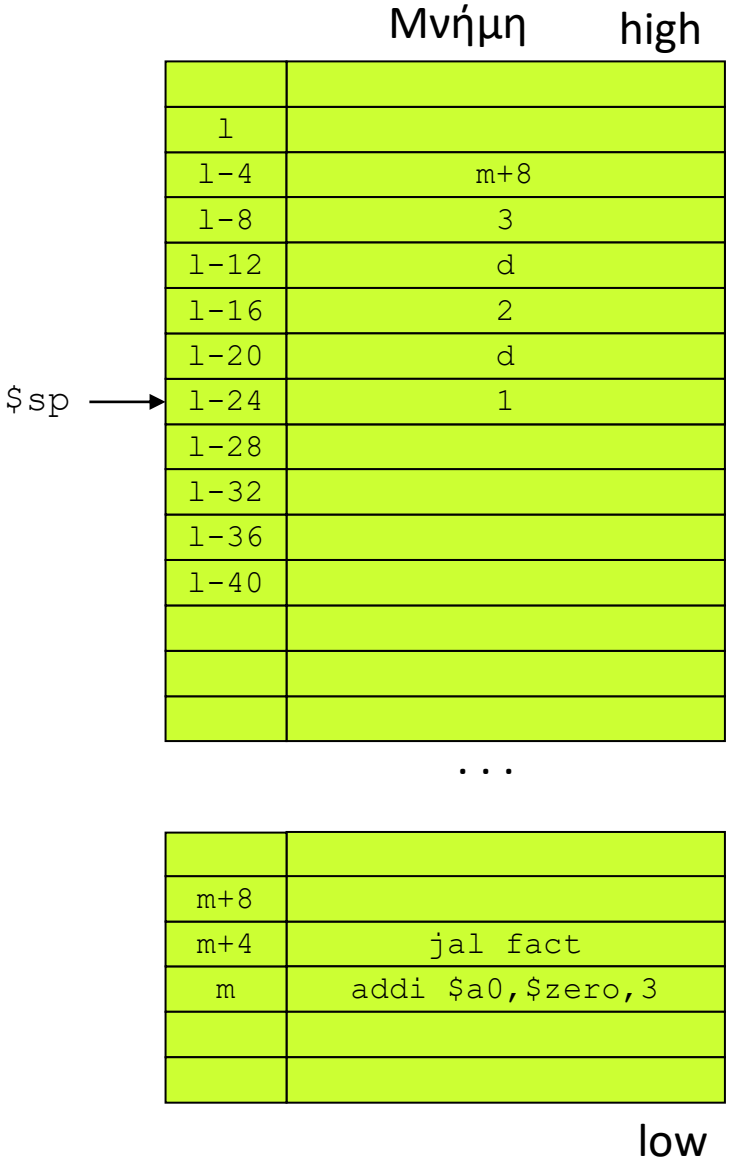
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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 ← PC
 jal fact
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0:

$a0: 0

$t0: 0
```



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
  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 ← PC
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

PC: βλ. PC

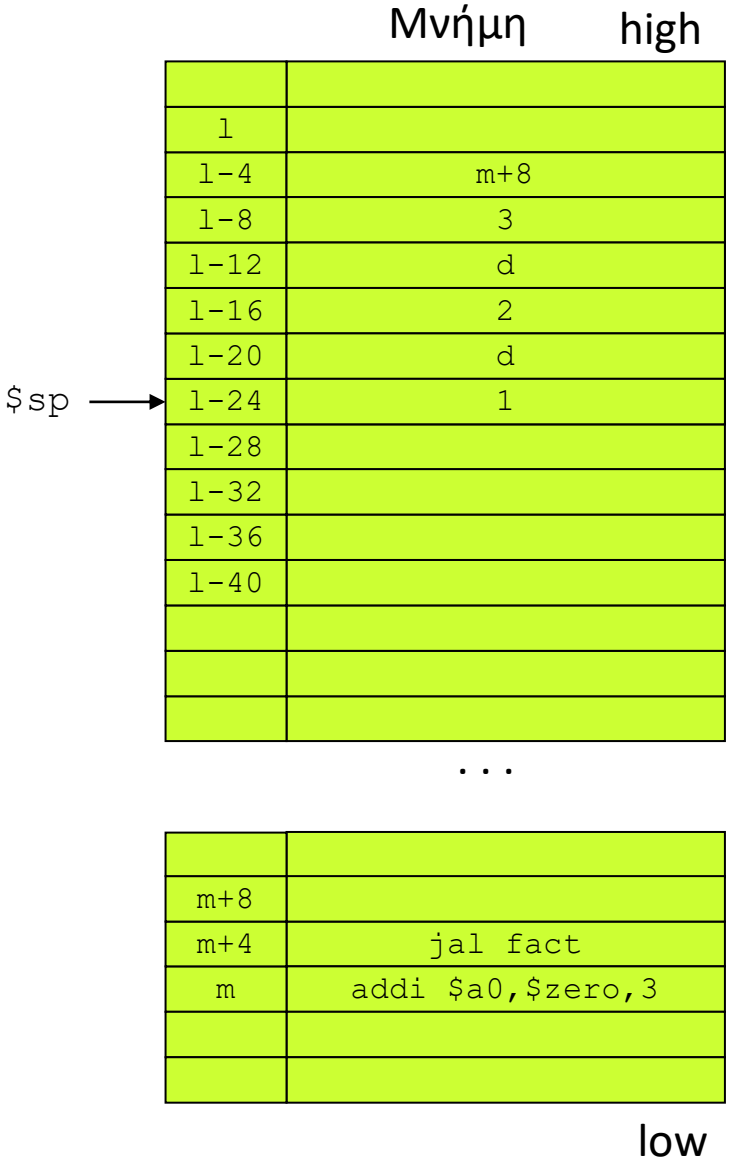
$sp: l-24

$ra: d

$v0:

$a0: 0

$t0: 0



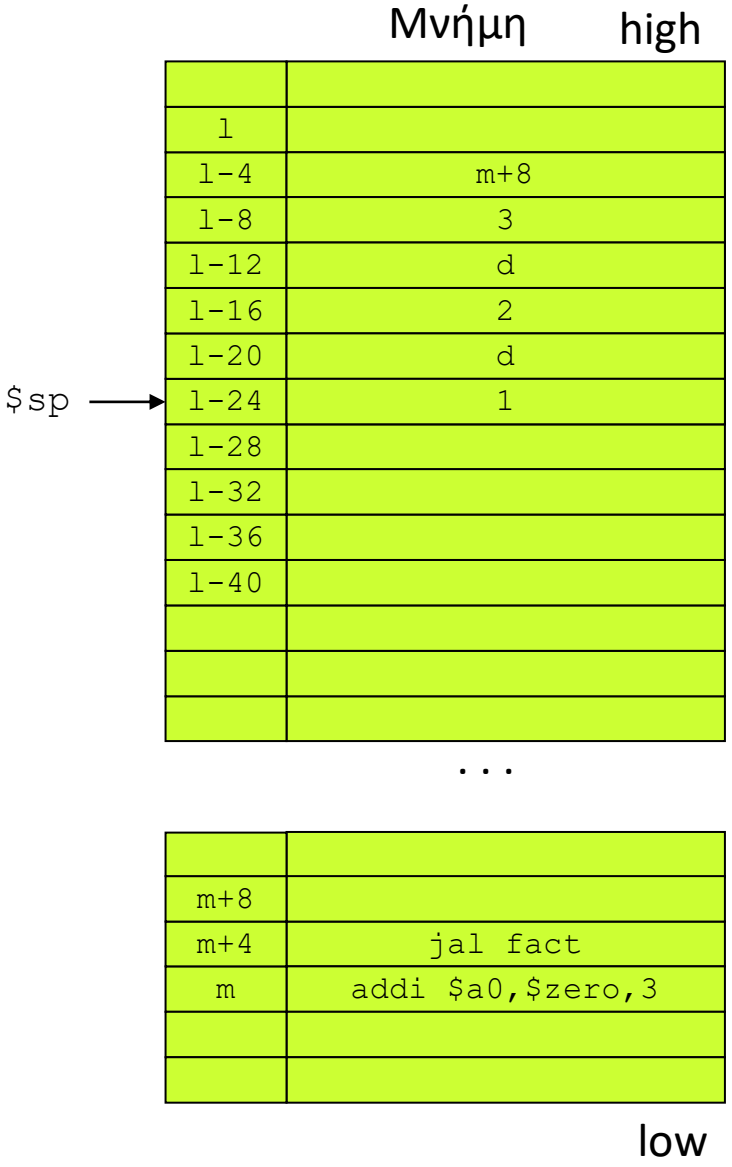
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
  addi $sp,$sp,-8
  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 ← PC
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0:

$a0: 0

$t0: 0
```



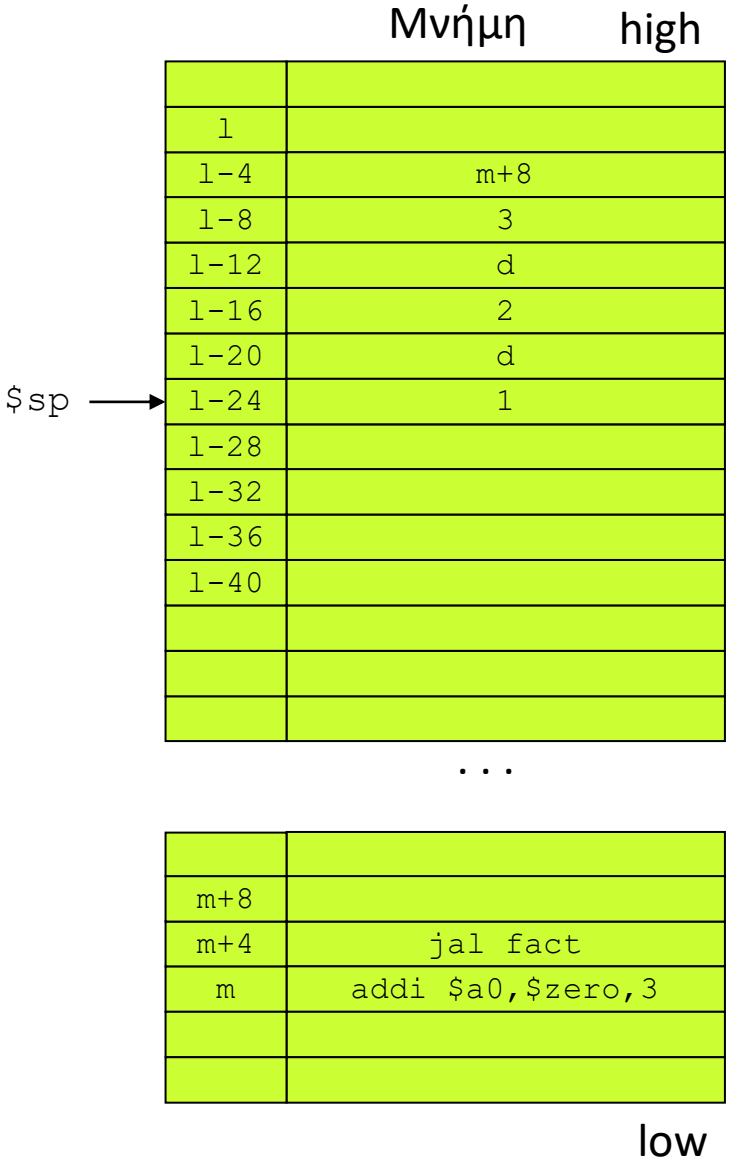
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

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
  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
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

```
PC:  βλ. PC
$sp: l-24
$ra: d
$v0:
$a0: 0
$t0: 0
```



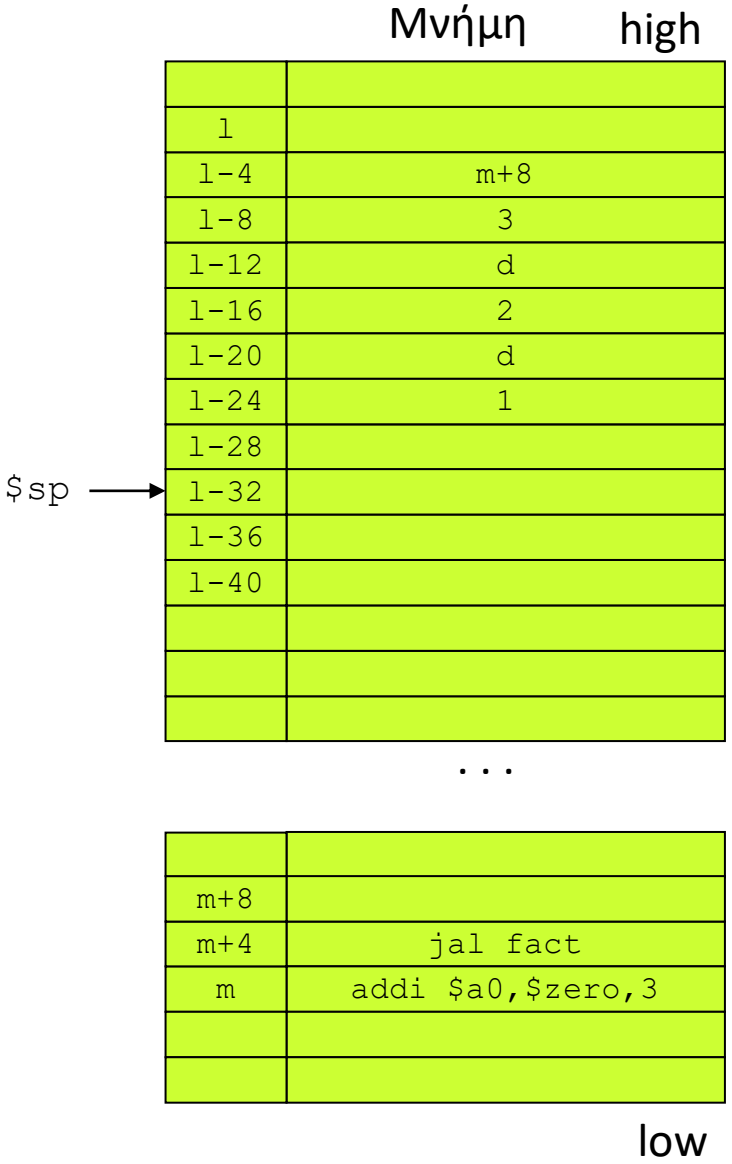
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

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
  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
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

```
PC:  βλ. PC

$sp: l-32

$ra: d

$v0:

$a0: 0

$t0: 0
```



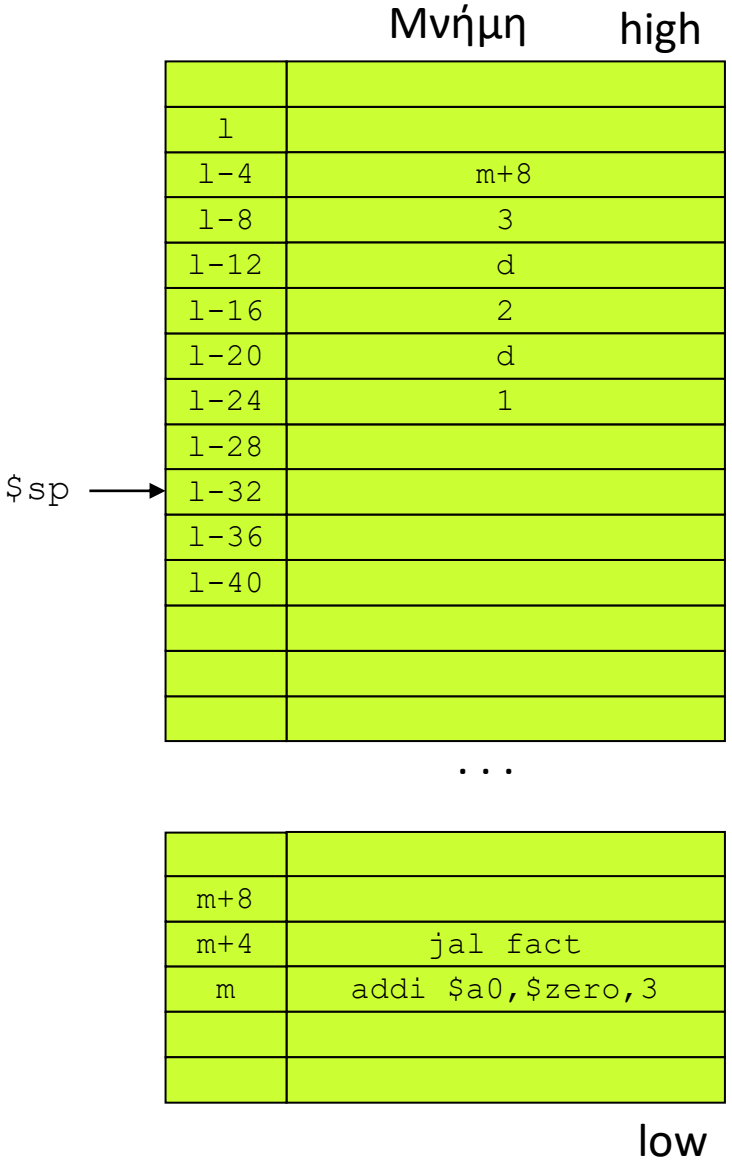
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 sw $ra,4($sp) #διεύθ. επιστροφής ← PC
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC
$sp: l-32
$ra: d
$v0:
$a0: 0
$t0: 0
```



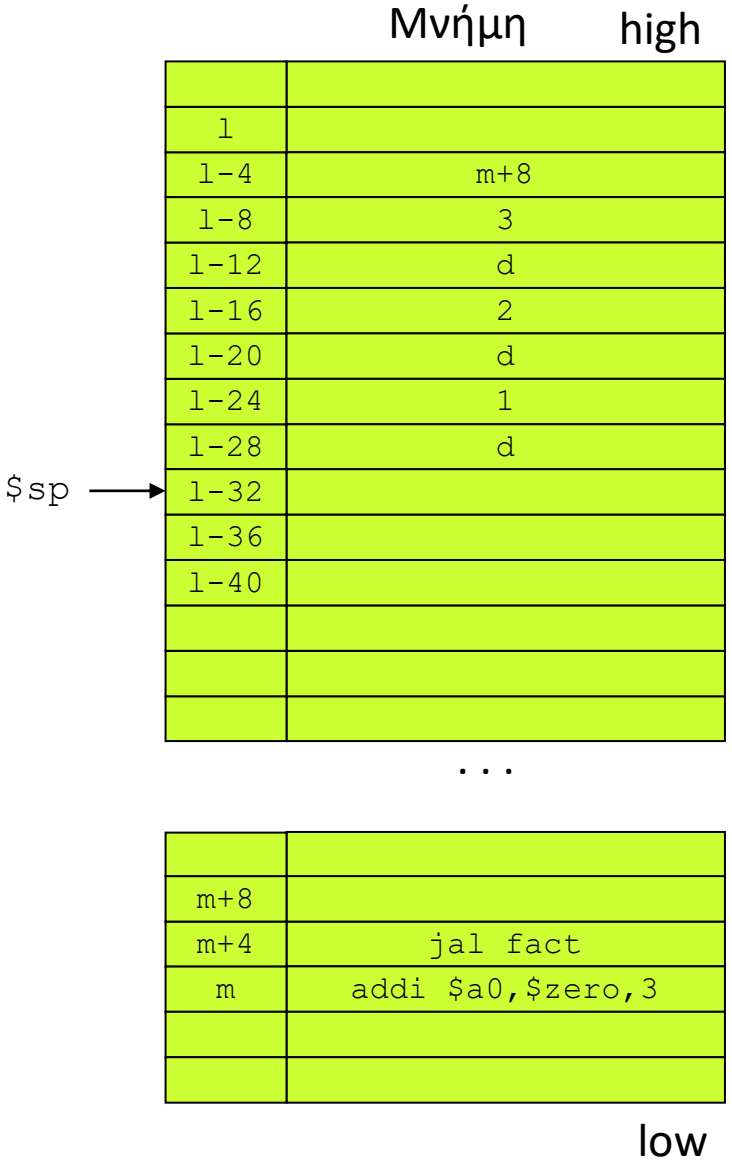
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 sw $ra,4($sp) #διεύθ. επιστροφής ← PC
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC
$sp: l-32
$ra: d
$v0:
$a0: 0
$t0: 0
```



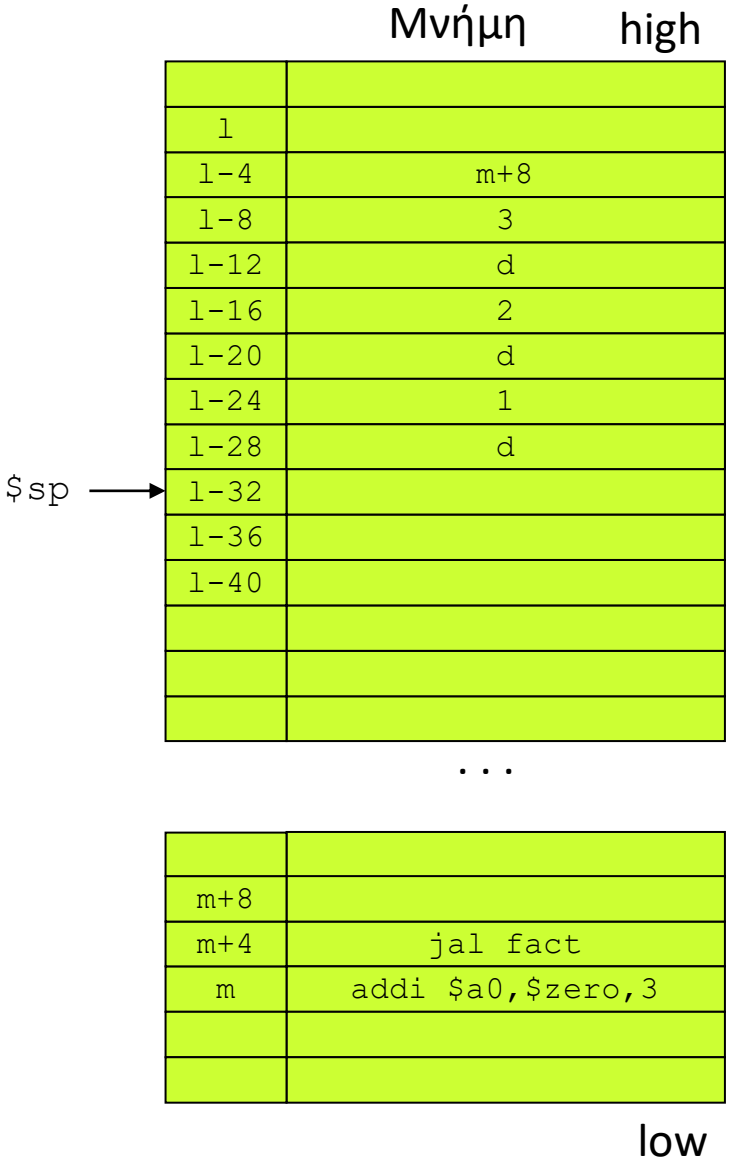
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-32

$ra: d

$v0:

$a0: 0

$t0: 0
```



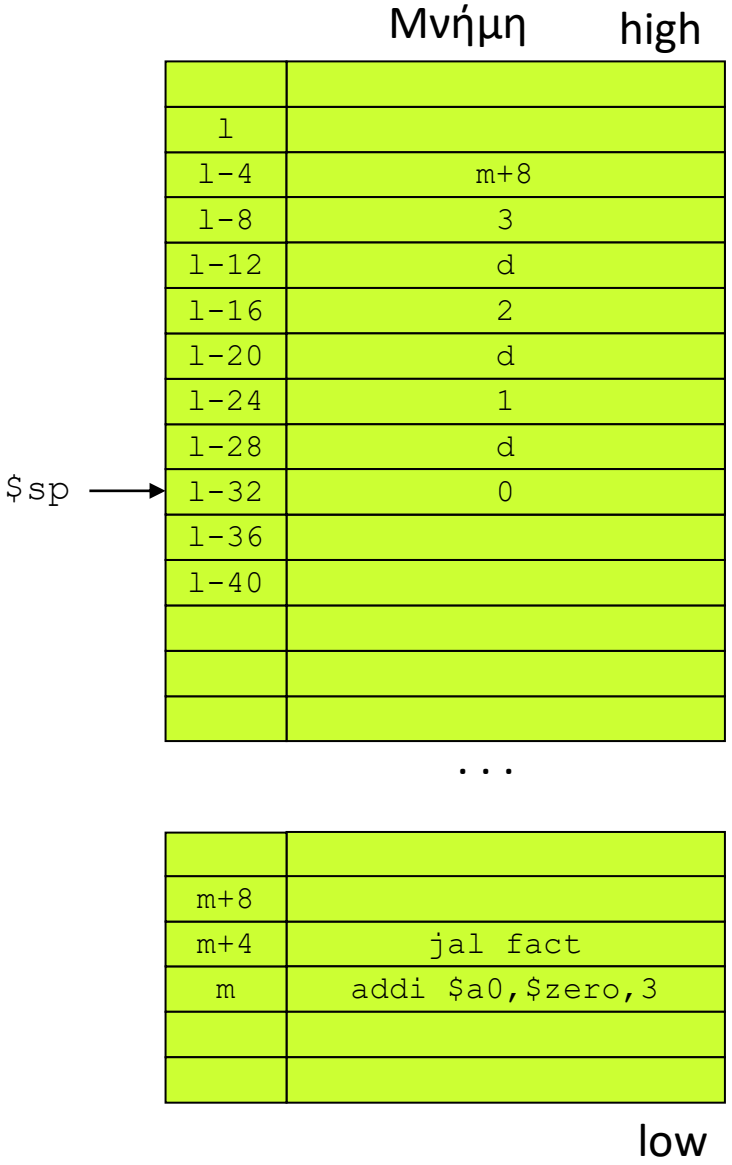
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-32

$ra: d

$v0:

$a0: 0

$t0: 0
```



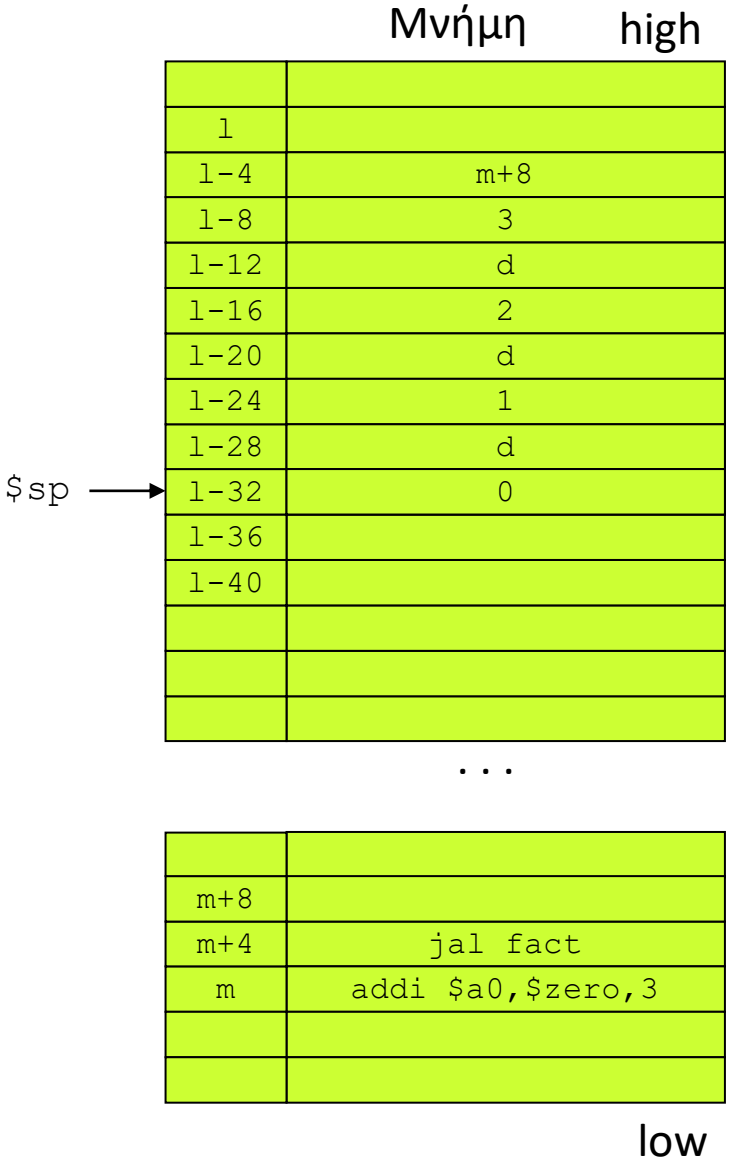
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-32

$ra: d

$v0:

$a0: 0

$t0: 0
```



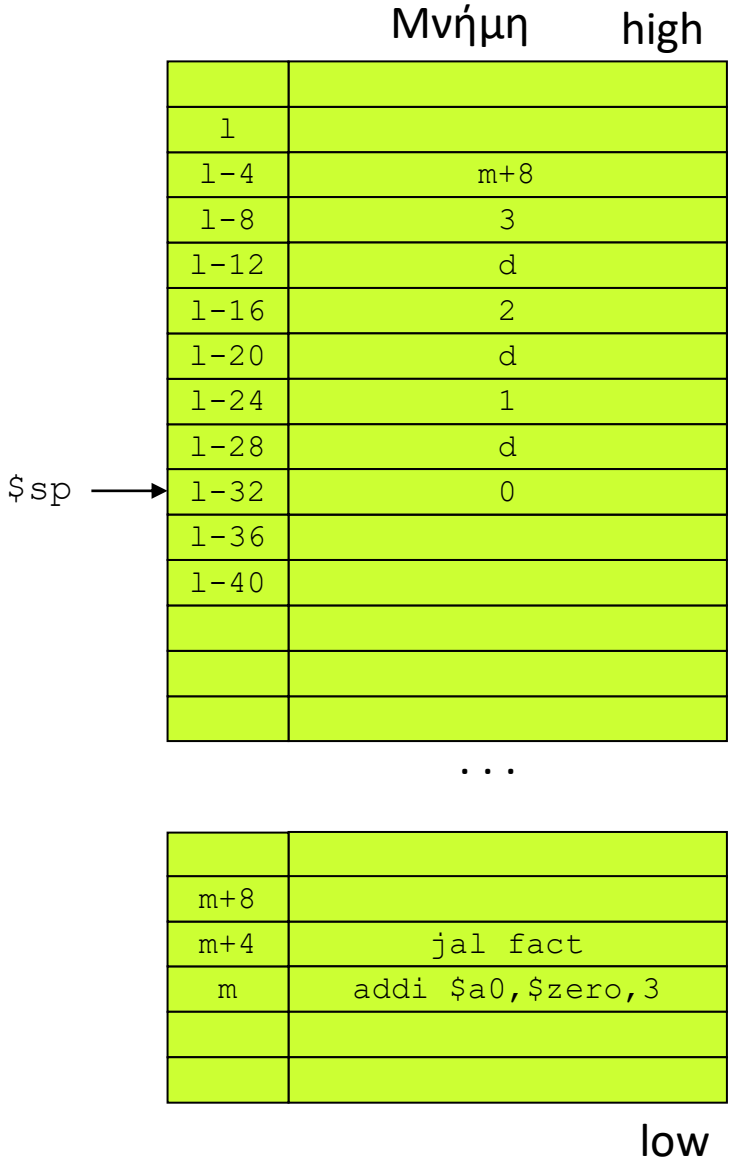
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC
$sp: l-32
$ra: d
$v0:
$a0: 0
$t0: 1
```



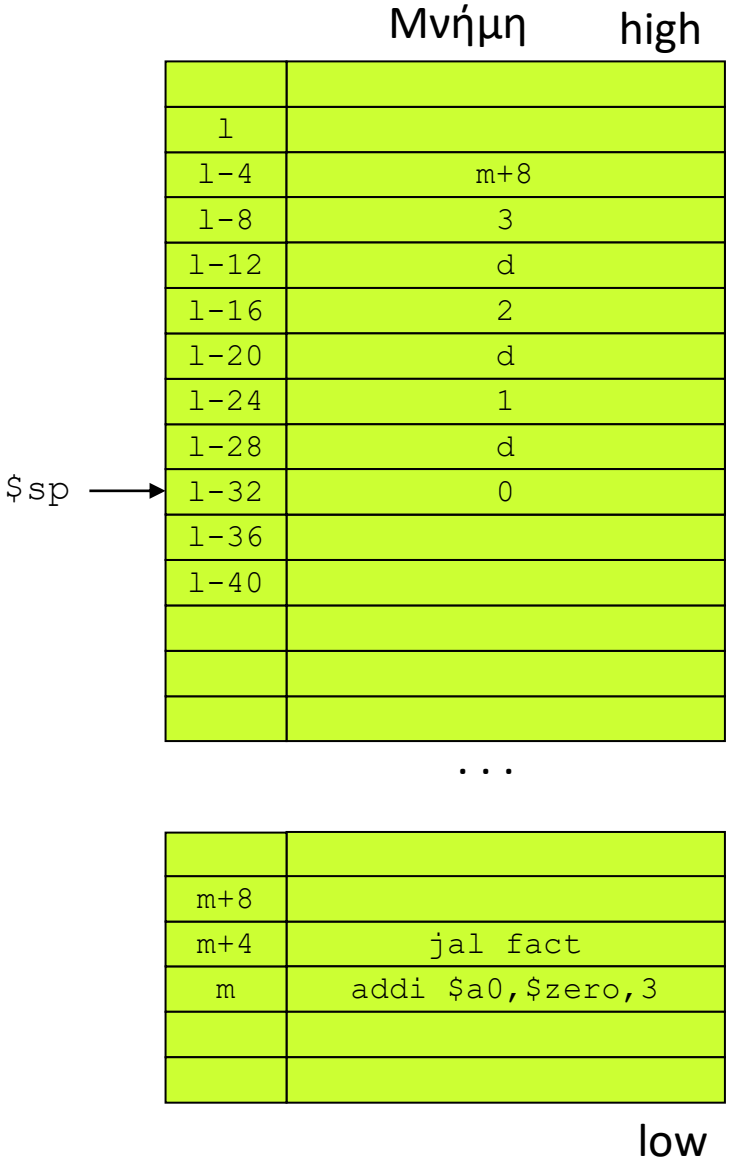
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC
$sp: l-32
$ra: d
$v0:
$a0: 0
$t0: 1
```



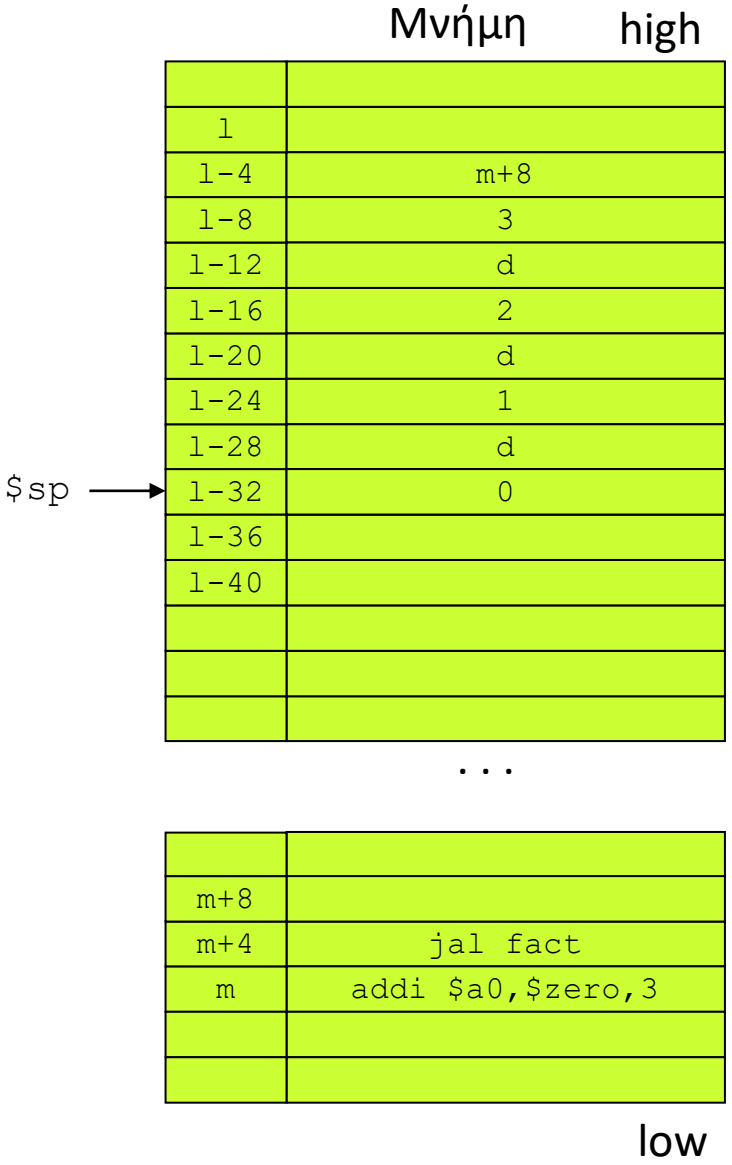
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-32

$ra: d

$v0:

$a0: 0

$t0: 1
```



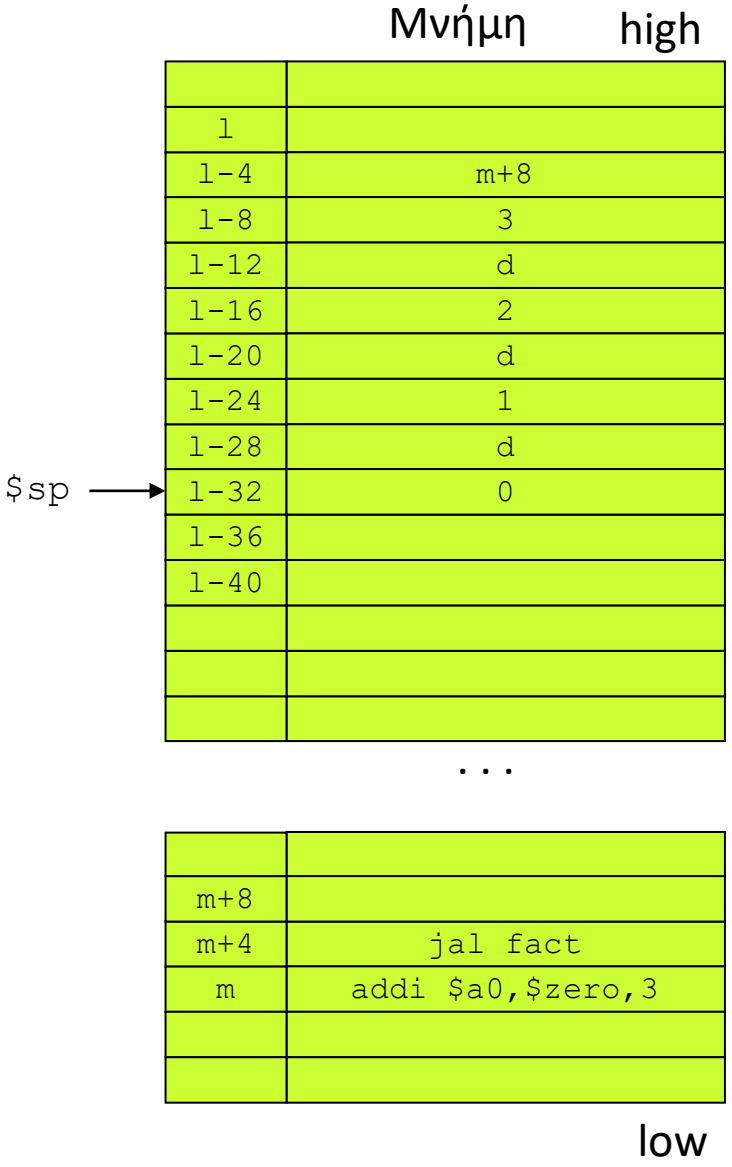
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

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

```
PC:  βλ. PC

$sp: l-32

$ra: d

$v0: 1

$a0: 0

$t0: 1
```



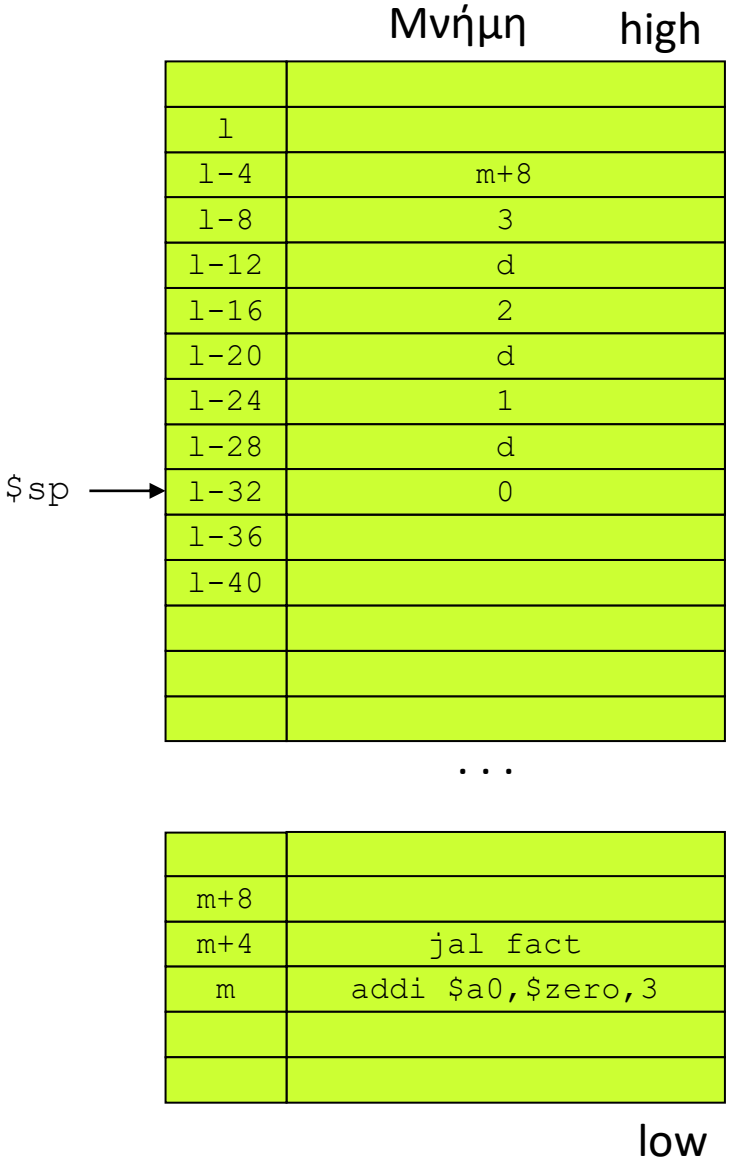
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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   ← PC
 jr $ra
L1:
 addi $a0,$a0,-1
 jal fact
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-32

$ra: d

$v0: 1

$a0: 0

$t0: 1
```



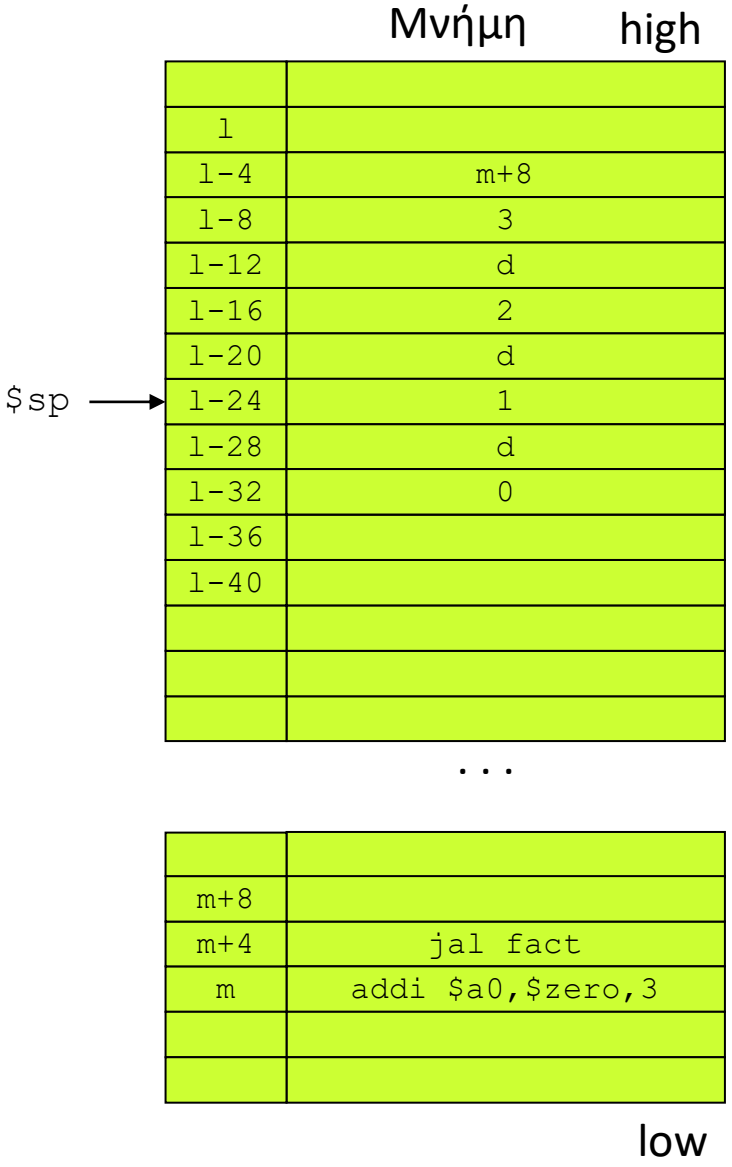
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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   ← PC
 jr $ra
L1:
 addi $a0,$a0,-1
 jal fact
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0: 1

$a0: 0

$t0: 1
```



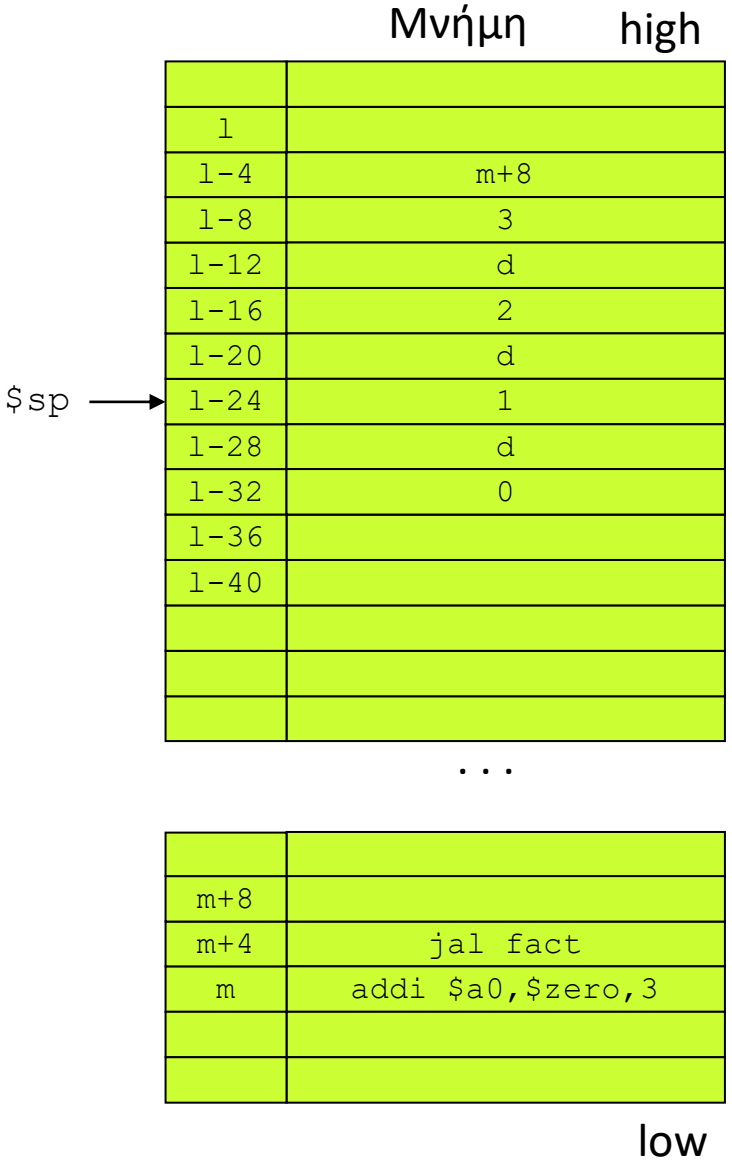
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
  addi $sp,$sp,-8
  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 ← PC
L1:
  addi $a0,$a0,-1
  jal fact
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra
```

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0: 1

$a0: 0

$t0: 1
```



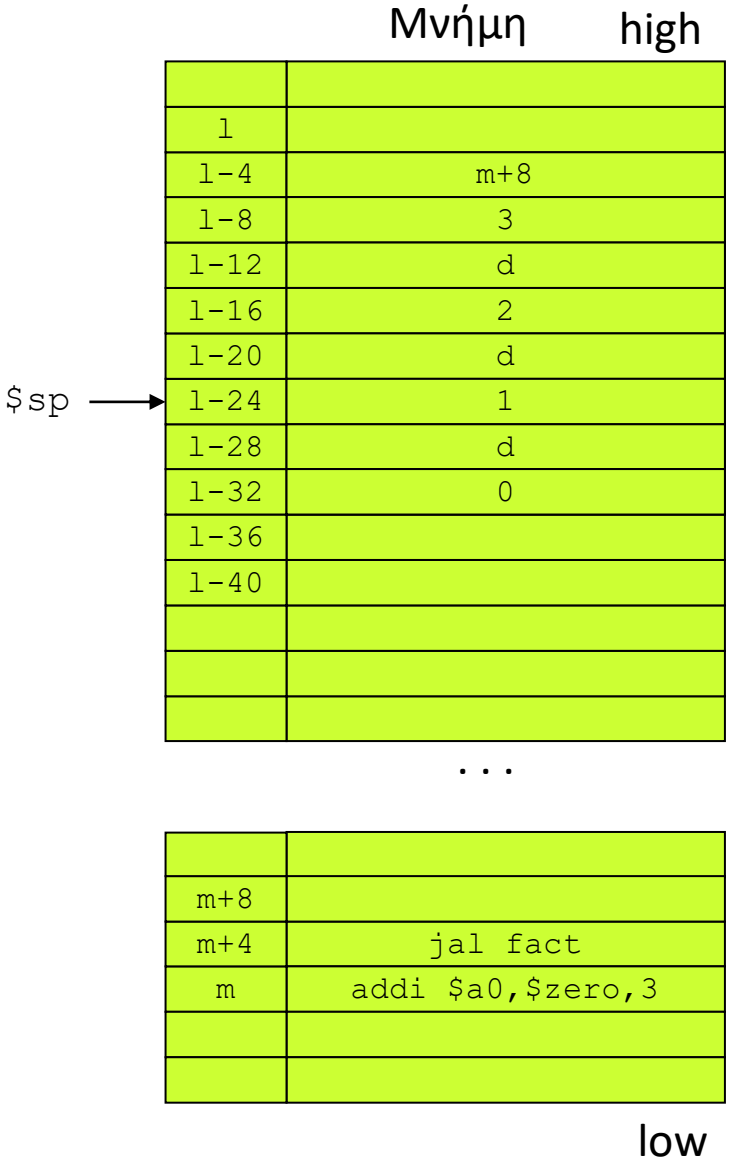
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d: lw $a0,0($sp)   ← PC
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0: 1

$a0: 0

$t0: 1
```



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
 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
d: lw $a0,0($sp)   ← PC
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

PC: βλ. PC

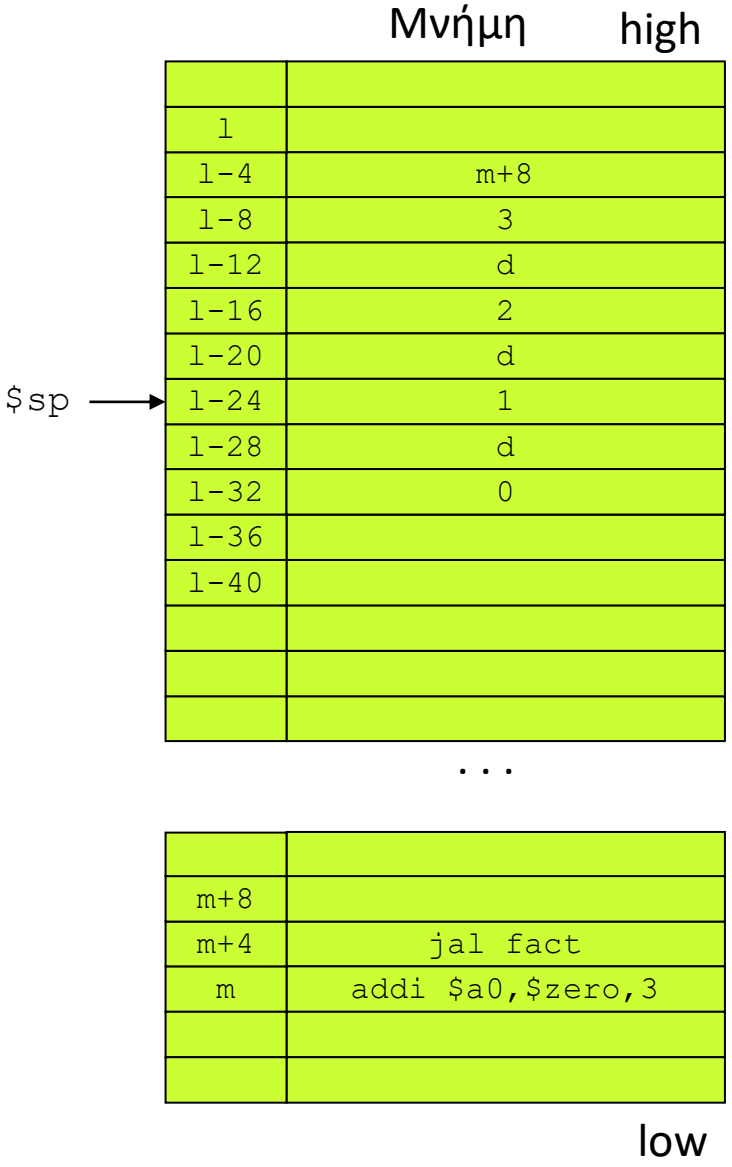
$sp: l-24

$ra: d

$v0: 1

$a0: 1

$t0: 1



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
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)   ← PC
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

PC: βλ. PC

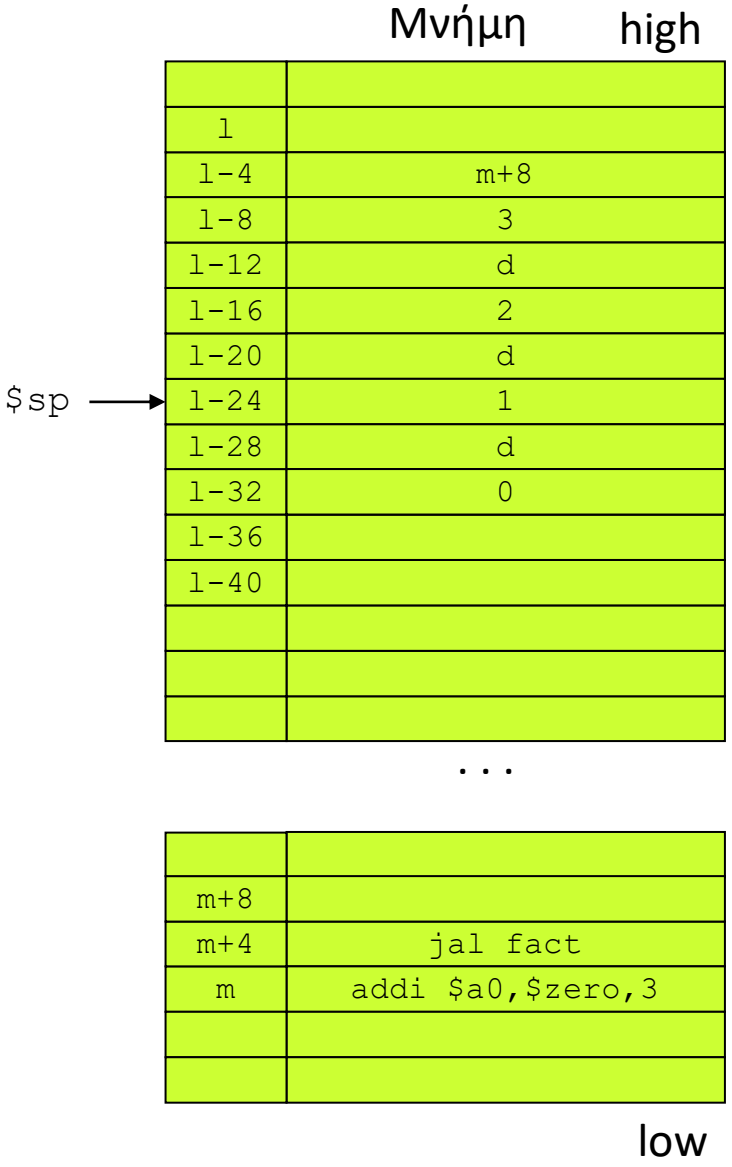
$sp: l-24

$ra: d

$v0: 1

$a0: 1

$t0: 1



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
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)   ← PC
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

PC: βλ. PC

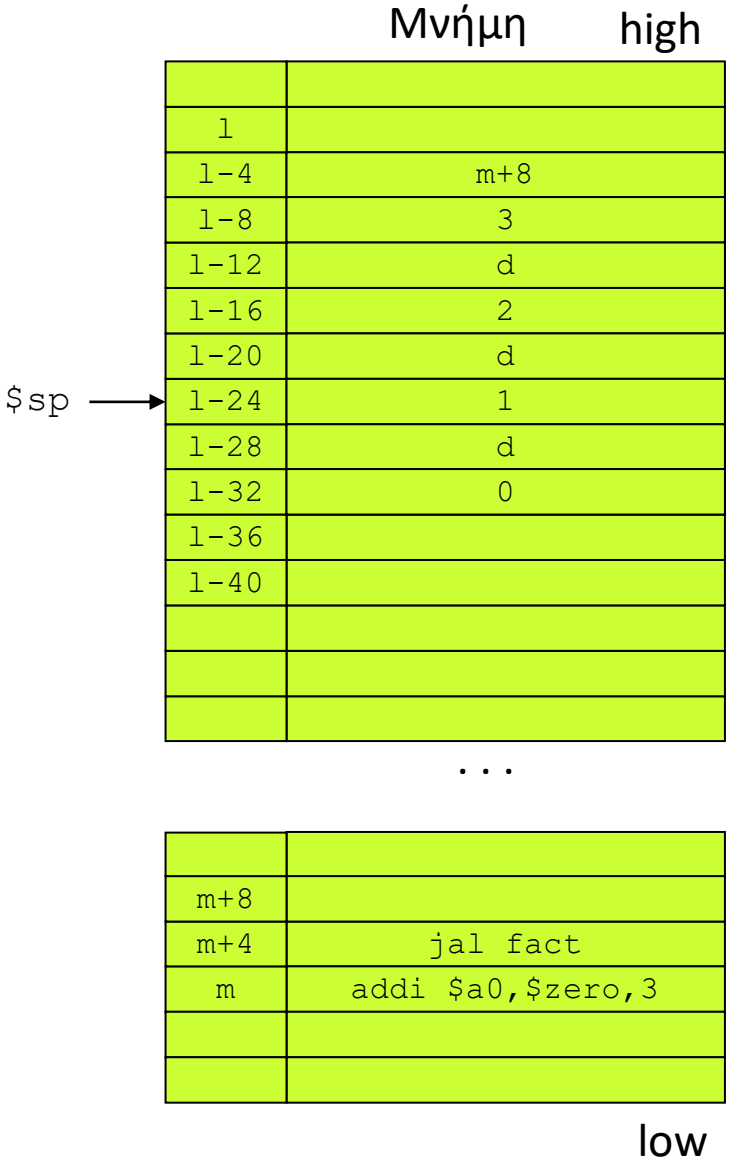
$sp: l-24

$ra: d

$v0: 1

$a0: 1

$t0: 1



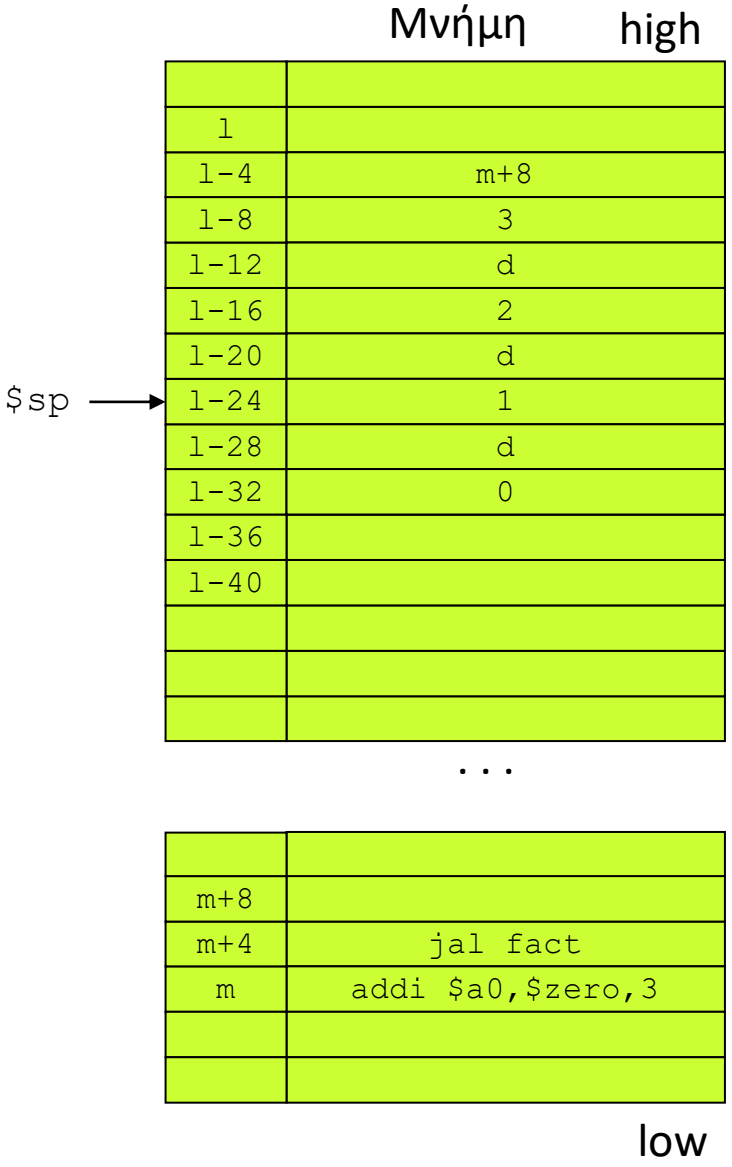
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
  addi $sp,$sp,-8
  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
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8   ← PC
  mul $v0,$a0,$v0
  jr $ra
```

```
PC:  βλ. PC

$sp: l-24

$ra: d

$v0: 1

$a0: 1

$t0: 1
```



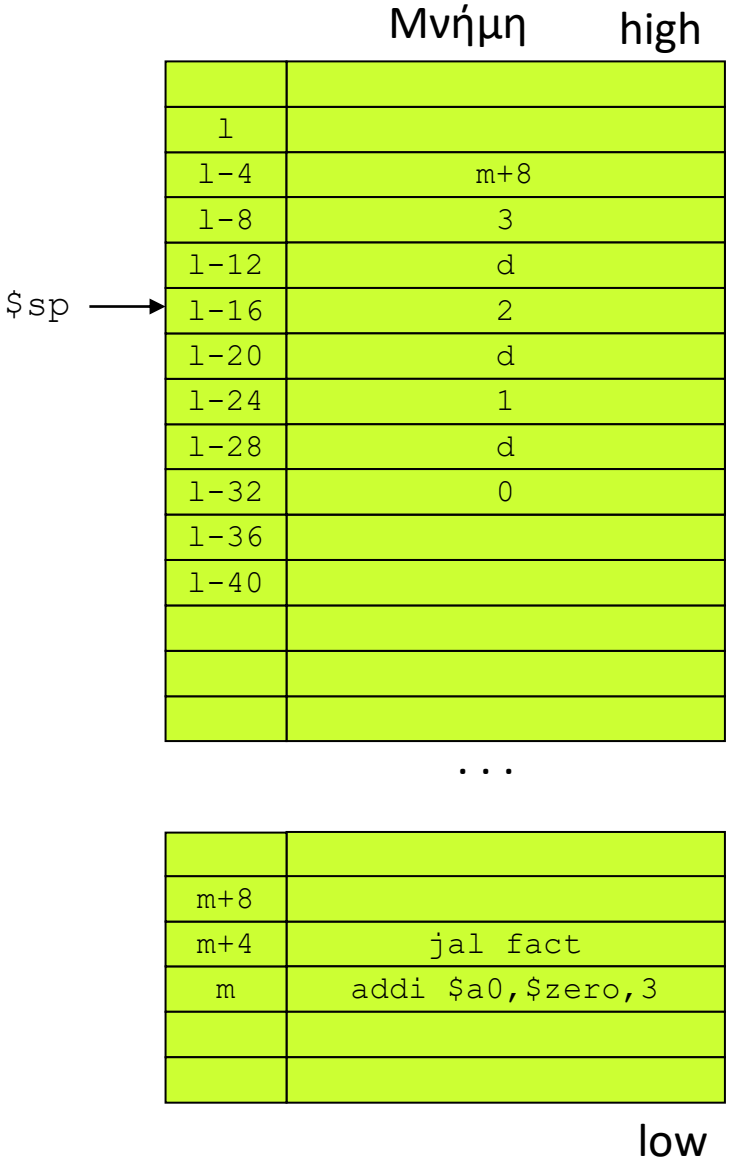
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8   ← PC
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0: 1

$a0: 1

$t0: 1
```



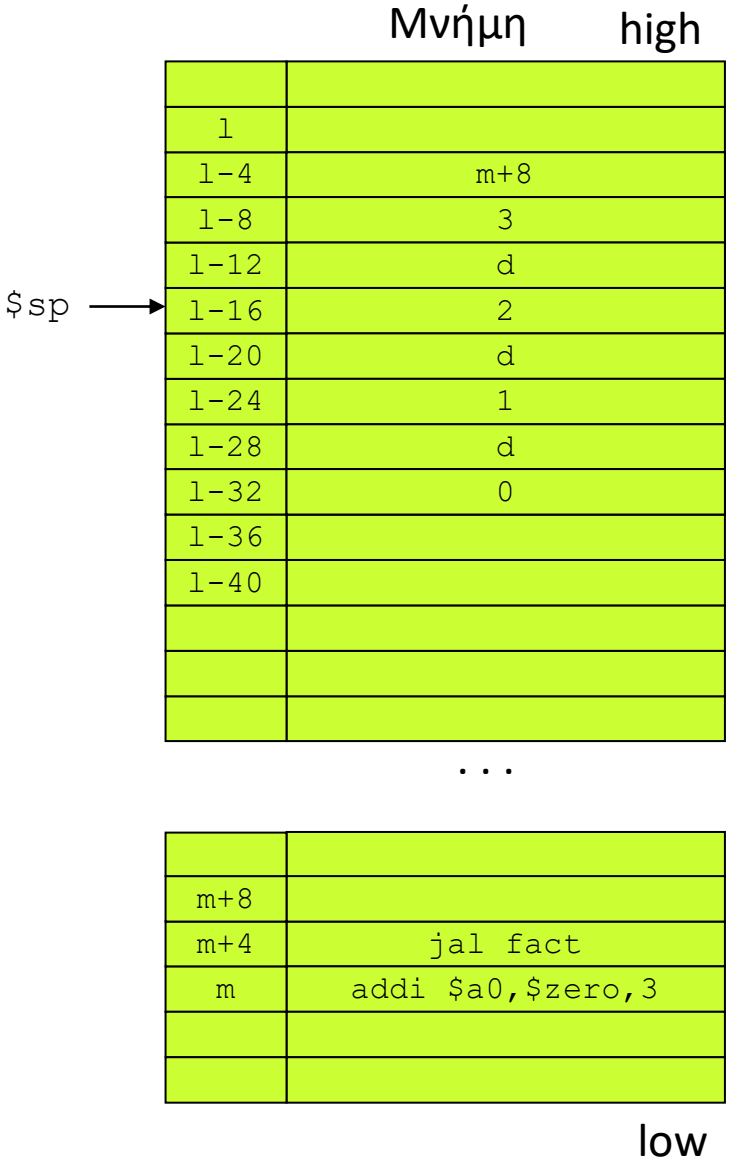
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0   ← PC
 jr $ra
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0: 1

$a0: 1

$t0: 1
```



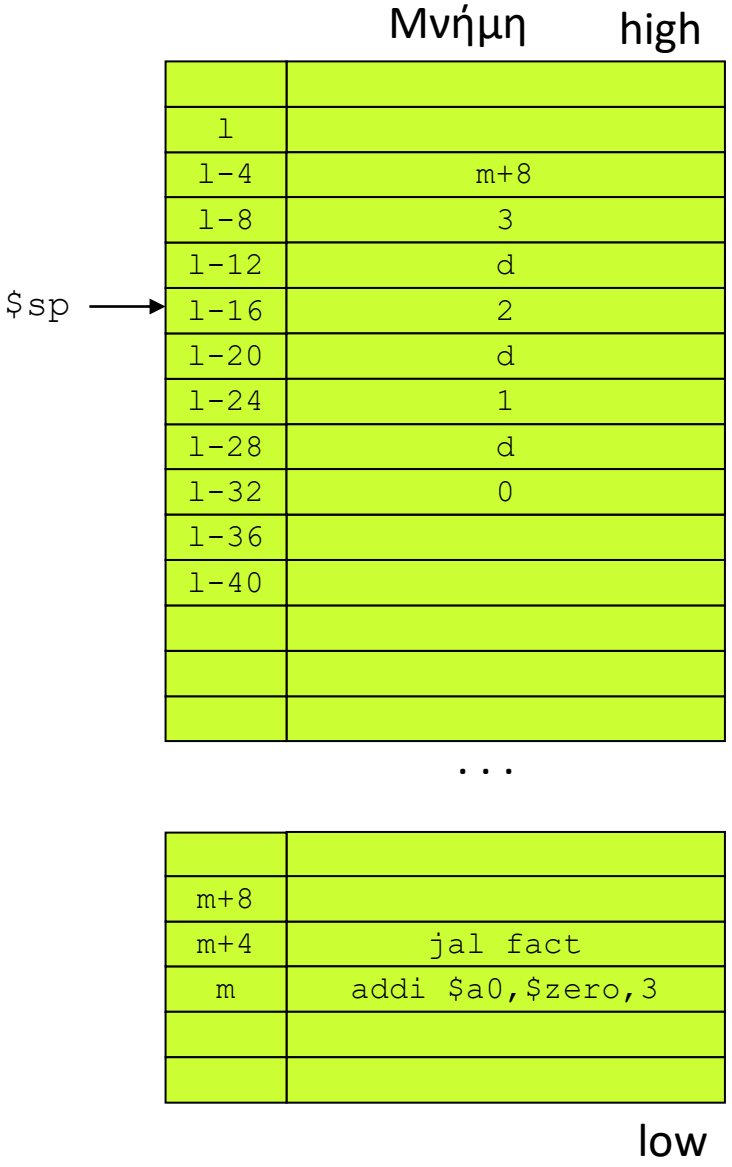
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0   ← PC
 jr $ra
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0: 1

$a0: 1

$t0: 1
```



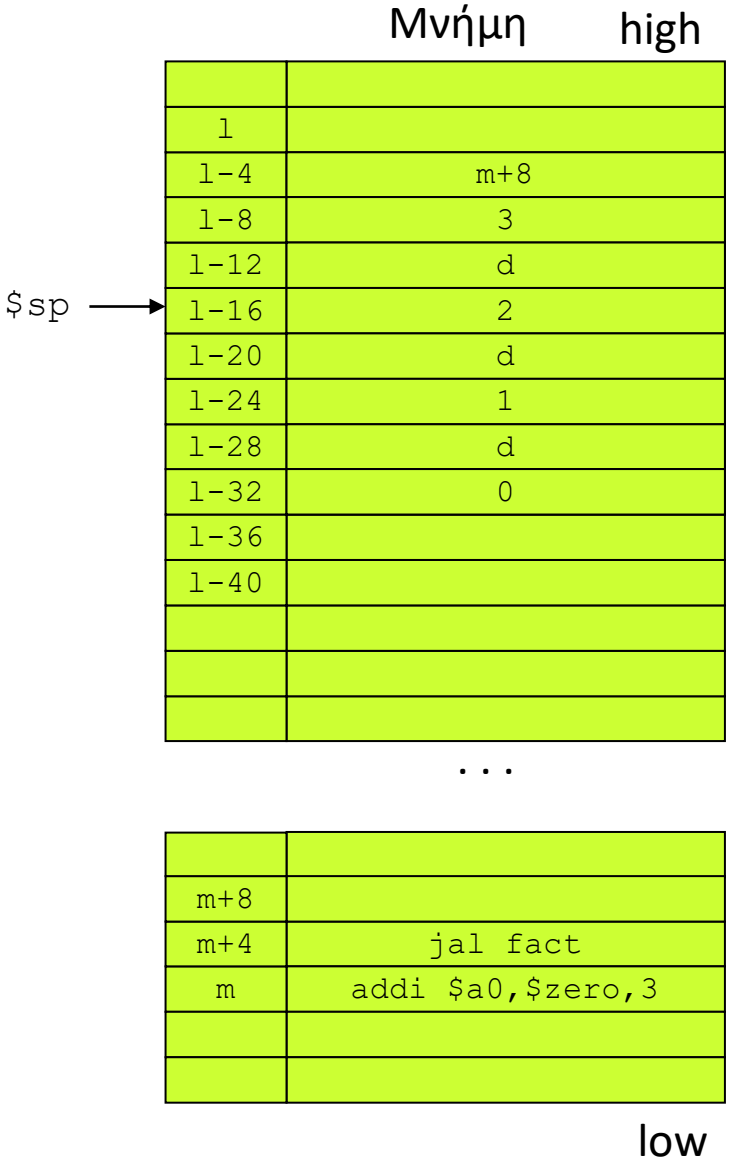
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
  addi $sp,$sp,-8
  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
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8
  mul $v0,$a0,$v0
  jr $ra ← PC
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0: 1

$a0: 1

$t0: 1
```



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
 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
d: lw $a0,0($sp)   ← PC
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

PC: βλ. PC

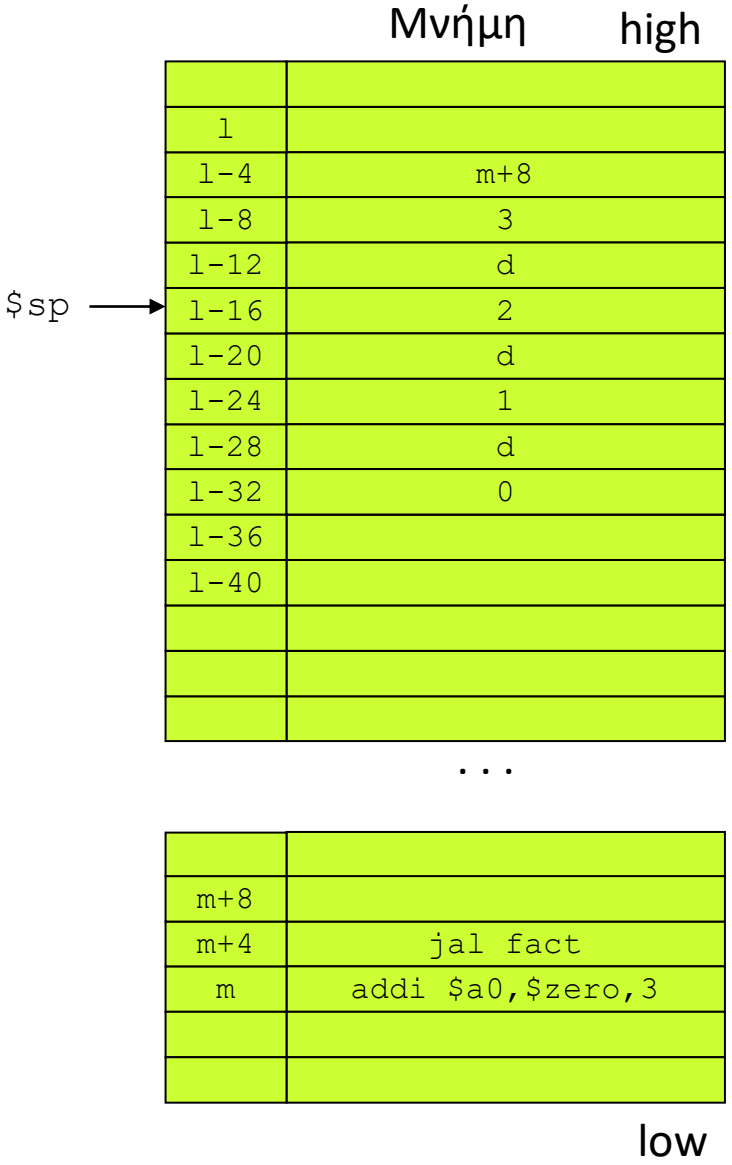
$sp: l-16

$ra: d

$v0: 1

$a0: 1

$t0: 1



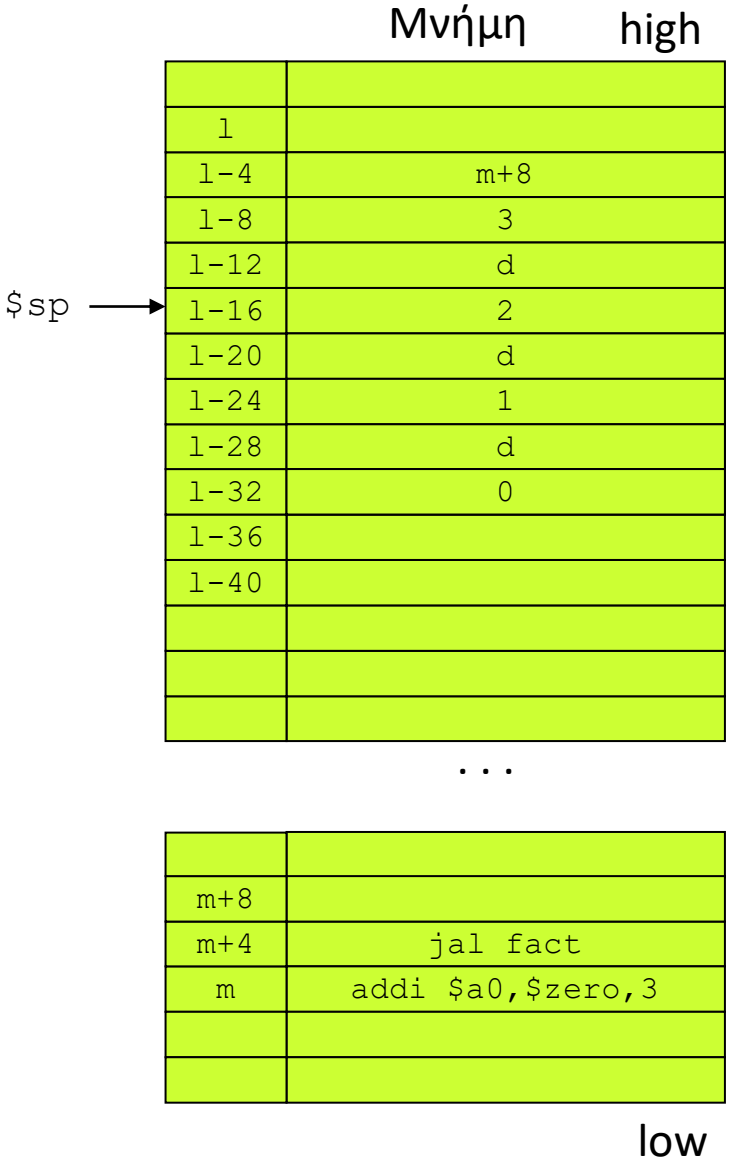
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d: lw $a0,0($sp)   ← PC
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0: 1

$a0: 2

$t0: 1
```



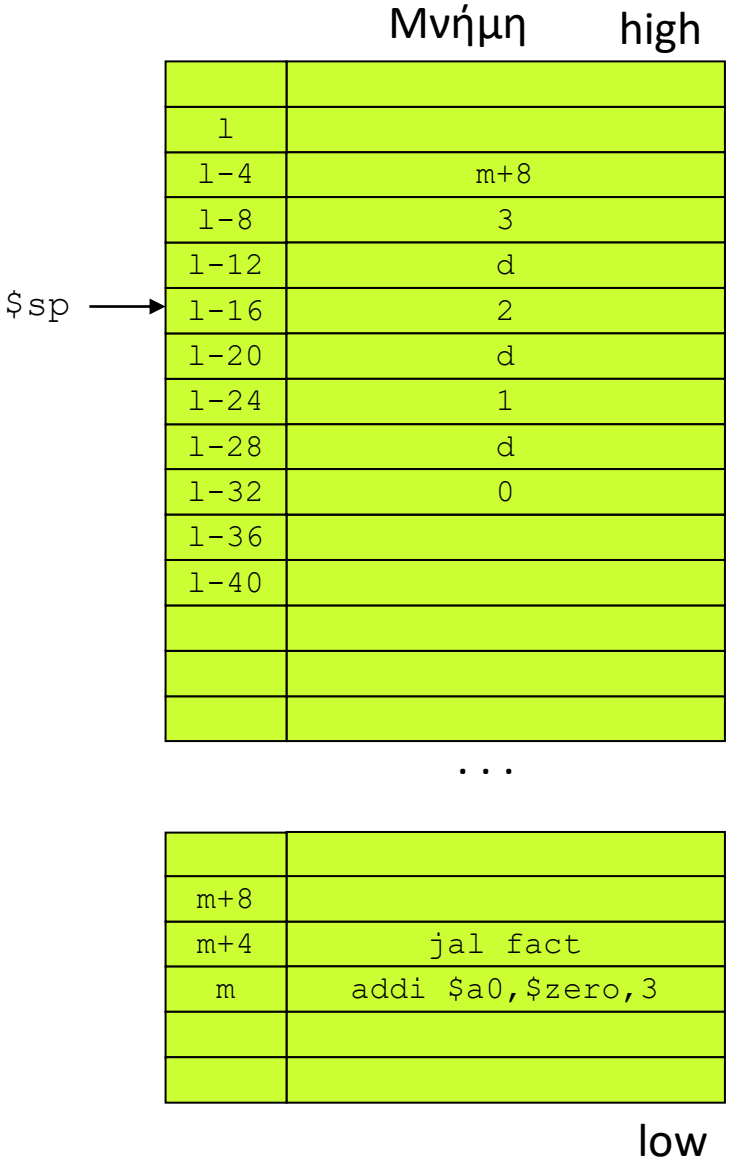
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)   ← PC
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0: 1

$a0: 2

$t0: 1
```



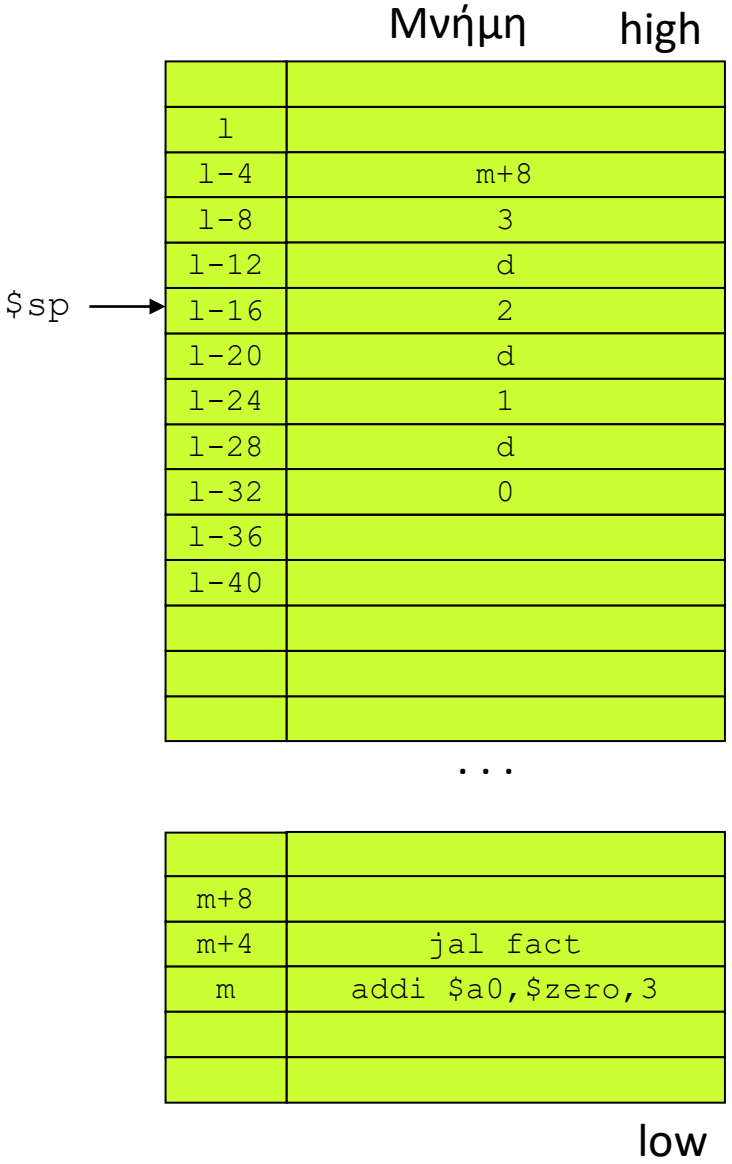
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)   ← PC
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0: 1

$a0: 2

$t0: 1
```



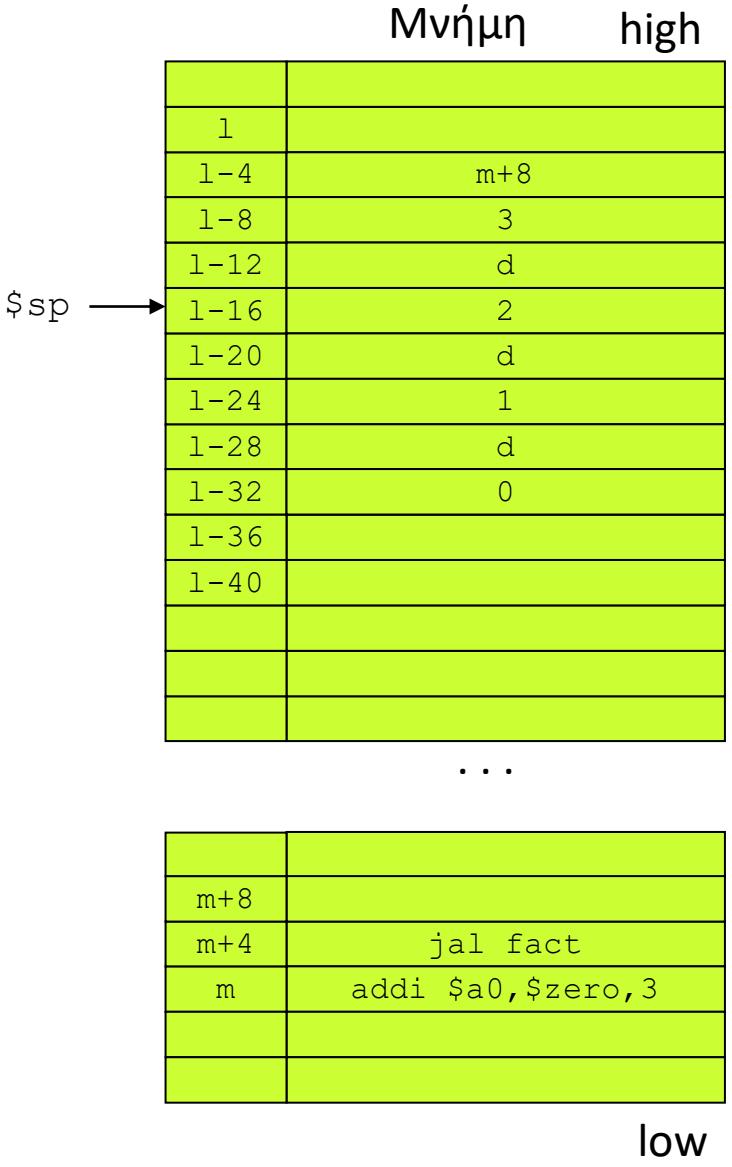
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8   ← PC
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-16

$ra: d

$v0: 1

$a0: 2

$t0: 1
```



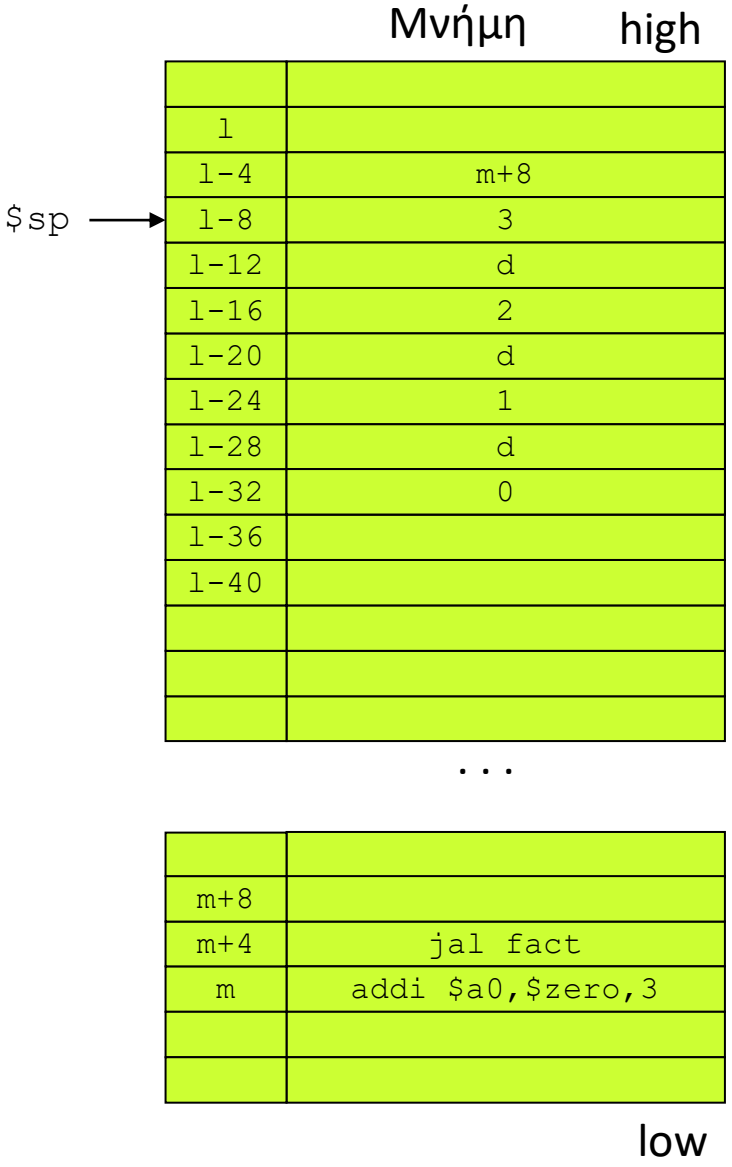
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8   ← PC
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: d

$v0: 1

$a0: 2

$t0: 1
```



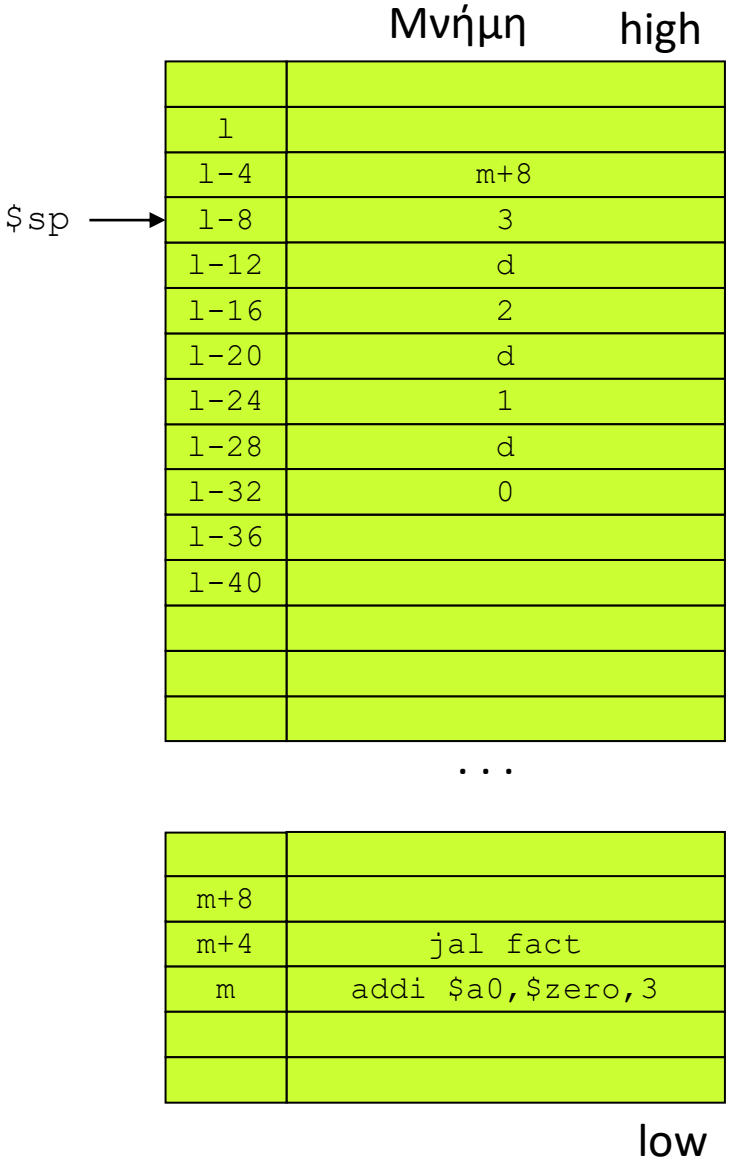
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0   ← PC
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: d

$v0: 1

$a0: 2

$t0: 1
```



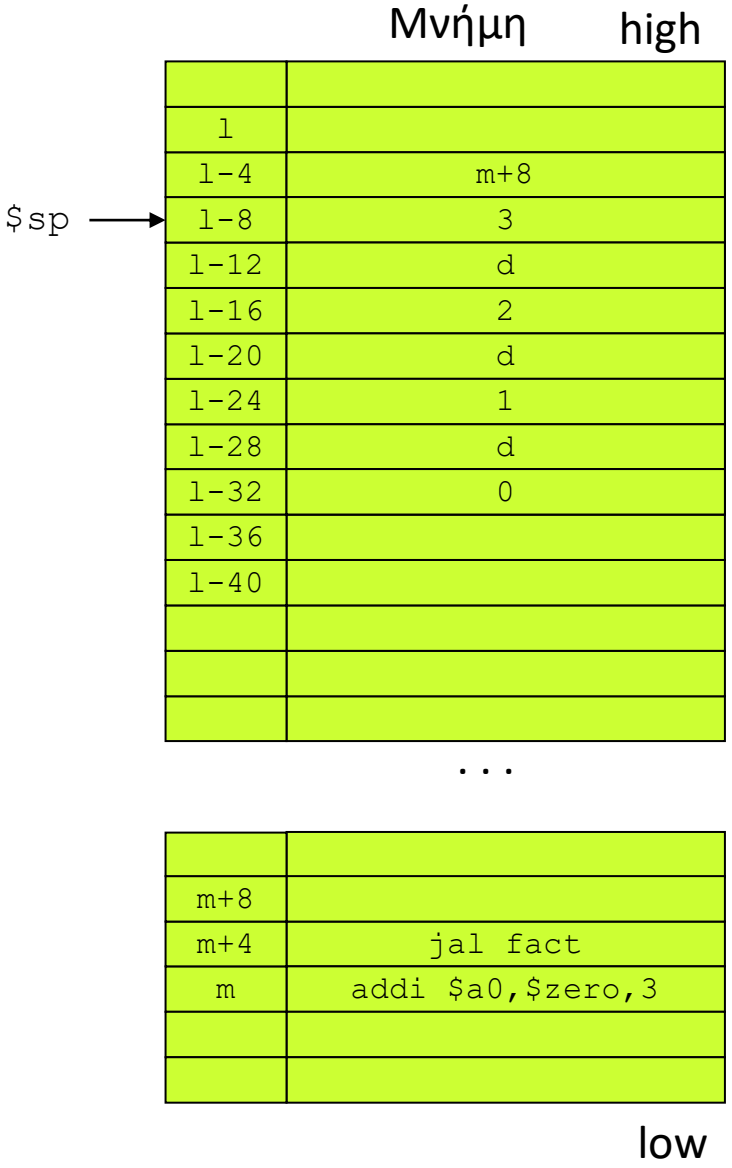
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0   ← PC
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: d

$v0: 2

$a0: 2

$t0: 1
```



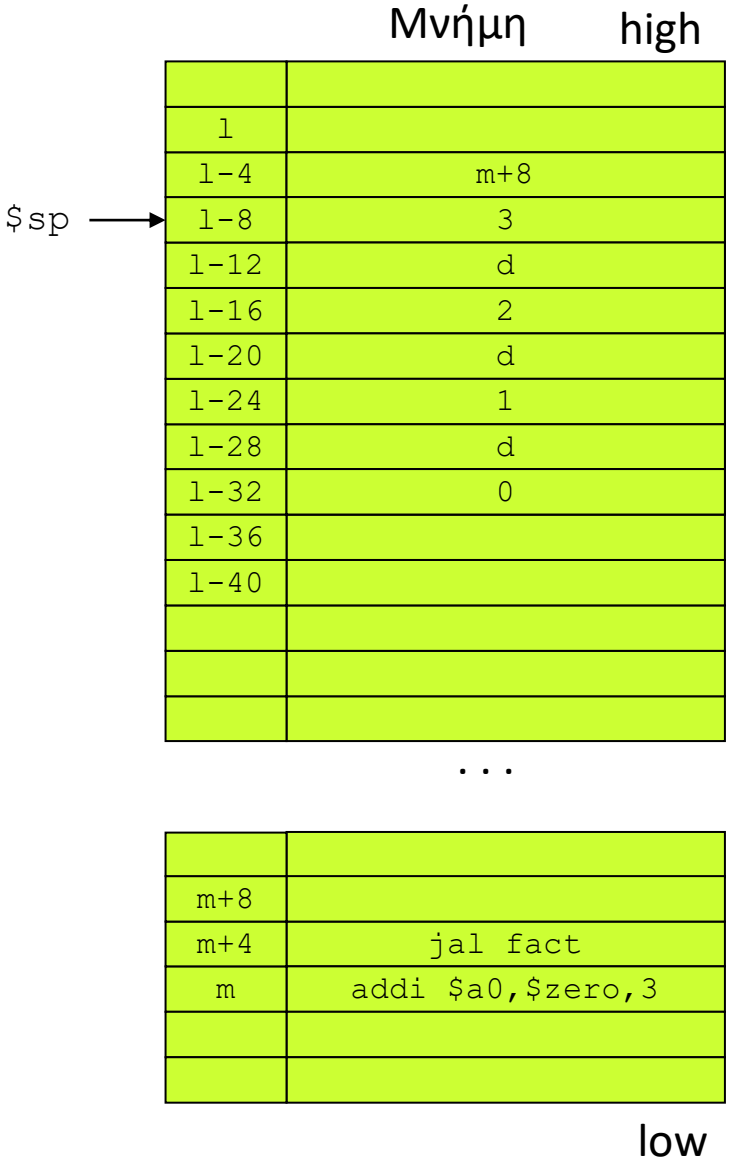
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra ← PC
```

```
PC:  βλ. PC

$sp: l-8

$ra: d

$v0: 2

$a0: 2

$t0: 1
```



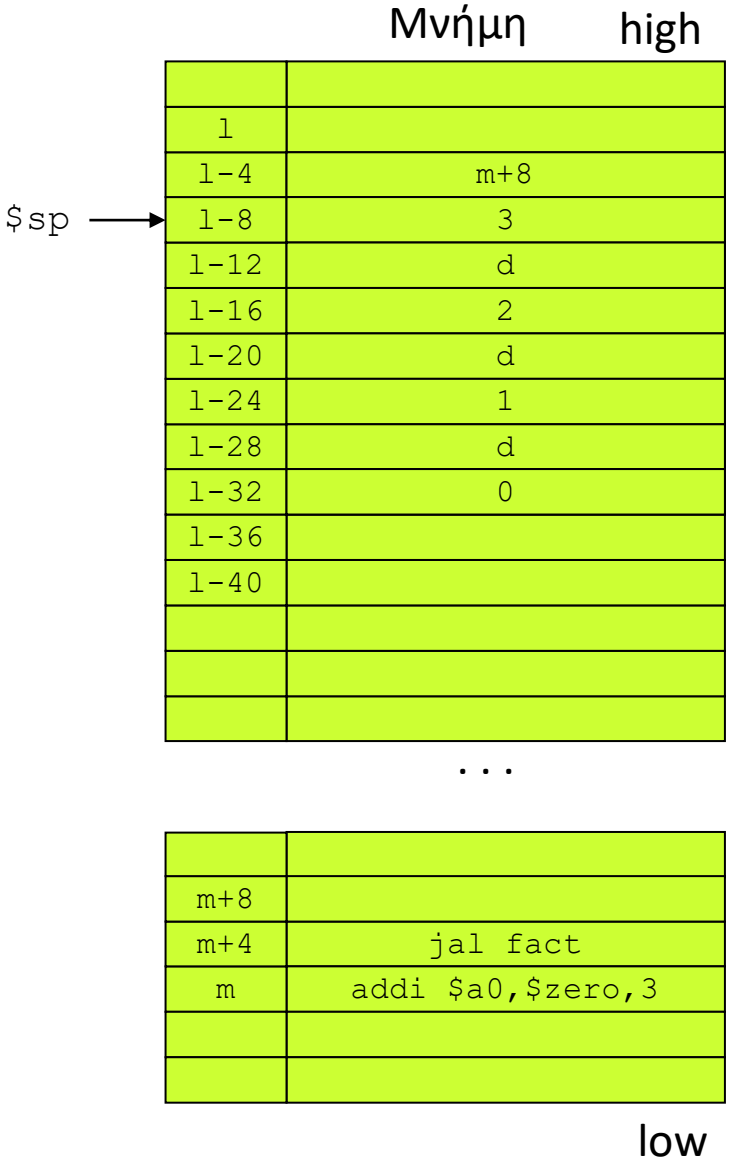
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d: lw $a0,0($sp)   ← PC
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: d

$v0: 2

$a0: 2

$t0: 1
```



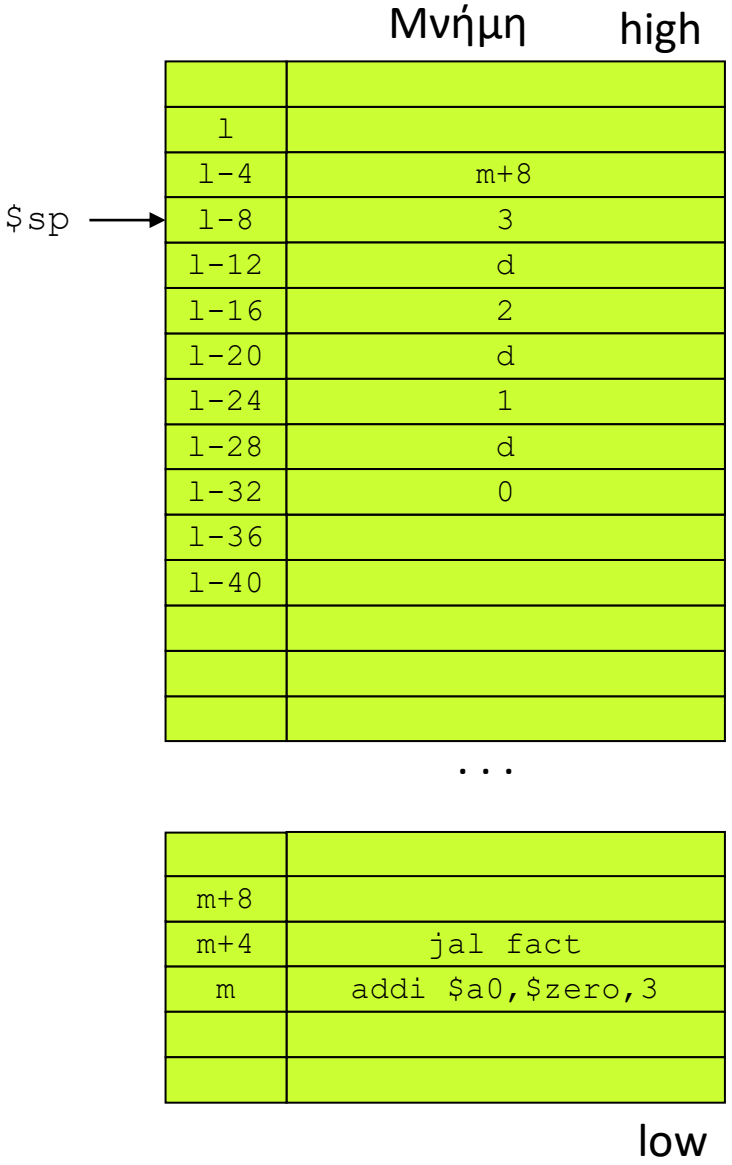
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d: lw $a0,0($sp)   ← PC
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: d

$v0: 2

$a0: 3

$t0: 1
```



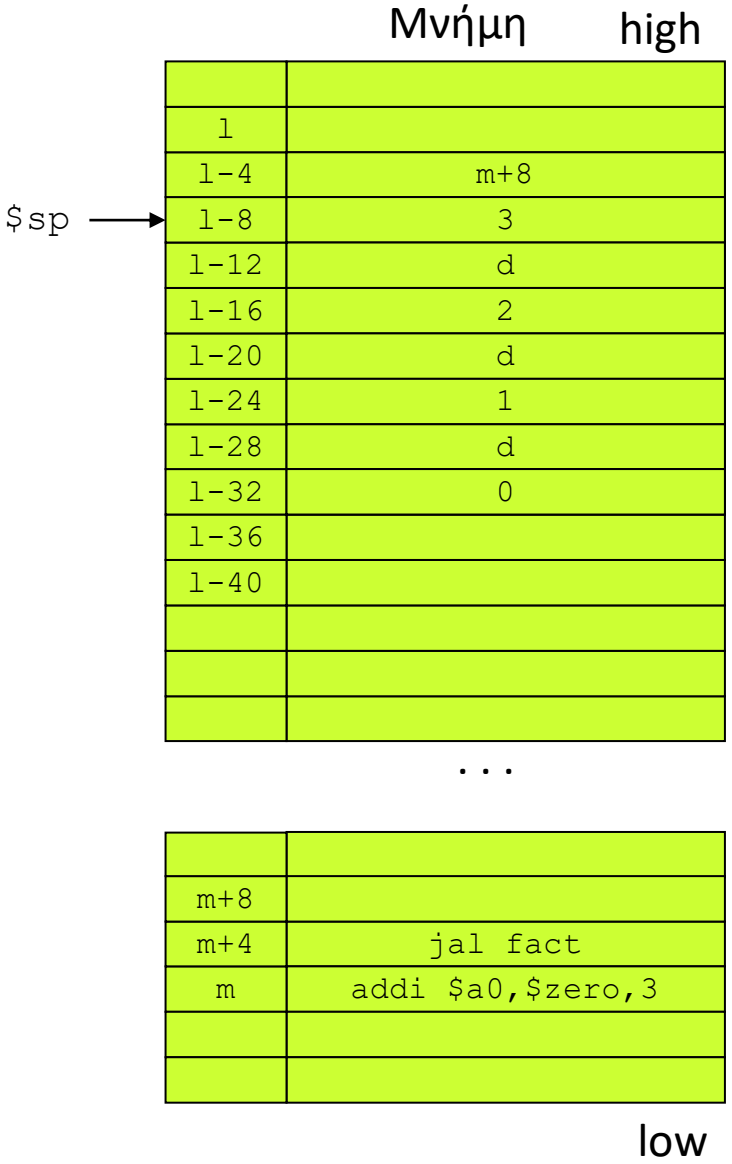
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)   ← PC
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: d

$v0: 2

$a0: 3

$t0: 1
```



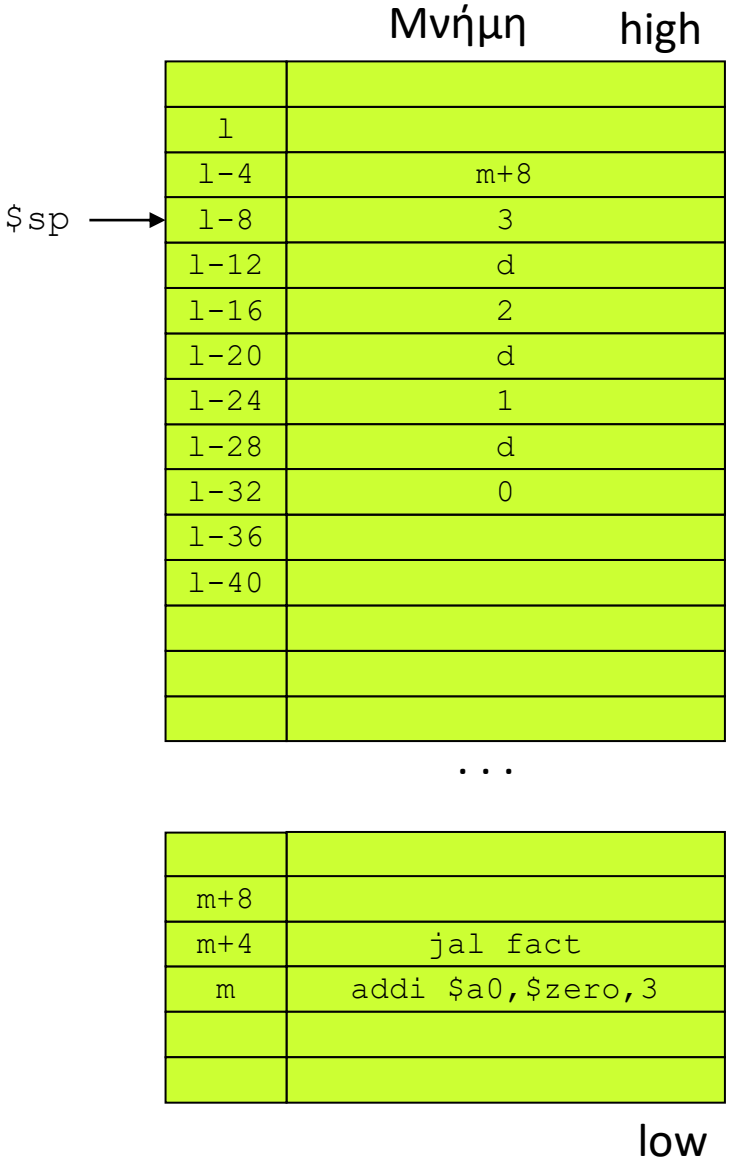
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)   ← PC
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0: 2

$a0: 3

$t0: 1
```



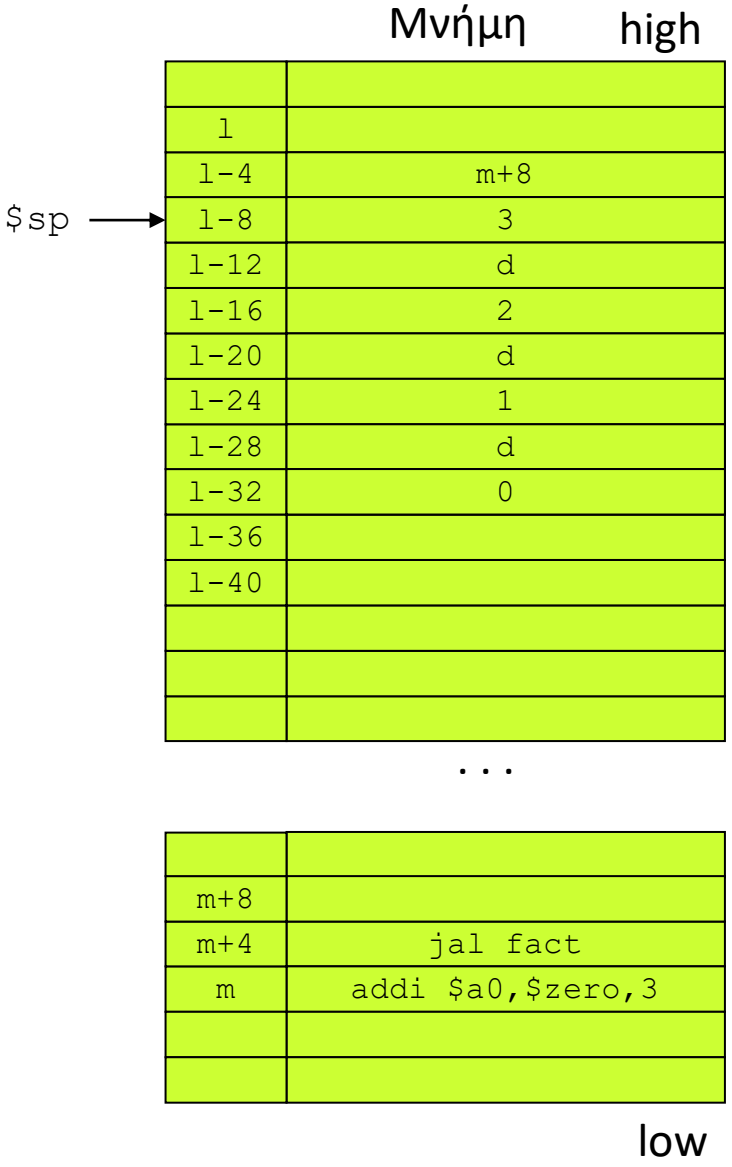
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8   ← PC
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  βλ. PC

$sp: l-8

$ra: m+8

$v0: 2

$a0: 3

$t0: 1
```



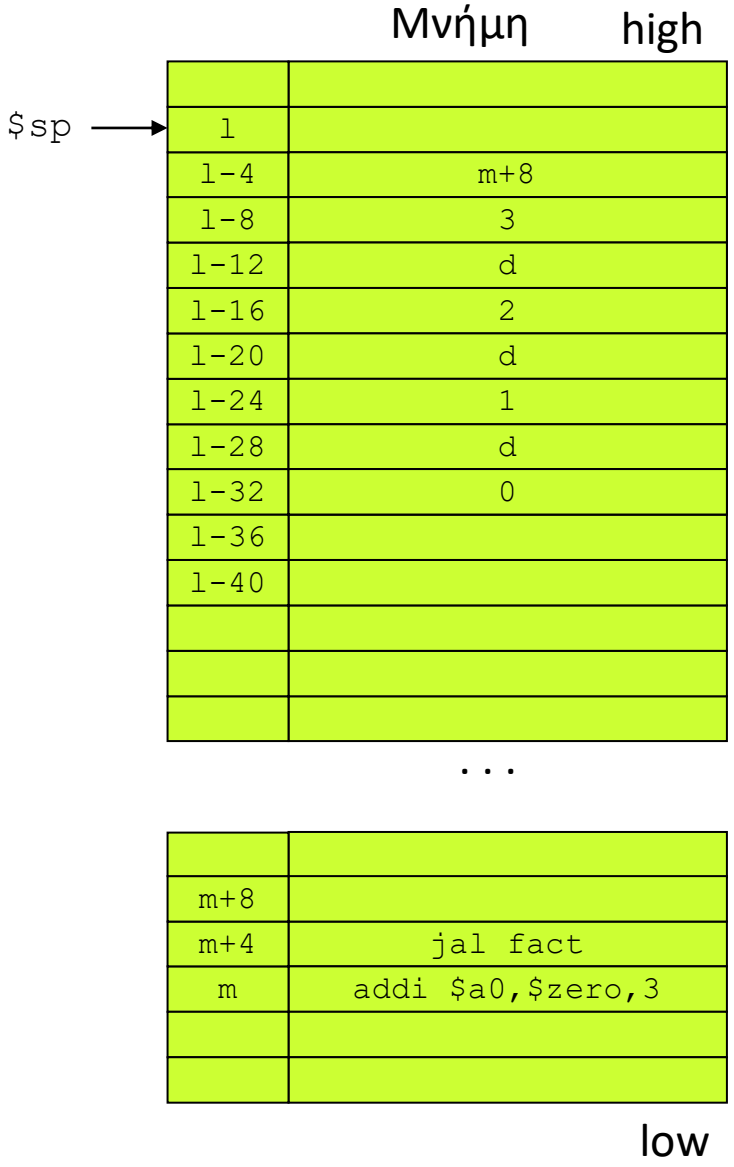
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
  addi $sp,$sp,-8
  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
d:lw $a0,0($sp)
  lw $ra,4($sp)
  addi $sp,$sp,8   ← PC
  mul $v0,$a0,$v0
  jr $ra
```

```
PC:  βλ. PC

$sp: l

$ra: m+8

$v0: 2

$a0: 3

$t0: 1
```



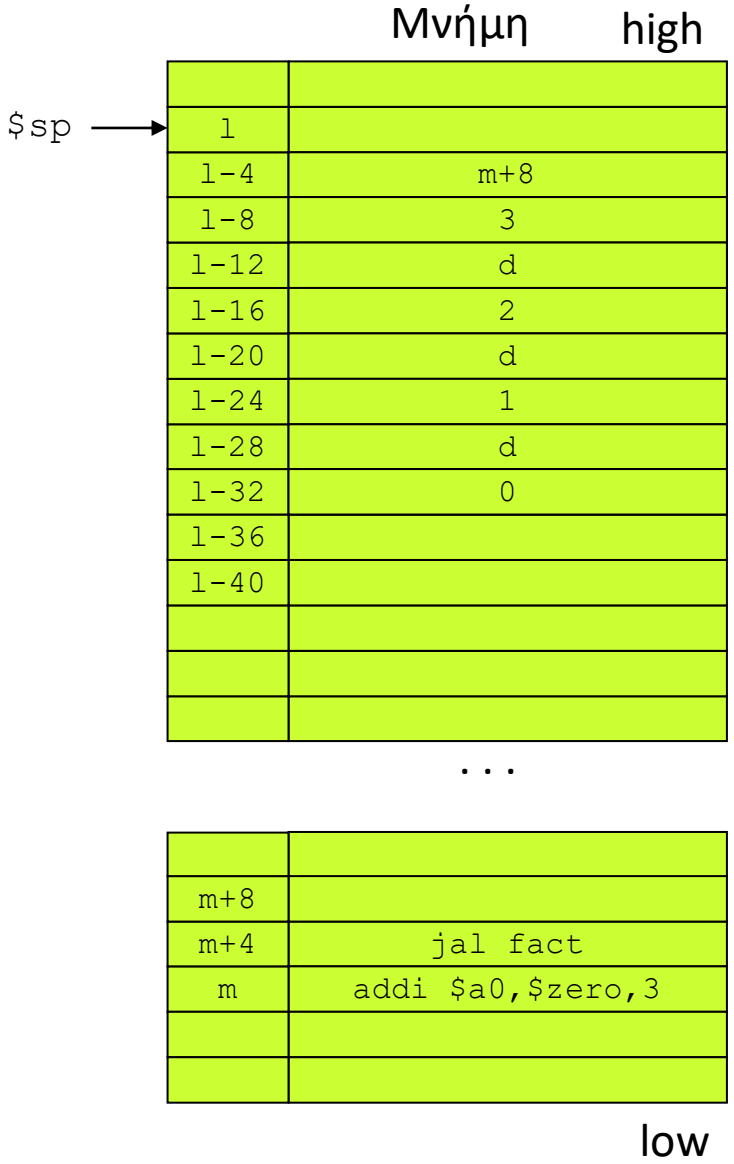
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0   ← PC
 jr $ra
```

```
PC:  βλ. PC

$sp: l

$ra: m+8

$v0: 2

$a0: 3

$t0: 1
```



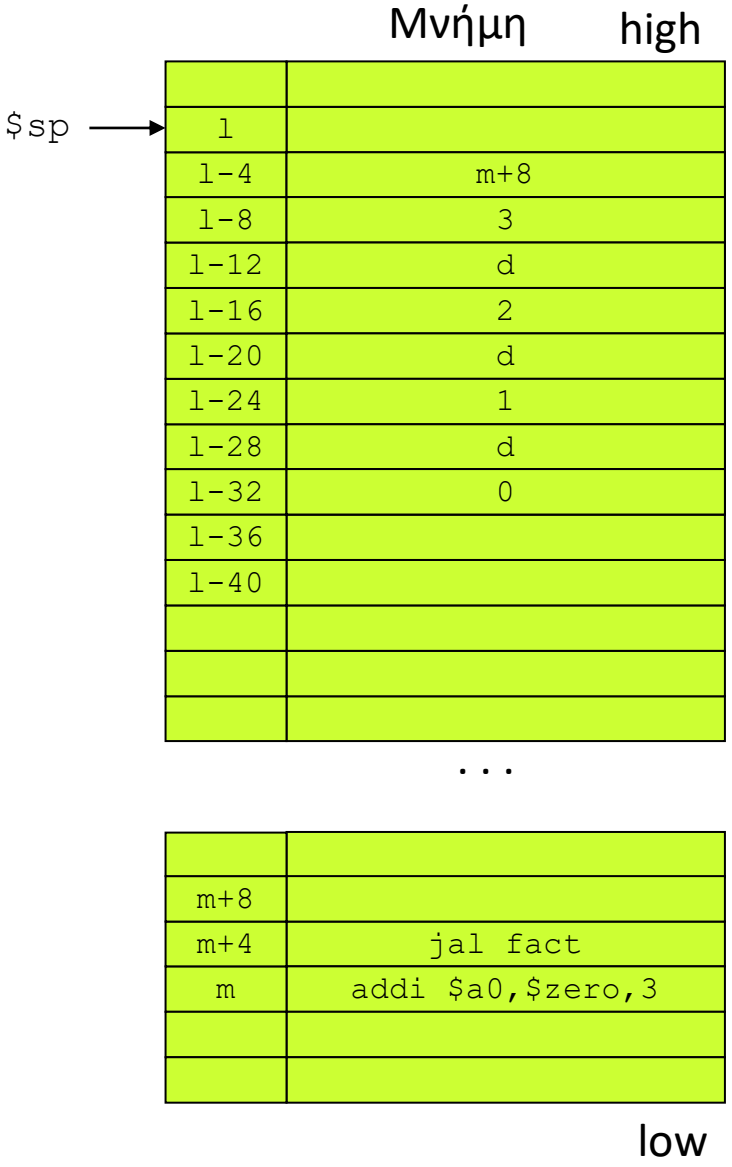
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0   ← PC
 jr $ra
```

```
PC:  βλ. PC

$sp: l

$ra: m+8

$v0: 6

$a0: 3

$t0: 1
```



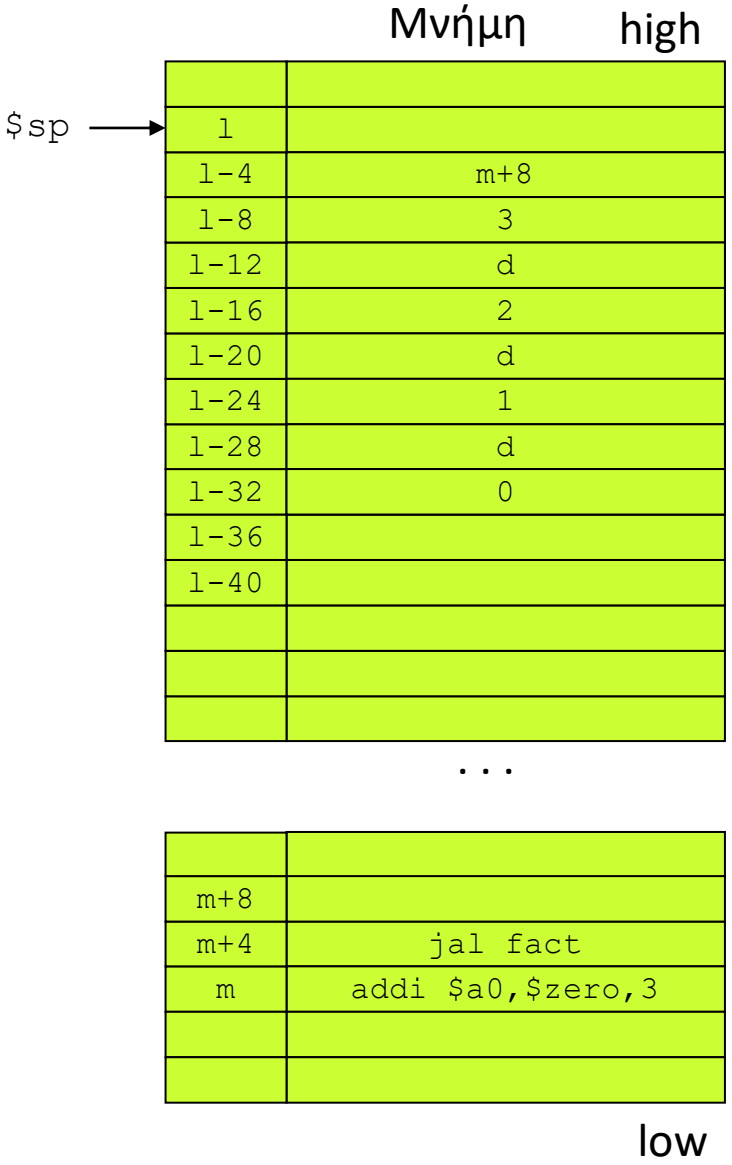
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra ← PC
```

```
PC:  βλ. PC

$sp: l

$ra: m+8

$v0: 6

$a0: 3

$t0: 1
```



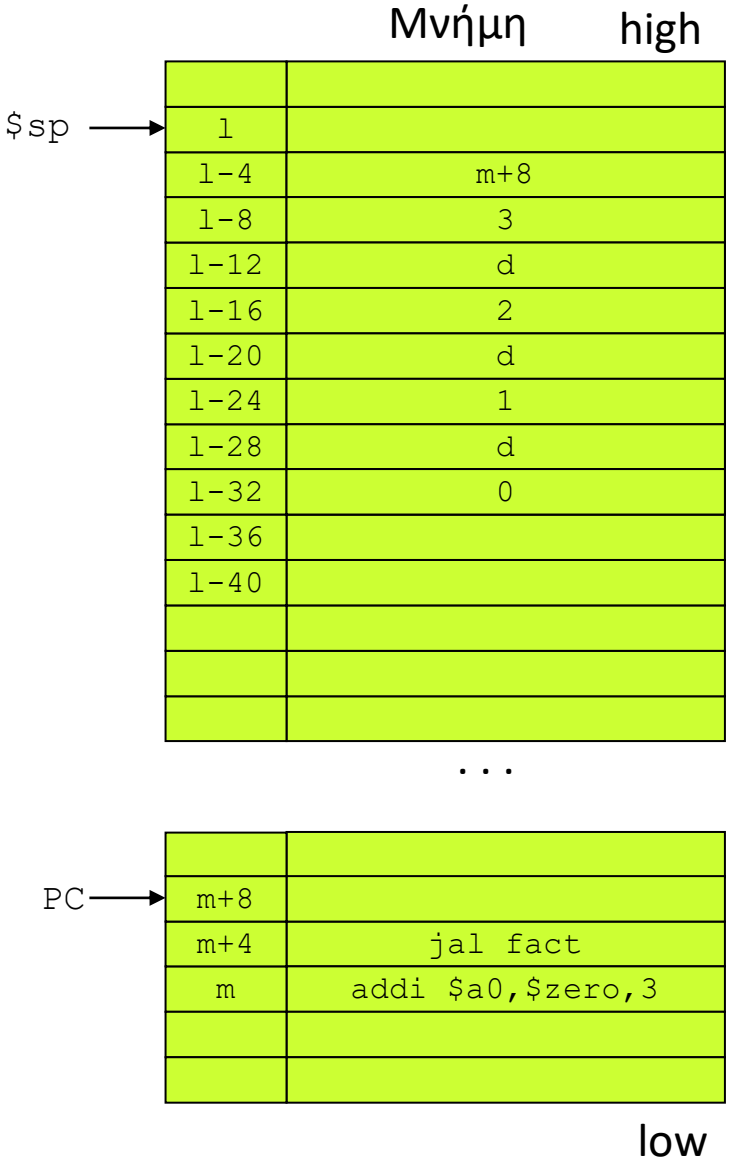
Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

```
PC:  m+8
$sp: l
$ra: m+8
$v0: 6
$a0: 3
$t0: 1
```



Υποστήριξη διαδικασιών στο υλικό των υπολογιστών

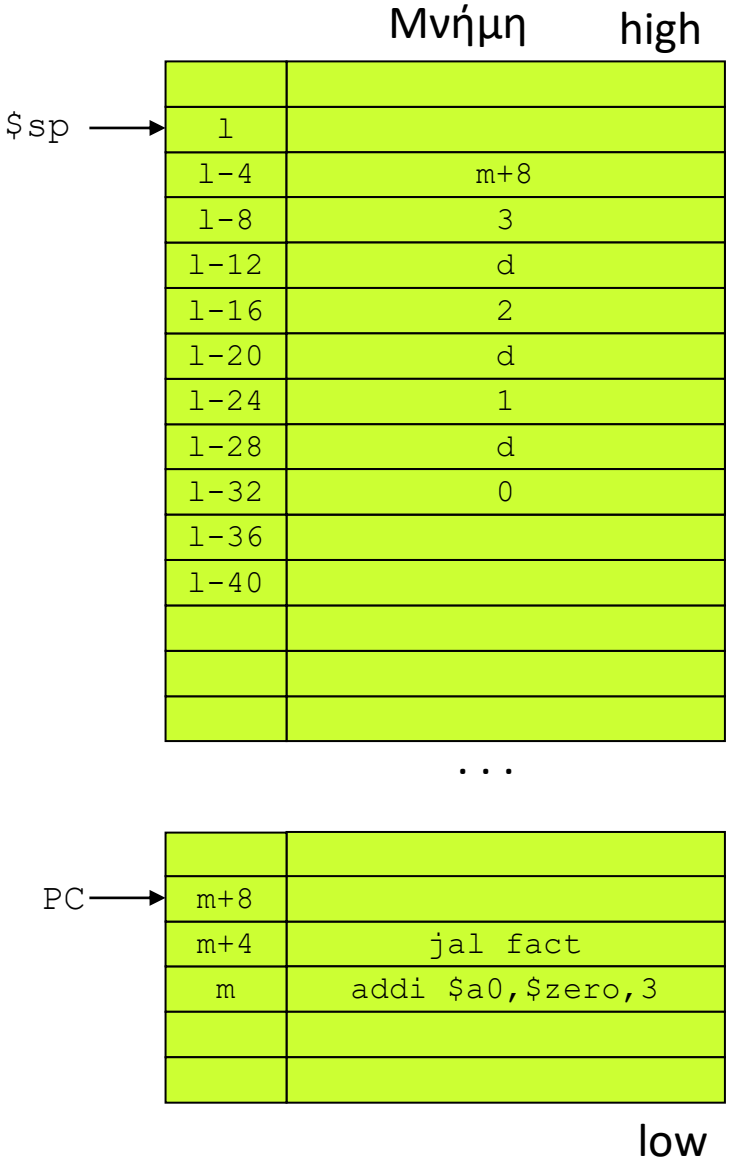
Εκτέλεση:

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

```
fact:
 addi $sp,$sp,-8
 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
d:lw $a0,0($sp)
 lw $ra,4($sp)
 addi $sp,$sp,8
 mul $v0,$a0,$v0
 jr $ra
```

finish

```
PC:  m+8
$sp: l
$ra: m+8
$v0: 6
$a0: 3
$t0: 1
```



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

Άσκηση – Βρείτε τα λάθη (1)

```
f:      sub     $s0,$a0,$a3
        sll     $v0,$s0,0x1
        add     $v0,$a2,$v0
        sub     $v0,$v0,$a1
        jr      $ra
```

Άσκηση – Βρείτε τα λάθη (1)

```
f:      sub     $s0,$a0,$a3
        sll     $v0,$s0,0x1
        add     $v0,$a2,$v0
        sub     $v0,$v0,$a1
        jr      $ra
```

Άσκηση – Βρείτε τα λάθη (2)

```
h:      addi    $sp,$sp,-8
        sw      $ra,4($sp)
        sw      $a0,0($sp)
        slti    $t0,$a0,2
        beq     $t0,$0,L1
        addi    $v0,$0,1
        addi    $sp,$sp,8
        jr      $ra
L1:     addi    $a0,$a0,-1
        jal     h
        lw      $a0,4($sp)
        lw      $ra,0($sp)
        addi    $sp,$sp,8
        mul     $v0,$a0,$v0
        jr      $ra
```

Άσκηση – Βρείτε τα λάθη (2)

```
h:      addi    $sp,$sp,-8
        sw      $ra,4($sp)
        sw      $a0,0($sp)
        slti    $t0,$a0,2
        beq     $t0,$0,L1
        addi    $v0,$0,1
        addi    $sp,$sp,8
        jr      $ra
L1:     addi    $a0,$a0,-1
        jal     h
        lw      $a0,4($sp)
        lw      $ra,0($sp)
        addi    $sp,$sp,8
        mul     $v0,$a0,$v0
        jr      $ra
```

Άσκηση – Βρείτε τα λάθη (3)

```
g:      addi    $sp,$sp,8
        sw      $ra,4($sp)
        sw      $s0,0($sp)
        move    $s0,$a2
        jal     h
        add     $v0,$v0,$s0
        lw      $ra,4($sp)
        lw      $s0,0($sp)
        addi    $sp,$sp,-8
        jr      $ra
```

Άσκηση – Βρείτε τα λάθη (3)

```
g:      addi    $sp,$sp,8
        sw      $ra,4($sp)
        sw      $s0,0($sp)
        move    $s0,$a2
        jal     h
        add     $v0,$v0,$s0
        lw      $ra,4($sp)
        lw      $s0,0($sp)
        addi    $sp,$sp,-8
        jr      $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:  addi  $sp,  $sp,
      sw    $ra,   ($sp)
      sw    $a0,  0($sp)
      bne   $a1,        ,cont
      addi     ,  $zero,
      addi  $sp,  $sp,
      jr    $ra
cont: addi       , $a1, -1
      jal   exp
      lw         ,  ($sp)
      mul        , $v0,
      lw         , 4($sp)
      addi  $sp, $sp,
      jr    $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:  addi  $sp,  $sp, -8
      sw    $ra,   ($sp)
      sw    $a0,  0($sp)
      bne   $a1,       ,cont
      addi     ,  $zero,
      addi  $sp,  $sp,
      jr    $ra
cont: addi       , $a1, -1
      jal   exp
      lw         ,  ($sp)
      mul        , $v0,
      lw         , 4($sp)
      addi  $sp, $sp,
      jr    $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:  addi  $sp,  $sp, -8
      sw    $ra,  4($sp)
      sw    $a0,  0($sp)
      bne   $a1,       ,cont
      addi     ,  $zero,
      addi  $sp,  $sp,
      jr    $ra
cont: addi       , $a1, -1
      jal   exp
      lw         ,  ($sp)
      mul        , $v0,
      lw         , 4($sp)
      addi  $sp, $sp,
      jr    $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:  addi  $sp,  $sp, -8
      sw    $ra,  4($sp)
      sw    $a0,  0($sp)
      bne   $a1,  $zero,cont
      addi      ,  $zero,
      addi  $sp,  $sp,
      jr    $ra
cont: addi        , $a1, -1
      jal   exp
      lw          ,  ($sp)
      mul         , $v0,
      lw          , 4($sp)
      addi  $sp, $sp,
      jr    $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:  addi  $sp,  $sp, -8
      sw    $ra,  4($sp)
      sw    $a0,  0($sp)
      bne   $a1,  $zero,cont
      addi  $v0,  $zero, 1
      addi  $sp,  $sp,
      jr    $ra
cont: addi        , $a1, -1
      jal   exp
      lw          ,  ($sp)
      mul         , $v0,
      lw          , 4($sp)
      addi  $sp, $sp,
      jr    $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:  addi  $sp,  $sp, -8
      sw    $ra,  4($sp)
      sw    $a0,  0($sp)
      bne   $a1,  $zero,cont
      addi  $v0,  $zero, 1
      addi  $sp,  $sp, 8
      jr    $ra
cont: addi       , $a1, -1
      jal   exp
      lw         ,  ($sp)
      mul        , $v0,
      lw         , 4($sp)
      addi  $sp, $sp,
      jr    $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:   addi   $sp,   $sp, -8
       sw     $ra,   4($sp)
       sw     $a0,   0($sp)
       bne    $a1,   $zero,cont
       addi   $v0,   $zero, 1
       addi   $sp,   $sp, 8
       jr     $ra
cont:  addi   $a1 ,  $a1, -1
       jal    exp
       lw           ,  ($sp)
       mul          , $v0,
       lw           , 4($sp)
       addi   $sp,  $sp,
       jr     $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:   addi   $sp,   $sp, -8
       sw     $ra,   4($sp)
       sw     $a0,   0($sp)
       bne    $a1,   $zero,cont
       addi   $v0,   $zero, 1
       addi   $sp,   $sp, 8
       jr     $ra
cont:  addi   $a1 ,  $a1, -1
       jal    exp
       lw     $a0 ,  0($sp)
       mul           , $v0,
       lw            , 4($sp)
       addi   $sp,  $sp,
       jr     $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:   addi   $sp,   $sp, -8
       sw     $ra,   4($sp)
       sw     $a0,   0($sp)
       bne    $a1,   $zero,cont
       addi   $v0,   $zero, 1
       addi   $sp,   $sp, 8
       jr     $ra
cont:  addi   $a1 ,  $a1, -1
       jal    exp
       lw     $a0 ,  0($sp)
       mul    $v0 ,  $v0, $a0
       lw           , 4($sp)
       addi   $sp,  $sp,
       jr     $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:   addi   $sp,   $sp, -8
       sw     $ra,   4($sp)
       sw     $a0,   0($sp)
       bne    $a1,   $zero,cont
       addi   $v0,   $zero, 1
       addi   $sp,   $sp, 8
       jr     $ra
cont:  addi   $a1 ,  $a1, -1
       jal    exp
       lw     $a0 ,  0($sp)
       mul    $v0 ,  $v0, $a0
       lw     $ra ,  4($sp)
       addi   $sp,  $sp,
       jr     $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
int exp(int x, int n) {
    if (n == 0) return 1;
    return x * exp (x, n-1);
}
```

```
exp:   addi   $sp,   $sp, -8
       sw     $ra,   4($sp)
       sw     $a0,   0($sp)
       bne    $a1,   $zero,cont
       addi   $v0,   $zero, 1
       addi   $sp,   $sp, 8
       jr     $ra
cont:  addi   $a1 ,  $a1, -1
       jal    exp
       lw     $a0 ,  0($sp)
       mul    $v0 ,  $v0, $a0
       lw     $ra ,  4($sp)
       addi   $sp,  $sp, 8
       jr     $ra
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, ____
LOOP:   slt   $t0, ____, ____
        ___   $t0, $zero, EXIT
I_LOOP: addi  ___, $s1, 1
        sll   $t0, $s1, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
int A[N];
int left = -1, int right = N;
int pivot, res;

pivot = A[0];

while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}

res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, ____
LOOP:   slt   $t0, ____, ____
        ___   $t0, $zero, EXIT
I_LOOP: addi  ___, $s1, 1
        sll   $t0, $s1, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;

pivot = A[0];

while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}

res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, ____
LOOP:   slt   $t0, ____, ____
        ___   $t0, $zero, EXIT
I_LOOP: addi  ___, $s1, 1
        sll   $t0, $s1, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];

while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}

res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, ____
LOOP:   slt   $t0, ____, ____
        ___   $t0, $zero, EXIT
I_LOOP: addi  ___, $s1, 1
        sll   $t0, $s1, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1      $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}

res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, ____
LOOP:   slt   $t0, ____, ____
        ___   $t0, $zero, EXIT
I_LOOP: addi  ___, $s1, 1
        sll   $t0, $s1, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1      $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, ____ 0($s3)
LOOP:   slt   $t0, ____, ____
        ___   $t0, $zero, EXIT
I_LOOP: addi  ___, $s1, 1
        sll   $t0, $s1, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1      $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        ___   $t0, $zero, EXIT
I_LOOP: addi  ___, $s1, 1
        sll   $t0, $s1, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  ___, $s1, 1
        sll   $t0, $s1, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1      $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1      $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, ____
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        ___   $t0, $zero, ____
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  ___, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, ____
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   ___, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    ___, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, ___, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        ____  $t0, $zero, ____
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1      $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        bne   $t0, $zero, J_LOOP
        slt   $t0, ___, ____
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        bne   $t0, $zero, J_LOOP
        slt   $t0, $s1, $s2
        ___   $t0, $zero, ____
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        bne   $t0, $zero, J_LOOP
        slt   $t0, $s1, $s2
        beq   $t0, $zero, LOOP
        ___   $a0, $s1, ____
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        bne   $t0, $zero, J_LOOP
        slt   $t0, $s1, $s2
        beq   $t0, $zero, LOOP
        sll   $a0, $s1, 2
        add   ___, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        bne   $t0, $zero, J_LOOP
        slt   $t0, $s1, $s2
        beq   $t0, $zero, LOOP
        sll   $a0, $s1, 2
        add   $a0, $a0, $s3
        sll   ___, ___, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        bne   $t0, $zero, J_LOOP
        slt   $t0, $s1, $s2
        beq   $t0, $zero, LOOP
        sll   $a0, $s1, 2
        add   $a0, $a0, $s3
        sll   $a1, $s2, 2
        add   $a1, $a1, ____
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        bne   $t0, $zero, J_LOOP
        slt   $t0, $s1, $s2
        beq   $t0, $zero, LOOP
        sll   $a0, $s1, 2
        add   $a0, $a0, $s3
        sll   $a1, $s2, 2
        add   $a1, $a1, $s3
        ___   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        bne   $t0, $zero, J_LOOP
        slt   $t0, $s1, $s2
        beq   $t0, $zero, LOOP
        sll   $a0, $s1, 2
        add   $a0, $a0, $s3
        sll   $a1, $s2, 2
        add   $a1, $a1, $s3
        jal   swap
        j     LOOP
EXIT:   add   $s4, ___, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση – Συμπληρώστε τα κενά

```
        lw    $s0, 0($s3)
LOOP:   slt   $t0, $s1, $s2
        beq   $t0, $zero, EXIT
I_LOOP: addi  $s1, $s1, 1
        sll   $t0, $s1, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $t1, $s0
        bne   $t0, $zero, I_LOOP
J_LOOP: addi  $s2, $s2, -1
        sll   $t0, $s2, 2
        add   $t0, $s3, $t0
        lw    $t1, 0($t0)
        slt   $t0, $s0, $t1
        bne   $t0, $zero, J_LOOP
        slt   $t0, $s1, $s2
        beq   $t0, $zero, LOOP
        sll   $a0, $s1, 2
        add   $a0, $a0, $s3
        sll   $a1, $s2, 2
        add   $a1, $a1, $s3
        jal   swap
        j     LOOP
EXIT:   add   $s4, $s2, $zero
```

Ζητούμενο: Δίνεται το παρακάτω πρόγραμμα σε C καθώς και μια μετάφραση του σε assembly MIPS. Η διεύθυνση του πρώτου στοιχείου του πίνακα A είναι αποθηκευμένη στον καταχωρητή $s3. Συμπληρώστε τα κενά. Σας υπενθυμίζουμε ότι ο καταχωρητής $0 (ή $zero) είναι πάντα μηδέν.

```
     $s3
int A[N];
int left = -1, int right = N;
int pivot, res;
  $s0
pivot = A[0];
          $s1       $s2
while (left < right) {
    while (A[++left] < pivot);
    while (A[--right] > pivot);

    if (left<right)
        swap(&A[left], &A[right]);
}
$s4
res = right
```

Άσκηση

Δώστε ένα πρόγραμμα σε assembly MIPS, το οποίο να μεταφέρει τα bits 17:11 του καταχωρητή $s0 στα bits 8:2 του καταχωρητή $s1, διατηρώντας τα υπόλοιπα bits του $s1 αμετάβλητα. Μπορείτε να χρησιμοποιήσετε τους προσωρινούς καταχωρητές $t, ενώ το πρόγραμμα σας δεν πρέπει να περιέχει παραπάνω από 6 εντολές

Άσκηση

Δώστε ένα πρόγραμμα σε assembly MIPS, το οποίο να μεταφέρει τα bits 17:11 του καταχωρητή $s0 στα bits 8:2 του καταχωρητή $s1, διατηρώντας τα υπόλοιπα bits του $s1 αμετάβλητα. Μπορείτε να χρησιμοποιήσετε τους προσωρινούς καταχωρητές $t, ενώ το πρόγραμμα σας δεν πρέπει να περιέχει παραπάνω από 6 εντολές

```
lui     $t0,0xFFFF
ori     $t0,$t0,0xFE03
and     $s1,$s1,$t0

srl     $t0,$s0,9
andi    $t0,$t0,0x1FC

or      $s1,$s1,$t0
```