


Il Linguaggio Assembly: Le procedure

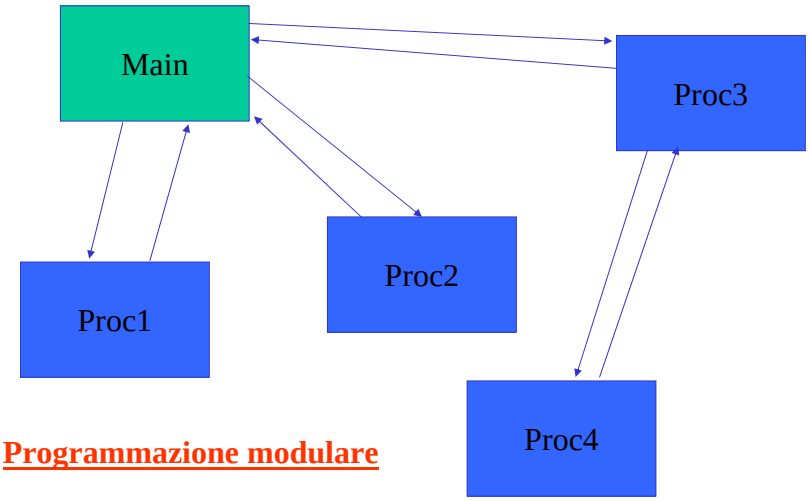
Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano

Riferimento sul Patterson: 2.6, 2.7


Sommario

Perché le procedure?



Programmazione modulare

Gli attori

Ci sono due ***attori:***
- Procedura chiamante.
- Procedura chiamata.

I due problemi delle procedure*:*
- Passaggio dei dati.
- Trasferimento del controllo.

```
f = f + 1;
if (f == g)
 res = funct(f,g)
else f = f -1;
……
```

```
int funct (int p1, int p2)
{ int out
   out = p1 * p2;
   return out;
}
```

I due moduli si parlano solamente attraverso i parametri:
-Parametri di input (argomenti della funzione).
-Parametri di output (valori restituiti dalla funzione).

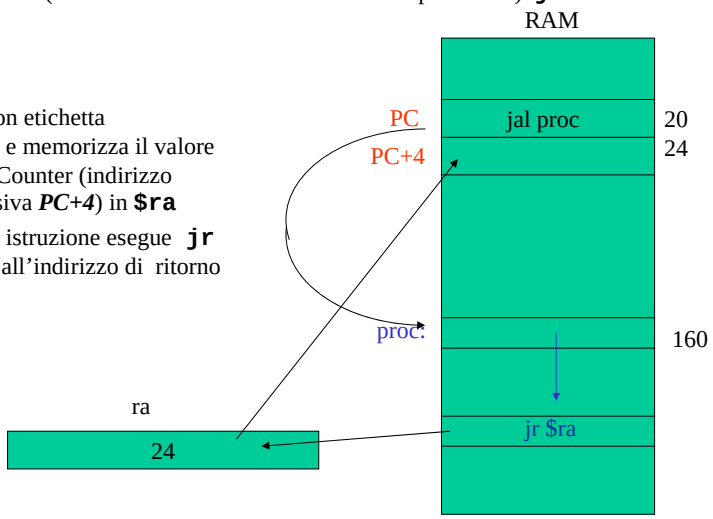
Meccanismo di chiamata::trasferimento del controllo

- Necessaria un'istruzione apposita che cambia il flusso di esecuzione (salta alla procedura) e salva l'indirizzo di ritorno (istruzione successiva alla chiamata di procedura): **`jal`** **(jump and link).**

•**jal Indirizzo_Procedura**

➢Salta all'indirizzo con etichetta **Indirizzo_Procedura** e memorizza il valore corrente del Program Counter (indirizzo dell'istruzione successiva ***PC+4***) in **`$ra`**

•La procedura come ultima istruzione esegue **`jr $ra`** per effettuare il salto all'indirizzo di ritorno della procedura.



La chiamata a procedura::trasferimento dati

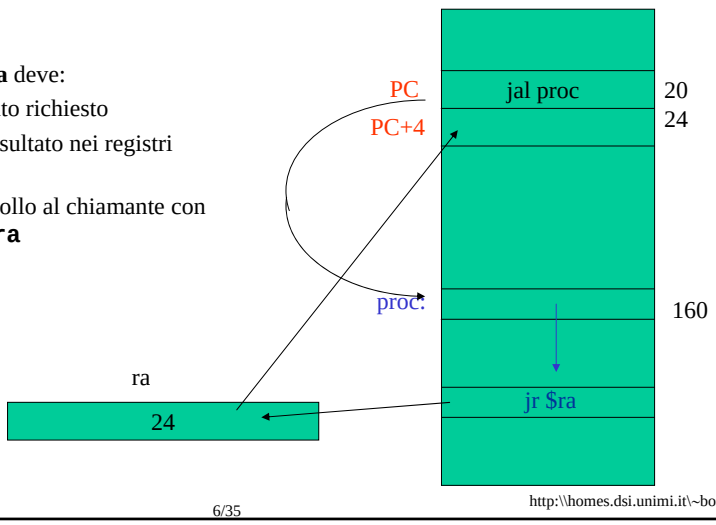
- Il programma **chiamante** deve:
  - Mettere i valori dei parametri da passare alla procedura nei registri **`$a0-$a3`**
  - Utilizzare l'istruzione **`jal address`** per saltare alla procedura e salvare il valore di **`(PC+4)`** nel registro **`$ra`**

•La procedura **chiamata** deve:

–Eseguire il compito richiesto

–Memorizzare il risultato nei registri **`$v0,$v1`**

–Restituire il controllo al chiamante con l'istruzione **`jr $ra`**



Uso dei registri nella chiamata delle procedure

Nome	Numero	Utilizzo
$zero	0	costante zero
$at	1	riservato per l'assemblatore
$v0-$v1	2-3	valori di ritorno di una procedura
$a0-$a3	4-7	argomenti di una procedura
$t0-$t7	8-15	registri temporanei (non salvati)
$s0-$s7	16-23	registri salvati
$t8-$t9	24-25	registri temporanei (non salvati)
$k0-$k1	26-27	gestione delle eccezioni
$gp	28	puntatore alla global area (dati)
$sp	29	stack pointer
$s8	30	registro salvato (fp)
$ra	31	indirizzo di ritorno

Esempio

Ci sono due ***attori:***
- Procedura chiamante.
- Procedura chiamata.

I due problemi delle procedure**:**
- Passaggio dei dati.
- Trasferimento del controllo.

f,g → $s0, $s1

```
        addi $s0, $s0, 1
        bne $s0, $s1, Else
        mov $a0,$s0
        mov $a1,$s1
        jal Funct
        j End
Else:   addi $s0, $s0, -1
End:    ……
```

```
Funct:  mul $v0, $a0, $a1
        jr $ra
```

Problemi

- Una procedura può avere bisogno di più registri rispetto ai 4 a disposizione per i parametri e ai 2 per la restituzione dei valori.
- Salvare i registri che una procedura potrebbe modificare, ma che il programma chiamante ha bisogno di mantenere inalterati.
- Fornire lo spazio necessario per le variabili locali alla procedura.
- Gestione di procedure annidate (procedure che richiamano al loro interno altre procedure) e procedure ricorsive (procedure che invocano dei 'cloni' di se stesse).

utilizzo dello stack

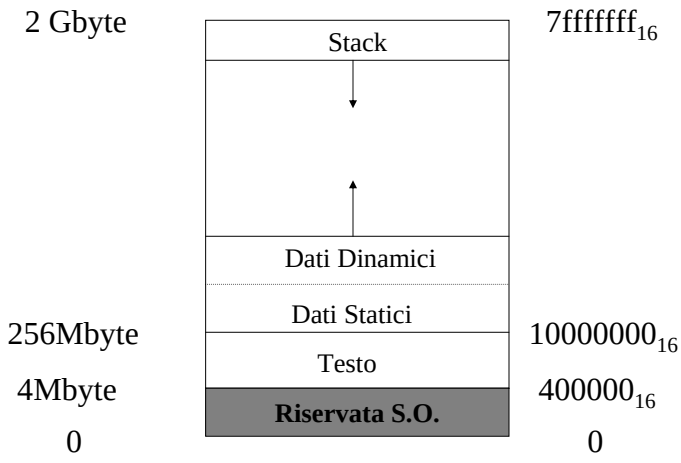
Sommario

Le procedure

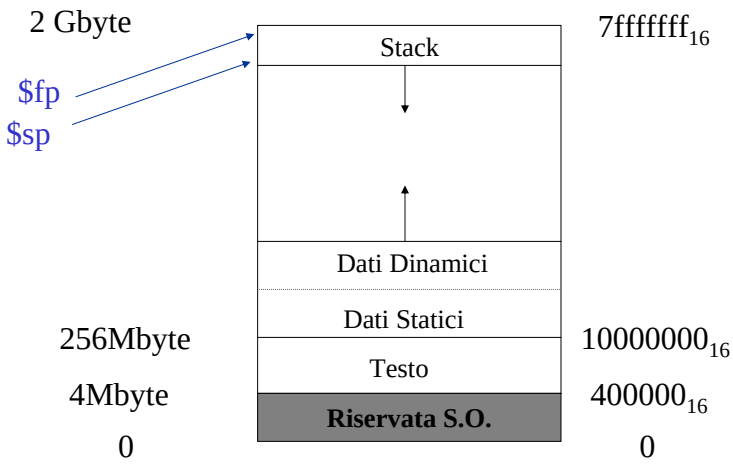
Lo stack

La chiamata a procedura

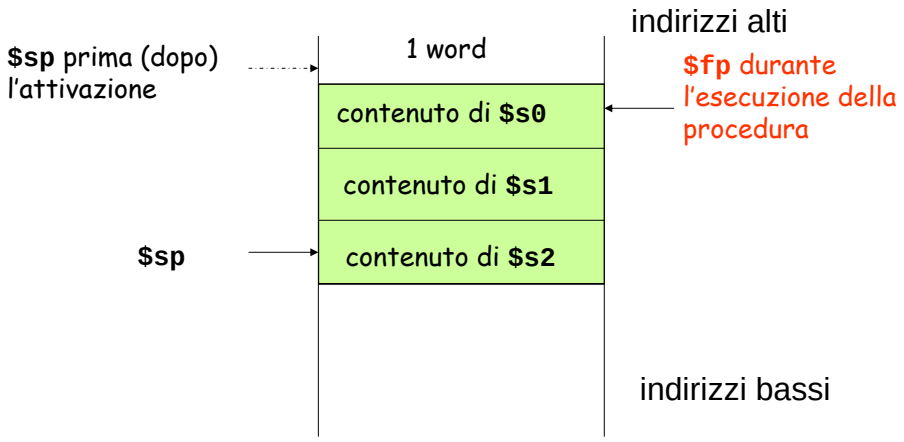
Organizzazione logica della memoria



Lo stack



Il frame pointer



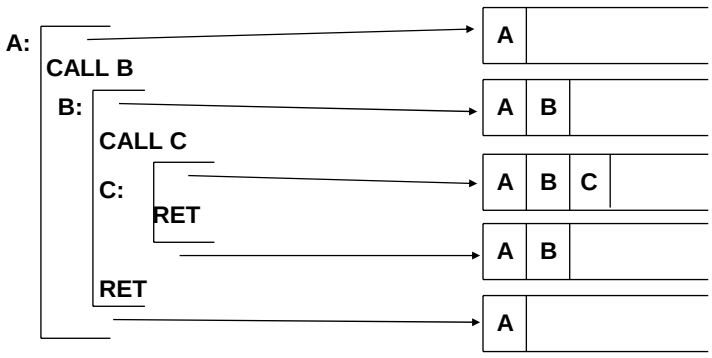
Uso dei registri: registro $sp

Nome	Numero	Utilizzo
$zero	0	costante zero
$at	1	riservato per l'assemblatore
$v0-$v1	2-3	valori di ritorno di una procedura
$a0-$a3	4-7	argomenti di una procedura
$t0-$t7	8-15	registri temporanei (non salvati)
$s0-$s7	16-23	registri salvati
$t8-$t9	24-25	registri temporanei (non salvati)
$k0-$k1	26-27	gestione delle eccezioni
$gp	28	puntatore alla global area (dati)
$sp	29	stack pointer
$s8	30	registro salvato (fp)
$ra	31	indirizzo di ritorno

$sp indica l'ultimo indirizzo dell'area di stack.

Calls: Why Are Stacks So Great?

Stacking of Subroutine Calls & Returns and Environments:



Some machines provide a memory stack as part of the architecture (e.g., VAX)

Sometimes stacks are implemented via software convention (e.g., MIPS)

Gestione dello stack nel MIPS

- Lo stack (pila) è una struttura dati costituita da una coda LIFO (last-in-first-out)
- Lo stack cresce **da indirizzi di memoria alti verso indirizzi bassi**
- Il registro **`$sp`** contiene l'indirizzo dell'ultima locazione utilizzata in cima allo stack.
- L'inserimento di un dato nello stack **(operazione di push)** avviene **decrementando `$sp`** per allocare lo spazio ed eseguendo una sw per inserire il dato.
- Il prelevamento di un dato dallo stack **(operazione di pop)** avviene eseguendo una lw ed **incrementando `$sp`** (per eliminare il dato), riducendo quindi la dimensione dello stack.

- Alla chiamata di procedura, lo spazio nello stack viene allocato **sottraendo** a **`$sp`** il numero di byte necessari:
  - Es: **`addi $sp,$sp,-24  #alloca 24 byte nello stack`**

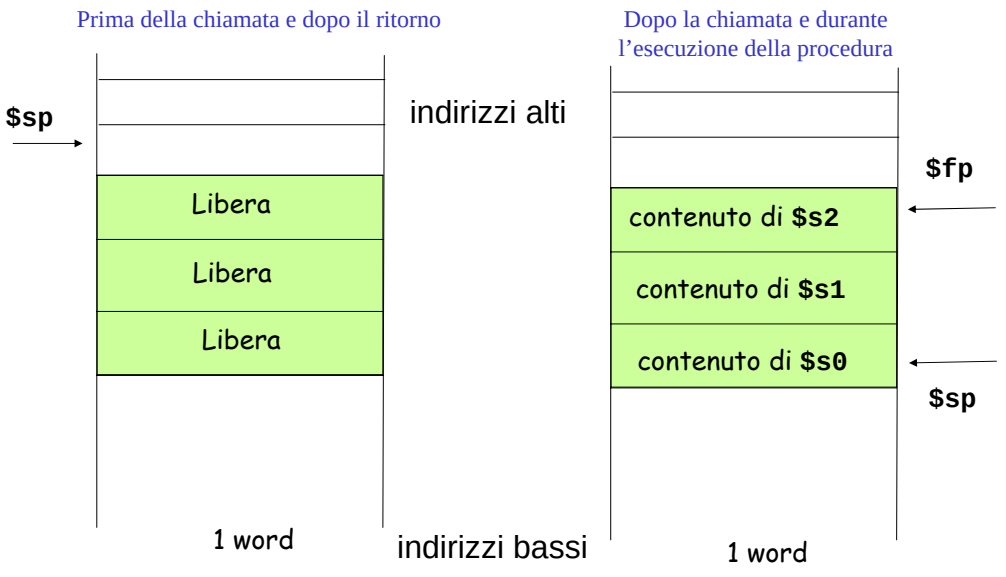
- Al rientro da una procedura il record di attivazione viene rimosso dalla procedura (deallocato) incrementando **`$sp`** della stessa quantità di cui lo si era decrementato alla chiamata
  - Es: **`addi $sp, $sp,24  #dealloca 24 byte nello stack`**

- È necessario liberare lo spazio allocato per evitare di riempire tutta la memoria

Esempio: procedura somma

```
Somma_algebrica:
  addi $sp,$sp,-12        # alloca nello stack lo spazio per i 3 registri
  sw $s0, 8($sp)          # salvataggio di $s0
  sw $s1, 4($sp)          # salvataggio di $s1
  sw $s2, 0($sp)          # salvataggio di $s2

  add $s0, $a0, $a1       # $t0 ← g + h
  add $s1, $a2, $a3       # $t1 ← i + j
  add $s2, $t0, $t1       # f ← $t0 - $t1

  add $v0, $s2, $zero     # restituisce f copiandolo nel reg. di ritorno $v0

  # ripristino del vecchio contenuto dei registri estraendolo dallo stack
  lw $s2, 0($sp)          # ripristino di $s0
  lw $s1, 4($sp)          # ripristino di $t0
  lw $s0, 8($sp)          # ripristino di $t1

  addi $sp, $sp, 12       # deallocazione dello stack per eliminare 3 registri
  jr $ra                        #ritorno al prog. chiamante
```


I registri nello stack



Sommario

Le procedure

Lo stack

La chiamata a procedura

Salvataggio dei registri

- Main e procedure lavorano sullo stesso Register File.
- Gli stessi registri possono essere utilizzati.

Occorre garantire che le variabili memorizzate nei registri dal main, non vengano toccate dalla procedura.

Potrei salvare l'intero register file in stack…. Oppure la parte che serve. Vengono definite delle convenzioni.

La chiamata ad una procedura intermedia

- Procedura **intermedia** è una procedura che ***ha*** annidate al suo interno chiamate ad altre procedure.

- Nel caso di procedure intermedia, il **chiamante** salva nello stack:
  - I registri temporanei di cui vuole salvare il contenuto, perché ne avrà bisogno dopo la chiamata (**$t0-$t9,...)**
  - I registri argomento (**$a0-$a3)** nel caso in cui il loro contenuto debba essere preservato (sono considerati registri temporanei).
  - Il contenuto dei registri **$v0, $v1** nel caso in cui il contenuto debba servire dopo la chiamata.

- Nel caso di procedure intermedia, il **chiamato** alloca nello stack:
  - I registri non temporanei che vuole utilizzare **`($s0-$s8)`**
  - Strutture dati locali (es: array, matrici) e variabili locali della procedura che non stanno nei registri.
  - I registri argomento nel caso in cui il loro contenuto debba servire dopo la chiamata.
  - I registri della procedura ($ra, $fp).

Lo stack pointer, **`$sp,`** è aggiornato per tener conto del numero di registri memorizzati nello stack; alla fine i registri vengono ripristinati e lo stack pointer riaggiornato.

Cosa occorre salvare in una procedura foglia?

Esempio - C

Eseguo s = (a+b) – (c+d) utilizzando due procedure annidate:

```
main
{
,…
res = opera(a,b,c,d)
….
}

int opera(int a, int b, int c, int d)
{
int t0 = a + b;
int t1 = c + d;
int result = diff(t0, t1);
return(result);
}

int diff (int t0, int t1)
{   retrun (t0 – t1);
}
```

Esempio – Assembly – Condizione iniziale

Suppongo a, b, c, d contenuti nei registri $s0, …. $s3; il risultato sarà inserito in $s5

```
main:
0x400  li $s0, 2
0x404  li $s1, 25
0x408  li $s2, 31
0x40C li $s3, 4

0x410  mov $a0, $s0
0x414  mov $a1, $s1
0x418  mov $a2, $s2
0x41C mov $a3, $s3,
0x420  jal opera
0x424  mov $s5, $v0
```

```
opera:
0x600    addi $sp, $sp, -4
0x604    sw $ra, 0($sp)
0x608    add $a0, $a0, $a1
0x60C    add $a1, $a2, $a3
0x610    jal diff
0x614    lw $ra, 0($sp)
0x618    addi $sp,$sp,4
0x61C    jr $ra
```

```
diff:
0x700    sub $v0, $a0, $a1
0x704    jr $ra
```

$a0 = 0	$ra = 0	$sp = 0x7fffffff

Esempio – Passo I – chiamata di opera

Suppongo a, b, c, d contenuti nei registri $s0, …. $s3; il risultato sarà inserito in $s5

```
main:
0x400  li $s0, 2
0x404  li $s1, 25
0x408  li $s2, 31
0x40C li $s3, 4

0x410  mov $a0, $s0
0x414  mov $a1, $s1
0x418  mov $a2, $s2
0x41C mov $a3, $s3,
0x420  jal opera
0x424  mov $s5, $v0
```

```
opera:
0x600    addi $sp, $sp, -4
0x604    sw $ra, 0($sp)
0x608    add $a0, $a0, $a1
0x60C    add $a1, $a2, $a3
0x610    jal diff
0x614    lw $ra, 0($sp)
0x618    addi $sp,$sp,4
0x61C    jr $ra
```

```
diff:
0x700    sub $v0, $a0, $a1
0x704    jr $ra
```

$a0 = 2	$ra = 0x424	$sp = 0x7ffffffc

Esempio – Passo II – chiamata di diff

Suppongo a, b, c, d contenuti nei registri $s0, …. $s3; il risultato sarà inserito in $s5

```
main:
0x400  li $s0, 2
0x404  li $s1, 25
0x408  li $s2, 31
0x40C li $s3, 4

0x410  mov $a0, $s0
0x414  mov $a1, $s1
0x418  mov $a2, $s2
0x41C mov $a3, $s3,
0x420  jal opera
0x424  mov $s5, $v0
```

```
opera:
0x600    addi $sp, $sp, -4
0x604    sw $ra, 0($sp)
0x608    add $a0, $a0, $a1
0x60C    add $a1, $a2, $a3
0x610    jal diff
0x614    lw $ra, 0($sp)
0x618    addi $sp,$sp,4
0x61C    jr $ra
```

```
diff:
0x700    sub $v0, $a0, $a1
0x704    jr $ra
```

$a0 = 27	$ra = 0x614	$sp = 0x7ffffffc

Come faccio a recuperare l'indirizzo di ritorno a main?

Storia della chiamata

Chiamante (main):

- Fare spazio nello stack per memorizzare nello stack i registri temporanei di eventuale interesse per il main (ad esempio i registri $t ed i registri $a) e memorizzarne il contenuto in stack (push).
- Mettere i parametri della procedura nei registri $a0, $a1, $a2, $a3 ed eventualmente in stack i parametri in eccesso.
- Trasferire il controllo alla procedura: definizione di un nome-etichetta per la procedura, es: **proc_name:**, ed esecuzione dell'istruzione *jal proc_name*.

Chiamato (procedura):

- Fare spazio nello stack per memorizzare i dati locali.
- Salvataggio nello stack dei registri variabile (registri $s) e dei registri temporanei di interesse per la procedura (registri $ra, $fp, $a, se si verificano delle chiamate ad altre procedure).

Prologo – corpo della procedura - Epilogo

Esempio

```
# Programma che stampa una stringa mediante procedura print
# Voglio utilizzare $s0 all'interno della procedura chiamata.
        .data
str:    .asciiz "benvenuti in xSPIM\n "
        .text
        .globl main
main:   la $s1, str             # $a0 ← ind. stringa da stampare
        li $s0, 16              # $v0 ← valore 16 da preservare
        move $a1, $s1
        jal stampa
        add $s0, $s0, $v0       # Devo utilizzare $v0 (il valore 16)
        li $v0, 10              # $v0 ← codice della exit
        syscall                 # esce dal programma

stampa: addi $sp, $sp,-4        # allocazione dello stack
        sw $s0, 0($sp)          # salvo $s0 che vado a
                                # modificare nella print
        li $s0, 4
        move $v0, $s0           # $v0 ← codice di print_string
        syscall                 # stampa della stringa
        lw $s0, 0($sp)          # ripristina il reg. $s0
        addi $sp,$sp,4          # deallocazione dello stack
        jr $ra
```


Procedure ricorsive

- Procedure che contengono una chiamata a se stesse al loro interno => il codice della procedura viene riutilizzato più volte, ogni volta con parametri diversi.

Calcolo del fattoriale di un numero intero

```
main(int argc, char *argv[])
{
    int n;
    printf("Inserire un numero intero\n");
    scanf("%d", &n);
    printf("Fattoriale: %d\n", fact(n));
}

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

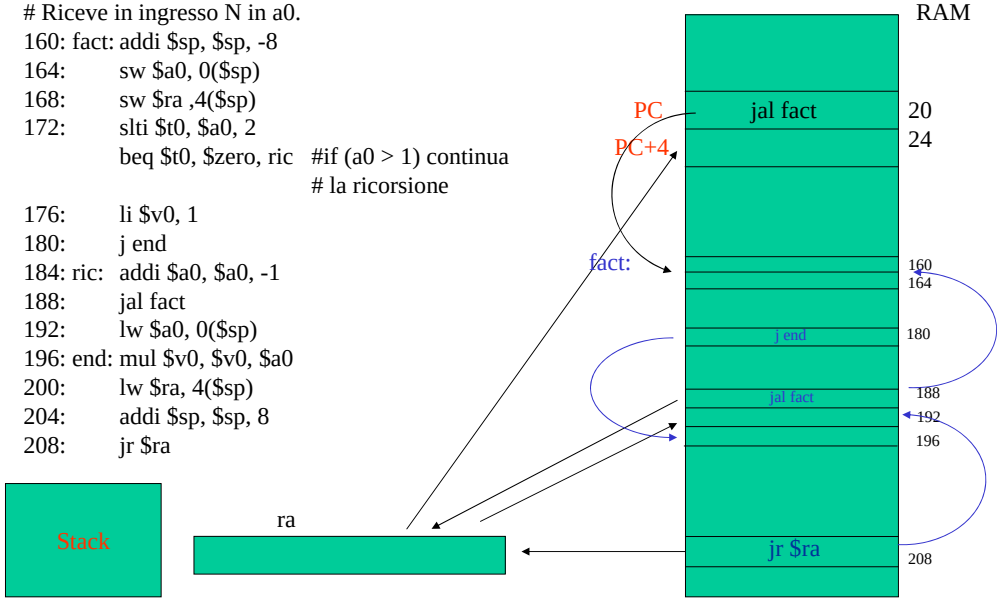
Procedure ricorsive, un passo oltre

Calcolo del fattoriale di un numero intero

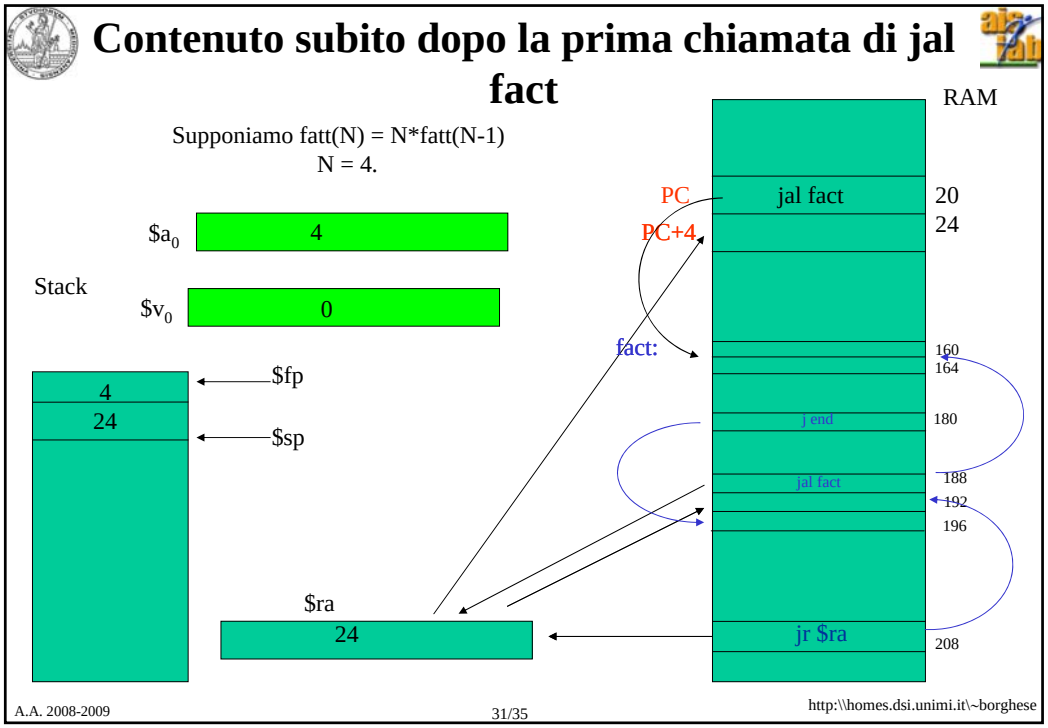
```
main(int argc, char *argv[])
{    int n;
     printf(”Inserire un numero intero\n");
     scanf("%d", &n);
     printf(”Fattoriale: %d\n", fact(n));
}

int fact(int m)
{  int res;
   if (m <= 1)
   {   res = 1;
       goto fine;
   }
   else
  {   res = fact(m-1);
       res = res * m;
  }
end;
  return(res);
}
```


La procedura ricorsiva fatt

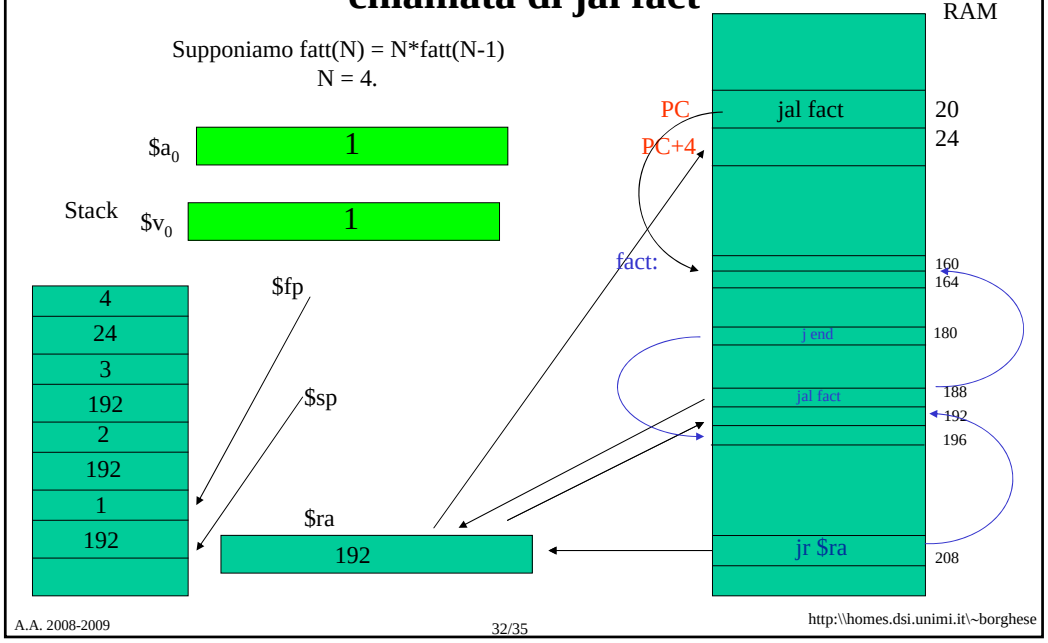
```
# Riceve in ingresso N in a0.
160: fact: addi $sp, $sp, -8
164:       sw $a0, 0($sp)
168:       sw $ra ,4($sp)
172:       slti $t0, $a0, 2
           beq $t0, $zero, ric   #if (a0 > 1) continua
                                 # la ricorsione
176:       li $v0, 1
180:       j end
184: ric:  addi $a0, $a0, -1
188:       jal fact
192:       lw $a0, 0($sp)
196: end:  mul $v0, $v0, $a0
200:       lw $ra, 4($sp)
204:       addi $sp, $sp, 8
208:       jr $ra
```



Contenuto subito dopo la prima chiamata di jal fact



Contenuto subito dopo l'ultima (quarta) chiamata di jal fact



Esempio, programma per il calcolo del fattoriale in Assembly – parte dichiarativa

```
# Programma che calcola il fattoriale iterativamente
        .data
prompt: .asciiz "Inserisci un numero intero"
output: .ascii "Il fattoriale è:"
        .text
        .globl main
main:
        li $v0, 4           # $v0 ← codice della print_string
        la $a0, prompt      # $a0 ←  indirizzo della stringa
        syscall             # stampa la stringa

        li $v0, 5           # $v0 ← codice della read_int
        syscall             # legge l'intero e lo carica in $v0
        move $s0, $v0       # Per pulizia, porta l'intero in
                            # un registro di varibili
```


Esempio, fattoriale – stampa risultato

```
# stampa il risultato

       move $a0, $v0        # $a0 ← $v0
       li $v0, 1            # $v0 ← codice della print_int
       syscall              # stampa l'intero in input

       li $v0, 4            # $v0 ← codice della print_string
       la $a0, output       # $a0 ←  indirizzo della stringa
       syscall              # stampa la stringa

       li $v0, 1            # $v0 ← codice della print_int
       move $a0, $t1        # $a0 ← n!
       syscall              # stampa n!

       li $v0, 10           # $v0 ← codice della exit
       syscall              # esce dal programma
```

Programma che somma i quadrati dei primi N numeri

```
# N e' memorizzato in t1.

.data
str:
        .asciiz "La soma da 0 a N vale "
.text
.globl main

main:

          li        $t1, 7    # N=7, interi di cui calcolare la somma dei quadrati
          li        $t6, 0    # t6 e' indice di ciclo
          li        $t8, 0    # t8 contiene la somma dei quadrati
Loop:
          mult      $t7, $t6 $t6        # t7 = t6 x t6
          addu      $t8, $t8, $t7       # t8 = t8 + t7
          addi      $t6, $t6, 1         # t6++
          blt       $t6, $t1, Loop      # if t6 < N stay in loop

          la        $a0, str
          li        $v0, 4              #print
          syscall

          li        $v0, 1              #print
          add       $a0, $t8, $zero
          syscall

          li $v0, 10                    # $v0 codice della exit
          syscall                       # esce dal programma
```


Sommario

Le procedure

Lo stack

La chiamata a procedura