# Il linguaggio assembly – Parte IV

Architettura
degli Elaboratori e delle Reti

Alberto Borghese
Università degli Studi di Milano
Dipartimento di Scienze dell'Informazione
email: borghese@dsi.unimi.it



## Struttura di una procedura

- Ogni procedura ha:
  - un prologo
    - Salvataggio dell'ambiente
  - un corpo
    - Esecuzione della procedura vera e propria
  - un epilogo
    - Ripristino dell'ambiente



## Struttura di una procedura

- Ogni procedura ha:
  - **un prologo**
    - Salvataggio dei registri di interesse
    - Mettere i parametri della procedura in un luogo accessibile.
    - Acquisire le risorse necessarie per memorizzare i dati interni alla procedura.
    - Trasferire il controllo alla procedura.
  - **un corpo**
    - Esecuzione della procedura vera e propria
  - **un epilogo**
    - Mettere il risultato in un luogo accessibile al programma chiamante.
    - Ripristino dei registri di interesse.
    - Liberare le risorse utilizzate dalla procedura
    - Restituzione del controllo alla procedura chiamante.



## Prologo

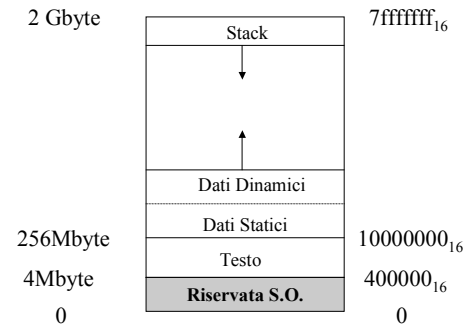
- Salvataggio dei registri di interesse nello stack (i registri $s e $f se vengono utilizzati dalla procedura internamente).
- Salvataggio dei parametri della procedura se utilizzati ancora dopo la chiamata (nei registri $a0, $a1, $a2, $a3 e nello stack).
- Fare spazio nello stack per memorizzare i dati locali.
- Trasferire il controllo alla procedura (definizione di un nome-etichetta per la procedura, es: **`proc_name:`**)

## Prologo – record di attivazione

- Determinazione della dimensione del record di attivazione
- Per determinare la dimensione del record di attivazione si deve stimare lo spazio per:
  - registri interi da salvare.
  - registri floating-point da salvare.
  - registri per argomenti.
  - registri per variabili locali.



## Organizzazione logica della memoria






## Prologo – creazione spazio in stack

- Allocazione dello spazio sullo stack: aggiornare il valore di **$sp:**

```
addi $sp,$sp,-dim_record_attivaz

# lo stack pointer viene decrementato
# della dimensione prevista per la procedura
```



## Prologo – salvataggio registri

- Salvataggio dei registri per i quali è stato allocato spazio nello stack:

```
sw reg,dim_record_attivaz-N($sp)
```

  - **N (N ≥ 4)** viene incrementato di 4 ad ogni salvataggio

## Esempio di salvataggio dei registri

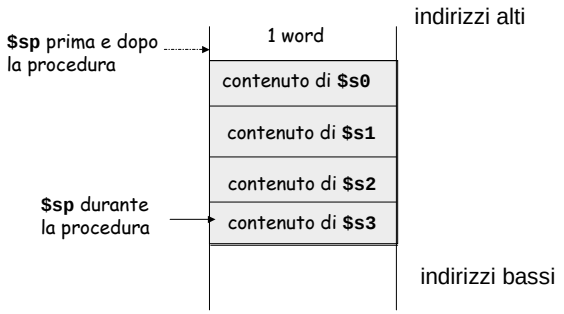
- Record di attivazione: 16 byte

```
addi $sp,$sp,-16

sw $s0, 12($sp) dim_record_attivazione - 4
sw $s1, 8($sp)  dim_record_attivazione - 8
sw $s2, 4($sp)  dim_record_attivazione - 12
sw $s3, 0($sp)  dim_record_attivazione - 16
```



## Esempio di salvataggio dei registri (cont.)






## Corpo della procedura

- Stesura delle istruzioni per l'esecuzione delle funzionalità previste dalla procedura



## Epilogo

- Ripristino dei registri di interesse dallo stack (i registri $s e $f se vengono utilizzati dalla procedura internamente).
- Restituzione dei parametri della procedura (dai registri $v0, $v1 e dallo stack).
- Eliminare lo spazio dello stack in cui sono stati memorizzati i dati locali.
- Trasferire il controllo al programma chiamante.

## Epilogo della procedura

- Ripristino dei registri salvati:
  `lw reg, dim_record_attivaz - N($sp)`
- Rimozione dello spazio allocato sullo stack:
  `addi $sp,$sp,dim_record_attivaz.`



## Cosa manca?

- Registro $ra.
- Restituzione del controllo al chiamante:
  `jr $ra`
- Si ritorna all'istruzione successiva a quella che ha chiamato la procedura.



## Procedura chiamante

- Salva i registri $t di cui vuole preservare il contenuto.
- Copia eventuali argomenti in numero superiore a quattro nello stack (oltre a quelli contenuti nei registri **$a0-$a3**)
- Esegue:
  `jal proc_name`



## Esempio 1

```
# Programma che stampa una stringa mediante
# procedura print
        .data
str:    .asciiz "benvenuti in xSPIM\n "
        .text
        .globl main
main:   la $a0, str             # $a0 ← ind. stringa da stampare
        li $v0, 16              # $v0 ← valore 16 da preservare
        jal print
        li $v0, 10              # $v0 ← codice della exit
        syscall                 # esce dal programma

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

## Esempio 2

```
# Programma che copia tramite una procedura gli interi positivi
# contenuti nell'array (elem) in un secondo array (pos_el)
```

- Inizializzazione
- Leggi gli interi
- Individua gli interi positivi
- Copia gli interi positivi
- Stampa gli interi positivi



## Esempio 2 – Versione in linguaggio C

```
main()
{       int i, k = 0, N=10, elem[10], pos_el[10];

        elem = leggi_interi(N);
        for (i=0; i<N;i++)
                if (elem[i] > 0)
                {       pos_el[k] = elem[i]; k++;
                }
        printf("Il numero di interi positivi e' %d",k);
        for (i=0; i<k; i++)
                printf(" %d",pos_el[i]);
        exit(0);
}
```



## Esempio 2 – estrazione di numeri positivi

```
# Programma che copia tramite una procedura gli interi positivi
# contenuti nell'array (elem) in un secondo array (pos_el)
        .data
elem:   .space 40               # alloca 40 byte per array elem
pos_el: .space 40               # alloca 40 byte per array pos_el
prompt: .asciiz "Inserire il successivo elemento \n"
msg:    .asciiz "Gli elementi positivi sono \n"
        .text
        .globl main

main:   la $s0, elem            # $s0 ← indirizzo di elem.
        la $s1, pos_el          # $s1 ← indirizzo di pos_el
        li $s2, 40              # $s2 ← numero di elementi (in byte).

# Ciclo di lettura dei 10 numeri interi
        move $t0, $s0           # indice di ciclo
        add $t1, $t0, $s2       # $t1 ← posizione finale di elem

loop:   li $v0, 4               # $v0 ← codice della print_string
        la $a0, prompt          # $a0 ←  indirizzo della stringa
        syscall                 # stampa la stringa prompt
```



## Esempio 2 (cont.)

```
# Lettura dell'intero
        li $v0, 5               # $v0 ← codice della read_int
        syscall                 # legge l'intero e lo carica in $v0

        sw $v0, 0($t0)          # memorizza l'intero in elem
        addi $t0, $t0, 4
        bne $t0, $t1, loop

# Fine della lettura, ora prepara gli argomenti per la proc.
        move $a0, $s0           # $a0 ← ind. array elem.
        move $a1, $s1           # $a1 ← ind. array pos_el
        move $a2, $s2           # $a2 ← dim. in byte dell'array

# Chiamata della procedura cp_pos
        jal cp_pos              # restituisce il numero di byte
                                # occupati dagli interi pos in $v0
```

## Esempio 2 (cont.)

```
# ciclo di stampa degli elementi positivi
# Gli indirizzi in cui e' racchiuso pos_elem vanno da $s2 a $t2
        add $t2, $s1, $v0       # $t2 ← ind. fine ciclo pos_el
        move $t0, $s1           # $t0 ← ind. di ciclo

loop_w: beq $t2, $t0, exit_main
        li $v0, 1               # $v0 ← codice della print_integer
        lw $a0, 0($t0)          # $a0 ← intero da stampare
        syscall                 # stampa dell'intero
        addi $t0, $t0, 4
        j loop_w

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



## Esempio 2 (procedure)

```
# Questa procedura esamina l'array elem ($a0) contenente $a2/4 elementi
# Resituisce in $v0, il numero di byte occupati dagli interi positivi e
# in pos_el gli interi positivi
cp_pos: addi $sp, $sp, -16      # allocazione dello stack
        sw $s0, 0($sp)
        sw $s1, 4($sp)
        sw $s2, 8($sp)
        sw $s3, 12($sp)
        move $s0, $a0           # $s0 ← ind. ciclo su elem
        add $s1, $a0, $a2       # $s1 ← ind. di fine ciclo su elem
        move $s2, $a1           # $s2 ← ind. ciclo su pos_el
next:   beq $s1, $s0, exit      # se esaminato tutti gli elementi di elem
                                #       salta alla fine del ciclo (exit).
        lw $s3, 0($s0)          # carica l'elemento da elem
        addi $s0, $s0, 4        # posiziona sull'elemento succ. di elem
        ble $s3, $zero, next    # se $s3 ≤ 0 va all'elemento succ. di elem
        sw $s3, 0($s2)          # Memorizzo il numero in pos_elem
        addi $s2, $s2, 4        # posiziona sull'elemento succ. di pos_el
        j next                  # esamina l'elemento successivo di elem
exit:   sub $v0, $s2, $a1       # salvo lo spazio in byte dei pos_el
        lw $s0, 0($sp)
        lw $s1, 4($sp)
        lw $s2, 8($sp)
        lw $s3, 12($sp)
        addi $sp,$sp,12         # deallocazione dello stack
        jr $ra                  # restituisce il controllo al chiamante
```



## Esercizi

- Estrazione di numeri pari da un vettore di N numeri interi.
- Scrivere un algoritmo di bubble-sort o quick-sort.
- Ricerca di una stringa (e.g. "nome") all'interno di un testo.



## Procedure annidate

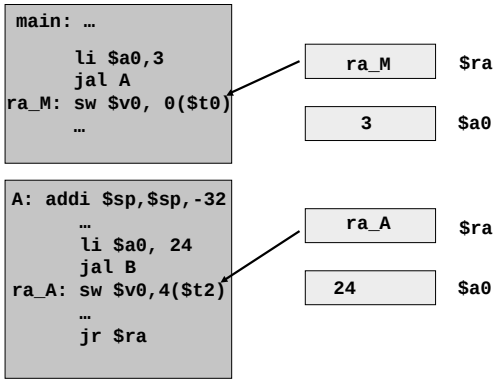
- Sono procedure che richiamano al loro interno altre procedure (non sono procedure foglia)
- Devono salvare nello stack un ambiente più ampio
- Rispetto alle procedure foglia, devono essere salvati anche:
  - i parametri di input della procedura ($a0, $a1, $a2, $a3)
  - l'indirizzo di ritorno ($ra)
    - la procedura chiamata all'interno di un'altra riscrive il contenuto di **$ra**
    - La procedura chiamata può richiedere dei parametri ed in questo caso i registri $ai verrebbero riscritti.

## Procedure ricorsive

- Procedure che contengono una chiamata a se stesse al loro interno
- Devono salvare nello stack
  - i parametri di input della procedura
  - l'indirizzo di ritorno
  - eventuali risultati intermedi

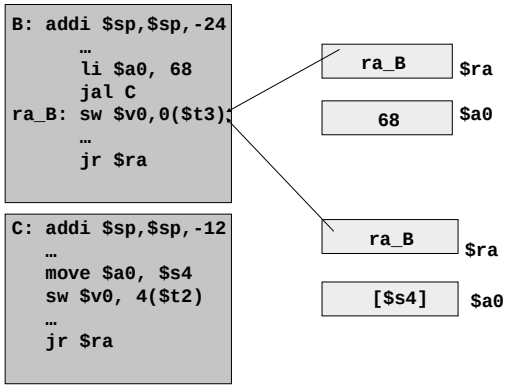


## Procedure ricorsive & annidate






## Procedure ricorsive & annidate






## Fattoriale

```
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 == 0)
    return(1);
  else
    return(m*fact(m-1));
}
```

## Fattoriale (Implementazione assembly)

```
# programma per il calcolo ricorsivo di n!
        .data
prompt: .asciiz "Inserire un numero intero:"
output: .ascii  "Fattoriale:"

        .text
        .globl main
main:
# Lettura dell'intero di cui si calcola il fattoriale
        li $v0, 4       # $v0 ← codice della print_string
        la $a0, prompt  # $a0 ← indirizzo della stringa
        syscall         # stampa della stringa

        li $v0, 5       # $v0 ← codice della read_int
        syscall         # legge l'intero n e lo carica in $v0
```



## Fattoriale (cont.)

```
# Calcolo del fattoriale
    move $a0, $v0       # $a0 ← n
    jal  fact           # chiama fact(n)
    move $s0, $v0       # $s0 ← n!

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

    move $a0, $s0       # $a0 ← n!
    li $v0, 1           # $v0 ← codice della print_int
    syscall             # stampa n!

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



## Fattoriale (procedura)

```
fact:
    addi $sp, $sp,-8          # alloca stack
    sw   $ra, 4($sp)          # salvo return address
    sw   $a0, 0($sp)          # salvo l'argomento n

    bgt  $a0, $zero, core     # se n > 0 salta a core
    li   $v0, 1               # $v0 ← 1
    j    end

core:
    addi $a0, $a0,-1          # decremento n → (n–1)
    jal  fact                 # chiama fact(n-1) in $v0

    lw   $a0, 0($sp)          # ripristino n in $v1
    mul  $v0, $a0, $v0        # ritorno n * fact (n-1)

end:
    lw   $ra, 4($sp)          # ripristino return address
    addi $sp, $sp, 8          # dealloca stack
    jr   $ra                  # ritorno al chiamante
```



## Fibonacci

```
main(int argc, char *argv[])
{
int n;

printf("Inserire un intero\n");
scanf("%d", &n);

printf("Numero di Fibonacci di %d = %d\n", n, fib(n));
}

int fib(int m)
{
if (m == 1)
  return(1);
else
  if (m == 0)
    return(0);
  else
    return(fib(m-1)+fib(m-2));
}
```

## Fibonacci (cont.)

```
        .data
prompt: .asciiz "Inserire un numero intero: \n"
output: .asciiz "Numero di Fibonacci: "

        .text
        .globl main
main:
        li $v0, 4
        la $a0, prompt
        syscall            # stampa la stringa

         li $v0, 5
        syscall            # legge l'intero
```



## Fibonacci (cont.)

```
# calcola fibonacci(n)

        move $a0, $v0
        move $a1, $0
        jal  fib

# stampa il risultato ed esci
        move $a1, $v0  # salva il valore restituito da fib in t0
        li $v0, 4
        la $a0, output
        syscall

        move $a0, $a1
        li $v0, 1
        syscall

        li $v0, 10
        syscall
```



## Fibonacci (cont.)

```
# Fibonacci. Legge la somma parziale in a1 e in a0 il numero.
# Restituisce v0 = fib(a0-1)->a1 + fib (a0-2)->v0.
fib:
        addi $sp, $sp,-12
        sw   $ra, 8($sp)
        sw   $a0, 4($sp)      #salva i parametri in input
        sw   $a1, 0($sp)      #salva il risultato della precedente
                              #chiamata di fib
        li   $t0, 1
        bgt  $a0, $t0, core   # se n> 1, continua,
                              # altrimenti restituisci n
        move $v0,$a0          # n = 0 or n = 1.
        j    return

core:
        addi $a0, $a0,-1      # n → n-1
        jal  fib              # chiama fib(n-1)
        move $a1, $v0         # salva fib(n-1) nello stack (a1)
        addi $a0, $a0,-1      # $a0 diventa n-2
        jal  fib              # esegue fib(n-2)
        add  $v0, $v0, $a1    # somma  fib(n-1) e fib(n-2)

return:
        lw $ra, 8($sp)
        lw $a0, 4($sp)
        lw $a1, 0($sp)
       addi $sp, $sp, 12
        jr $ra
```



## Esempio - Numeri primi – bozza in C

```
main(int argc, char *argv[])
{
int primes_test = [1 2 3 5 7 11 13 17 19 23];
int primes[10], n_primes, n_primes_test = 10, i;

printf("Inserire un intero\n"); scanf("%d", &n);

printf("I numeri primi sono: ");

[n_primes primes] = find_primes(n, n_primes_test, primes_test, primes)
for (i=0; i< n_primes; i++) printf("%d ",primes[i]);
exit(0);
}

[int n_primes, int primes[10]]  = find_primes(int n, int n_primes_test, int primes_test[10],
          int primes[10])
{  int t = n_primes_test, k = 0, l = n;
   while (t > 0)
   {  [primes[10], t] = find_first_prime(l, n_primes_test, primes_test[10],  primes[10], t);
   }
   return(….);
}
```

## Numeri primi - Assembly

```
        .data
prompt: .asciiz "Inserire un numero intero: "
output: .asciiz "I numeri primi sono: "
primes_test: .word 1 2 3 5 7 11 13 17 19 23
primes: .space 40
spazio: .asciiz " "

        .text
        .globl main
# $s0 ($a0) contiene il numero di numeri primi da testare (in byte).
# $s1 ($a1) contiene l'indirizzo del vettore dei numeri primi da testare
# $s2 ($a2) contiene N, il numero primo da scomporre
# $s3 ($a3) contiene l'indirizzo del vettore dei numeri primi di N
# $s4 ($v0) contiene il numero di numeri primi di N (in byte).

main:
       li $s0, 40     # Impostato alla lunghezza di primes_test (in byte)
       la $s1, primes_test

       li $v0, 4
        la $a0, prompt
        syscall              # stampa prompt
```



## Numeri primi (cont.)

```
            li $v0, 5
           syscall        # legge l'intero N
            move $s2, $v0

            la $s3, primes

            move $a0, $s0  # prepara la chiamata
            move $a1, $s1  # prepara la chiamata
            move $a2, $s2  # prepara la chiamata
            move $a3, $s3  # prepara la chiamata

            jal find_primes

            move $s4, $v0  # numero di primi di N (in byte)

     # scrivo anche il numero 1 come costituendo di N
            lw $t0, 0($s1)
            add $t1, $s4, $s3
            sw $t0, 0($t1)
            addi $s4, $s4, 4
```



## Numeri primi (cont.)

```
        # stampa il risultato ed esci
               li $v0, 4
               la $a0, output
               syscall

               move $t0, $s3
               add $t1, $s4, $t0
        Loop_w:
                beq $t0, $t1, fine

               li $v0, 4             # Stampa uno spazio
               la $a0, spazio
               syscall

                lw $a0, 0($t0)
                li $v0, 1
               syscall

                addi $t0, $t0, 4
                j Loop_w
        fine:
               li $v0, 10
               syscall
```



## Numeri primi (cont.)

```
    find_primes:
           addi $sp, $sp, -8
           sw $ra, 0($sp)
           sw $a0, 4($sp)

           move $v0, $zero       # memorizza il numero di primi di N
                                 # in numero di byte
           addi $a0, $a0, -4     # elemento di prime_test

    # in find_first_prime:
    #       a2 viene diviso via via per tutti i primi
    #       a0 viene decrementato via via che i primi sono esaminati.

           jal find_first_prime

           lw $ra, 0($sp)
           lw $a0, 4($sp)
           addi $sp, $sp, 8
           jr $ra
```

# Numeri primi (cont.)

```
find_first_prime:
        addi $sp, $sp, -4
        sw $ra, 0($sp)

        beq $a0, $zero, fine_1    # il ciclo termina quando ci si posiziona sul
                                  # primo elemento di primes_test (cioe' base + 4)
                                  # (primo elemento di prime_test che e' = 1)
        add $t0, $a1, $a0
        lw $t2, 0($tx0)           # estraggo il numero primo da testare
        addi $a0, $a0, -4         # mi posiziono sul numero primo successivo da
                                  # esaminare
        rem $t3, $a2, $t2         # guardo se $t2 e' un divisore di N
        bgtz $t3, oltre           # se non e' un divisore di N, passo al successivo
                                  # numero primo da esaminare
        div $a2, $a2, $t2         # divido N per il numero primo trovato
        add  $t4, $v0, $a3        # $t4 indirizzo su primes in cui scrivere
        sw $t2, 0($t4)            # memorizzo il numero primo trovato
        addi $v0, $v0, 4          # punto al nuovo elemento in primes
oltre:
        jal find_first_prime
fine_1:
        lw $ra, 0($sp)
        addi $sp, $sp, 4
        jr $ra
```