UNIVERSITÀ DEGLI STUDI DI MILANO
matteo.re@unimi.it
https://homes.di.unimi.it/re/arch2-lab-2015-2016.html

DIPARTIMENTO DI INFORMATICA
Laboratorio di Architetture degli Elaboratori II
Turno B : (G – Z)

6

Procedure

SOMMARIO:

1) Procedure e funzioni
2) Convenzioni di chiamata in MIPS
3) Convenzioni chiamante
4) Linkage (andata & ritorno)
5) Passaggio di argomenti
6) Restituire un risultato
7) Convenzioni chiamato
8) Call frame
9) Step fondamentali chiamata a procedura/funzione
10) Esempio : calcolo potenza
11) Esempio : calcolo sommatoria di sei numeri
12) Esercizi

Ultimo aggiornamento: 21/4/2016

Procedure e funzioni

Nei linguaggi ad alto livello (ad es. C/C++) le procedure sono definite come funzioni che non ritornano alcun valore o ritornano un tipo speciale detto **void** che svolge il ruolo di segnaposto, o «tipo di ritorno fittizio».

Parleremo dunque di **funzione** solo nel caso in cui il blocco di codice chiamato restituisca al chiamante almeno un valore. In caso contrario il termine appropriato da utilizzare è **procedura**.

La **chiamata** di una procedura/funzione richiede due tipi di attività:

- Linkage
- Passaggio di argomenti (nel caso in cui essi siano richiesti)

Chiamata a procedura/funzione

(scenario ad alto livello)

```
f = f + 1;

if (f == g)
        res = funct(f,g);

else
        f = f - 1;
```

Procedura **chiamante** (**caller**)

```
int funct (int p1, int p2){

        int out;
        out = p1 * p2;
        return out;
}
```

Procedura **chiamata** (**callee**)

- Caller e callee comunicano attraverso il passaggio dei parametri di input alla funzione e di output restituito (nel caso della chiamata a funzione)

Convenzioni di chiamata in MIPS

Le convenzioni di chiamata riguardano:

- Utilizzo dei registri
- Passaggio di argomenti
- Backup dei valori dei registri

Possiamo identificare due classi principali di procedure/funzioni:

- Procedure/funzioni foglia (leaf) : sono procedure/funzioni che **non effettuano chiamate ad altre procedure/funzioni** (incluse esse stesse)
- Procedure/funzioni non-foglia (not-leaf) : sono procedure che contengono chiamate ad altre procedure/funzioni

Le convenzioni di chiamata standard definiscono una serie di azioni specifiche sia per la procedura/funzione chiamante che per quella chiamata.

Formato di procedura/funzione

Il formato di base per la dichiarazione di una procedura utilizza la direttiva **global** , una direttiva che dichiara un punto di ingresso (**.ent**) ed una label alla quale corrisponde l'inizio della procedura/funzione. In generale la fine di una procedura viene indicata dalla direttiva **end** . La sintassi generale è la seguente:

```
.globl  procedureName
.ent    procedureName
procedureName:

# code goes here

.end  procedureName
```

L'utilizzo della direttiva **end** in QtSpim è opzionale ma è buona norma utilizzarla per rendere più leggibile il codice.

Convenzioni (chiamante) I

Le convenzioni per la procedura/funzione (d'ora in poi procedura per brevità) chiamante servono a definire un modo standardizzato per soddisfare le specifiche necessità della procedura chiamante. In particolare:

- La procedura chiamante è responsabile per il salvataggio del contenuto di qualsiasi registro il cui valore potrebbe essere modificato dalla procedura chiamata (**$a0 - $a3**, **$t0 - $t9**, **$v0**, **$v1**, **$f0 - $f10**, **$f16 - $f18**) che risultino necessari dopo il completamento dell'esecuzione della procedura chiamata.
- La procedura chiamante deve passare tutti gli argomenti richiesti dalla procedura chiamata:
  - Primo argomento va posto in **$a0** (se di tipo intero) o in **$f12** (se float a precisione singola o double)
  - Secondo argomento va posto in **$a1** (se di tipo intero) o in **$f14** (se float a precisione singola o double)
  - Terzo argomento va posto in **$a3** (solo se è di tipo intero)
  - Se il terzo argomento è di tipo float deve essere messo sullo stack
  - Quarto argomento va posto in **$a3** (solo se è di tipo intero)
  - Se il quarto argomento è di tipo float deve essere messo sullo stack
  - **EVENTUALI ARGOMENTI RIMANENTI VANNO POSTI SULLO STACK** (in ordine **inverso** a quello in cui **compaiono** nel set di argomenti della procedura chiamata. LIFO).
- Deviazione del flusso di controllo (salto a codice procedura, chiamata a procedura)

Convenzioni (chiamante) II

E' necessario tenere presente una differenza importanze riguardante le possibili modalità attraverso le quali vengono trasferiti gli **argomenti** dalla procedura chiamante a quella chiamata:

- Nel caso in cui essi siano passati per **riferimento** verrà utilizzata *load address* (**la**) per ottenerli.
- Nel vaso in cui vengano passati per valore verrà utilizzata **load** per ottenerli.

La chiamata vera e propria (il salto alla procedura chiamata) dovrebbe essere effettuata mediante **jal** come segue:

jal <nomeprocedura>

Al momento del **completamento** della procedura il chiamante **deve**:

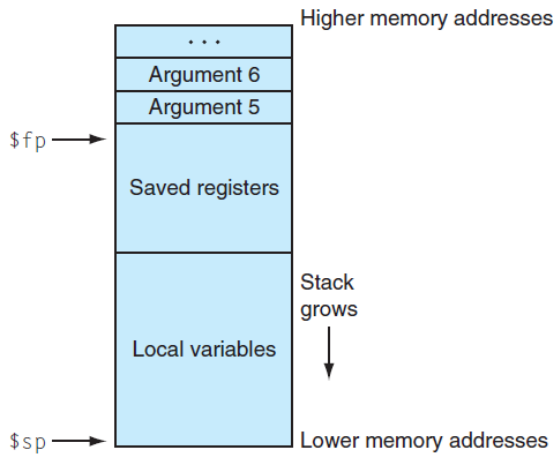
- Ripristinare i valori dei registri dei quali aveva effettuato il backup prima di chiamare la procedura chiamata
- Ripristinare lo stack pointer **$sp** in modo opportuno nel caso in cui alcuni argomenti siano stati passati attraverso lo stack

Chiamata a procedura

- Le **informazioni di servizio** (registri callee-saved e variabili locali) di una procedura sono contenute nello **stack frame**: un segmento di stack associato ad **ogni** procedura (anche detto *record di attivazione*).

- Chiamate e ritorni da procedure seguono un ordine **LIFO**: **l'ultima** procedura invocata è la **prima** a terminare.

- Gli **stack frame** sono allocati sullo stack seguendo la sequenza delle chiamate (lo stack è gestito con una convenzione LIFO)



Linkage (andata E ritorno) I

Il termine Linkage si riferisce al processo di andata alla procedura chiamata e al **ritorno nella posizione corretta** nel codice della procedura chiamante. Esso non contempla in alcun modo il passaggio di argomenti.

Le operazioni di linkage di base vengono realizzate utilizzando le istruzioni **jal** (jump and link) e **jr** (jump to reg(content)). **Entrambe** le istruzioni utilizzano il registro **$ra** . Il valore di questo registro è settato all'indirizzo di ritorno durante la procedura di chiamata.

La chiamata a procedura richiede la conoscenza del **nome** della procedura.

jal <procName> # al posto di <procName> inserire il nome della procedura

jal copia il contenuto di **$pc** in **$ra** e poi salta al punto di ingresso della procedura chiamata. **$pc** contiene l'indirizzo della **prossima** istruzione da eseguire (quella **dopo** la chiamata) e quindi è il punto corretto a cui tornare dopo il suo completamento.

SE la procedura chiamata non chiama altre procedure non ci sono altre necessità rispetto a **$ra**. Ma se effettua chiamate il valore di **$ra** va preservato (backup).

Linkage (andata E ritorno) II

Il ritorno da una procedura viene effettuato mediante l'istruzione **jr** . Il registro da utilizzare (come avrete intuito) è $ra. Quindi la modalità standard di ritorno è la seguente:

jr $ra

Se la procedura chiamata è di tipo foglia (non chiama nessuno) non vi è necessità di salvare il valore di $ra. In caso contrario il suo valore va preservato e va messo sullo stack e **recuperato** dalla procedura stessa (nel suo ruolo di **chiamante** di un'altra procedura). Solo così la procedura sarà in grado di tornare **alla sua procedura chiamante** in modo corretto.

Le eventuali operazioni di backup del valore di $ra e del suo ripristino vengono tipicamente svolte una sola volta e sono poste all'inizio ed alla fine della procedura non-foglia.

Passaggio di argomenti I

A seconda dello specifico contesto gli argomenti possono essere passati per valore o per riferimento. Il passaggio di argomenti avviene mediante un utilizzo congiunto dei registri e dello stack.

- Passaggio per valore: effettua una **copia** del valore degli argomenti da passare alla procedura chiamata. Di conseguenza i valori originali **non possono essere modificati**.
- Passaggio per riferimento (per indirizzo): si basa sul passaggio degli **indirizzi delle variabili** coinvolte. In questo caso è possibile modificare il loro valore.

Le convenzioni per il passaggio di argomenti sono le seguenti (già viste prima):

- Primo argomento va posto in **$a0** (se di tipo intero) o in **$f12** (se float a precisione singola o double)
- Secondo argomento va posto in **$a1** (se di tipo intero) o in **$f14** (se float a precisione singola o double)
- Terzo argomento va posto in **$a3** (solo se è di tipo intero)
- Se il terzo argomento è di tipo float deve essere messo sullo stack
- Quarto argomento va posto in **$a3** (solo se è di tipo intero)
- Se il quarto argomento è di tipo float deve essere messo sullo stack
- **EVENTUALI ARGOMENTI RIMANENTI VANNO POSTI SULLO STACK** (in ordine **inverso** a quello in cui **compaiono** nel set di argomenti della procedura chiamata. LIFO).

Passaggio di argomenti II

Se il primo argomento è un intero si utilizza **$a0** e **$f12** non va utilizzato. Se il primo argomento è di tipo float **$f12** viene utilizzato e **$a0** non va utilizzato. Ogni argomento aggiuntivo (oltre i 4 nel caso degli interi ed oltre il secondo nel caso dei float) **va passato usando lo stack**.

	1st	2nd	3rd	4th	5th	Nth
integer	**$a0**	**$a1**	**$a2**	**$a3**	stack	stack
	or	or	or	or		
floating-point value	**$f12**	**$f14**	stack	stack	stack	stack

E' possibile utilizzare questa tabella come riferimento. Gli **indirizzi** sono **interi** (anche se puntano a dei float) e quindi gli indirizzi vanno passati attraverso i **registri per gli interi**.

Risultato/i di funzione

I registri **$v0** o **$v0/$v1** vengono utilizzati per restituire valori interi da chiamate a funzione. I registri **$f0** e **$f2** vengono utilizzati per restituire dei valori float da chiamate a funzione.

Convenzioni backup registri

Le convenzioni MIPS riguardanti le chiamate richiedano che solo alcuni registri (non tutti) siano preservati tra una chiamata e l'altra.

I registri **$s0 - $s7** devono essere salvati (backup) e rispristinati dalla procedura

I registri **$f20 - $f30** devono essere salvati (backup) e ripristinati dalla procedura

Questo richiede che **mentre si scrive una procedura** i registri **$s0 - $s7** o **$f20 - $f30** siano messi sullo stack e rispristinati se la procedura li utilizza e li modifica. Quando una procedura **viene chiamata** i valori da passare tra le procedure vengono posti nei registri **$s0 - $s7** o **$f20 - $f30** .

I registri per gli interi **$t0 - $t9** e i registri floating point **$f4 - $f10** e **$f16 - $f18** (precisione singola o precisione doppia) sono utilizzati per valori temporanei (backup non richiesto).

Altre convenzioni per utilizzo registri

I registri **$at**, **$k0** e **$k1** sono riservati per l'assemblatore e per il sistema operativo (non dovrebbero essere utilizzati nei programmi). Il registro **$fp** punta al frame di chiamata della procedura posto sullo stack. Esso può essere utilizzato quando alcuni argomenti sono passati tra procedure per mezzo dello stack.

Il registro **$gp** viene utilizzato per puntare ad aree dati globalmente accessibili (non si utilizza se non in casi estremamente particolari).

Riepilogo convenzioni chiamato

Le convenzioni per la procedura chiamata (callee) sono le seguenti:

- Registri di cui è necessario fare il backup (verranno modificati) → push sullo stack
  - Questi includono **$s0-$s7** , **$f20-$f30**, **$ra**, **$fp** e (eventualmente) **$gp**
  - Se la procedura è non-foglia **$ra** **DEVE** essere salvato
  - Se **$fp** viene alterato nel corso della procedura esso deve essere salvato (sempre se si passano valori tramite lo stack)
  - Spazio per le variabili locali dinamiche **DEVE** essere creato sullo stack
- Se si altera il contenuto di **$sp** questo andrebbe fatto in una singola operazione
- Se uno o più argomenti vengono passati attraverso lo stack **$fp** sovrebbe essere impostato come segue:
  - **$fp = $sp + (frame size)**
  - **Motivo:** questo imposta **$fp** in modo che punti al PRIMO argomento posto sullo stack

La procedura può accedere ai primi 4 argomenti di tipo intero accedendo ai registri **$a0 - $a3** e ai primi due argomenti di tipo float accedendo ai registri **$f12 - $f14**.

Argomenti e valori di ritorno

E' possibile accedere ad argomenti passati usando lo stack utilizzando **$fp** .
La procedura dovrebbe porre i valori di ritorno (se ce ne sono) in **$v0** e **$v1** .

Prima di ritornare : ripristino dei registri salvati

Inclusi **$s0 - $s7 , $fp , $ra , $gp** **se li abbiamo messi (push) sullo stack**

Ritornare alla procedura chiamante mediante l'istruzione: **jr $ra**

Riepilogo convenzioni chiamata

- Le chiamate a procedura avvengono secondo questa convenzione:

A cura del chiamante

1. **Salvataggio dei registri caller-saved**: `$t0 … $t9, $a0 … $a3` e **passaggio parametri**: il chiamante alloca i parametri di input in registri a cui la procedura accederà per leggerli.
2. **Deviazione del flusso di controllo** al blocco di istruzioni della procedura (chiamata a procedura).

A cura del chiamato

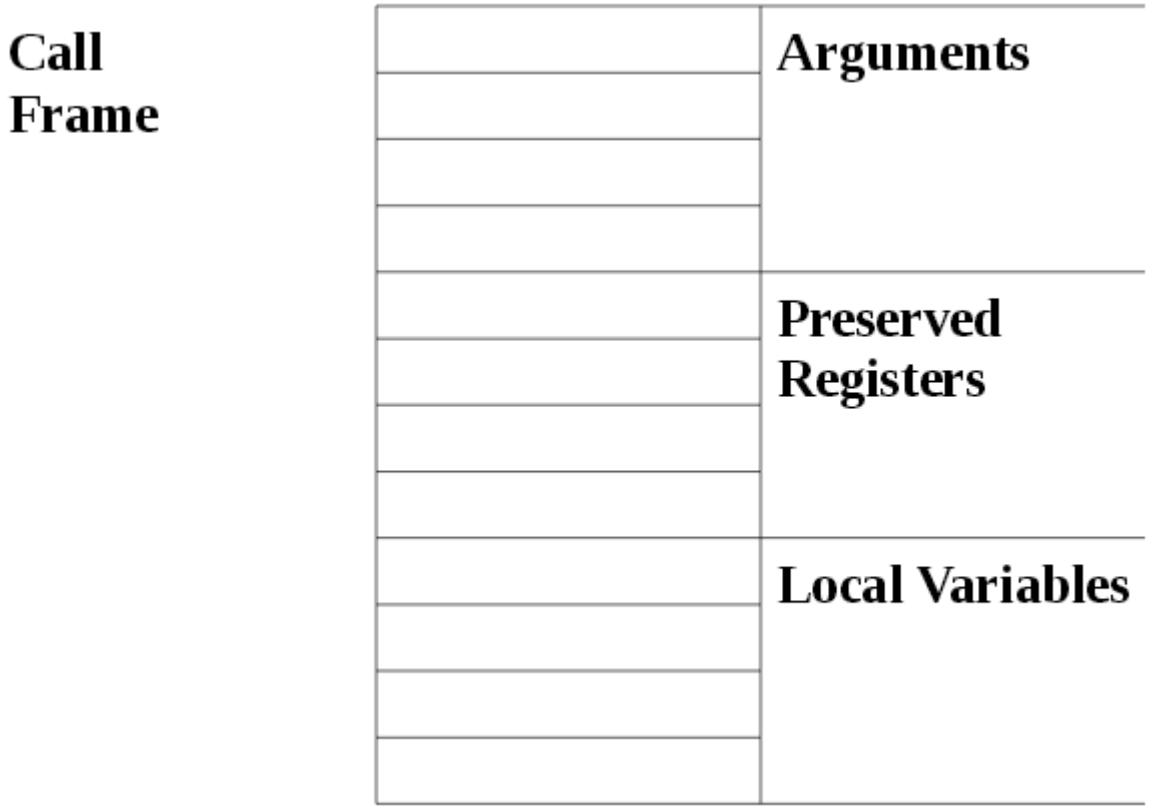
3. **Allocazione dello stack frame del chiamato e salvataggio dei registri callee-saved**: `$s0 … $s7, $fp, $ra`.
4. **Esecuzione della procedura**.
5. **Ritorno dei valori di output (se presenti), ripristino dei registri callee-saved e restituzione del controllo al chiamante** (che riprenderà la sua esecuzione dall'istruzione successiva alla chiamata a procedura).

Nota: «chiamante» e «chiamato» scaturiscono da una distinzione concettuale di due segmenti di codice assembly funzionalmente indipendenti (potremmo anche chiamarli «moduli»). Un chiamante e un chiamato non fanno/richiedono assunzioni sulla loro interazione che vadano oltre la convenzione indicata sopra (punti 1-5).

Call frame

Il call frame della procedura (detto anche activation record) è il nome con cui ci si riferisce alle informazioni poste sullo stack. Il call frame della procedura include parametri passati tra procedure e registri salvati. Inoltre, se esistono variabili **locali** dinamiche, lo spazio per esse è allocato sullo stack.

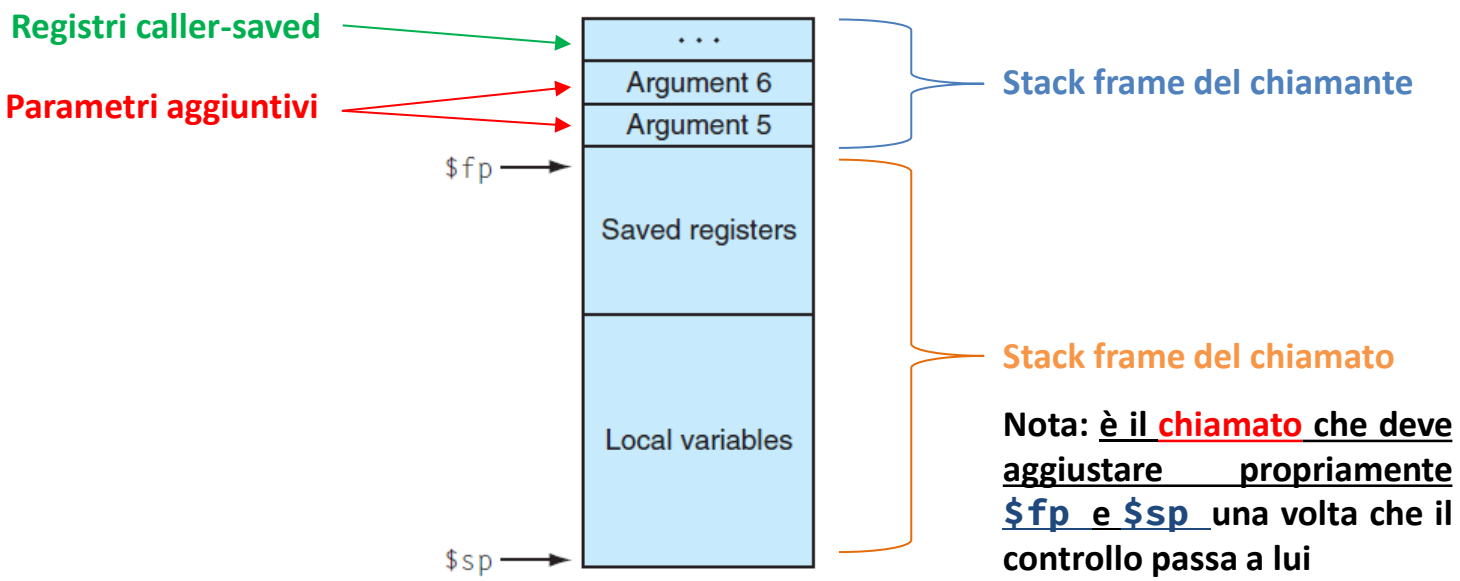
Ogni parte del call frame può avere dimensioni diverse a seconda del numero e della dimensione degli argomenti passati, degli eventuali registri salvati e del numero e dimensione delle variabili locali.



Step fondamentali di chiamata a procedura/funzione

Chiamante: Passaggio dei parametri (step 1)

- Dopo aver salvato sullo stack i registri caller-saved, il chiamante alloca i valori da passare nei registri `$a0, $a1, $a2, $a3`
- Nel caso si debbano passare più di 4 parametri? Per il quinto parametro e i successivi bisogna usare lo stack, **come?**
- Ci sono diverse convenzioni, ad esempio MIPS specifica un meccanismo abbastanza efficiente ma intricato, mentre *gcc* ne utilizza uno più semplice. Noi ci ispiriamo a quello di *gcc*.



Chiamante: Deviazione del flusso di controllo (step 2)

- Utilizzare l'istruzione **Jump and Link** per passare alla procedura (jump) e salvare l'indirizzo di ritorno (link)

- Salto incondizionato all'istruzione che sta alla label `procedura`, salva l'indirizzo della prossima istruzione (PC+4) in `$ra`

```
jal procedura
```

Chiamato: Stack frame e registri callee-saved (3)

- Creazione dello **stack frame**: sottrarre allo stack pointer corrente la dimensione del frame (che dipende da ciò che la procedura necessita).

```
$sp = $sp – DIM_FRAME
```

- Salvare sullo stack i registri callee-saved (se necessario):
  - `$s0 … $s7` se il chiamato necessita di sovrascriverli (il chiamante si aspetta che restino inalterati);
  - `$fp` se il chiamato crea uno stack frame (sempre);
  - `$ra` se il chiamato chiamerà a sua volta un'altra procedura.

- Settaggio del **frame pointer**: sommare allo stack pointer la dimensione del frame – 4 (ricordiamo che $fp punta alla prima parola del frame).

```
$fp = $sp + DIM_FRAME - 4
```

A questo punto la procedura può eseguire le proprie istruzioni (step 4)

Chiamato: Nota sullo Stack Frame (record di attivazione)

- L'ultimo frame allocato sullo stack è associato alla procedura **P** correntemente in fase di esecuzione.

- Fintanto che **P** non ha terminato, `$fp` contiene l'indirizzo della prima parola allocata nel frame di **P**

- Supponiamo che P chiami un'altra procedura **Q** (P è di tipo non-foglia) : **Q** **dovrà** salvare sullo stack il registro `$fp` prima di rilocarlo sul proprio frame e, prima di restituire il controllo a **P**, ripristinarlo. Questo per garantire a **P** la non alterazione del suo frame pointer (è lo stesso principio che vale per ogni calle-saved register).

- A cosa serve il frame pointer? Può essere comodo per indirizzare sullo stack variabili locali della procedura usandolo come base address del frame (la stessa cosa **non vale** per lo stack pointer che varia ogni volta che viene fatta una POP o un PUSH).

Chiamato: operazioni di ritorno (step 5)

- Se il chiamato è una *funzione* avrà un valore di ritorno da restituire al chiamante: viene lasciato nel registro `$v0` (e `$v1`)

- Ripristino dei registri callee-saved che erano stati salvati precedentemente (assegnando il loro valore iniziale presente sullo stack) (**nota: incluso $fp!**)

- Deallocazione (pop) dello stack frame: sommare la dimensione del frame a $sp

```
$sp = $sp – DIM_FRAME
```

- Ritorno del controllo al chiamante con:

```
jr $ra
```

Esempio 1 : calcolo potenza

```
# data declarations
.data

x:          .word     3
y:          .word     5
answer:     .word     0

# main routine
.text                                          CHIAMANTE
.globl      main
.ent        main
main:
            lw        $a0, x
            lw        $a1, y
            jal       power
            sw        $v0, answer

            li        $v0, 10          # syscall exit
            syscall                    # terminate program
```

Esempio 1 : calcolo potenza

```
# power function

.globl      power
.ent        power
power:

            li          $v0, 1
            li          $t0, 0

powLoop:
            mul         $v0, $v0, $a0
            add         $t0, $t0, 1
            blt         $t0, $a1, powLoop

            jr          $ra
.end        power
```

CHIAMATO
 # funzione per il calcolo di x^y
 # ritorna risultato in $v0
 # argomenti: x ($a0), y ($a1)

 # pseudoistruzione branch if less than

Esempio 2 : calcolo sommatoria

```
# Summation of 6 arguments:
# data declarations
.data

num1:    .word    3
num2:    .word    5
num3:    .word    3
num4:    .word    5
num5:    .word    3
num6:    .word    5
sum:     .word    0

.test
.globl   main
.ent     main
main:
```

(… continua)

CHIAMANTE

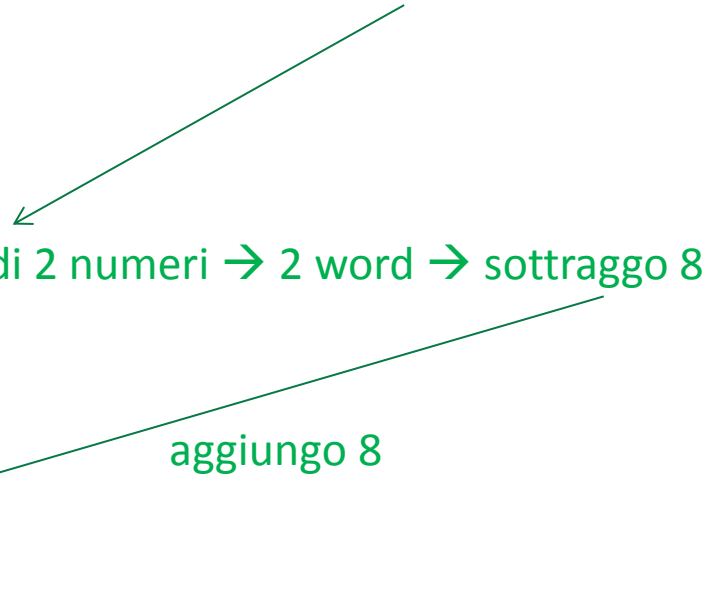
```
# funzione per la sommatoria
# di sei numeri. Primi 4 argo-
# menti passati mediante $a0 .. $a3
# gli ultimi due sullo stack
```

Esempio 2 : calcolo sommatoria

```
…
main:
```

```
        lw      $a0, num1
        lw      $a1, num2
        lw      $a2, num3
        lw      $a3, num4
        lw      $t0, num5
        lw      $t1, num6
        subu    $sp, $sp, 8
        sw      $t0, 0($sp)
        sw      $t1, 4($sp)
        jal     addem
        sw      $v0, sum

        addu    $sp, $sp, 8     # pulizia stack

        li      $v0, 10
        syscall                 # terminazione
.end main
```

CHIAMANTE
 # funzione per la sommatoria
 # di sei numeri. Primi 4 argo-
 # menti passati mediante $a0 .. $a3
 # gli ultimi due sullo stack



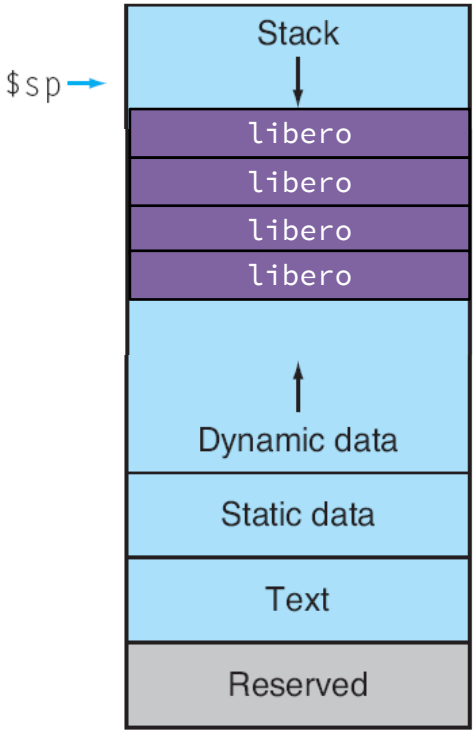
Esempio 2 : calcolo sommatoria

```
# Summation of 6 arguments:
.globl      addem
.ent        addem
addem:

            subu        $sp, $sp, 4     #preserva registri
            sw          $fp, ($sp)
            addu        $fp, $sp, 4     # imposta frame pointer

            li          $v0, 0
            add         $v0, $v0, $a0           # num1
            add         $v0, $v0, $a1           # num2
            add         $v0, $v0, $a2           # num 3
            add         $v0, $v0, $a3           # num 4
            lw          $t0, ($fp)              # num 5  (stack)
            add         $v0, $v0, $t0
            lw          $t0, 4($fp)             # num 6 (stack)
            add         $v0, $v0, $t0
# restore registers
            lw          $fp,  ($sp)
            addu        $sp, $sp, 4
            jr          $ra
.end addem
```

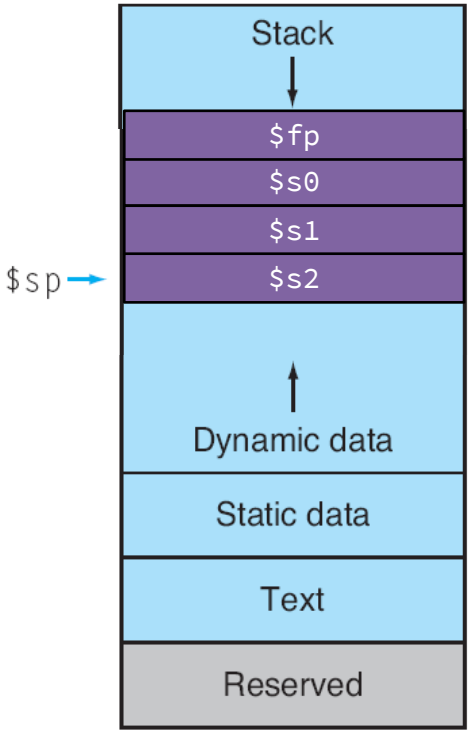
```
CHIAMATO
# funzione per la sommatoria
# di sei numeri. Primi 4 argo-
# menti passati mediante $a0 .. $a3
# num5 in ($fp), num6 in 4($fp)
# ritorna: $v0 = num1+num2+num3+
# +num4+num5+num6
```

Esempio

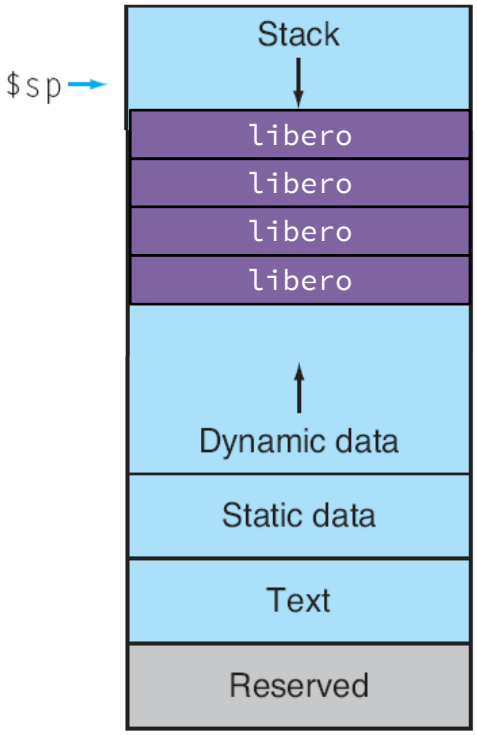
Prima della chiamata



Dopo la chiamata, durante l'esecuzione della procedura



Dopo il ritorno al chiamante



Esercizio 6.1

- Si scriva una procedura assembly, chiamata **Elaboratore**, che esegue la somma, differenza, moltiplicazione e divisione tra due numeri interi.

- Input: i due operandi e un terzo parametro per la selezione dell'operazione.

- Output: risultato (nel caso della divisione restituisce anche il resto).

- Si scriva poi il main dove:
  - vengono chiesti all'utente operandi e operatore;
  - il risultato dell'operazione è mostrato a terminale.

Esercizio 6.2

- Si scriva un programma che
  - chieda all'utente di inserire un array di interi di dimensione arbitraria.
  - invochi una procedura P
  - stampi il valore ritornato da P

- La procedura P è definita come segue:
  - Input: l'array inserito dall'utente e un parametro k
  - se k=0 la procedura calcola la somma di tutti gli interi in posizione (indice nell'array) dispari
  - se k=1 sommerà quelli in posizioni pari.

- *Suggerimento: allocare l'array staticamente in memoria e passare alla procedura il base address (passaggio per indirizzo).*

Esercizio 6.3

- Si scriva un programma che chieda all'utente di inserire due array di interi A1 e A2 di dimensione arbitraria e stampi a video un messaggio se ogni elemento di A1 è presente in A2

- Si utilizzi una procedura P così definita
  - Input: un array A e un intero i
  - Output: 1 se i è contenuto in A, 0 altrimenti