Architettura Elaboratori Elettronici

ESERCITAZIONI

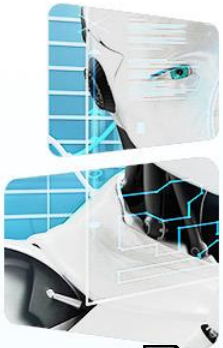
FUNZIONI

Franco Liberati

liberati@di.uniroma1.it

Argomenti

- Funzioni
- Annidamento di funzione statico: Activation Frame
- Annidamento di funzione dinamico

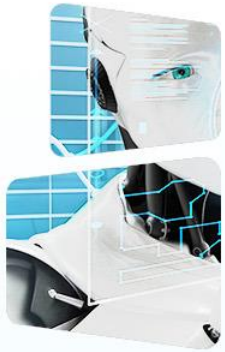


Funzioni

Generalità

- Una **funzione** (o sub routine) è una sequenza di istruzioni che esegue un compito definito e che viene usata *come se* fosse una unità di codice
  - È identificata da un nome, elabora degli argomenti e restituisce un risultato
  - È una astrazione presente in (quasi) tutti i linguaggi ad alto livello
  - Il suo uso garantisce una maggiore leggibilità del codice
  - L'impiego di sub routine permette il riuso del codice
  - L'intero programma assembly è visto come un insieme di sub routine in cui c'è una routine principale detta MAIN

L'assembly fornisce solo le istruzioni e i registri che consentono di realizzare delle funzioni
Una funzione in assembler è più un insieme di convenzioni che una struttura sintattica predefinita



Funzioni

Esempio Linguaggio alto livello

```
Read(a);
Read(b);
Read (c);
media=calcola_media(a,b,c);
mediano=calcola_mediano(a,b,c);
massimo=calcola_massimo(a,b,c);
Print(media);
Print(mediano);
Print(massimo);
```

```
function calcola_media (x,y,z)
{
int m;
m=(x+y+z)/3;
return m
}

function calcola_mediano (x,y,z)
{
for(i=0; i<MAXINT;i++){t[i]=0;}
t[x]=1;t[y]=1;t[z]=1;
Int mediano=0; int m=0;
for(i=0; i<MAXINT;i++)
        {
                if (t[i]==1) mediano++;
                if (mediano==2) m=i;
        }
return m
}

function calcola_massimo(x,y,z)
{
Int m;
If (x>=y)&&(x>=z){m=x;}
If (y>=x)&&(y>=z){m=y;}
If (z>=x)&&(z>=y){m=z;}
return m;
}
```

Funzioni MARS

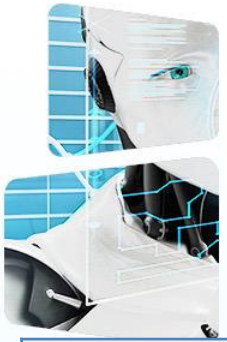
Chiamata di una subroutine

- In MIPS/MARS per eseguire (invocare o chiamare) una sub routine si usa l'istruzione **jal** (*jump and link*)

jal *function-name*

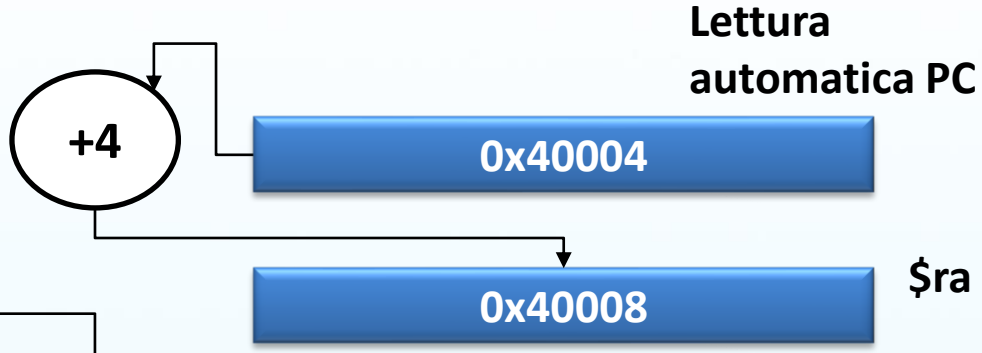
- Il MIPS memorizza l'indirizzo dell'istruzione che *segue* **jal** nel registro ***$ra*** e salta all'etichetta ***function-name***

Il valore di $ra è copiato al termine della fase di fetch dell'istruzione JAL



Funzioni MARS

PROGRAMMA	
0x40000	…
0x40004	**JAL PIPPO**
0x40008	…
0x4000c	…
0x40010	…



PIPPO	
0x42000	
0x42004	
0x42008	
0x4200c	
0x42010	**JR $RA**

Funzioni MARS

Ritorno da una subroutine

- Per tornare da una funzione si usa l'istruzione **jr** (*jump to register*)

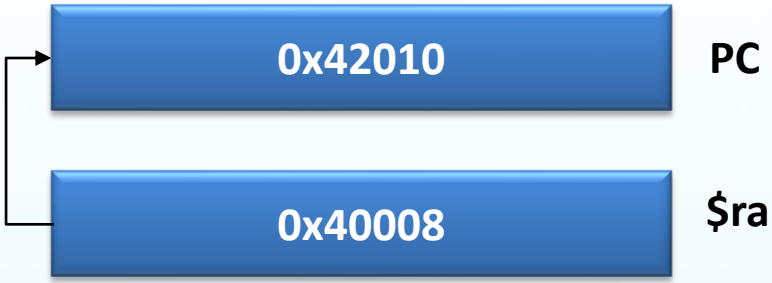
jr *$ra*

- Salta all'indirizzo contenuto nel registro ***$ra***

*L'istruzione **jr** forza il program counter al valore contenuto nel registro che segue (che può anche non essere $ra, ma bensì un qualsiasi registro che contenga l'indirizzo del valore di ritorno)*

Funzioni MARS

PROGRAMMA	
0x40000	…
0x40004	**JAL PIPPO**
0x40008	…
0x4000c	…
0x40010	…



PIPPO	
0x42000	
0x42004	
0x42008	
0x4200c	
0x42010	**JR $RA**

Funzioni MARS

Argomenti funzione - parametri

- Gli **argomenti** di una sub routine sono convenzionalmente immessi nei registri preservanti **$a0, $a1, $a2, $a3**

- Il **risultato** è riportato in un registro preservante: per convenzione si impiega **$v0** (se i risultati sono due o c'è la necessita di scambiare il risultato di una funzione annidata una volta si usa anche **$v1**)

- La scelta dei registri preservanti è motivata dal fatto che l'istruzione **JAL** (*jump and link*) azzera i registri temporanei

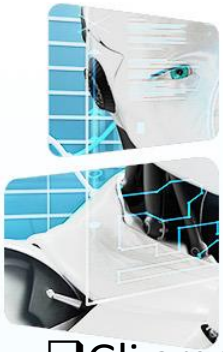
Funzioni MARS

Esempio

Calcolo del massimo di due numeri con una funzione

```
            .text
            .globl main
main:
            lw $a0,x
            lw $a1,y
            jal massimo_fra_due
            move $a0,$v0
            li $v0,1
            syscall
            li $v0,10
            syscall
```

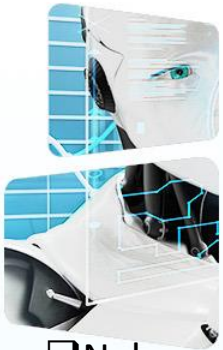
```
massimo_fra_due:
            move $t0, $a0
            move $t1, $a1
            move $v0,$t0
            blt $t1,$t0, fine
            move $v0,$t1
fine:
            jr $ra
```



Funzioni MARS

Passaggio parametri

- Gli argomenti, nei linguaggi ad alto livello, possono essere passati per **valore** o per **riferimento**
  - Nel **passaggio per valore** la routine elabora i valori contenuti nelle variabili passate come argomenti
    - Quando il passaggio dei parametri avviene per valore, alla funzione è in effetti passata solo una copia dell'argomento. Grazie a questo meccanismo il valore della variabile nel programma chiamante non è modificato.
  - Nel **passaggio per riferimento** alla routine si passa l'indirizzo in memoria dove risiedono le variabili (in questo caso le modifiche apportate all'interno delle routine sulle variabili cambiano i valori al termine della chiamata di funzione)
    - In passaggio di parametri per riferimento (o reference), alla funzione è passato l'indirizzo e non il valore dell'argomento. Questo approccio richiede meno memoria rispetto alla chiamata per valore, e soprattutto consente di modificare il valore delle variabili che sono ad un livello di visibilità (*scope*) esterno alla funzione o al metodo



Funzioni MARS

Passaggio per valore

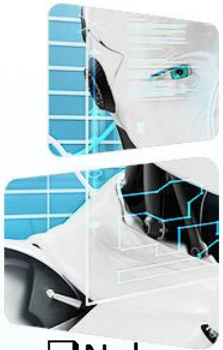
- Nel **passaggio per valore** la routine elabora i valori contenuti nelle variabili passate come argomenti

```
int main(){
 int pippo=2; int pluto=3;
int ris;
ris=scambiaesomma(pippo,pluto);
cout<<"Il valore di pippo ("<<pippo<<")
dopo lo scambio è"<<pippo << "e la
somma è"<<ris;
}
```

```
int scambiaesomma(int pippo,int pluto){
        int x;
        x=pippo;
        pippo=pluto;
        pluto=x;
        return (pippo+pluto;)
}
```

OUTPUT

Il valore di pippo (2) dopo lo scambio è 2 e la somma è 5



Funzioni MARS

Passaggio per valore in MARS

- Nel **passaggio per valore** la routine elabora i valori contenuti nelle variabili passate come argomenti

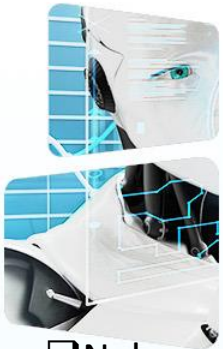
```
lw $a0,pippo
lw $a1, pluto
jal scambia_e_somma
move $a0,$v0
li $v0,1
syscall
```

```
scambia_e_somma:
                    move $t1,$a0
                    move $t0,$a1
                    add$v0,$t0,$t1
                    jr $ra
```

```
.data
pippo: .word 2
pluto: .word 5
```

OUTPUT

Il valore di pippo (2) dopo lo scambio è 2 e la somma è 5



Funzioni MARS

Passaggio per riferimento

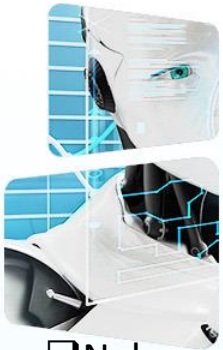
- Nel **passaggio per valore** la routine elabora i valori contenuti nelle variabili passate come argomenti

```
int main(){
 int pippo=2; int pluto=3;
int ris;
ris=scambiaesomma(&pippo,&pluto);
cout<<"Il valore di pippo ("<<pippo<<")
dopo lo scambio è"<<pippo << "e la
somma è"<<ris;
}
```

```
int scambiaesomma(int *pippo,int *pluto){
        int x;
        x=pippo;
        pippo=pluto;
        pluto=x;
        return (pippo+pluto;)
}
```

OUTPUT

Il valore di pippo (2) dopo lo scambio è 3 e la somma è 5



Funzioni MARS

Passaggio per riferimento

- Nel **passaggio per valore** la routine elabora i valori contenuti nelle variabili passate come argomenti

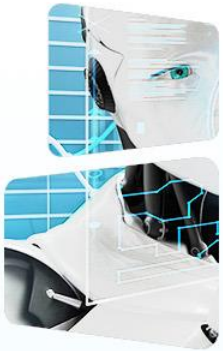
```
la $a0,pippo
la $a1, pluto
#passaggio parametro  per indirizzo
jal scambia_e_somma
move $a0,$v0
li $v0,1
syscall
```

```
scambia_e_somma:
        lw $t0,($a0)
        lw $t1,($a1)
        move $t2,$t0
        move $t0,$t1
        move $t1,$t2
        sw $t0,($a0)
        sw $t1,($a1)
        #salvataggio valori
        add $v0,$t0,$t1
        jr $ra
```

```
.data
pippo: .word 2
pluto: .word 5
```

OUTPUT

Il valore di pippo (2) dopo lo scambio è 3 e la somma è 5



Funzioni MARS

Annidamento funzione problema

- Un **annidamento di routine di profondità nota (o annidamento statico)** si verifica, almeno, quando all'interno di una sub routine chiamata dal programma principale è richiesta una ulteriore chiamata a sub routine (cioè con profondità 1)

Funzioni MARS

Annidamento funzione problema

- L'annidamento di funzione si può risolvere -quando è nota la profondità - utilizzando dei registri supplementari nei quali memorizzare i valori di $ra. Ma questa è una soluzione limitante (perché non si possono sfruttare più di 9 registri preservanti: $s0,…,$s9)

MAIN:

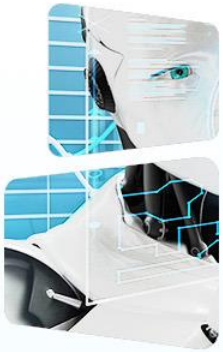
```
lw $a0,pippo
lw $a1, pluto
jal F1
move $a0,$v0
li $v0,1
syscall
```

F1:

```
move $s0,$ra
lw $t0,$a0
lw $t1, $a1
div $a0,$t1,$t0
rem $a1,$t1,$t0
jal F2
move $ra,$s0
jr $ra
```

F2:

```
lw $t0,$a0
lw $t1, $a1
mul $v0,$t0,$t1
jr $ra
```



Funzioni MARS

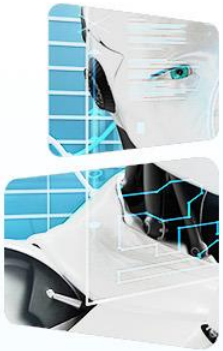
Limiti e soluzione

- Come si gestisce un numero di argomenti immessi o in uscita dalla sub routine se sono molti (es.: più di 4) nel caso del **passaggio per parametri**?

- Come si gestisce un **annidamento di routine di profondità nota (o annidamento statico)**, che si verifica, almeno, quando all'interno di una sub routine chiamata dal programma principale è richiesta una ulteriore chiamata a sub routine (cioè con profondità 1)

ACTIVATION FRAME

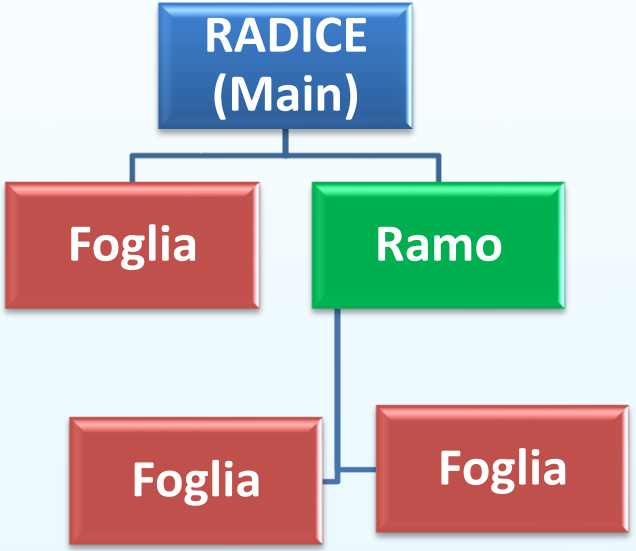
Activation Frame

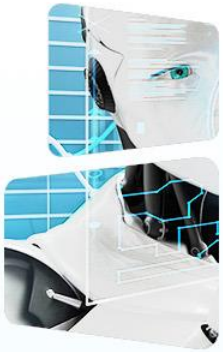


Funzioni MARS

Nomenclatura

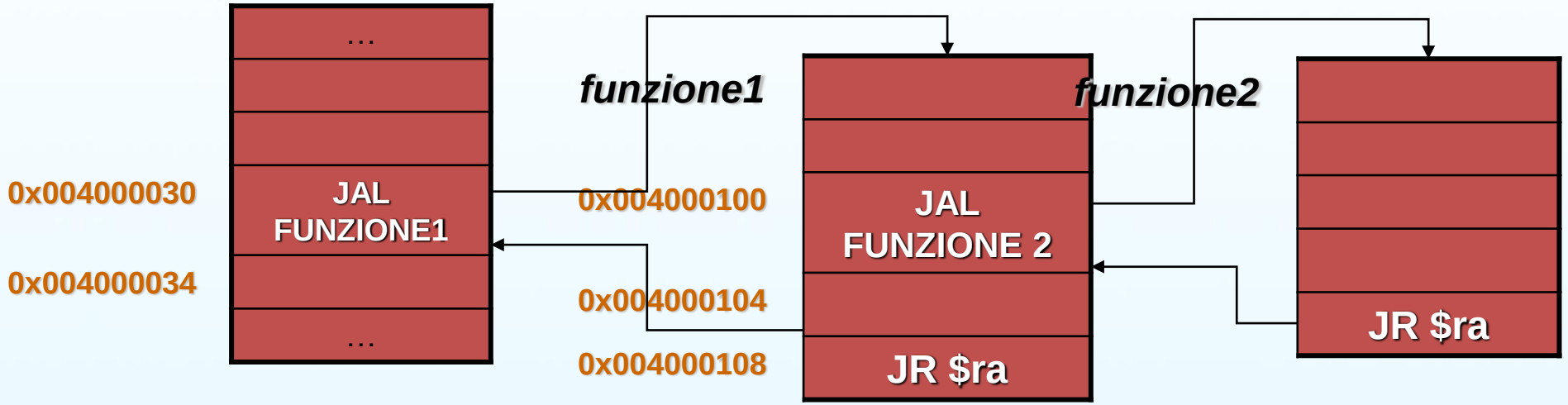
- Una funzione può essere classificata:
  - ***Foglia:*** Se non chiama alcuna altra funzione al suo interno
  - ***Ramo:*** Se è sia chiamante che chiamata
- La funzione principale, quella primordiale (**main**) è detta ***radice***
- Un annidamento di funzioni prevede che ci sia almeno una sub routine ramo



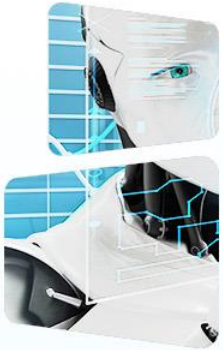


Funzioni MARS

Nomenclatura



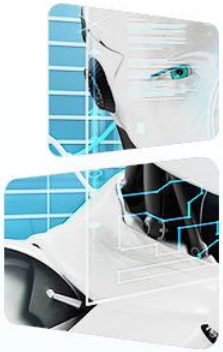
1. Il solo registro $ra non è sufficiente se la funzione chiamata è a sua volta una funzione chiamante (*vedi figura*)
2. Si deve gestire in maniera adeguata il passaggio degli argomenti tra subroutine
3. Si devono monitorare le variabili temporanee necessario al calcolo



Activation frame MARS

Generalità

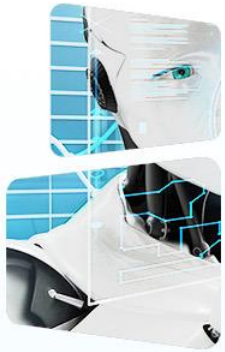
- La chiamata ad un sub-routine può richiedere uno spazio di memoria per il **salvataggio temporaneo degli argomenti** di ingresso e quelli di uscita nonché dell'**indirizzo di ritorno**
- Per questo motivo si ricorrere allo **spazio di memoria di attivazione** durante la chiamata di una sub-routine (**activation frame**)



Activation frame MARS

Utilità

- L’**activation frame** è utile per:
  - Salvare il valore dell’indirizzo di ritorno nel caso di annidamento
  - Memorizzare i valori passati alla funzione come argomenti di ingresso
  - Salvare registri i cui valori possono essere modificati da una funzione, ma che la routine chiamante si aspetta che siano preservati
  - Fornire spazio nel caso debba usare i registri per elaborare molti dati



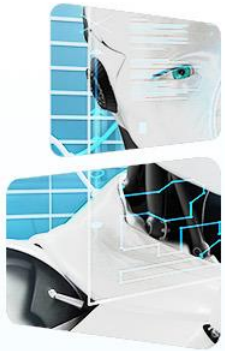
Activation frame MARS

Esempio

```
Main ()
{
int x,y,z;
x=255;
y=65000;
z= massimo_fra_quadrati(x,y);
stampa (z)
}
```

```
massimo_fra_quadrati(int a, int b)
{
int x,y,c;
 x=a*a;
 y=b*b;
 c=massimo(x,y);
 return c;
}
```

```
massimo(int c, int d)
{
int m;
m=c;
If (d>c){m=d;}
return m;
}
```



Activation frame MARS

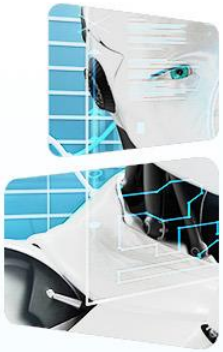
Esempio

```
lw $t0,valore1
lw $t1,valore2
move $a0,$t0,
move $a1,$t1
jal MAX_QUAD
sw $v0,z
```

```
.data
valore1: .word 2312
valore2:.word 1105
MainActFrame: .space 4 # spazio per $ra
```

```
MAX_QUAD:
        sw $ra, MainActFrame
        mul$t0,$a0,$a0
        mul$t1,$a1,$a1
        move $a0,$t0
        move $a1,$t1
        jal MAX
        lw $ra,MainActFrame
        move $v0,$v1
        jr $ra
```

```
MAX:
        move $t0,$a0
        move $t1,$a1
        move $t2,$t0
        blt $t1,$t0, fine
        move $t2,$t1
fine:
        move $v1,$t2
        Jr $ra
```



Activation frame MARS

Impiego

- L’activation frame per gestire il ritorno da subroutine **deve essere definito** per la radice e ogni funzione ramo mentre deve essere usato in ogni subroutine ramo (escluso la radice); ricordandosi che l’uso avviene nella subroutine successiva

Activation frame MARS

Esempio di impiego

```
Main ()
{
Read(x);
Read(y);
 z= Batman(x,y);
Print (z)
}
```

```
Batman (int a, int b)
{
 a=a+1; b=b+2;
 c=Robin(a,b);
 return c;
}
```

```
Robin (int c, int d)
{
f=c*2;g=d*5;
z=Joker (f,c);
return z
}
```

```
Joker (int c, int d)
{
return c*d;
}
```

Activation frame MARS

Esempio di impiego

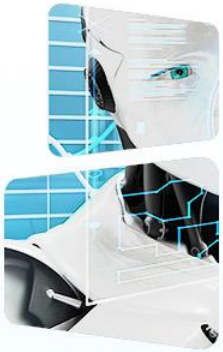
```
lw $t0,x
lw $t1,y
move $a0,$t0
move $a1,$t1
jal Batman
sw $v0,z
```

```
Batman:
        sw $ra, MainActFrame
        add $a0,$a0,1
        add $a1,$a1,2
        jal Robin
        move $v0,$v1
        lw $ra, MainActFrame
        jr $ra
```

```
Robin:
        sw $ra, BatmanFrame
        mul $a0,$a0,2
        mul $a1,$a1,5
        jal Joker
        move $v0,$v1
        lw $ra, BatmanFrame
        jr $ra
```

```
.data
MainActFrame: .space 4 # spazio per $ra
BatmanFrame: .space 4 # spazio per $ra
```

```
Joker:
    mul $v1,$a0,$a1
    jr $ra
```



Activation frame MARS

Impiego

- L’**activation frame per il trasferimento dei dati** deve essere definito nella subroutine ramo e utilizzata in quella foglia

Activation frame MARS

Esempio di impiego con più 4 argomenti

```
Main ()
{
 leggi x;
 leggi y;
 z= superman(x,y);
}
```

```
Superman (int a, int b)
{
 a=a+1;
b=b+2;
c=a*3
d=b*4
 c=Supergirl(a,b,c,d);
 return c;
}
```

```
Supergirl (int a, int b int c, int d)
{
e=c*2;f=d*5;
z=Luthor (a,b,c,d,e,f);
return z
}
```

```
Luthor(int a, int b,int c, int d,int e, int f)
{
 int temp8= a*b*c;
int  temp9=d*e*f;
return temp8/temp9;
}
```

Activation frame MARS

Esempio di impiego con più 4 argomenti

```
lw $t0,x
lw $t1,y
move $a0,$t0,
move $a1,$t1
jal Superman
sw $v0,z
```

```
.data
MainActFrame: .space 4 # spazio per $ra
SupermanFrame: .space 4 # spazio per $ra
LuthorFrame: . space 8
```

```
Superman:
        sw $ra, MainActFrame
        add $a0,$a0,1
        add $a1,$a1,2
        mul $a2,$a0,3
        mul $a3,$a1,4
        jal Supergirl
        move $v0,$v0
        lw $ra, MainActFrame
        jr $ra
```

```
Supergirl:
        sw $ra, SupermanFrame
        mul $t0,$a2,2
        mul $t1,$a3,5
        sw $t0, LuthorFrame
        sw $t1, LuthorFrame+4
        jal Luthor
        move $v0,$v1
        lw $ra, SupermanFrame
        jr $ra
```

```
Luthor:
        lw $t0, LuthorFrame
        lw $t1, LuthorFrame+4
        mul $a0,$a0,$a1
        mul $t9,$a0,$a2
        mul $t0,$a3,$t0
        mul $t8,$t1,$t0
        div $v1,$t9,$t8
        jr $ra
```

Activation frame MARS

Esempio di impiego con più 4 argomenti (variante)

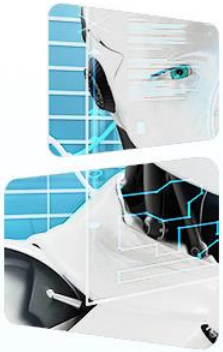
```
lw $t0,x
lw $t1,y
move $a0,$t0,
move $a1,$t1
jal Superman
sw $v0,z
```

```
.data
MainActFrame: .space 4 # spazio per $ra
SupermanFrame: .space 4 # spazio per $ra
LuthorFrame: . space 8
```

```
Superman:
        sw $ra, MainActFrame
        add $a0,$a0,1
        add $a1,$a1,2
        mul $a2,$a0,3
        mul $a3,$a1,4
        jal Supergirl
        move $v0,$v0
        lw $ra, MainActFrame
        jr $ra
```

```
Supergirl:
        sw $ra, SupermanFrame
        mul $t0,$a2,2
        mul $t1,$a3,5
        la $s0, LuthorFrame
        sw $t0, 0($s0)
        sw $t1, 4($s0)
        jal Luthor
        move $v0,$v1
        lw $ra, SupermanFrame
        jr $ra
```

```
Luthor:
        la $s0, LuthorFrame
        lw $t0, ($s0)
        lw $t1, 4($s0)
        mul $a0,$a0,$a1
        mul $t9,$a0,$a2
        mul $t0,$a3,$t0
        mul $t8,$t1,$t0
        div $v1,$t9,$t8
        jr $ra
```



Activation frame MARS

limite

- Cosa accade quando c'è un **annidamento dinamico** (cioè ripetuto più volte in relazione ad un numero di volte variabile in accordo a dei parametri immessi in ingresso)?

Calcolo del fattoriale in modalità ricorsiva

$$f(n) = \begin{cases} n \cdot f(n-1) & se\ n > 0 \\ 1 & se\ n = 0 \end{cases}$$

Activation frame MARS

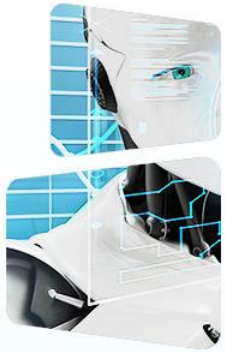
limite

Calcolo del fattoriale in modalità ricorsiva

$$f(n) = \begin{cases} n \cdot f(n-1) & \text{se } n > 0 \\ 1 & \text{se } n = 0 \end{cases}$$

```
#include <stdio.h>
int main(void)
        {
        int n;  printf("Inserire un intero > 0 : ");
        scanf("%d", &n);
        printf("Il fattoriale e' %d\n", fattoriale(n));
        return 0;
        }
```

```
int fattoriale(int n)
        {
        if (n == 0) return 1;
        else return n*fattoriale(n-1);
}
```



Activation frame MARS

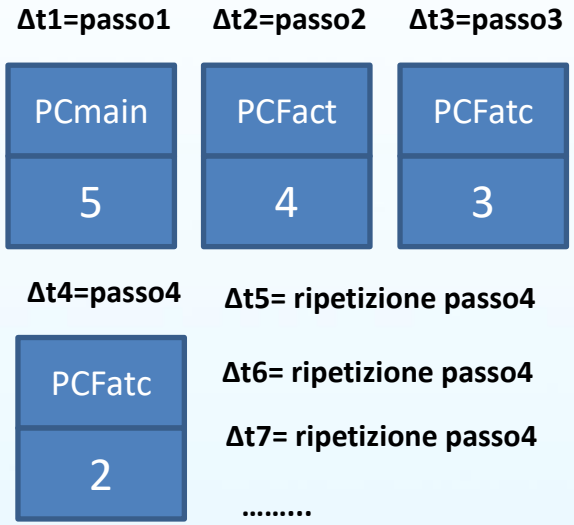
limite

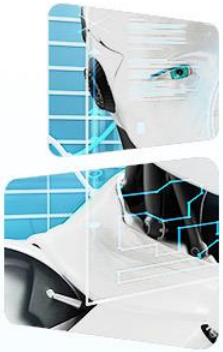
$$f(n) = \begin{cases} n \cdot f(n-1) & \text{se } n > 0 \\ 1 & \text{se } n = 0 \end{cases}$$

```
…
lw $t0,x
move $a0,$t0
jal fact
…

.data
    me: .space 8
```

```
fact:
        ble $a0,1,base
        la $t9, factFrame
        sw $ra,($t9)
        sw $a0,4($t9)
        subu $a0,$a0,1
        jal fact
        la $t9, factFrame
        lw $t0, 4($t9)
        mul $v0,$v0,$t0
        j end
base:
        li $v0,1
end:
        la $t9, factFrame
        lw $ra,($t9)
        jr $ra
```





Activation frame MARS

limite e soluzione

- Un singolo frame per funzione non è sufficiente nel caso di sub routine con annidamento dinamico (anche note come funzioni ricorsive)
- Da qui il nome: “frame di attivazione” perché è necessario un frame per ogni attivazione (chiamata) di funzione
- Come si può allocare del nuovo spazio in memoria ogni volta che chiamo una funzione?

FINE