SALTO A SUBROUTINE

Dott. Franco Liberati

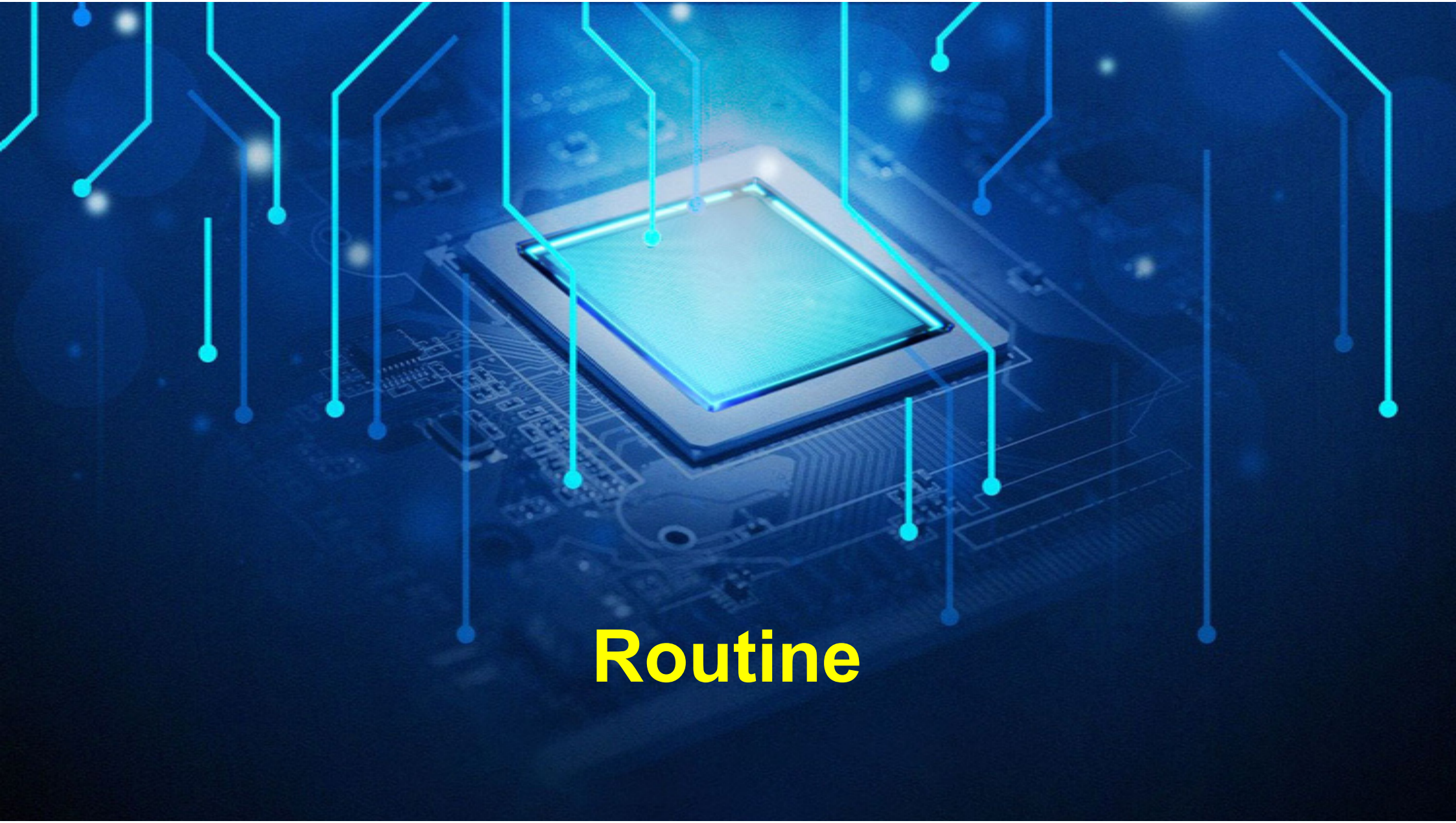
Argomenti

01

Routine
Passaggio degli argomenti (valore e riferimento)
Limiti del salto a subruotine

02

Activation Frame
Limite dell'Activation Frame


Routine

Routine

Generalità

Una **subroutine** (o funzione) 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
- Una funzione è una astrazione presente in (quasi) tutti i linguaggi ad alto livello
- Il suo uso garantisce una maggiore leggibilità del codice e una migliore gestione degli errori (debug)
- 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 soubroutine pertanto è più un insieme di convenzioni che una struttura sintattica predefinita

Routine

Generalità

```
BEGIN
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);
END
```

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

Routine in MIPS

Chiamata

Per eseguire (invocare o chiamare) una subroutine 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***

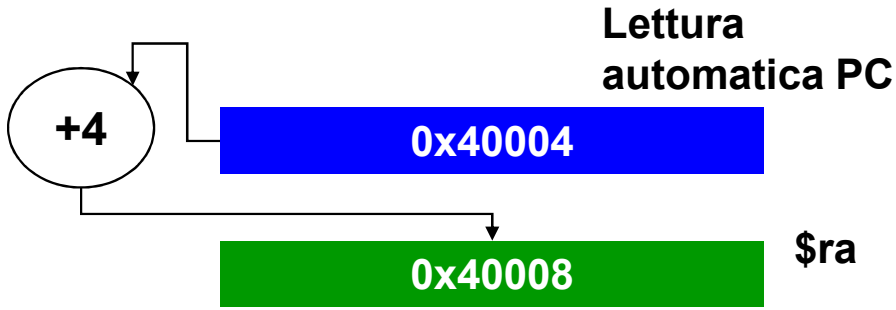
Il valore del program counter in $ra è copiato al termine della fase di fetch dell'istruzione JAL

Routine in MIPS

Chiamata

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

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



Routine in MIPS

Ritorno da chiamata

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

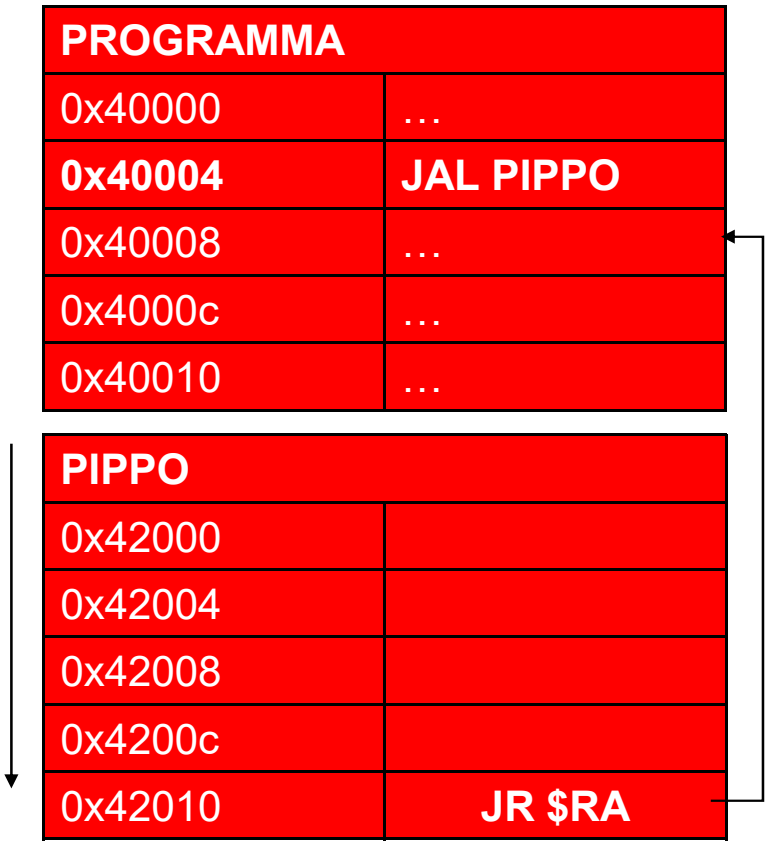
jr *$ra*

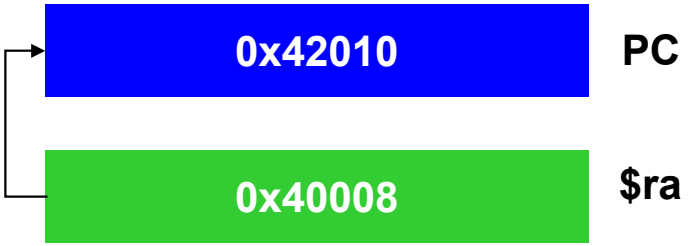
Salta all'indirizzo contenuto nel registro ***$ra***

*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)*

Routine in MIPS

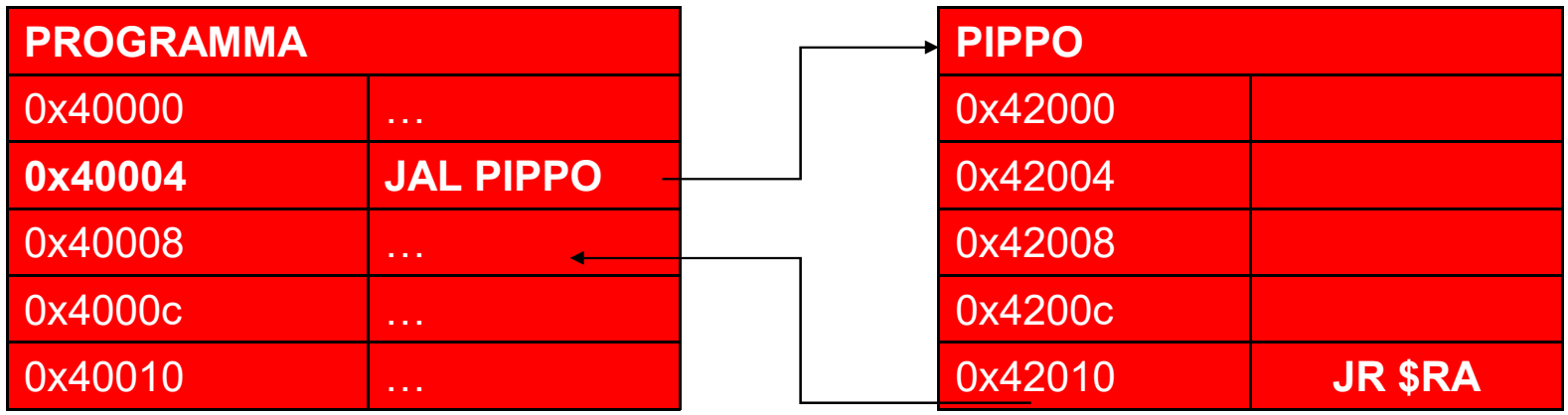
Ritorno da chiamata





Routine in MIPS

Uso subroutine



1. Sono eseguite le istruzioni della ***funzione chiamante*** **PROGRAMMA** che precedono la chiamata (**jal**)
2. **Chiamata**: l'istruzione **jal** salva l'indirizzo della prossima istruzione (da eseguire al ritorno) in **$ra** e salta all'inizio della ***funzione chiamata*** **PIPPO**
3. Sono eseguite tutte le istruzioni di **PIPPO** fino alla **jr** conclusiva
4. **Ritorno**: l'istruzione **jr** salta alla prima istruzione dopo l'istruzione di chiamata
5. Sono eseguite le rimanenti istruzioni della funzione chiamante

Routine in MIPS

Passaggio degli argomenti

Una routine di solito lavora su un certo numero di **argomenti** in ingresso e propone un **risultato**

```
main ()
        {
                int d,a,b;
                a=2;
                b=3;
                d=somma(a,b);
                print (d);
        }
```

```
int somma(int a,int b)
        {
                int risultato;
                risultato=a+b;
                return risultato;
        }
```

Routine in MIPS

Passaggio degli argomenti

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 **$v0** (se i risultati sono due, si usa anche **$v1**)

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

Routine in MIPS

Esempio (Massimo tra due numeri)

Realizzazione di una funzione che calcola il massimo tra due numeri
La funzione è: int **MAXDUE**(*int* **a**,*int* **b**) che prende sue argomenti in ingresso e restituisce un valore in uscita

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

```
MAXDUE:
        move $t0, $a0
        move $t1, $a1
        move $t2,$t0
        blt $t1,$t0, fine
        move $t2,$t1
fine:
        move $v0,$t2
        jr $ra
```

Routine in MIPS

Esempio del riuso di una funzione (Massimo tra tre numeri)

Realizzazione di una funzione che calcola il massimo tra tre numeri

Il programma applica due volte la funzione int **MAXDUE**(*int* **a**,*int* **b**)

MAXTRE(a,b,c)=MAX(a,MAXDUE(b,c))

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

```
MAXDUE:
        move $t0, $a0
        move $t1, $a1
        move $t2,$t0
        blt $t1,$t0, fine
        move $t2,$t1
fine:
        move $v0,$t2
        jr $ra
```

Routine in MIPS

Passaggio degli argomenti

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 viene 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)
  - Nel 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

Routine in MIPS

Passaggio per valore

Nel **passaggio per valore**, per i linguaggi ad alto livello, la routine elabora i valori contenuti nelle variabili passate come argomenti

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

```
int scambiaedividi(int pippo,int*pluto){
        int x;
        x=pippo;
        pippo=pluto;
        pluto=x;
        return (pippo/pluto;)
}
```

Soluzione: Dopo la funzione Pippo è 2, Pluto è 3 e il risultato è 1

Routine in MIPS

Passaggio per valore in MIPS

Nel **passaggio per valore**, per i linguaggi asemblativi, la routine elabora gli operandi contenuti nei registri e non ne modifica il valore in memoria

```
        .text
        .globl main
main:
        lw $a0,pippo
        lw $a1, pluto
        jal SCAMBIA_E_DIVIDI
        move $t0,$v0
        li $v0,10
        syscall
.data
pippo: .word 2
pluto: .word 5
```

```
SCAMBIA_E_DIVIDI:
        move $t1,$a0
        move $t0,$a1
        div $v0,$t0,$t1
        jr $ra
```

Routine in MIPS

Passaggio per riferimento

Nel **passaggio per riferimento**, per i linguaggi ad alto livello, la routine elabora i valori contenuti nelle variabili passate come argomenti

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

```
scambiaedividi (int *pippo,int *pluto){
        int* x;
        *x=*pippo;
        *pippo=*pluto;
        *pluto=*x;
        return (*pippo\*pluto);
}
```

Soluzione: Dopo la funzione Pippo diventa 3, Pluto diventa 2 e il risultato è 1

Routine in MIPS

Passaggio per riferimento in MIPS

Nel **passaggio per riferimento**, per i linguaggi asemblativi, la routine elabora gli operandi contenuti nelle loro locazioni di memoria

```
        .text
        .globl main
main:
        la $a0,pippo
        la $a1, pluto
        jal SCAMBIA_E_DIVIDI
        move $t0,$v0
        li $v0,10
        syscall
        .data
pippo: .word 2
pluto: .word 5
```

```
                .text
SCAMBIA_E_DIVIDI:
                lw $t0,($a0)
                lw $t1,($a1)
                move $t2,$t0
                move $t0,$t1
                move $t1,$t2
                #salvataggio valori
                sw $t0,($a0)
                sw $t1,($a1)
                div $v0,$t0,$t1
                jr $ra
```

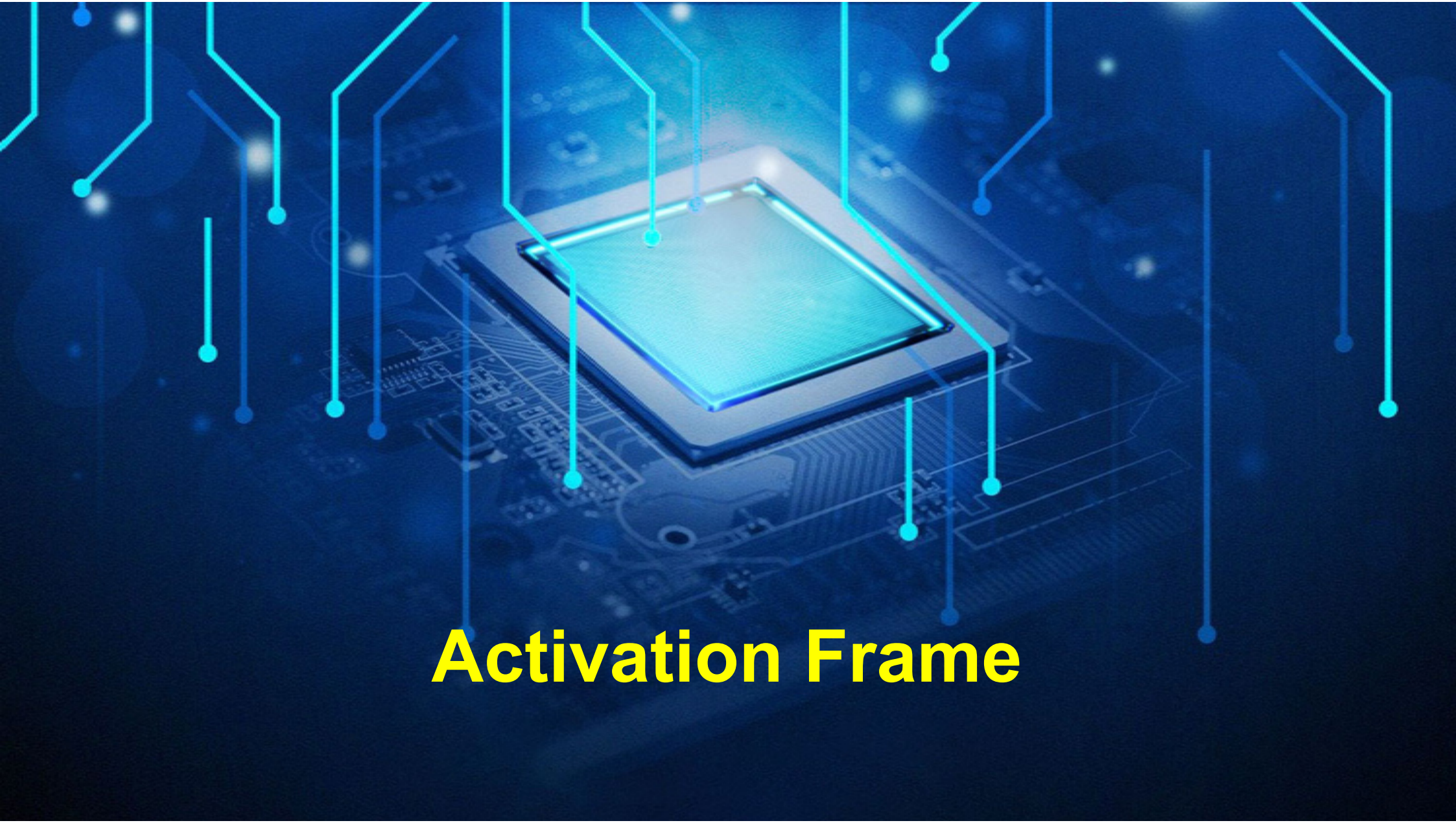
Routine in MIPS

Limiti

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

Come si gestisce l'**annidamento di routine**, che si verifica quando all'interno di una sub routine chiamata dal programma principale è richiesta, almeno, una ulteriore chiamata a sub routine

Activation Frame


Activation Frame

Activation Frame

Generalità

L'**activation frame** è un'area di memoria definita, riservata e gestita dal programmatore per:

- 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
- Salvare il valore dell'indirizzo di ritorno nel caso di **annidamento**

La definizione dell'activation frame avviene grazie alla direttiva **.space n** che alloca n locazioni di un byte di memoria

Esempio:

```
actFrm: .space 4 #riserva 4byte
```

Activation Frame

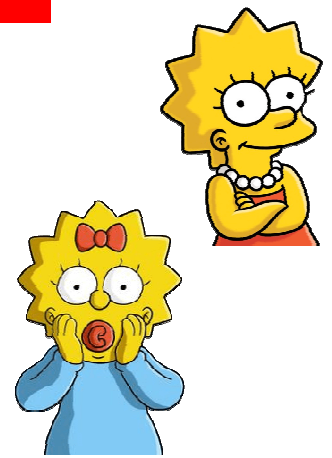
Approfondimento: problema del salvataggio dell'indirizzo di ritorno

100	MAIN
104	
108	
112	JAL F1
116	
120	
124	

116 416

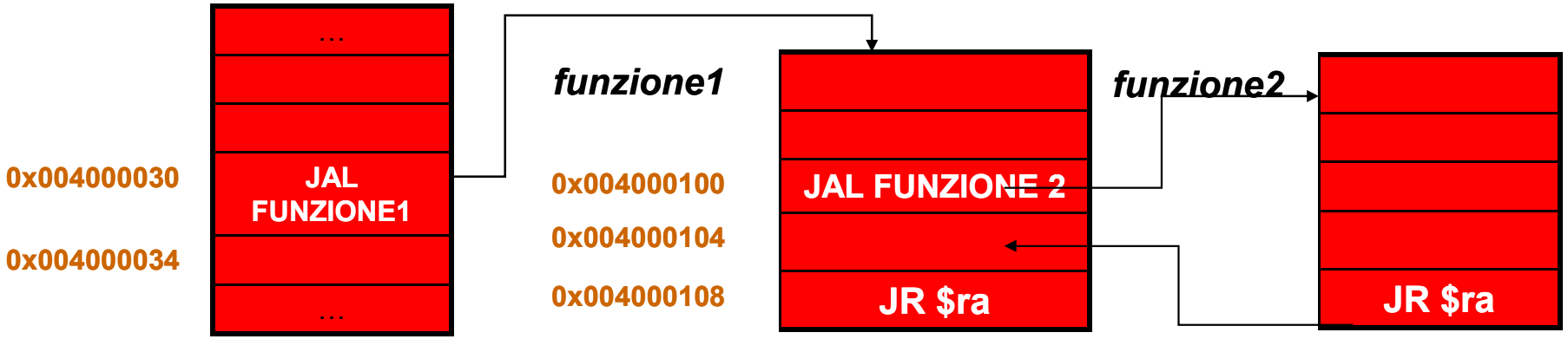
400	F1
412	JAL F2
416	
	JR RA

800	F2
820	JR RA



Activation Frame

Approfondimento: problema del salvataggio dell'indirizzo di ritorno



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

Da qui il nome: "*frame di attivazione*" perché è necessario un frame per ogni *attivazione* (chiamata) di funzione

Activation Frame

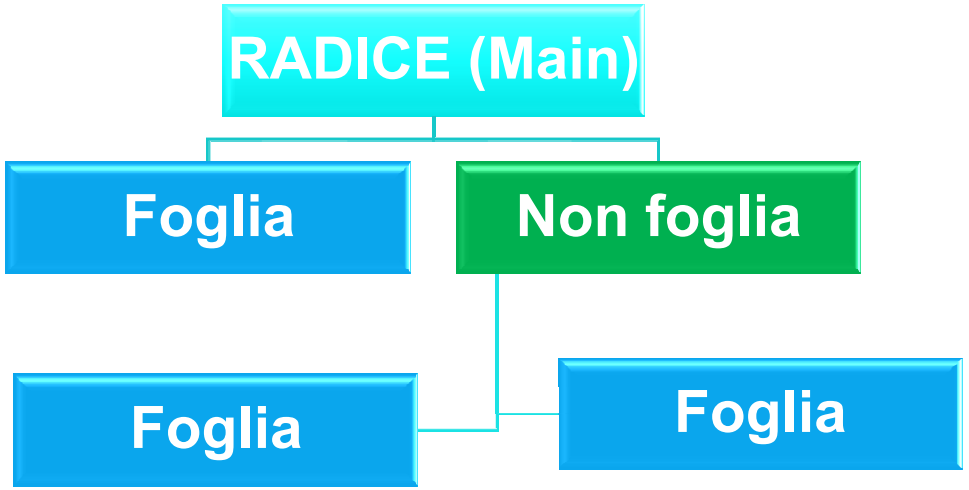
Approfondimento: problema del salvataggio dell'indirizzo di ritorno

Una funzione può essere classificata:

- ***Foglia:*** Se non chiama alcuna altra funzione al suo interno
- ***Non-foglia:*** 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 NON FOGLIA



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

Activation Frame

Esempio uso dell'Activation Frame

Calcolo del massimo fra i quadrati di due valori

```
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

Esempio uso dell'Activation Frame

Calcolo del massimo fra i quadrati di due valori

```
        .text
        .globl main
main:
        lw $t0,valore1
        lw $t1,valore2
        move $a0,$t0
        move $a1,$t1
        jal MAX_QUAD
        sw $v0,z

        li $v0,10
        syscall

        .data
valore1: .word 2312
valore2:.word 1105
MainActFrame: .space 4
# spazio per indirizzo di ritorno
```

```
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

Esempio uso dell'Activation Frame

Gestione indirizzo di ritorno da annidamento di subroutine (numero finito di chiamate)

```
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

Esempio uso dell'Activation Frame

Gestione indirizzo di ritorno da annidamento di subroutine (numero finito di chiamate)

```
.text
.globl main
main:
        lw $t0,x
        lw $t1,y
        move $a0,$t0,
        move $a1,$t1
        jal BATMAN
        sw $v0,z
        li $v0,10
        syscall
.data
MainAF: .space 4
BatmanAF: .space 4
```

```
BATMAN:
        sw $ra, MainAF
        add $a0,$a0,1
        add $a1,$a1,2
        jal ROBIN
        lw $ra, MainAF
        move $v0,$v0
        #per coerenza
        #non necessario
        jr $ra
```

```
ROBIN:
        sw $ra, BatmanAF
        mul $a0,$a0,2
        mul $a1,$a1,5
        jal JOKER
        lw $ra, BatmanAF
        move $v0,$v1
        jr $ra
```

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

Activation Frame

Approfondimento: problema della gestione dei parametri

L'activation **frame riguardo al trasferimento dei dati** consente di:

- ❖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

L'activation **frame riguardo al trasferimento dei dati** deve essere definito nella subroutine chiamante e utilizzata in quella chiamata

Activation Frame

Approfondimento: problema della gestione dei parametri

```
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=Robin(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

Approfondimento: problema della gestione dei parametri

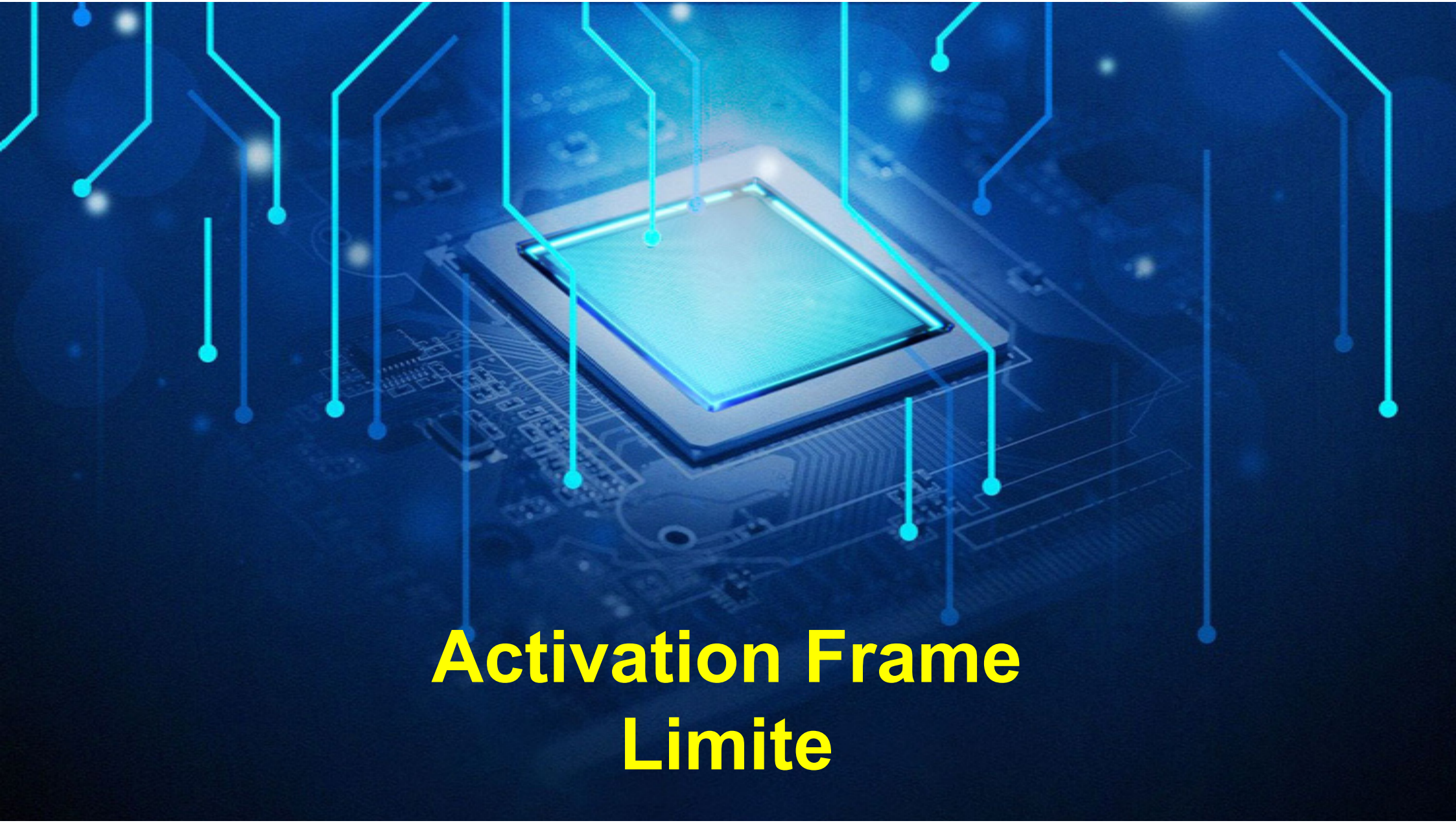
```
        .text
        .globl main
main:
        lw $t0,x
        lw $t1,y
        move $a0,$t0,
        move $a1,$t1
        jal Superman
        sw $v0,z

.data
MainAF: .space 4
SuperAF: .space 4
LuthorAF: . space 8
```

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

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

```
Luthor:
        la $s0, LuthorAF
        lw $t0, ($s0)
        lw $t1, 4($s0)
        mul $a0,$a0,$a1
        mul $t9,$a0,$a2
        mul $t0,$a3,$t0
        mul $t8,$t1,$t0
        div $v0,$t9,$t8
        jr $ra
```


Activation Frame
Limite

Activation Frame

Limite

Cosa accade quando c'è un annidamento dinamico (cioè ripetuto più volte a seconda dei parametri immessi in ingresso)?

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

```
#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

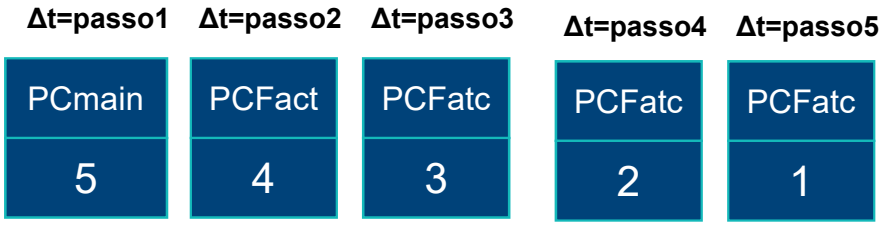
Limite

```
.text
.globl main
main
        lw $t0,x
        move $a0,$t0
        jal fact
        sw $v0,risultato

        li $v0,10
        syscall

        .data
factAF: .space 8
```

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



Activation Frame

Limite

Un singolo frame per funzione ***non è sufficiente*** nel caso di sub routine con annidamento dinamico (anche note come **funzioni *ricorsive*)**

Come si può allocare del nuovo spazio in memoria ogni volta che chiamo una funzione?

Stack

FINE