

# **Architettura Elaboratori Elettronici ESERCITAZIONI**

**SALTO A SUBROUTINE**

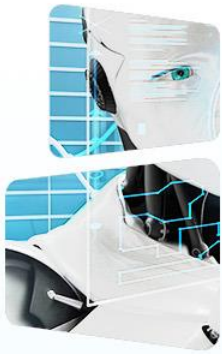
**Franco Liberati**  
**[liberati@di.uniroma1.it](mailto:liberati@di.uniroma1.it)**



# Argomenti

- ☐ Salto a subroutine/funzione
- ☐ Salto a subroutine/funzione nel MARS
- ☐ Il concetto di MACRO



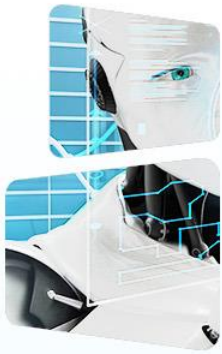


# Salto a Subroutine

L'istruzione di **salto a sub-routine** muta l'esecuzione da un punto ad un altro di un programma salvando l'indirizzo dell'istruzione successiva al salto

```
READ (a)
READ (b)
Massimo=MAX(a,b)
PRINT (Massimo)
```

```
MAX (val1,val2)
{
  IF (val1>val2)
    {mass=val1}
  ELSE
    {mass=val2}
  return (mass)
}
```



# Salto a Subroutine

La funzione che evoca la subroutine è detta **funzione chiamante**; la subroutine evocata è detta **funzione chiamata**

```
READ (a)
READ (b)
Massimo=MAX(a,b)
PRINT (Massimo)
```

**funzione chiamante**

```
MAX (val1,val2)
{
  IF (val1>val2)
    {mass=val1}
  ELSE
    {mass=val2}
  return (mass)
}
```

**funzione chiamata**



# Salto a Subroutine

Formalmente una **subroutine** in un linguaggio ad **alto** livello è costituito da

- un tipo del valore restituito dalla funzione (**type**)
- una **parola chiave** (function)
- il **nome della funzione** (name\_subroutine)
- dei **parametri di ingresso** acquisiti dalla funzione (var\_par)
- Il corpo della funzione
- una **istruzione di ritorno** alla funzione principale che riporta anche il valore (o **parametro di uscita**) che vuole essere riportato alla funzione chiamante

```
int function massimo(int a, int b)
{
    int max;
    if (a>b) {max=a}
    else {max=b}
    return max;
}
```

# SALTO A SUBROUTINE NEL MARS





# Salto a Subroutine MARS

Una **subroutine in MARS** richiede

- il **nome della funzione** (name\_subroutine) richiamato dalla istruzione **JAL**
- i **parametri di ingresso** sono posti nei registri \$a0,\$a1,\$a2,\$a3
- L'**etichetta** che indica l'inizio della funzione
- Il corpo della funzione
- Il **parametro di uscita** è posto nel registro \$v0 e/o \$v1
- L' **istruzione di ritorno** **JR \$RA**

```
...  
move $a0,$t0  
move $a1,$t1  
JAL MASSIMO  
...  
...  
li $v0,10  
syscall
```

**MASSIMO:**

```
        move $v0,$a0  
        bgt $a0,$a1,fine  
        move $v0,$a1  
  
fine:  
        jr $ra
```



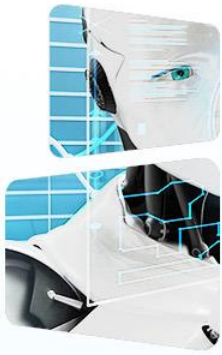


# Salto a Subroutine MARS

In MIPS/MARS istruzione di salto a sub-routine sposta l'esecuzione da un punto ad un altro di un programma salvando l'indirizzo dell'istruzione successiva al salto in un registro speciale (\$RA)

<b>jal target</b>	<i>Salto incondizionato all'istruzione <b>target</b> e salvataggio dell'indirizzo della prossima istruzione in <b>ra</b></i>
<b>jr rsource</b>	<i>Salto incondizionato all'istruzione che ha indirizzo memorizzato nel registro <b>rsource</b></i>
<b>jalr rsource, rdest</b>	<i>Salto incondizionato all'istruzione che ha indirizzo memorizzato nel registro <b>rsource</b> e salvataggio dell'indirizzo della prossima istruzione in <b>rdest</b></i>





# Salto a Subroutine MARS

```
Main:
lw $t0, pippo
lw $t1, paperino
add $t2, $t0, $t1
...
jal batman
...
...
li v0, 10
break
```

```
xor $t7, $t7, $t7
add $t7, $t6, $t5
```

```
...
...
...
...
...
...
...
```

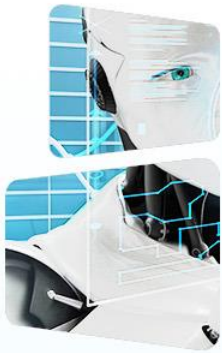
```
jr $ra
```



# Salto a Subroutine MARS

Registri preservanti e non

L'istruzione di salto a sub-routine implica (anche se nel simulatore MARS non sempre accade) un azzeramento dei registri non preservanti  $\$t0, \dots, \$t9$ ; mentre gli operandi contenuti nei registri preservanti  $\$s0, \$s9$  non sono modificati



# Salto a Subroutine MARS

Convenzione sui registri per i parametri di ingresso e di uscita

Quando si richiama una subroutine è convenzione porre i parametri di ingresso (i valori che devono essere manipolati dalla funzione) nei registri preservanti \$a0,\$a1,\$a2,\$a3

Il valore risultante di una subroutine è sito nel registro preservante \$v0



# Salto a SubRoutine

ESEMPIO

Realizzare un programma che svolga il massimo tra tre numeri

Approccio canonico

**maxtre(a,b,c)**

Riuso di funzione

**leggi a,b,c**

**t= MASSIMO(a,b)**

**max=MASSIMO(t,c)**

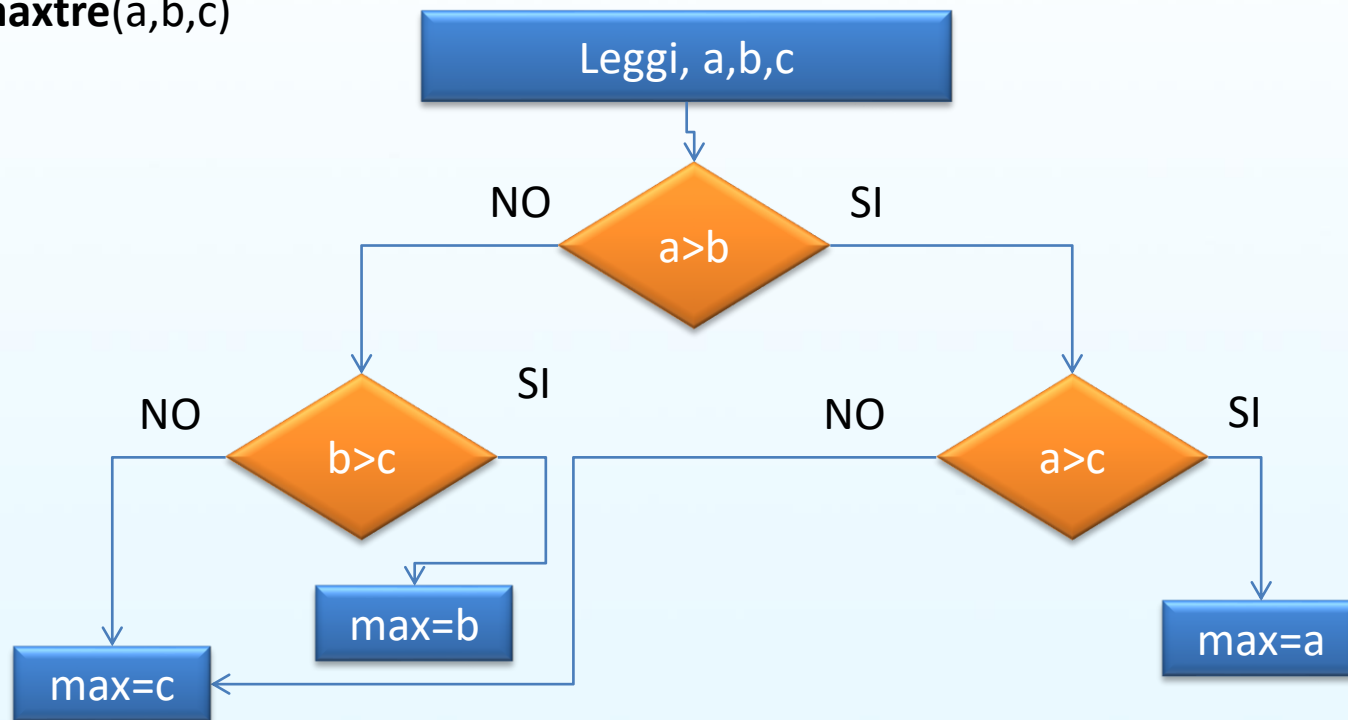
**conserva max**



Approccio canonico  
**maxtre(a,b,c)**

# Salto a SubRoutine

ESEMPIO



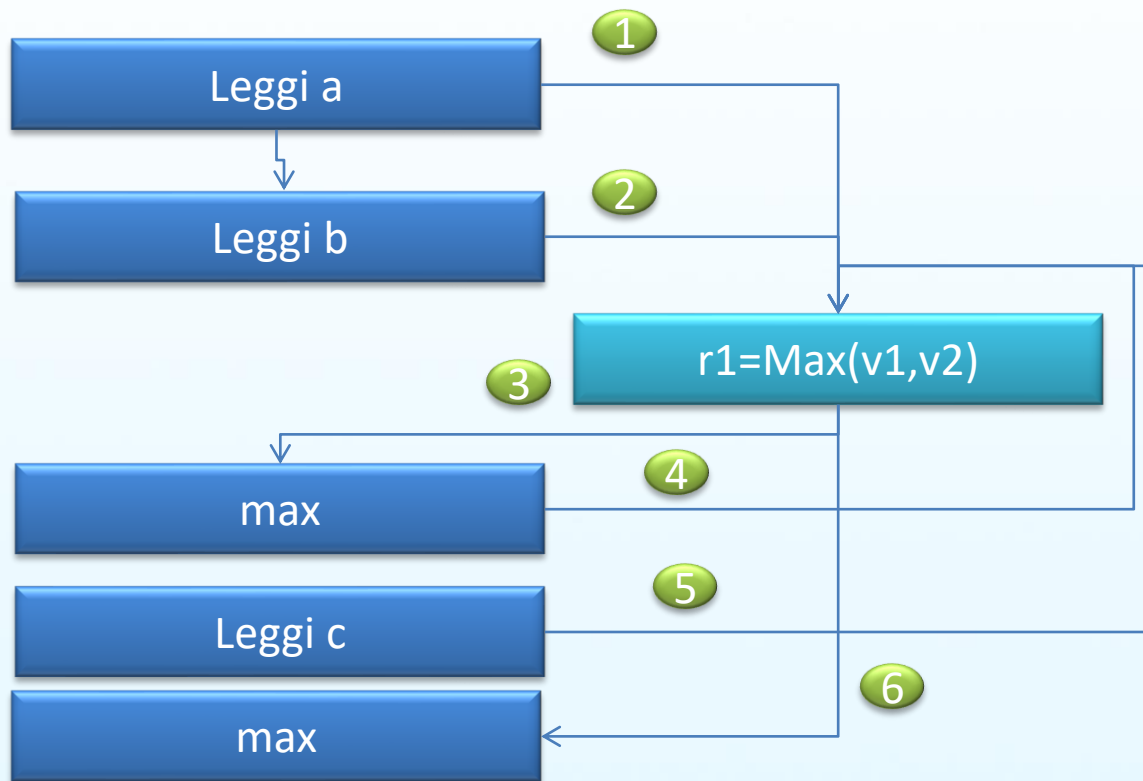


Riuso di funzione

leggi a,b,c  
t= MASSIMO(a,b)  
max=MASSIMO(t,c)  
conserva max

# Salto a SubRoutine

ESEMPIO





# Salto a SubRoutine

## ESEMPIO

```
.text
.globl main

main:
    lw $a0,x      #lettura primo valore
    lw $a1,y      #lettura secondo valore
    jal MASSIMO #salto a funzione
    move $a0,$v0 #recupero massimo dalla funzione
    lw $a1,z      #lettura terzo valore
    jal MASSIMO #salto a funzione
    move $a0,$v0 #recupero massimo dalla funzione
    move $t0,$a0 #massimo in T0

    li $v0,10
    syscall
```

### MASSIMO:

# PARAMETRI INGRESSO: A0 e A1 valori interi

# PARAMETRO USCITA: V0 massimo tra A0 e A1

move \$v0,\$a0 #Imposto A0 come massimo

bgt \$a0,\$a1,fine #Se A0>A1 allora finisco

move \$v0,\$a1 #Imposto A1 come massimo

fine:

jr \$ra

### .data

x:.word 45

y:.word 100

z:.word 77





# MACRO





# MACRO

## Generalità

Quando si definisce **una macro** si rimpiazza la sua etichetta con le linee di codice racchiuse tra le direttive di inizio e fine macro; si dice anche che c'è una inclusione diretta delle istruzioni (*raw source code*)

La sostituzione avviene in fase di assemblaggio o per mezzo di un pre-assemblatore (Assembly PreProcessor, APP), cioè un programma che opera sul codice tra la fase di compilazione e quella di assemblaggio

Il programma APP abilita il pre-assemblatore che sostituisce i nomi delle costanti con i loro valori e le macro nonché inserisce il testo dei file richiesti con la direttiva **.include**. **Pertanto quando si utilizza una macro non c'è un sovra lavoro dovuto a un salto in memoria, il passaggio di parametri e il ritorno al programma principale**

Malgrado questo è compito del programmatore gestire bene i registri e le etichette che sono usate all'interno della macro evitando nomi comuni, sovrascritture di registri significativi (cioè già usati nel programma principale) e l'uso di identificatori generici



# MACRO

## Definizione

**.macro**

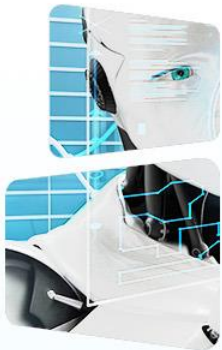
Istruzione1

Istruzione2

...

Istruzionien

**.end\_macro**



```
.text
```

```
.globl main
```

```
.macro FINEPROGRAMMA
```

```
    li $v0,10
```

```
    syscall
```

```
.end_macro
```

```
main:
```

```
    lw $t0,Joker
```

```
    lw $t1,Goblin
```

```
    li $t2,0
```

```
ciclo:
```

```
    bltz $t1,fine
```

```
    add $t2,$t2,$t0
```

```
    sub $t1,$t1,1
```

```
    j ciclo
```

```
fine:
```

```
    FINEPROGRAMMA
```

```
#definizione macro
```

```
#istruzione macro
```

```
#istruzione macro
```

```
#chiusura macro
```

```
#chiamata macro
```

```
 #(sostituzione codice)
```

# MACRO

## Esempio

```
.data
```

```
Joker:.word 4
```

```
Goblin:.word 6
```



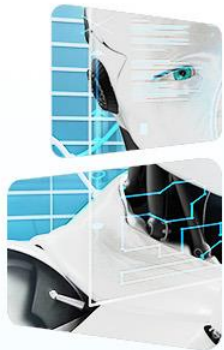
# Direttiva EQV

Generalità

Analogamente alla MACRO esiste la direttiva **EQV** che consente di assegnare un nome ad un registro (in generale sostituisce un identificatore con una stringa)

Segue la logica: definisci una volta e usa molte volte

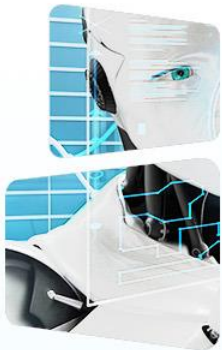
Utile per avere un debug più semplice



# Direttiva EQV

Generalità

.eqv ETICHETTA registro



```
.text
.globl main
.macro FINEPROGRAMMA
    li $v0,10
    syscall
.end_macro
.eqv VAR1 $t0
.eqv VAR2 $t1
.eqv TOT $t2
main:
    lw VAR1,x
    lw VAR2,y
    li TOT,0

ciclo:
    bltz VAR2,fine
    add TOT,TOT,VAR1
    sub VAR2,VAR2,1
    j ciclo

fine:
    FINEPROGRAMMA
```

```
#definizione macro
#istruzione macro
#istruzione macro
#chiusura macro
```

```
#chiamata macro
#(sostituzione codice)
```

# Direttiva EQV

## Esempio

```
.data
Joker:.word 4
Goblin:.word 6
```



FINE

