

**Architettura  
Elaboratori  
Elettronici  
ESERCITAZIONI  
STRUTTURE ITERATIVE**

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



# Argomenti

- ☐ Selezione

  - ☐ Il costrutto IF

- ☐ Cicli

  - ☐ Il costrutto WHILE

  - ☐ Il costrutto DO... WHILE

  - ☐ Il costrutto FOR





# Strutture Iterative

- ❑ Un programma è una sequenza di istruzioni; con i salti si può controllare l'**ordine di esecuzione**
- ❑ Le istruzioni, in genere, sono eseguite sequenzialmente in accordo all'ordine in cui sono scritte:
  - ❑ La CPU non ha una visione di insieme del programma: “vede” solo l'istruzione che deve eseguire e il contenuto dei registri
  - ❑ La CPU esegue una istruzione alla volta nell'ordine scritto in memoria (l'indirizzo è presente nel PC)



# Strutture Iterative

- ❑ Fanno eccezione le **istruzioni di salto** che fanno proseguire l'esecuzione all'indirizzo specificato non necessariamente contiguo all'ultimo elaborato
- ❑ L'istruzione che è eseguita dopo un salto è quella che si trova all'indirizzo specificato come destinazione del salto
  - ❑ Se il **salto è condizionato** e la condizione è falsa si prosegue normalmente con l'istruzione successiva
  - ❑ Se il **salto è incondizionato** si prosegue all'indirizzo specificato senza valutare la verità di una condizione

# SALTO CONDIZIONALE IF





# Salto condizionale IF

- ❑ Si esegue una istruzione solo se una condizione (espressione) è vera
- ❑ In linguaggio ad alto livello è spesso presente il costrutto IF la condizione e poi l'esecuzione di una o più istruzioni incluse tra {...}

If (espressione) {istruzione}

Esempio linguaggio alto livello:

```
if (Batman>0) {  
    Robin=Batman+2;  
}
```



# Salto condizionale IF

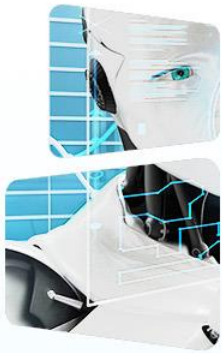
If (espressione) {istruzione}

- ❑ In assembler un salto condizionale si concretizza con:
1. Il codice per valutare l'espressione
  2. il salto condizionato per eseguire l'istruzione da dover elaborare
  3. Il salto incondizionato per evitare l'esecuzione dell'istruzione nel caso di condizione non verificata

```
if (Batman>0) {  
    Robin=Batman+2;  
}
```

```
    b $t0,ramo_THEN ❶  
    j salto ❸  
ramo_THEN: ❷  
    addi $t1,$t2,2  
salto:
```



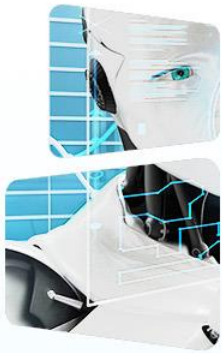


# Salto condizionale IF

Definite due variabili intere , Batman e Robin, in memoria se Batman ha un valore maggiore di Robin allora il valore di Robin diventa uguale al valore di Batman incrementato di due

```
If (Batman>0)      {  
                    Robin=Batman+2;  
                    }
```





# Salto condizionale IF

```
.text
.globl main
main:
    lw $t0,Batman
    lw $t1,Robin
    bgtz $t0, ISTRUZIONE
    j FINE_IF
ISTRUZIONE:
    add $t1,$t0,2
FINE_IF:
    sw $t1,Robin
    li $v0,10
    syscall
```

```
If (Batman>0) {
    Robin=Batman+2
}
```

```
# carica Batman
# carica Robin
# salto se condizione è vera
# salto se condizione è falsa
# INIZIO ISTRUZIONE
# imposta Robin con Robin=Batman+2
# FINE ISTRUZIONE
# salva $t1 in Robin
```

```
.data
Batman: .word 5
Robin: .word 0
```



# Salto condizionale IF (estensione)

- ❑ Si eseguono più istruzioni solo se una condizione (espressione) è vera
- ❑ In linguaggio ad alto livello si esplicita la condizione (dentro le parentesi tonde) dopo il costrutto IF e poi nel corpo dell'IF (dentro le parentesi graffe) si esplicitano le istruzioni

**If (espressione) {istruzioni}**

Esempio linguaggio alto livello:

```
if (Batman>0)      {  
    Batman=Batman+2;  
    Robin=Robin·10;  
}
```

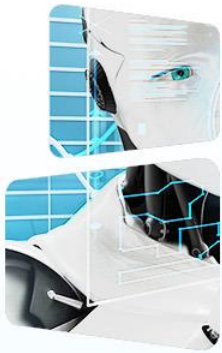


# Salto condizionale IF (estensione)

- ❑ Si eseguono più istruzioni solo se una condizione (espressione) è vera
- ❑ In assembler un salto condizionale si concretizza con:
  1. il salto condizionato per eseguire il blocco di istruzioni del “ramo then”
  2. Le istruzioni da svolgere nel caso di condizione verificata
  3. il salto incondizionato per evitare l'esecuzione del “ramo then”

**If (espressione) {istruzioni}**

```
b $t0,ramo_THEN ❶  
j fine_IF ❸  
ramo_THEN: ❷  
    addi $t1,$t2,2  
fine_IF:
```



# Salto condizionale IF (estensione)

Definite due variabili intere, Batman e Robin, in memoria. Se Batman ha un valore maggiore di zero allora il valore di Batman si incrementa di due e il valore di Robin si moltiplica per dieci

```
If (Batman>0)      {  
                    Batman=Batman+2;  
                    Robin=Robin*10;  
                    }
```



# Salto condizionale IF (estensione)

```
.text  
.globl main  
main:
```

**RAMO\_THEN:**

**FINE\_IF:**

```
lw $t0,Batman  
lw $t1,Robin  
bgtz $t0, RAMO_THEN  
j FINE_IF
```

```
add $t0,$t0,2  
mul $t1,$t1,10
```

```
sw $t0,Batman  
sw $t1,Robin  
li $v0,10  
syscall
```

```
If (Batman>0) {  
    Batman=Batman+2;  
    Robin=Robin*10;  
}
```

```
# carica Batman  
# carica Robin  
# va al RAMO THEN  
# Esclude il blocco THEN dall'elaborazione  
# INIZIO RAMO THEN  
# Batman=Batman+2  
# Robin=Robin*10  
# FINE RAMO THEN  
# salva $t0 in variabile Batman  
# salva $t1 in variabile Robin
```

```
.data  
Batman: .word 5  
Robin: .word 0
```



# Salto condizionale IF (estensione)

Ottimizzazione codice

```
If (Batman>0) {  
    Batman=Batman+2;  
    Robin=Robin*10;  
}
```

```
.text  
.globl main  
main:
```

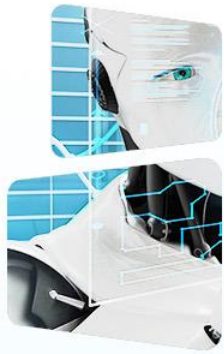
```
lw $t0,x  
lw $t1,y  
blez $t0,FINE_IF  
add $t0,$t0,2  
mul $t1,$t1,10
```

**FINE\_IF:**

```
sw $t0,Batman  
sw $t1,Robin  
li $v0,10  
syscall
```

```
# carica Batman  
# carica Robin  
# Esclusione delle operazioni sottostanti se la condizione inversa è vera  
# INIZIO RAMO THEN (prima istruzione)  
# Blocco THEN (seconda istruzione)  
# FINE RAMO THEN  
# salva $t0 in variabile Batman  
# salva $t1 in variabile Robin
```

```
.data  
Batman: .word 5  
Robin: .word 0
```



# Salto condizionale IF THEN ELSE

- ❑ Si esegue del codice solo se una condizione (espressione) è vera altrimenti si esegue altra serie di istruzioni
- ❑ Nei linguaggi ad alto livello si usa il costrutto IF la condizione e delle istruzioni tra {} segue il costrutto ELSE e altre istruzioni racchiuse tra {}

If (espressione) {istruzioni}  
else{istruzioni}

Esempio linguaggio alto livello:

```
if (Batman>0){Batman=Batman+1;  
Robin=Batman+2;}  
else {Batman=Batman-1;  
Robin=Batman-2;}
```





# Salto condizionale IF THEN ELSE

- ❑ Si esegue del codice solo se una condizione (espressione) è vera altrimenti si esegue altro codice
- ❑ In assembler un salto condizionale completo si concretizza con:
  1. Codice per valutare
  2. Salto condizionato per svolgere il “ramo then”
    - a. Alla fine del “ramo then” salto incondizionato oltre il “ramo else”
  3. Salto incondizionato per svolgere il “ramo else”

If (espressione) {istruzioni}  
else{istruzioni}

```
b $t0,ramo_THEN      ❶  
j ramo_ELSE          ❷  
  
ramo_THEN:           ❸  
...  
j fine_IF  
ramo_ELSE:  
...  
...  
fine_IF:
```



# Salto condizionale IF THEN ELSE

```
.text
.globl main
main:
```

```
    lw $t0,Batman
    lw $t1,Robin
    bgtz $t0, RAMO_THEN
    j RAMO_ELSE
RAMO_THEN:
    add $t0,$t0,1
    add $t1,$t0,2
    j FINE_IF
RAMO_ELSE:
    sub $t0,$t0,1
    sub $t1,$t0,2
FINE_IF:
    sw $t0,Batman
    sw $t1,Robin
    li $v0,10
    syscall
```

```
if (Batman>0){Batman=Batman+1;
               Robin=Batman+2;}
else {Batman=Batman-1;
      Robin=Batman-2;}
```

```
# carica Batman
# carica Robin
# salto se condizione vera al BLOCCO IF
# salto incondizionato in caso condione falsa al BLOCCO ELSE
# INIZIO BLOCCO IF
# Batman=Batman+1
# Robin=Batman+2
# FINE BLOCCO IF
# INIZIO BLOCCO ELSE
#Batman=Batman-1
# Robin=Batman-2
# FINE BLOCCO IF
# salva $t1 in Batman
# salva $t1 in Robin
```

```
.data
Batman: .word 5
Robin: .word 0
```



```
.text  
.globl main  
main:
```

```
lw $t0,Batman  
lw $t1,Robin  
blez $t0,RAMO_ELSE  
add $t0,$t0,1  
add $t1,$t0,2  
j FINE_IF
```

RAMO\_ELSE:

```
sub $t0,$t0,1  
sub $t1,$t0,2
```

FINE\_IF:

```
sw $t0,Batman  
sw $t1,Robin  
li $v0,10  
syscall
```

# Salto condizionale IF THEN ELSE

(variante)

```
If (Batman>0){Batman=Batman+1;  
Robin=Batman+2;}  
else {Batman=Batman-1;  
Robin=Batman-2;}
```

```
# carica Batman  
# carica Robin  
# salto se condizione opposta è vera al BLOCCO ELSE  
# INIZIO RAMOTHEN  
# BLOCCO THEN  
# FINE RAMO THEN SALTO OLTRE IL BLOCCO ELSE  
# INIZIO RAMO ELSE  
#x=x-1  
# y=x-2  
# FINE BLOCCO IF  
# salva $t1 in Batman  
# salva $t1 in Robin
```

```
.data  
Batman: .word 5  
Robin: .word 0
```

# CICLO CONDIZIONALE WHILE





# Ciclo condizionale WHILE

- ❑ Si esegue più volte del codice finché una condizione (espressione) è vera. Non si sa se l'espressione è vera all'inizio (quindi va controllata prima di eseguire il codice)
- ❑ In un linguaggio ad alto livello si ha il costrutto WHILE nel quale si valuta la condizione. Se la condizione è vera si eseguono le istruzioni all'interno del corpo del while (cioè quello racchiuso tra parentesi graffe)

```
while (espressione) {  
    istruzione1  
    istruzione2  
    ...  
    istruzionen  
}
```

Esempio linguaggio alto livello:

```
while(Batman>0){  
    Robin=Robin+Batman;  
    Batman=Batman-1;  
}
```

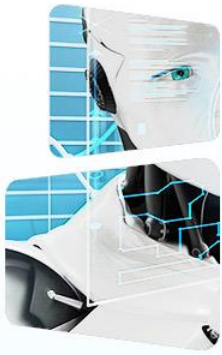


# Ciclo condizionale WHILE

- ❑ Si esegue più volte del codice finché una condizione (espressione) è vera. Non si sa se l'espressione è vera all'inizio (quindi va controllata prima di eseguire il codice)
- ❑ In assembler un ciclo condizionale si concretizza con:
  1. Il codice per valutare l'espressione
  2. Salto incondizionato per saltare il blocco delle istruzioni se la condizione è falsa
  3. Salto incondizionato al punto 1 per valutare la condizione

```
while (espressione) {  
    istruzione1  
    istruzione2  
    ...  
    istruzionen  
}
```

```
controllo_WHILE:  
    b $t0,blocco_WHILE ❶  
    j fine_WHILE        ❷  
blocco_WHILE:  
    ...  
    j controllo_WHILE  ❸  
fine_WHILE:
```



# Ciclo condizionale WHILE

Si ripete l'incremento di Robin della quantità di Batman che va a diminuire di una unità ad ogni ciclo fino a quando il valore di Batman è maggiore di 0

```
while(Batman>0){  
    Robin=Robin+Batman;  
    Batman=Batman-1;  
}
```





# Ciclo condizionale WHILE

```
while(Batman>0){  
    Robin=Robin+Batman;  
    Batman=Batman-1;  
}
```

```
.text  
.globl main  
main:
```

**CONTROLLO\_WHILE:**

```
lw $t0,Batman  
lw $t1,Robin  
  
bgtz $t0,BLOCCO_WHILE  
j FINE_WHILE
```

**BLOCCO\_WHILE:**

```
add $t1,$t1,$t0  
sub $t0,$t0,1  
j CONTROLLO_WHILE
```

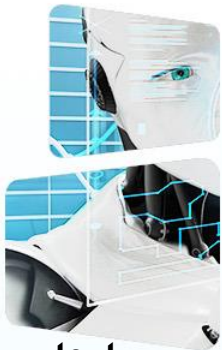
**FINE\_WHILE:**

```
sw $t1,Batman  
sw $t1,Robin  
li $v0,10  
syscall
```

```
# Carica Batman  
# Carica Robin  
# INIZIO WHILE  
# CONTROLLO CONDIZIONE  
# SALTO A FINE WHILE  
# INIZIO BLOCCO WHILE  
# Istruzione di somma  
# Istruzione di sottrazione  
# FINE BLOCCO WHILE e ritorno al controllo
```

```
# Salva risultato in Batman  
# Salva risultato in Robin
```

```
.data  
Batman: .word 5  
Robin: .word 0
```



```
.text
.globl main
main:
```

**WHILE:**

**END\_WHILE:**

```
lw $t0,Batman
lw $t1,Robin
```

```
blez $t0,END_WHILE
add $t1,$t1,$t0
sub $t0,$t0,1
jWHILE
```

```
sw $t1,Batman
sw $t1,Robin
li $v0,10
syscall
```

# Ciclo condizionale WHILE

(variante)

```
while(Batman>0){
    Robin=Robin+Batman;
    Batman=Batman-1;
}
```


```
# Carica Batman
# Carica Robin
```

```
# Esce dal CICLO WHILE se condizione è falsa
# INIZIO BLOCCO WHILE (somma)
# BLOCCO WHILE (sottrazione)
```

**# FINE BLOCCO WHILE**

```
# Salva risultato in Batman
# Salva risultato in Robin
```

```
.data
Batman: .word 5
Robin: .word 0
```



# CICLO CONDIZIONALE CON ALMENO UNA ESECUZIONE DI UN GRUPPO DI ISTRUZIONI **DO WHILE**





# Ciclo condizionale DO WHILE

- ❑ Si vuole eseguire più volte del codice finché una condizione è vera. Si sa se che l'espressione è vera all'inizio (quindi si esegue il codice almeno una volta e poi si controlla l'espressione)
- ❑ In un linguaggio ad alto livello si usa il costrutto DO e si riporta il corpo delle istruzioni (tra parentesi graffe) alla fine si valuta una condizione che se non è vera prosegue il flusso dei dati altrimenti ripete le istruzioni nel corpo delle istruzioni

```
do {  
    istruzione1  
    istruzione2  
    ...  
    istruzionen  
}  
while (espressione)
```

Esempio linguaggio alto livello:

```
do {  
    Robin=Robin+Batman;  
    Batman=Batman-1;  
}  
while (Batman>0)
```



# Ciclo condizionale DO WHILE

- ☐ Si vuole eseguire più volte del codice finché una condizione è vera. Si sa se che l'espressione è vera all'inizio (quindi si esegue il codice almeno una volta e poi si controlla l'espressione)
- ☐ In assembler un ciclo condizionale si concretizza con:
  1. Etichetta prima del codice
  2. Codice da ripetere finché la condizione è vera
  3. Salto condizionato al punto 1 se la condizione è vera

```
do {  
    istruzione1  
    istruzione2  
    ...  
    istruzionen  
}  
while (espressione)
```

**DO:**

```
...  
...  
...  
b $t0,DO ③
```

①  
②



# Ciclo condizionale DO WHILE

```
.text
.globl main
main:
```

```
    lw $t0,Batman
    lw $t1,Robin
DOWHILE:
    add $t1,$t1,$t0
    sub $t0,$t0,1
    bgtz $t0,DOWHILE
    sw $t0,Batman
    sw $t1,Robin
```

```
li $v0,10
syscall
```

```
# Carica Batman
# Carica Robin
# INIZIO IL BLOCCO DO
```

```
# Effettua somma
# Effettua sottrazione
# Controlla la condizione WHILE: se è vera cicla altrimenti finisce
# Salva risultato di Batman
# Salva risultato di Robin
```

```
do      {
        Robin=Robin+Batman;
        Batman=Batman-1;
      }
while (Batman>0)
```

```
.data
Batman: .word 5
Robin: .word 0
```

# CICLO ITERATIVO FOR







# Ciclo iterativo FOR

- ❑ Si vuole eseguire un blocco di istruzioni, un numero prestabilito di volte
- ❑ In un linguaggio ad alto livello si ha il costrutto FOR seguito dalla dichiarazione di un indice; una condizione di ripetizione; l'incremento dell'indice (non necessariamente unitario). Segue il corpo delle istruzioni che devono essere ripetuto un numero esatto di volte

```
for (inizializzazione, condizione, incremento){  
    istruzione1  
    istruzione 2  
    istruzione n  
}
```

Esempio linguaggio alto livello:

```
for(i=0;i<5;i=i+1)  
{  
    totale=totale+costo_prodotto;  
}
```

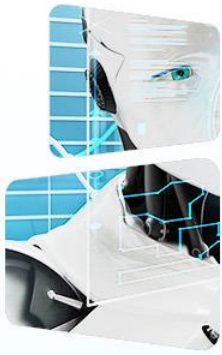


- ❑ Si vuole eseguire un blocco di istruzioni, un numero prestabilito di volte
- ❑ In assembler un ciclo iterativo si concretizza con:
  1. Inizializzare i registri che mi servono prima di iniziare il ciclo (compreso l'indice)
  2. Valutazione della condizione
  3. Codice da ripetere il numero stabilito di volte
  4. Incremento il registro usato come indice
  5. Salto condizionato al punto 2 se la condizione è vera

## Ciclo iterativo FOR

```
for (inizializzazione, condizione, incremento){  
    istruzione1  
    istruzione 2  
    istruzione n  
}
```

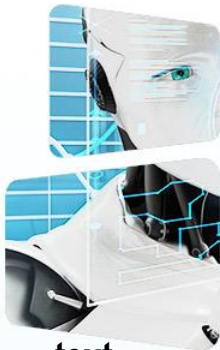
```
li $t0,0                #indice  
li $t1, valore_terminazione  
  
FOR:  
    b $t0,$t1, blocco_FOR  
    j fine_FOR  
  
blocco_FOR:  
    ....  
    add $t1,$t1,1  
    j FOR  
  
fine_FOR:
```



# Ciclo iterativo FOR

Somma 5 volte il valore 15 e memorizza il risultato in una variabile totale

```
for(i=0;i<5;i++)  
{  
  totale=totale+15;  
}
```



# Ciclo iterativo FOR

```
for(i=0;i<5;i=i+1)
{
    totale=totale+valore_15;
}
```

```
.text
.globl main
main:
```

```
li $t9,0
li $t0,15
li $t1,0
```

**FOR:**

```
blt $t9,5,BLOCCOFOR
j FINE
```

**BLOCCOFOR:**

```
add $t1,$t1,$t0
addi $t9,$t9,1
j FOR
```

**FINE:**

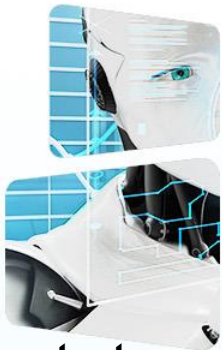
```
sw $t1,totale
li $v0,10
syscall
```

**# INIZIALIZZAZIONE INDICE i**  
**# lettura del valore da sommare**  
**#Valore della somma che andrà nella variabile totale**

**# CONTROLLO CONDIZIONE FOR (cioè i<5)**  
**# TERMINAZIONE FOR**  
**# INIZIO BLOCCO FOR**  
**# Operazione (somma della spesa)**  
**# INCREMENTO INDICE FOR**  
**# SALTO AL NUOVO CONTROLLO FOR**

**# Salva risultato in totale**

```
.data
totale: .word 0
```



```
.text
.globl main
main:
```

```
li $t9,0
li $t0,15
li $t1,0
```

**FOR:**

```
bgt $t9,4,FINE_FOR
add $t1,$t1,$t0
addi $t9,$t9,1
j FOR
```

**FINE\_FOR:**

```
sw $t1,totale
li $v0,10
syscall
```

# Ciclo iterativo FOR

(variante)

```
for(i=0;i<5;i=i+1)
{
    totale=totale+valore_15;
}
```

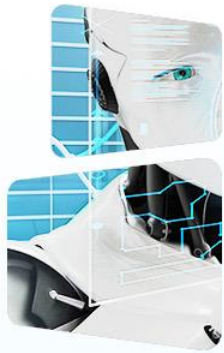
```
# INIZIALIZZAZIONE INDICE i
# lettura del valore da sommare
#Valore della somma che andrà nella variabile totale
#
# CONTROLLO CONDIZIONE FOR
# INIZIO BLOCCO FOR
# INCREMENTO INDICE
#CICLO

# Salva risultato in totale
```

**.data**  
totale: .word 0

# ESERCIZIO



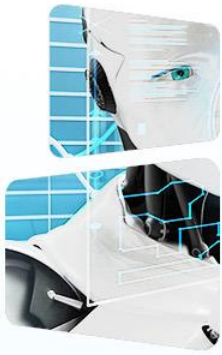


# ESERCIZIO I

La spesa.

Calcolare la spesa sommando il prezzo (valori interi) di sei prodotti immessi da tastiera





# ESERCIZIO I

La spesa.

Calcolare la spesa sommando il prezzo (valori interi) di sei prodotti immessi da tastiera

```
totale=0  
for(i=0;i<6;i++)  
{  
    leggi x  
    totale=totale+x;  
}
```



# ESERCIZIO I

text  
.globl main  
main:

li \$t9,0  
li \$t0,0

**FOR:**

bge \$t9,6, **FINE\_FOR**  
li \$v0,5  
syscall  
move \$t1,\$v0  
add \$t0,\$t0,\$t1  
add \$t9,\$t9,1  
j **FOR**

**FINE\_FOR:**

sw \$t0,totale

li \$v0,10  
syscall

.data  
totale: .word 0

# Inizializza indice  
# inizializza il registro che ospiterà totale

# CONTROLLO CONDIZIONE NEGATA FOR  
# Servizio di lettura intero  
# Chiamata del servizio  
# spostamento del valore letto da tastiera  
# totale=totale+x  
# incremento indice  
#CICLO FOR

# Salva risultato in totale

```
totale=0
for{i=0;i<6;i++}
{
    leggi x
    totale=totale+x;
}
```



# ESERCIZIO I

(variante)

```
.text
.globl main
main:
```

```
li $t9,0
lw $t8, limite
li $t0,0
```

```
# Inizializza indice
# indice limite
# inizializza il registro che ospiterà totale
```

**FOR:**

```
bge $t9,$t8, FINE_FOR
li $v0,5
syscall
move $t1,$v0
add $t0,$t0,$t1
add $t9,$t9,1
j FOR
```

```
# CONTROLLO CONDIZIONE NEGATA FOR
# Servizio di lettura intero
# Chiamata del servizio
# spostamento del valore letto da tastiera
# totale=totale+x
# incremento indice
#CICLO FOR
```

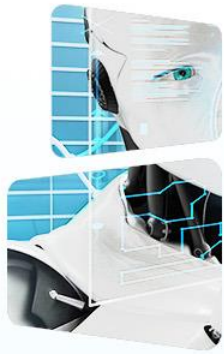
**FINE\_FOR:**

```
sw $t0,totale
li $v0,10
syscall
```

```
# Salva risultato in totale
```

```
.data
totale: .word 0
limite: .word 6
```

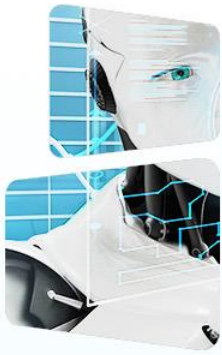
```
totale=0
for{i=0;i<6;i++}
{
    leggi x
    totale=totale+x;
}
```



## ESERCIZIO II

Sommatoria.

Svolgere il totale di valori (interi) immessi da tastiera con terminazione quando il valore è zero



## ESERCIZIO II

Sommatoria.

Svolgere il totale di valori (interi) immessi da tastiera con terminazione quando il valore è zero

```
totale=0  
leggi x  
while{x!=0}  
{  
    totale=totale+x;  
    leggi x  
}
```



## ESERCIZIO II

```
.text  
.globl main  
main:
```

```
li $t0,0  
li $v0,5  
syscall  
move $t1,$v0
```

**WHILE:**

```
beqz $t1,END_WHILE  
add $t0,$t0,$t1  
li $v0,5  
syscall  
move $t1,$v0  
j WHILE
```

**END\_WHILE:**

```
sw $t1,totale  
li $v0,10  
syscall
```

```
.data  
totale: .word 0
```

```
# inizializza il registro che ospiterà totale  
# Servizio di lettura intero  
# Chiamata del servizio  
# spostamento del valore letto da tastiera
```

```
#Esce dal CICLO WHILE se condizione è falsa  
# sommatoria  
# Servizio di lettura intero  
# Chiamata del servizio  
# spostamento del valore letto da tastiera  
# SALTO CICLO WHILE
```

```
# Salva risultato in totale
```

```
totale=0  
leggi x  
while(x!=0)  
{  
  
}  
}
```

```
totale=totale+x;  
leggi x
```



```
.text
.globl main
main:
```

```
    li $t0,0
DOWHILE:
    li $v0,5
    syscall
    move $t1,$v0
    add $t0,$t0,$t1
    bnez $t1,DOWHILE
    sw $t1,totale
    li $v0,10
    syscall
```

```
.data
totale: .word 0
```

## ESERCIZIO II

(variante per rendere il ciclo indipendente  
dal valore di terminazione)

# inizializza il registro che ospiterà totale

# Servizio di lettura intero

# Chiamata del servizio

# spostamento del valore letto da tastiera

# sommatoria

# ripete ciclo se condizione è vera

# Salva risultato in totale

```
totale=0
do{
    leggi x
    totale=totale+x;
} while{x!=0}
```

FINE

