Architettura Elaboratori Elettronici

ESERCITAZIONI

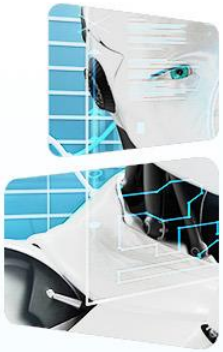
ISTRUZIONI MARS

Franco Liberati

liberati@di.uniroma1.it

Argomenti

- Istruzioni di spostamento
- Istruzioni logiche-aritmetiche
- Istruzioni di confronto
- Istruzioni di salto
  - Condizionato
  - Incondizionato
- Istruzione di salto a funzione



Suddivisione in classi

Le istruzioni del linguaggio assembly possono essere divise nelle seguenti categorie:

Istruzioni “Load and Store”

Spostano dati tra la memoria e i registri generali del processore
(i valori non sono modificati, ma solo spostati)

Istruzioni “Load Immediate”

Caricano, nei registri, valori costanti

Istruzioni “Data Movement”

Spostano dati tra i registri del processore

Istruzioni aritmetico/logiche

Effettuano operazioni aritmetico, logiche o di scorrimento sui registri del processore
(il valore risultante modifica i condition code della ALU)

Istruzioni di confronto

Effettuano il confronto tra i valori contenuti nei registri

Istruzioni di salto condizionato

Spostano l’esecuzione da un punto ad un altro di un programma in presenza di certe condizioni

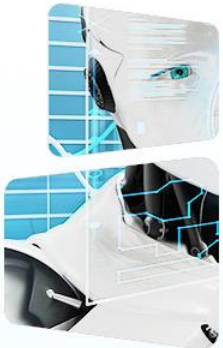
Istruzioni di salto non condizionato

Spostano l’esecuzione da un punto ad un altro di un programma

Istruzioni di sistema o comando

Influenzano lo svolgimento del programma (HALT, NOP,BREAK , TRAP …)

RAPPRESENTAZIONE DELLE PAROLE NEGLI ELABORATORI ELETTONICI



Little endian / Big endian

- **Definizione di endianness:** disposizione di una parola (word) nei byte di memoria
- **Little Endian**: memorizza prima la “little end” (ovvero i bit meno significativi) della word nelle locazioni di memoria con indirizzo più basso
- **Big Endian**: memorizza prima la “big end” (ovvero i bit più significativi) della word nelle locazioni di memoria con indirizzo più basso

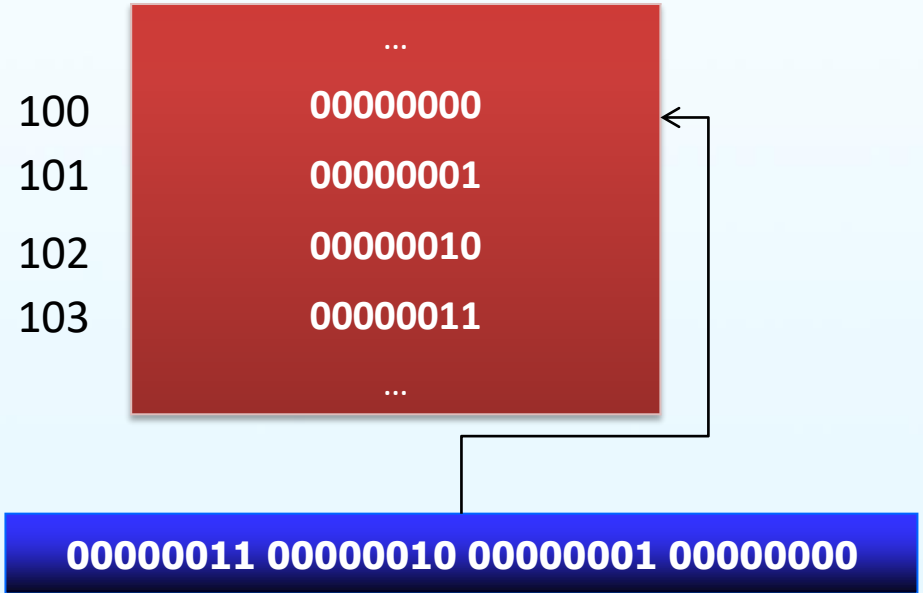
Terminologia ripresa da “I viaggi di Gulliver” di Jonathan Swift, in cui due fazioni sono in lotta per decidere se le uova sode devono essere aperte a partire dalla "little end" o dal "big end" dell'uovo

- **NB:**
  - I processori x86 sono little-endian
  - MIPS può essere little endian o big endian (L’endianness di SPIM dipende dal sistema in cui viene eseguito, quindi in generale è little endian)
  - Quando bisogna affrontare i problemi di endianness?
    - Quando si "mischiano" operazioni su 8, 16 e 32 bit
    - Se si utilizzano operazioni uniformi, non vi sono problemi di sorta

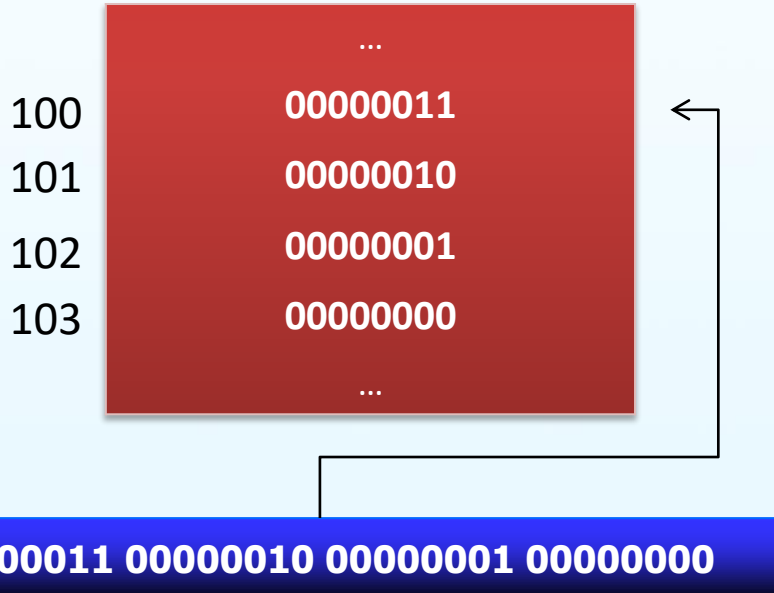
Little endian / Big endian

Esempio

Little Endian:



Big Endian:



Little endian / Big endian

Esempio

Little Endian:

lh $t0,value

#copia 16 bit a partire dalla locazione
#con indirizzo value

	...
100	00001100
101	10100001
102	01100010
103	00000011
	...

00000000 00000000 10100001 00001100

Big Endian:

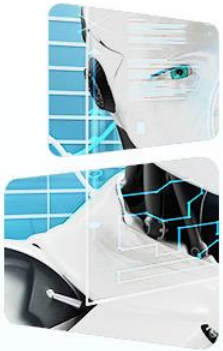
lh $t0,value

#copia 16 bit a partire dalla locazione
#con indirizzo value

	...
100	00000011
101	01100010
102	10100001
103	00011000
	...

00000000 00000000 01100010 00000011

ISTRUZIONI DI SPOSTAMENTO



Generalità architettura Load-Store

L'architettura di MIPS è di tipo **Load-and-Store**

In pratica, la maggioranza delle istruzioni di MIPS elaborano operandi siti nei registri interni al processore e non accedendo ad essi direttamente nelle celle di memoria nei quali risiedono

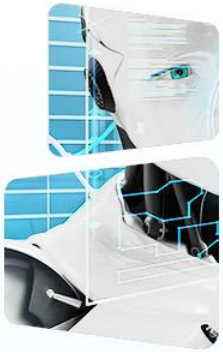
Esempio:

```
add $t0, $t1, $t2      #Somma $t1+$t2 e mette il risultato in $t0
```

Per questo motivo

LOAD: il dato è presente in una locazione di memoria è copiato in un registro

STORE: il valore di un registro è memorizzato in una locazione di memoria



Istruzioni di spostamento

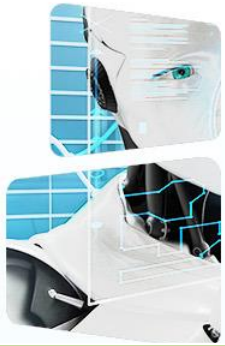
lb rdest, address	*Copia un byte sito all'indirizzo address nel registro rdest*
lbu rdest, address	*Copia un unsigned-byte sito all'indirizzo address nel registro rdest*
lh rdest, address	*Copia un halfword sito all'indirizzo address nel registro rdest*
lhu rdest, address	*Copia un unsigned-halfword sito all'indirizzo address nel registro rdest*
lw rdest, address	*Copia una word sita all'indirizzo address nel registro rdest*
la rdest, address	*Copia un indirizzo address nel registro rdest*
sb rsource, address	*Memorizza un byte all'indirizzo address prelevandolo dal registro rsource*
sh rsource, address	*Memorizza un halfword all'indirizzo address prelevandolo dal registro rsource*
sw rsource, address	*Memorizza una word all'indirizzo address prelevandola dal registro rsource*

Istruzione LOAD

Esempio

Indirizzo in memoria della variabile valore (262160)	Contenuto (1.437.364.264)
00000000 00000100 00000000 00010000	01010101 10101100 01110000 00101000

lb rdest, address	lb $t0,valore	00000000 00000000 00000000 00101000
lh rdest, address	lh $t0,valore	00000000 00000000 01110000 00101000
lw rdest, address	lw $t0,valore	01010101 10101100 01110000 10101000
la rdest, address	la $t0,valore	00000000 00000100 00000000 00010000



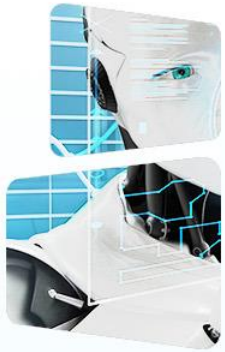
Istruzione STORE

Esempio

Registro	Contenuto
$t0	11010001 10000001 11010101 10101010

sb rsource, address	sb $t0,variabile1
sh rsource, address	sh $t0,variabile2
sw rsource, address	sw $t0,variabile3

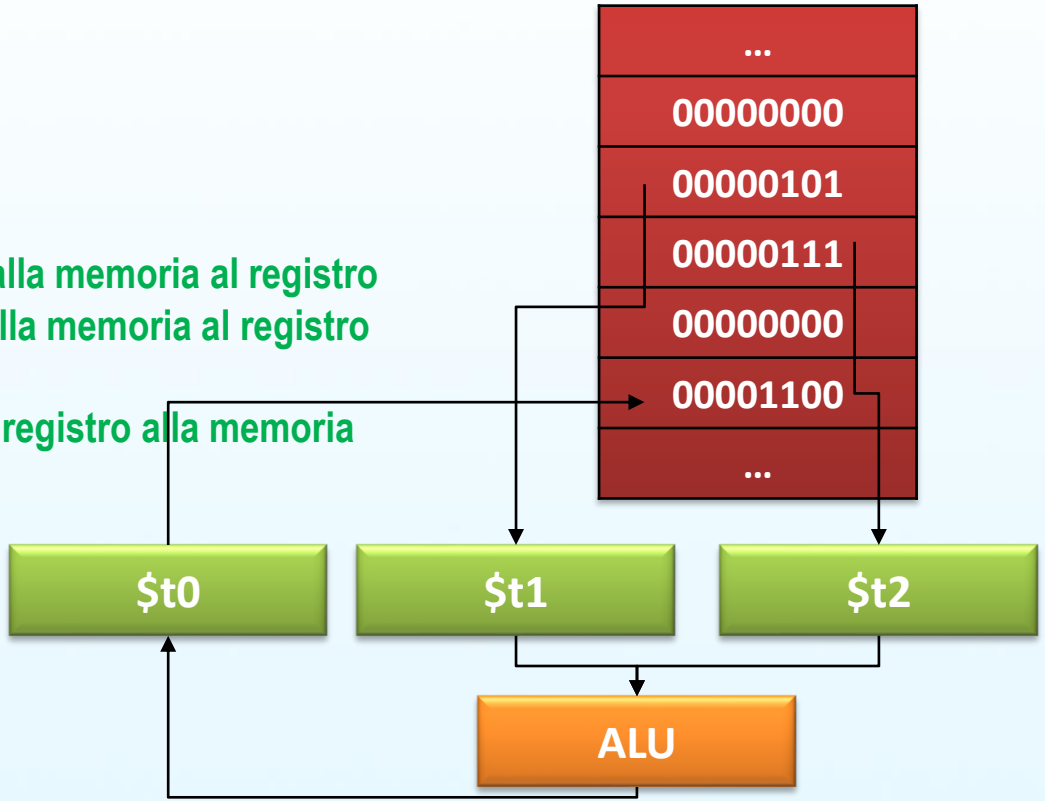
Indirizzo	Contenuto
variabile1	00000000 00000000 00000000 10101010
variabile2	00000000 00000000 11010101 10101010
variabile3	11010001 10000001 11010101 10101010

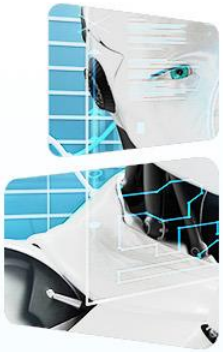


Istruzioni LOAD-STORE

Esempio

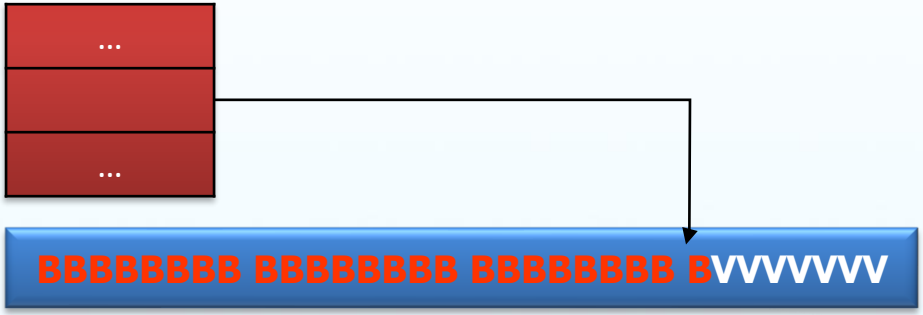
```
.text
.globl main
main:
        lb $t1, operandA  #copia l'operando dalla memoria al registro
        lb $t2, operandB #copia l'operando dalla memoria al registro
        add $t0, $t1, $t2 #somma gli operandi
        sb $t0, risultato #copia l'operando dal registro alla memoria
.end

.data
operandA: .byte 5
operandB: .byte 7
risultato: .byte 0
```



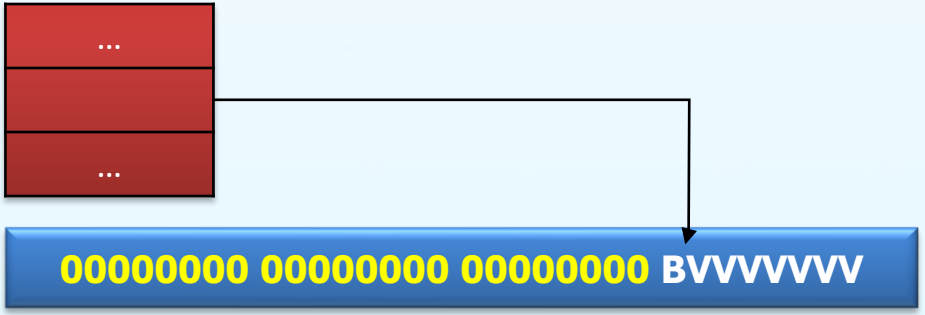


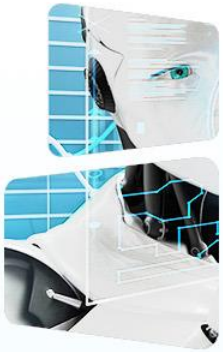
Differenza tra SIGNED/UNSIGNED LB

❑**lb $t0, address**
Copia l'operando di un byte presente nell'indirizzo address *nel byte meno significativo del registro*
Il bit del segno viene esteso



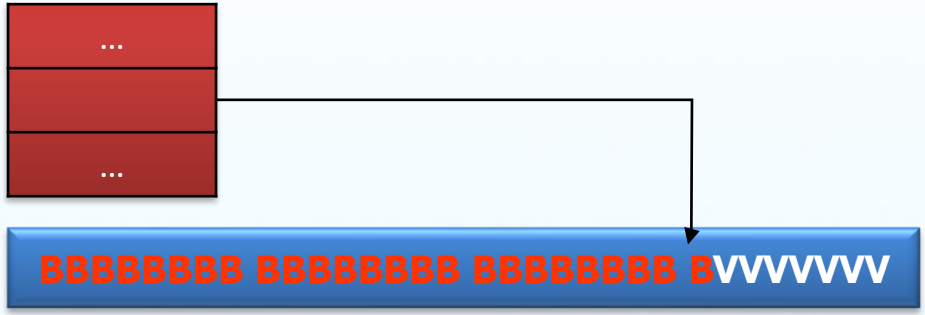
❑**lbu $t0, address**
Copia l'operando di un byte presente nell'indirizzo address *nel byte meno significativo del registro*
Gli altri tre byte sono posti a zero



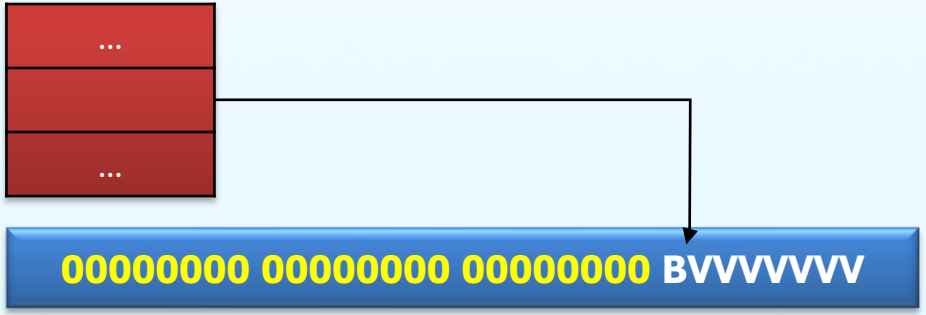


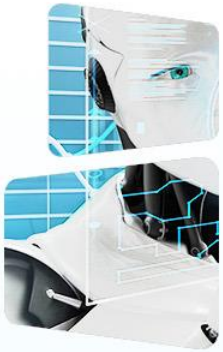
Differenza tra SIGNED/UNSIGNED LH

❑**lh $t0, address**
Copia l'operando di due byte presenti a partire dall'indirizzo address *nei due byte meno significativi del registro*
Il bit del segno viene esteso



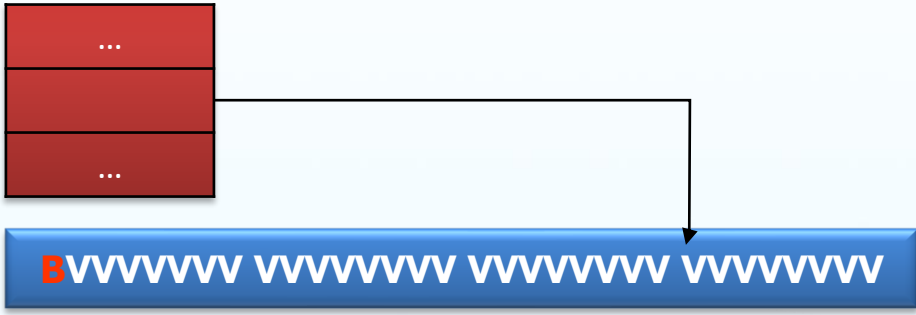
❑**lhu $t0, address**
Copia l'operando di due byte presenti a partire dall'indirizzo address *nei due byte meno significativi del registro*
Gli altri tre byte vengono posti a zero

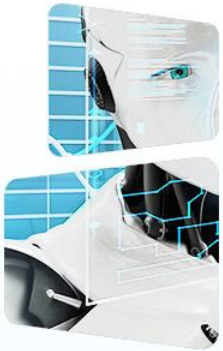




Differenza tra SIGNED/UNSIGNED LW

❑**lw $t0, address**
Copia l'operando di quattro byte presente a partire dall'indirizzo address *nel registro*
Non c'è alcuna estensione del bit del segno

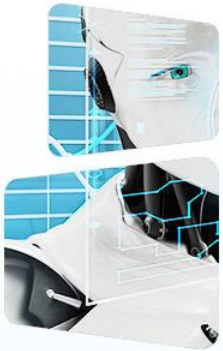




Acquisizione indirizzo etichetta

- L'istruzione **lw $t0, address** copia **il contenuto della word** indirizzata dall'etichetta address nel registro t0
- L'istruzione **la $t0, address** acquisisce **l'indirizzo indicato** dall'etichetta address nel registro t0 (utile per la gestione di strutture dati come i vettori e le stringhe)

```
.text
lw  $t0,val          #In $t0 si trova il dato numerico 250
la  $t1,val          #In $t1 si trova l'indirizzo di memoria dove risiede il valore

.data
val:   .byte 250
```



Acquisizione indirizzo etichetta

Esempio

- L'istruzione **lw $t0, address** copia **il contenuto della word** indirizzata dall'etichetta address nel registro t0
- L'istruzione **la $t0, address** acquisisce **l'indirizzo indicato** dall'etichetta address nel registro t0

(utile per la gestione di strutture dati come i vettori e le stringhe)

lw $t0, address

0x00400030	12
0x00400031	34
0x00400032	56
0x00400033	78

$t0=0x78563412

00101110 00111000 00100010 00001100

la $t0, address

0x00400030	12
0x00400031	34
0x00400032	56
0x00400033	78

$t0=0x00400030

00000000 00101000 00000000 00011110

Inizializzazione di un registro con un valore

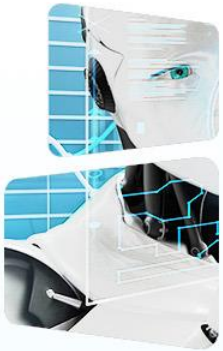
(Indirizzamento immediato)

- Per copiare un operando direttamente in un registro si utilizza l'operazione **load immediate**
- Si evita di definire una valore in memoria il valore numerico è nella stessa istruzione
- Il valore numerico è anche detto ***IMMEDIATE VALUE***
- Le istruzioni sono più lunghe

li rdest, imm	***Sposta il valore imm presente nell'istruzione nel registro rdest***
lui rdest, imm	***Muove la halfword imm presente nell'istruzione nella parte più alta dell'halfword del registro rdest***

Esempio

```
li $t0, 234          #Mette nel registro $t0 il valore 234
```



Spostamento tra registri

(indirizzamento a registro)

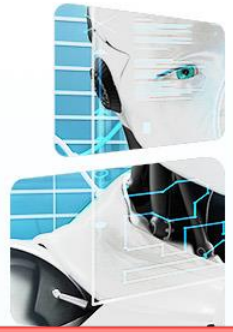
- La maggioranza delle istruzioni di MIPS operano tramite i registri interni al processore

- Per copiare i valori tra registri si utilizzano le operazioni di **data movement**

move rdest, rsource	*Muove il registro rsource nel registro rdest*
mfhi rdest	*Muove il registro hi nel registro rdest*
mflo rdest	*Muove il registro lo nel registro rdest*
mthi rsource	*Muove il registro rsource nel registro hi*
mtlo rsource	*Muove il registro rsource nel registro lo*

Esempio

```
move $t0, $t1        #Copia il contenuto del registro $t1 in $t0
```

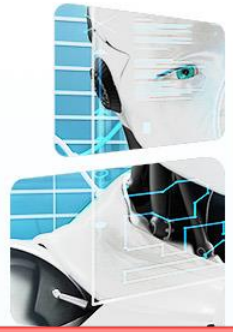


Spostamento tra registri

Esempio

Scambio dei valori contenuti in due registri

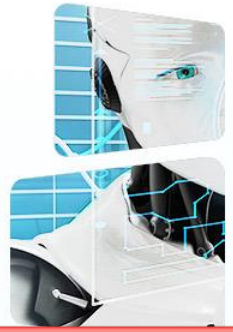
Prima: $t0=5 e $t1=3 Dopo: $t0=3 e $t1=5



Spostamento tra locazioni di memoria

Esempio

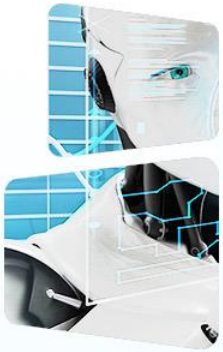
Definiti due valori in memoria, Diabolik e Ginko, scambiare la loro posizione



Spostamento tra locazioni di memoria

Esempio (variante)

Definiti due valori in memoria, Diabolik e Ginko, scambiare la loro posizione

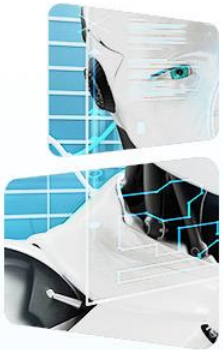


Istruzione di Spostamento

Relazione con i Condition Code

Codice	C	N	Z	W	P
LOAD	X	X	X	X	X
MOVE	X	X	X	X	X
STORE	X	X	X	X	X

ISTRUZIONI LOGICHE-ARITMETICHE



Istruzioni Logiche Artimetiche

Preludio

❑Gli elaboratori elettronici hanno come principale motivo di esistenza la possibilità di svolgere un gran numero di operazioni in tempi rapidissimi

Esempio

```
add $t2, $t0, $t1    #Somma il contenuto di $t1e $t0 e lo mette in $t2
xor $t2,$t2,$t2      #Inizializza il registro $t2 a zero
```

NB: Nel MARS le istruzioni aritmetiche operano su operandi siti nei registri.
I registri sono usati per tre motivi principali:

- ❖Costruiti con ottima tecnologia
- ❖Sono interni alla CPU (non richiedono accessi in memoria)
- ❖Consentono di trovare i dati su cui operare utilizzando pochi bit (RISC)

Istruzioni Artimetiche

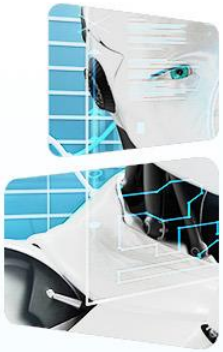
Istruzioni: somma, sottrazione, opposto

add rd, rs, rt	rd = rs + rt (con overflow)
addu rd, rs, rt	rd = rs + rt (senza overflow)
addi rd, rs, imm	rd = rs + imm (con overflow)
addiu rd, rs, imm	rd = rs + imm (senza overflow)
sub rd, rs, rt	rd = rs - rt (con overflow)
subu rd, rs, rt	rd = rs - rt (senza overflow)
neg rd, rs	rd = - rs (con overflow)
negu rd, rs	rd = - rs (senza overflow)
abs rd, rs	rd = |rs|

Istruzioni Artimetiche

Istruzioni: moltiplicazione, divisione, resto

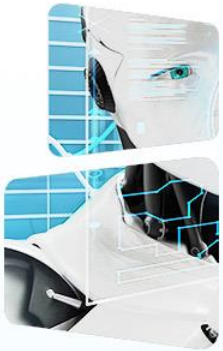
mult rs, rt	*hi,lo = rs * rt (signed, overflow impossibile)*
multu rs, rt	*hi,lo = rs * rt (unsigned, overflow impossibile)*
mul rd, rs, rt	*rd = rs * rt (senza overflow)*
mulo rd, rs, rt	*rd = rs * rt (con overflow)*
mulou rd, rs, rt	*rd = rs * rt (con overflow, unsigned))*
div rs, rt	*hi,lo = resto e quoz. di rs / rt (signed con overflow)*
divu rs, rt	*hi,lo = resto e quoz. di rs / rt (unsigned, no overflow)*
div rd, rs, rt	*rd = rs / rt (signed con overflow)*
divu rd, rs, rt	*rd = rs / rt (unsigned)*
rem rd, rs, rt	*rd = resto di rs / rt (signed)*
remu rd, rs, rt	*rd = resto di rs / rt (unsigned)*



Istruzioni Logiche

Lista istruzioni: and, or, xor, not

and rd, rs, rt	*rd = rs AND rt*
andi rd, rs, imm	*rd = rs AND imm*
or rd, rs, rt	*rd = rs OR rt*
ori rd, rs, imm	*rd = rs OR imm*
xor rd, rs, rt	*rd = rs XOR rt*
xori rd, rs, imm	*rd = rs XOR imm*
nor rd, rs, rt	*rd = NOT (rs OR rt)*
not rd, rs	*rd = NOT rs*

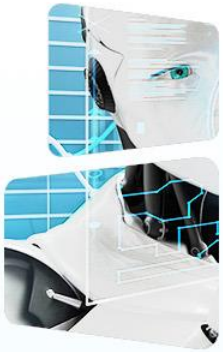


Istruzioni Logiche-Artimentiche

Shift e Rotate

- Lo **Shift** (spostamento logico-aritmentico) sposta *n* bit verso destra/sinistra e perde i bit dalla parte della movimentazione; mentre è colmato con 0 nella direzione opposta
- Il **Rotate** (rotazione o spostamento logico) sposta *n* bit verso destra/sinistra e mantiene i bit dalla parte della movimentazione riproponendo i bit spostati nella direzione opposta

sll rd, rs, rt	*rd = rs è shiftato verso sinistra di rt mod 32 bits*
srl rd, rs, rt	*rd = rs s è shiftato verso destra di rt mod 32 bits*
rol rd, rs, rt	*rd = rs è ruotato verso sinistra di rt mod 32 bits*
ror rd, rs, rt	*rd = rs è ruotato verso destra di rt mod 32 bits*



Istruzioni Logiche-Artimentiche

Shift e Rotate - esempio

sll $t2, $t1,3

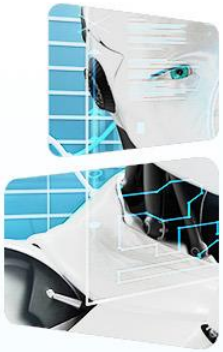
$t1	**00000000000000000000000010011101**	157
$t2	**00000000000000000000010011101000**	1256

srl $t2, $t1,3

$t1	**00000000000000000000000000010000**	16
$t2	**00000000000000000000000000000010**	2

ror $t2, $t1,3

$t1	**00000000000000000000000010011101**	157
$t2	**10100000000000000000000000010011**	2684354579



Istruzioni Logiche-Artimentiche

Immediate e Unsigned

Versioni immediate (i)

❑Le versioni immediate (i) delle istruzioni precedenti utilizzano un valore immediato al posto di uno degli operandi

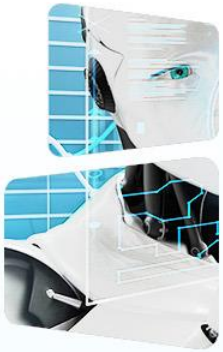
❑Non sempre è necessario specificare la **i**; c'è un riconoscimento implicito da parte dell'assemblatore

Esempio: **add $t0, $t0, 1**

Versioni unsigned (u)

❑ Le versioni unsigned delle istruzioni non gestiscono le problematiche relative all'overflow

❑ Dettagli in seguito quando si parlerà delle eccezioni



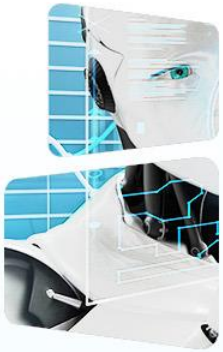
Istruzioni Logiche-Artimentiche

Esempio I

Realizzare un programma che in linguaggio SPIM prenda un valore residente in memoria (di tipo byte) e metta 0 o 1 se il numero è rispettivamente pari o dispari

```
.text
.globl main
main:
        lb $t0,pippo
        li $t1,2
        rem $t2,$t0,$t1
        sb  $t2,risb
li $v0,10
syscall
```

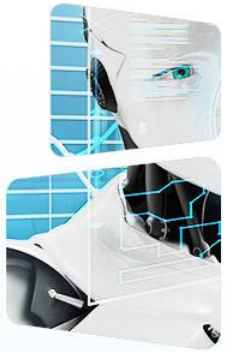
```
.data
pippo: .byte 100
risb: .byte 0
```



Istruzioni Logiche-Artimentiche

Esempio II

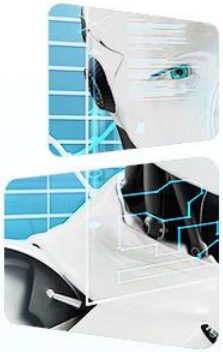
Si scriva un programma in linguaggio assembly che definito un valore intero in memoria riporti in $t0 l'intero precedente, in $t1 il numero corrente e in $t2 il successivo



Istruzioni Logiche-Artimentiche

Esempio III

Si scriva un programma in linguaggio assembly che definiti due numeri in memoria riporta la loro media (fra interi) in $t2 Esempio: Batman=5 Robin=8 **media**=(5+8)/2=**6**



Istruzioni Logiche-Artimentiche

Esempio IV

Scrivere un programma che definito in memoria il prezzo al dettaglio e lo sconto e riporta in $t2 il prezzo scontato (arrotondato all'intero)

Istruzioni Logiche-Aritmetiche

Relazione con i Condition Code

Codice	C	N	Z	W	P
ADD	1/0	1/0	1/0	1/0	1/0
CMP	1/0	1/0	1/0	1/0	1/0
NEG	1/0	1/0	1/0	1/0	1/0
SUB	1/0	1/0	1/0	1/0	1/0
AND	0	1/0	1/0	0	1/0
OR	0	1/0	1/0	0	1/0
XOR	0	1/0	1/0	0	1/0
NOT	0	1/0	1/0	0	1/0
SL	1/0	1/0	1/0	1/0	1/0
SR	1/0	1/0	1/0	1/0	1/0
ROL	1/0	0	0	0	1/0
ROR	1/0	0	0	0	1/0

ISTRUZIONI DI CONFRONTO

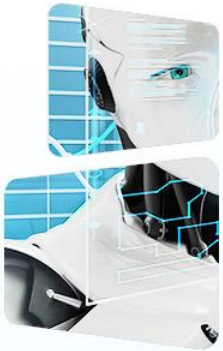
Istruzioni di confronto

Set less than

- Le istruzioni di confronto sono di solito usate dall'assemblatore e sono usate per creare pseduo-istruzioni utili ai programmatori per i salti condizionati

slt rd, rs, rt	***Setta il registro rd a 1 se rs < rt, 0 altrimenti (con overflow)***
sltu rd, rs, rt	***Setta il registro rd a 1 se rs < rt, 0 altrimenti (no overflow)***
slti rd, rs, imm	***Setta registro rd a 1 se rs < imm, 0 altrimenti (con overflow)***
sltiu rd, rs, imm	***Setta registro rd a 1 se rs < imm, 0 altrimenti (no overflow)***
sle, sgt, sge, seq, sne	***Analoghe per testare <=, >, >=, = e <>***

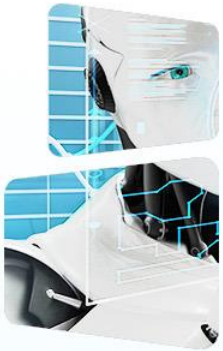
ISTRUZIONI DI SALTO CONDIZIONATO



Istruzioni di salto condizionato

Introduzione

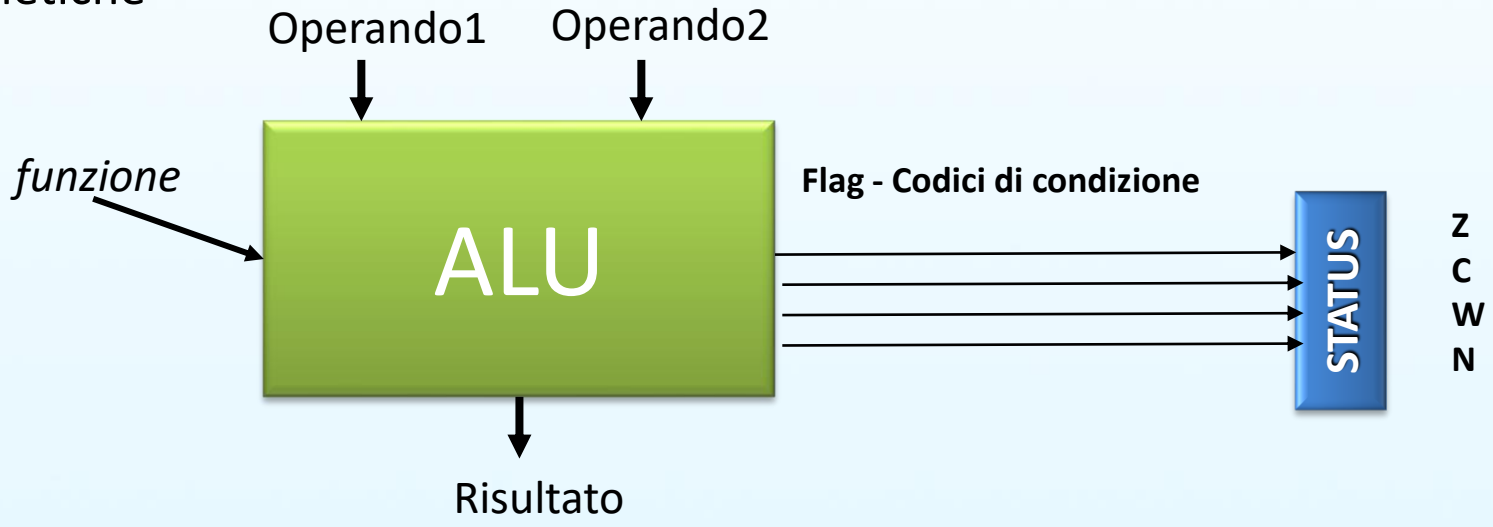
- Le **istruzioni di salto condizionato** considerano uno/due operandi e in relazione al verificarsi di una condizione di verità derivata da un confronto effettuano un salto ad un'altra locazione del programma

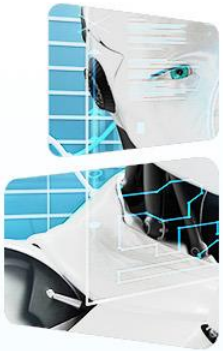


Istruzioni di salto condizionato

Introduzione

- Le istruzioni di salto condizionato fanno riferimento a i *flag* o codici di condizione presenti nello STATUS REGISTER derivanti da operazione logico-aritmetiche





Istruzioni di salto condizionato

Introduzione

- Quando si verifica una condizione c'è un **salto** all'indirizzo interno al programma
- Tecnicamente il salto è una sovrascrittura del program counter con l'indirizzo mascherato dall'etichetta

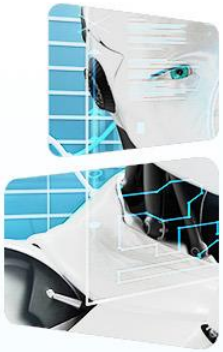
```
100        li $t0,56
101        li $t1,122
102        bgt $t1,$t0,salto
103        mul $t9,$t0,$t1
104 salto: li $t9,0
```

PC: 101

PC: 102

PC: 104

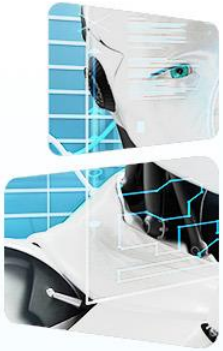
PC: 105



Istruzioni di salto condizionato

Set

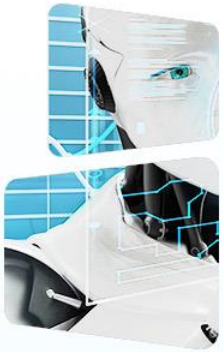
beq rs, rt, target	*Salta all'istruzione puntata da target se rs = rt*
bne rs, rt, target	*Salta all'istruzione puntata da target se rs != rt*
bge rs, rt, target bgeu rs, rt, target	*Salta all'istruzione puntata da target se rs >= rt* *Comparazione senza considerare il segno*
bgt rs, rt, target bgtu rs, rt, target	*Salta all'istruzione puntata da target se rs > rt* *Comparazione senza considerare il segno*
ble rs, rt, target bleu rs, rt, target	*Salta all'istruzione puntata da target se rs <= rt* *Comparazione senza considerare il segno*
blt rs, rt, target bltu rs, rt, target	*Salta all'istruzione puntata da target se rs < rt* *Comparazione senza considerare il segno*



Istruzioni di salto condizionato

Set con riferimento al registro $ZERO

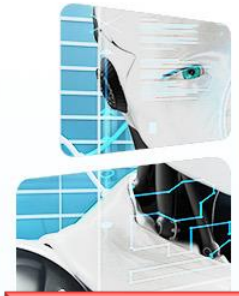
bgez rs, target	*Salta all'istruzione puntata da target se rs >= 0*
bgtz rs, target	*Salta all'istruzione puntata da target se rs > 0*
blez rs, target	*Salta all'istruzione puntata da target se rs <= 0*
bltz rs, target	*Salta all'istruzione puntata da target se rs < 0*
beqz rs, target	*Salta all'istruzione puntata da target se rs = 0*
bnez rs, target	*Salta all'istruzione puntata da target se rs != 0*



Istruzioni di salto condizionato

Analisi dei condition code

Mnemonico	Significato	Flag interessato
EQ	=	Z=1
NE	!=	Z=0
CS	> (unsigned)	C=1
	< (unsigned)	C=0
HI	>	C=1 e Z=0
LS	<	C=0 e Z=1
MI	negativo	N=1
PL	positivo	N=0

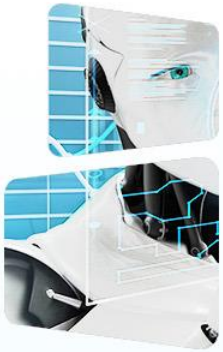


Istruzioni di salto condizionato

Esempio I

Realizzare un programma che valuta il massimo fra due numeri x,y (di tipo byte) definiti in memoria. Riportare il risultato in memoria

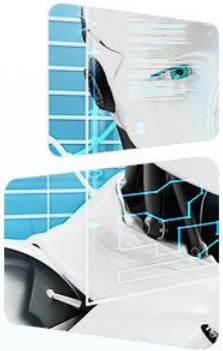
ISTRUZIONI DI SALTO INCONDIZIONATO



Istruzioni di salto incondizionato

Introduzione

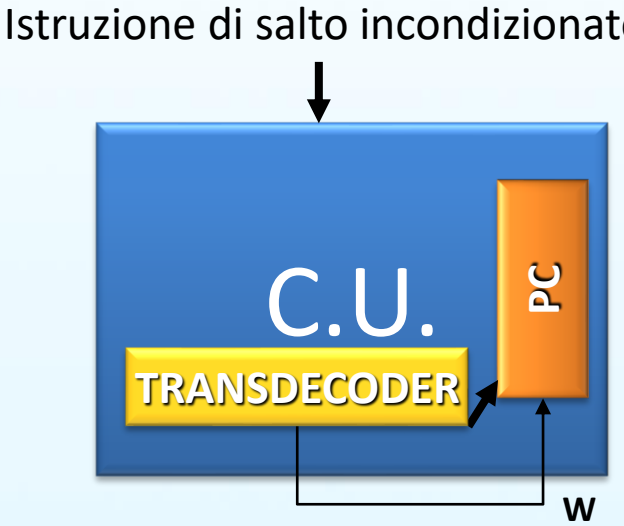
- Le istruzioni di salto incondizionato effettuano un salto ad un'altra locazione del programma senza valutare la veridicità di alcuna condizione

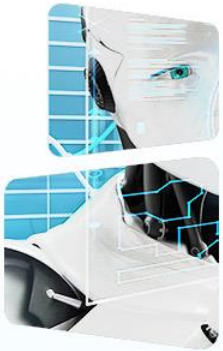


Istruzioni di salto incondizionato

Introduzione

- Le istruzioni di **salto incondizionato** intervengo direttamente mutando il valore del PC



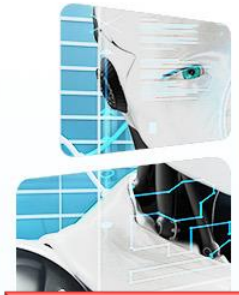


Istruzioni di salto incondizionato

Introduzione

- Le istruzioni di **salto incondizionato** sono utili per realizzare cicli e strutture iterative

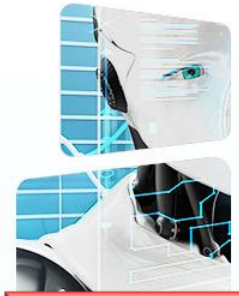
j target	*Salto incondizionato all'istruzione puntata da target*



Istruzioni di salto

Esempio I

Realizzare un contatore da 0 a 100

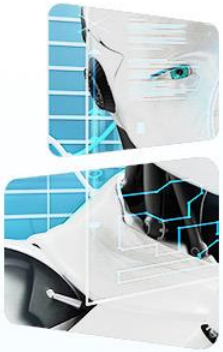


Istruzioni di salto

Esempio II

Realizzare un programma che in linguaggio SPIM definisca un valore di tipo byte residente in memoria e metta in $t2 il valore 1 o 2 se il numero è rispettivamente multiplo di 7 o no

PROGETTAZIONE DI UN APPLICATIVO INFORMATICO



Progettazione Applicativo Informatico

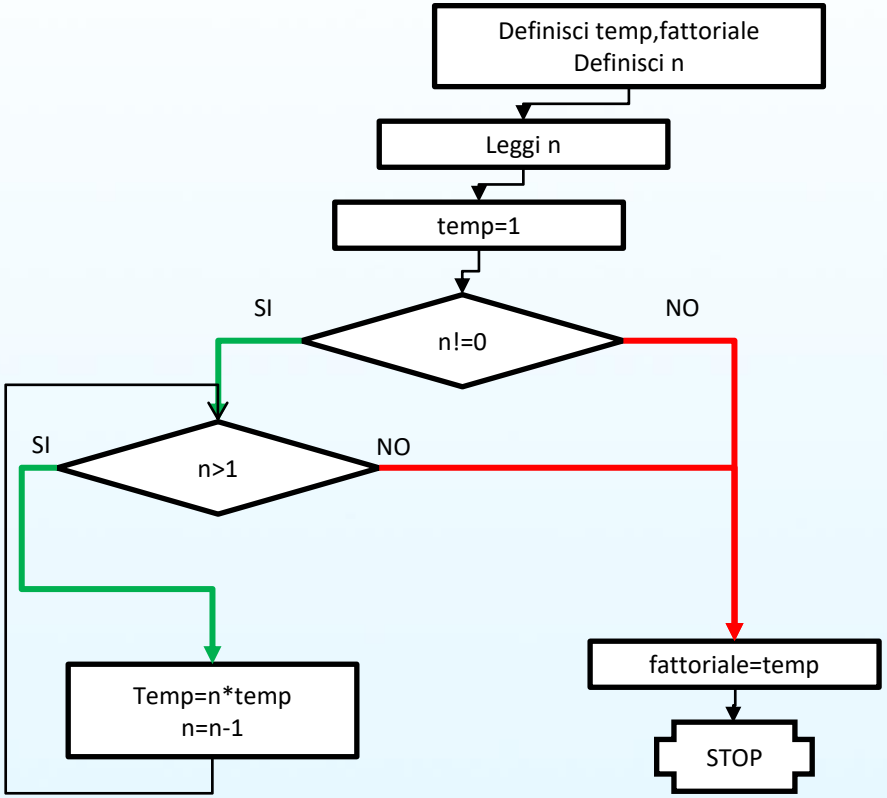
Implementazione diagramma di flusso

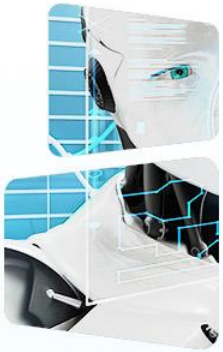
Effettuare il fattoriale di un numero n ($n \geq 0$ e $n \in \mathbb{N}$)

Il fattoriale di un numero N è definito come:

$N! = N \bullet N-1 \bullet N-2 \bullet \ldots \bullet 3 \bullet 2 \bullet 1$

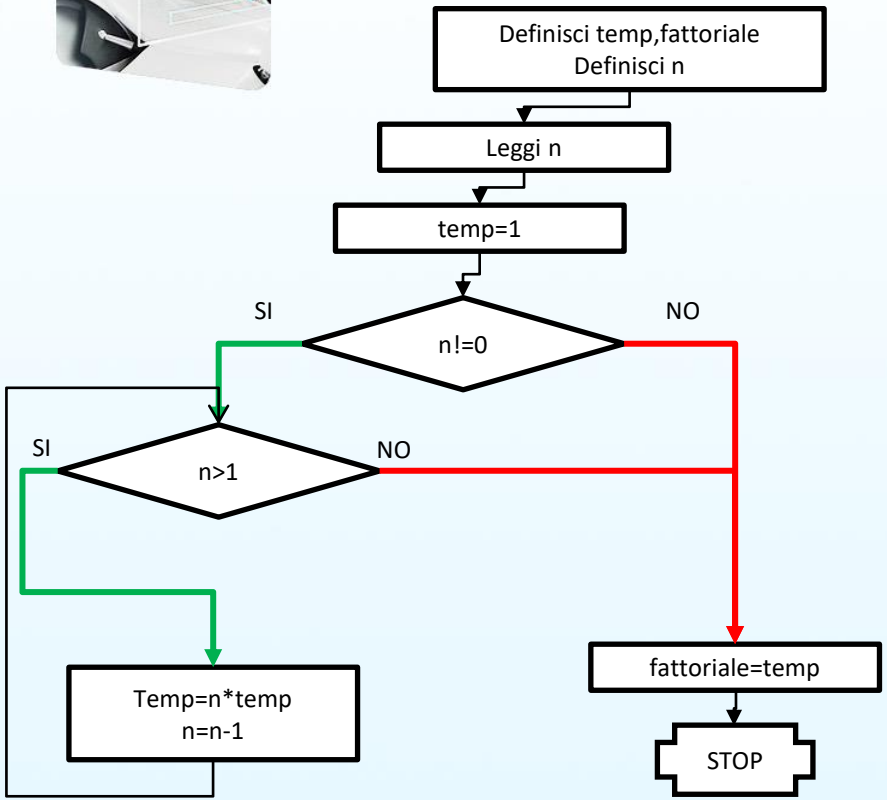
NB: 0!=1



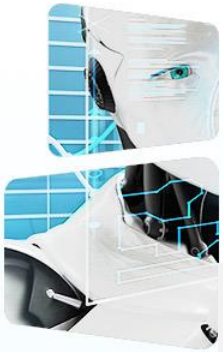


Progettazione Applicativo Informatico

Scrittura linguaggio alto livello




```
main()
{
int n,temp, fattoriale;
temp=1;
if (n!=0){
            while (n>1)
                        {
                        temp=temp*n;
                        n=n-1;
                        }
            }
fattoriale=temp;
}
```



Progettazione Applicativo Informatico

Scrittura linguaggio assemblativo

```
main()
{
int n,temp, fattoriale;
temp=1;
if (n!=0){
        while (n>1)
                {
                temp=temp*n;
                n=n-1;
                }
        }
fattoriale=temp;
}
```

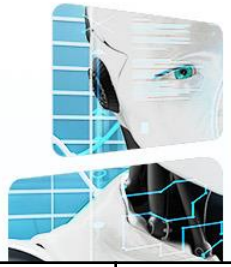
```
.text
        lw $t0,n
        lw $t1,temp
        li $t5,1
        beqz $t0, fine
ciclo:
        ble $t0,$t5, fine
        mul $t1,$t1,$t0
        sub $t0,$t0,1
        j ciclo
fine:
        sw $t1, fattoriale
.data
fattoriale: .word 0
temp: .word 1
n: .word 35
```

Progettazione Applicativo Informatico

```
.text
            lw $t0,n
            lw $t1,temp
            li $t5,1
            beqz $t0, fine
ciclo:
            ble $t0,$t5, fine
            mul $t1,$t1,$t0
            sub $t0,$t0,1
            j ciclo
fine:
            sw $t1, fattoriale
.data
fattoriale: .word 0
temp: .word 1
n: .word 35
```

Assemblatore

Indirizzo	Etichetta	Istruzione
400		lw $t0,n
404		lw $t1,temp
408		li $t5,1
412		beqz $t0,fine
416	ciclo	ble $t0,$t5,fine
420		mul $t1,$t1,$t0
424		sub $t0, $t0,1
428		j ciclo
432	fine:	sw $t1,fattoriale
10000	fattoriale	0
10004	temp	1
10008	n	35



Progettazione Applicativo Informatico

Assemblatore: risoluzione etichette – tabella simboli

Indirizzo	Etichetta	Istruzione
400		lw $t0,n
404		lw $t1,temp
408		li $t5,1
412		beqz $t0,fine
416	ciclo:	ble $t0,$t5,fine
420		mul $t1,$t1,$t0
424		sub $t0, $t0,1
428		j ciclo
432	fine:	sw $t1,fattoriale
10000	fattoriale	0
10004	temp	1
10008	n	35

Tempo	TABELLA DEI SIMBOLI	
	ETICHETTA	VALORE
T1	n	?
T2	Temp	?
T3	fine	?
T4	ciclo	416
T5	fine	432
T6	fattoriale	?
T7	fattoriale	10000
T8	temp	10004
T9	n	10008

Indirizzo	Etichetta	Istruzione
400		lw $t0,n
404		lw $t1,temp
408		li $t5,1
412		beqz $t0,fine
416	ciclo:	ble $t0,$t5,fine
420		mul $t1,$t1,$t0
424		sub $t0, $t0,1
428		j 416
432	fine:	sw $t1,fattoriale
10000	fattoriale	0
10004	temp	1
10008	n	35

Progettazione Applicativo Informatico

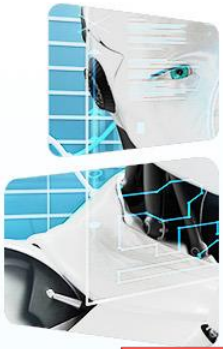
Assemblatore: risoluzione etichette – tabella simboli

Indirizzo	Etichetta	Istruzione
400		lw $t0,n
404		lw $t1,temp
408		li $t5,1
412		beqz $t0,fine
416	ciclo:	ble $t0,$t5,fine
420		mul $t1,$t1,$t0
424		sub $t0, $t0,1
428		j 416
432	fine:	sw $t1,fattoriale
10000	fattoriale	0
10004	temp	1
10008	n	35

Tempo	TABELLA DEI SIMBOLI	
	ETICHETTA	VALORE
T1	n	10008
T2	Temp	10004
T3	fine	432
T4	ciclo	416
T5	fine	432
T6	fattoriale	10000
T7	fattoriale	10000
T8	temp	10004
T9	n	10008

Indirizzo	Etichetta	Istruzione
400		lw $t0,10008
404		lw $t1,10004
408		li $t5,1
412		beqz $t0,432
416	ciclo:	ble $t0,$t5,432
420		mul $t1,$t1,$t0
424		sub $t0, $t0,1
428		j 416
432	fine:	sw $t1,10000
10000	fattoriale	0
10004	temp	1
10008	n	35

ESERCIZIO RIEPILOGATIVO



Esercizio

Individuazione numero primo

Implementare un programma che letto un intero da tastiera stampa su videoterminale un messaggio che indica se il numero è primo o no

NB: un numero è primo se, oltre ad essere divisibile per 1, è divisibile solamente per se stesso ovvero:
$(n,x)=1 \; \forall \; x \in N-\{0\}$ con $x=1 \wedge x=n$ *cioè* $(n,1)=1 \wedge (n,n)=1$

FINE