SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Architettura degli Elaboratori
Lez. 4 – ASM: Vettori e Matrici

Prof. Andrea Sterbini – sterbini@di.uniroma1.it

Argomenti

- **Argomenti della lezione**
  - Vettori: manipolazione con indici e con puntatori
  - Matrici a 2, 3 ed N dimensioni
  - Esempi di programmi

Argomenti

- **Argomenti della lezione**
  - Vettori: manipolazione con indici e con puntatori
  - Matrici a 2, 3 ed N dimensioni
  - Esempi di programmi

- **Vettore:**

  sequenza di **N elementi** di dimensioni uguali

  consecutivi in memoria

  indirizzabili per indice (da 0 a N-1)

  dimensione totale = N * dimensione_elemento

Argomenti

- **Argomenti della lezione**
  - Vettori: manipolazione con indici e con puntatori
  - Matrici a 2, 3 ed N dimensioni
  - Esempi di programmi

- **Vettore:**

  sequenza di **N elementi** di dimensioni uguali

  consecutivi in memoria

  indirizzabili per indice (da 0 a N-1)

  dimensione totale = N * dimensione_elemento

- Si possono definire staticamente nella zona **.data** del programma assembly

usando una **etichetta** per indicare l'**indirizzo del primo elemento** del vettore

Argomenti

- **Argomenti della lezione**
  - Vettori: manipolazione con indici e con puntatori
  - Matrici a 2, 3 ed N dimensioni
  - Esempi di programmi

- **Vettore:**

  sequenza di **N elementi** di dimensioni uguali

  consecutivi in memoria

  indirizzabili per indice (da 0 a N-1)

  dimensione totale = N * dimensione_elemento

- Si possono definire staticamente nella zona **.data** del programma assembly

usando una **etichetta** per indicare l'**indirizzo del primo elemento** del vettore

- Per indirizzare l'**elemento i-esimo** bisogna aggiungere l'offset **i*dimensione_elemento**

Vettori di byte in memoria

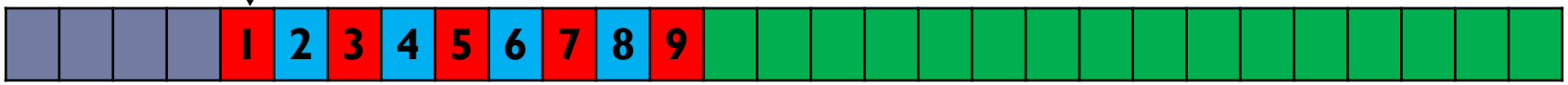
vettore di **byte** (valori interi da 0 a 255)

```
label1:  .byte 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Vettori di byte in memoria

vettore di **byte** (valori interi da 0 a 255)

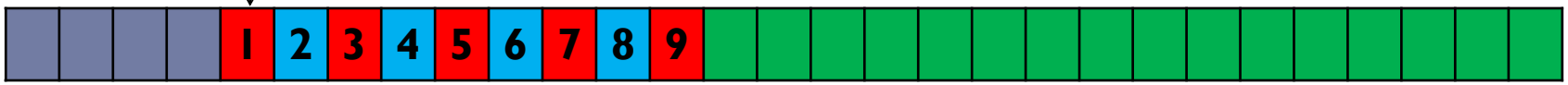
```
label1:  .byte 1, 2, 3, 4, 5, 6, 7, 8, 9
```



Vettori di byte in memoria

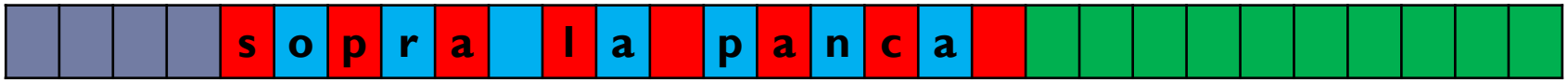
vettore di **byte** (valori interi da 0 a 255)

```
label1:  .byte 1, 2, 3, 4, 5, 6, 7, 8, 9
```



testo: vettore di caratteri (byte) seguiti da 0 (zero)

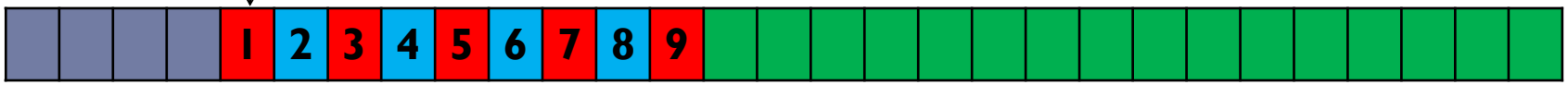
```
label2:  .asciiz  "sopra la panca"
```



Vettori di byte in memoria

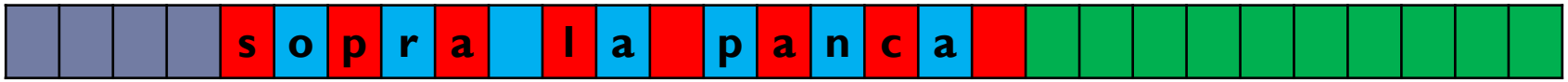
vettore di **byte** (valori interi da 0 a 255)

```
label1:  .byte 1, 2, 3, 4, 5, 6, 7, 8, 9
```

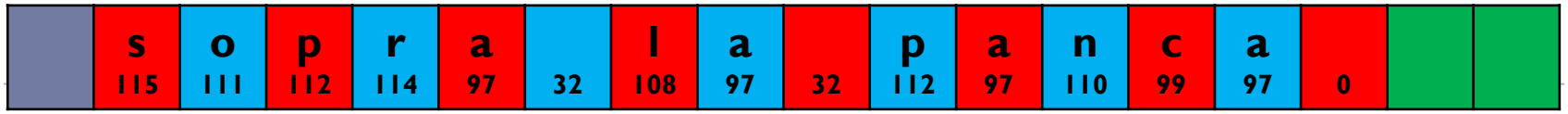


testo: vettore di caratteri (byte) seguiti da 0 (zero)

```
label2:  .asciiz  "sopra la panca"
```



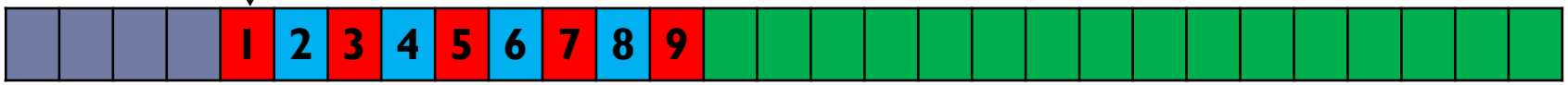
Viene memorizzata come sequenza dei codici ASCII dei caratteri inseriti nella direttiva **.asciiz**



Vettori di byte in memoria

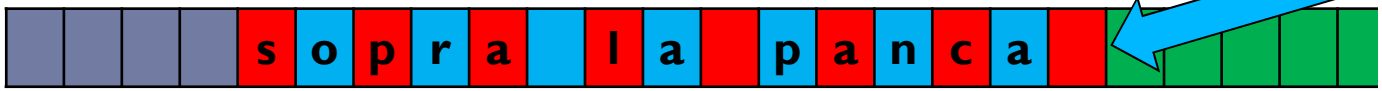
vettore di **byte** (valori interi da 0 a 255)

```
label1:  .byte 1, 2, 3, 4, 5, 6, 7, 8, 9
```



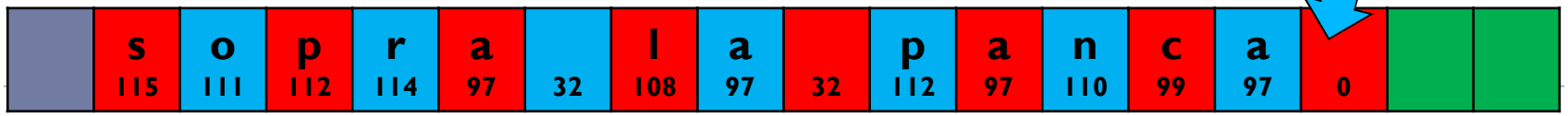
testo: vettore di caratteri (byte) seguiti da 0 (zero)

```
label2:  .asciiz  "sopra la panca"
```



Codice 0 che ne indica la fine

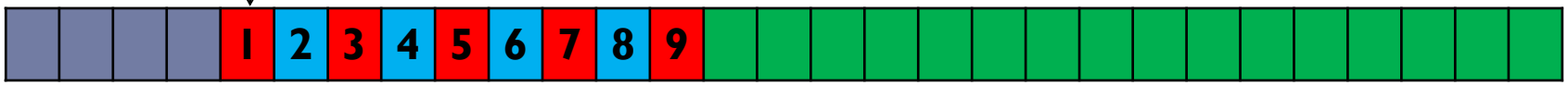
Viene memorizzata come sequenza dei codici ASCII dei caratteri inseriti nella direttiva **.asciiz**



Vettori di byte in memoria

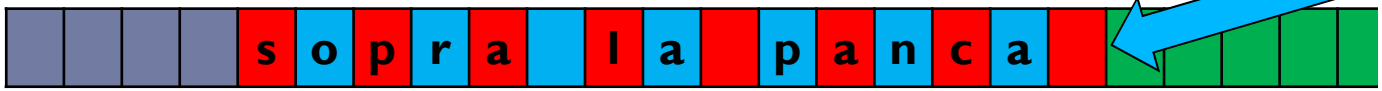
vettore di **byte** (valori interi da 0 a 255)

```
label1:  .byte 1, 2, 3, 4, 5, 6, 7, 8, 9
```



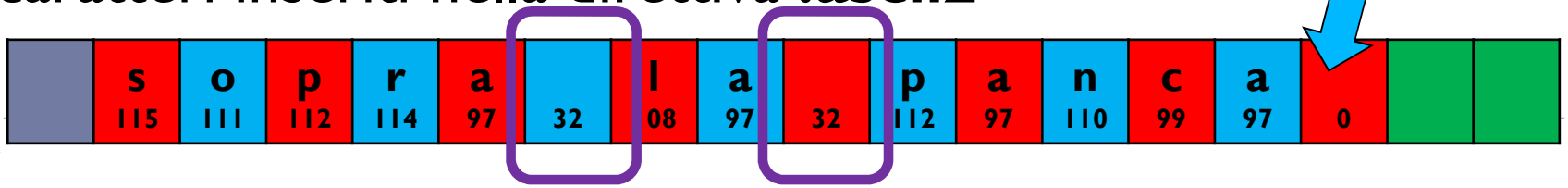
testo: vettore di caratteri (byte) seguiti da 0 (zero)

```
label2:  .asciiz  "sopra la panca"
```



Codice 0 che ne indica la fine

Viene memorizzata come sequenza dei codici ASCII dei caratteri inseriti nella direttiva **.asciiz**



I vettori di word

vettore di **word**, numeri a 32 bit in Ca2 (da -2^31 a +(2^31)-1) codificati in 4 byte

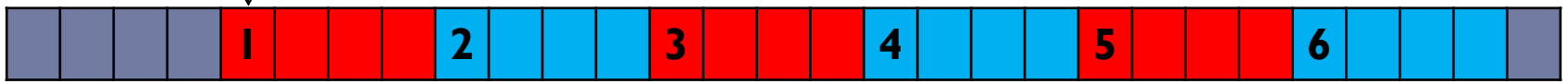
```
label3:  .word    1, 2, 3, 4, 5, 6
```
(6 elementi di 4 byte ciascuno)

I vettori di word

vettore di **word**, numeri a 32 bit in Ca2 (da -2^31 a +(2^31)-1) codificati in 4 byte

```
label3:  .word    1, 2, 3, 4, 5, 6
```

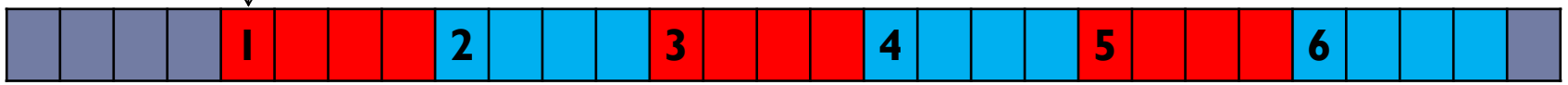
(6 elementi di 4 byte ciascuno)



I vettori di word

vettore di **word**, numeri a 32 bit in Ca2 (da -2^31 a +(2^31)-1) codificati in 4 byte

```
label3:  .word    1, 2, 3, 4, 5, 6
```
(6 elementi di 4 byte ciascuno)

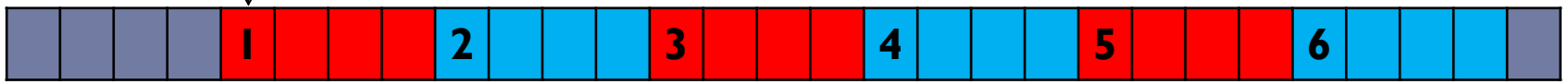



```
label4:  .word    100:0
```
(100 elementi di valore 0)

I vettori di word

vettore di **word**, numeri a 32 bit in Ca2 (da -2^31 a +(2^31)-1) codificati in 4 byte

```
label3:  .word    1, 2, 3, 4, 5, 6
```
(6 elementi di 4 byte ciascuno)




```
label4:  .word    100:0
```
(100 elementi di valore 0)

Il processore MIPS permette l'ordinamento dei byte di una word nei due modi:

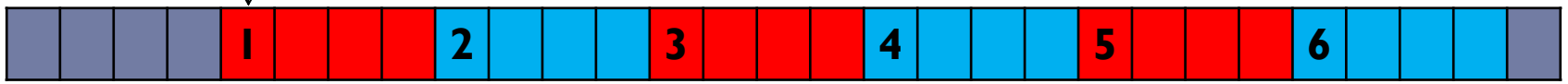
- **Big-endian** (o network-order, usato da Java e dalle CPU SPARC Sun/Oracle)

i byte della word sono memorizzati dal MSB (most significative byte) al LSB (least…)

I vettori di word

vettore di **word**, numeri a 32 bit in Ca2 (da -2^31 a +(2^31)-1) codificati in 4 byte

`label3:` `.word` `1, 2, 3, 4, 5, 6` (6 elementi di 4 byte ciascuno)

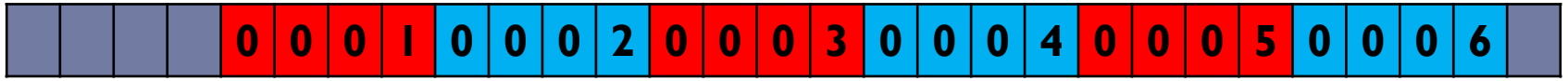


`label4:` `.word` `100:0` (100 elementi di valore 0)

Il processore MIPS permette l'ordinamento dei byte di una word nei due modi:

- **Big-endian** (o network-order, usato da Java e dalle CPU SPARC Sun/Oracle)

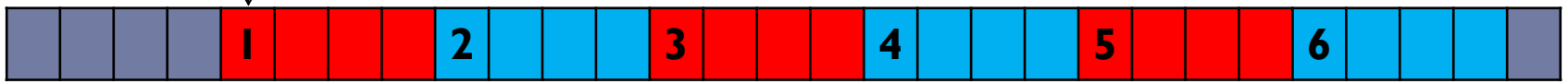
i byte della word sono memorizzati dal MSB (most significative byte) al LSB (least…)



I vettori di word

vettore di **word**, numeri a 32 bit in Ca2 (da -2^31 a +(2^31)-1) codificati in 4 byte

`label3:` `.word` `1, 2, 3, 4, 5, 6` (6 elementi di 4 byte ciascuno)

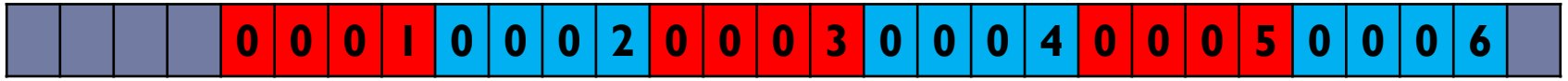


`label4:` `.word` `100:0` (100 elementi di valore 0)

Il processore MIPS permette l'ordinamento dei byte di una word nei due modi:

- **Big-endian** (o network-order, usato da Java e dalle CPU SPARC Sun/Oracle)

i byte della word sono memorizzati dal MSB (most significative byte) al LSB (least…)



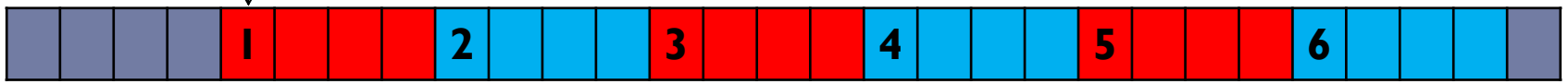
- **Little-endian** (usato dalle CPU Intel, ovvero su OSX, Windows, …, e da **MARS**)

i byte della word sono memorizzati dal LSB al MSB (ovvero a rovescio)

I vettori di word

vettore di **word**, numeri a 32 bit in Ca2 (da -2^31 a +(2^31)-1) codificati in 4 byte

`label3:` `.word` `1, 2, 3, 4, 5, 6` (6 elementi di 4 byte ciascuno)

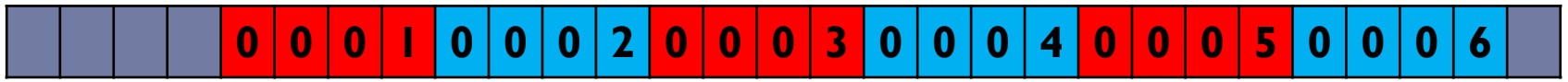


`label4:` `.word` `100:0` (100 elementi di valore 0)

Il processore MIPS permette l'ordinamento dei byte di una word nei due modi:

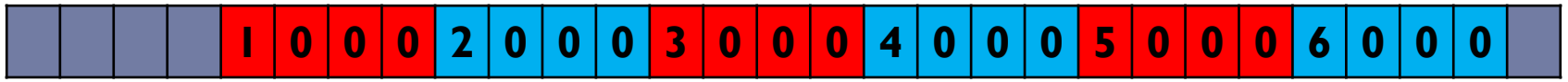
- **Big-endian** (o network-order, usato da Java e dalle CPU SPARC Sun/Oracle)

i byte della word sono memorizzati dal MSB (most significative byte) al LSB (least…)



- **Little-endian** (usato dalle CPU Intel, ovvero su OSX, Windows, …, e da **MARS**)

i byte della word sono memorizzati dal LSB al MSB (ovvero a rovescio)



MARS e i vettori

MARS è little-endian

→ nella finestra dei dati le **stringhe** sono visualizzate come **gruppi di 4 caratteri rovesciati**

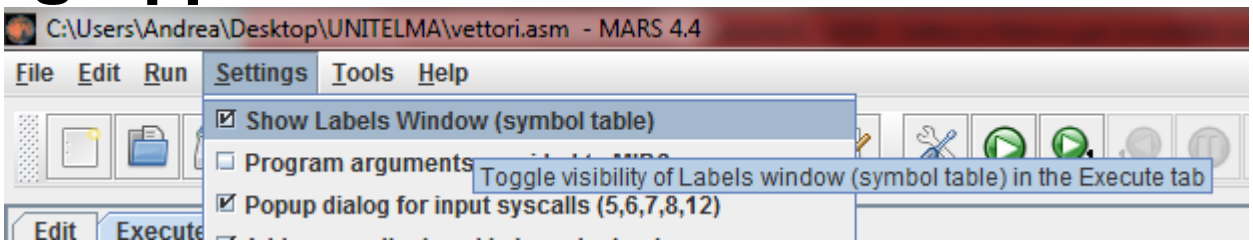
- per il resto non ci sono differenze (a meno che non vogliate accedere ai singoli byte di un dato memorizzato in una word)

```
label1: .asciiz    "sopra la panca"
```

MARS e i vettori

MARS è little-endian

→ nella finestra dei dati le **stringhe** sono visualizzate come **gruppi di 4 caratteri rovesciati**

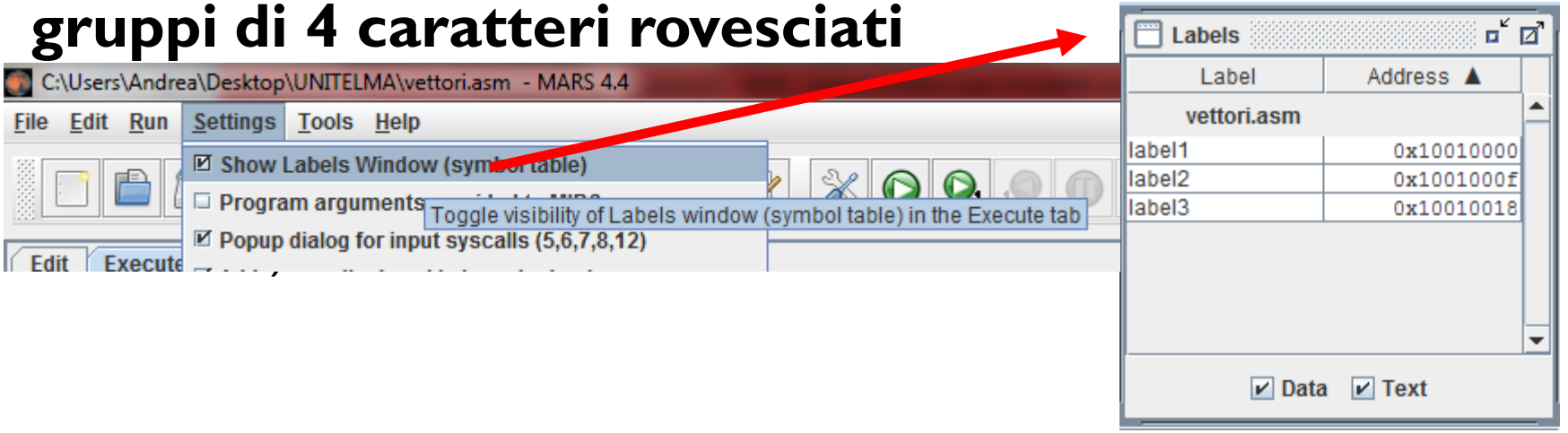



```
label1:  .asciiz    "sopra la panca"
```

MARS e i vettori

MARS è little-endian

→ nella finestra dei dati le **stringhe** sono visualizzate come **gruppi di 4 caratteri rovesciati**

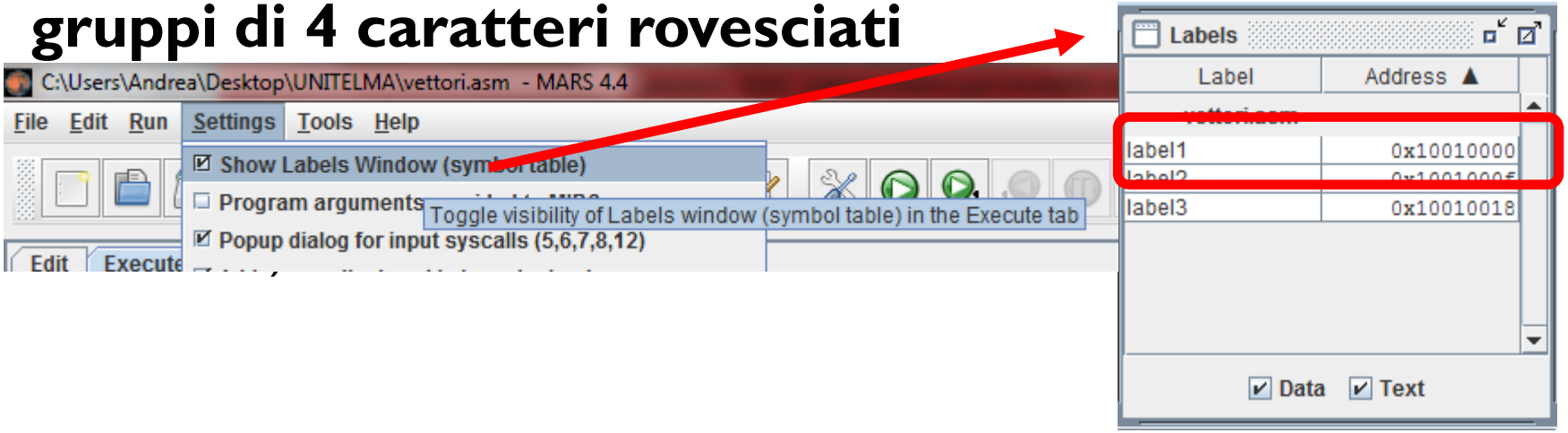



```
label1:  .asciiz    "sopra la panca"
```

MARS e i vettori

MARS è little-endian

→ nella finestra dei dati le **stringhe** sono visualizzate come **gruppi di 4 caratteri rovesciati**

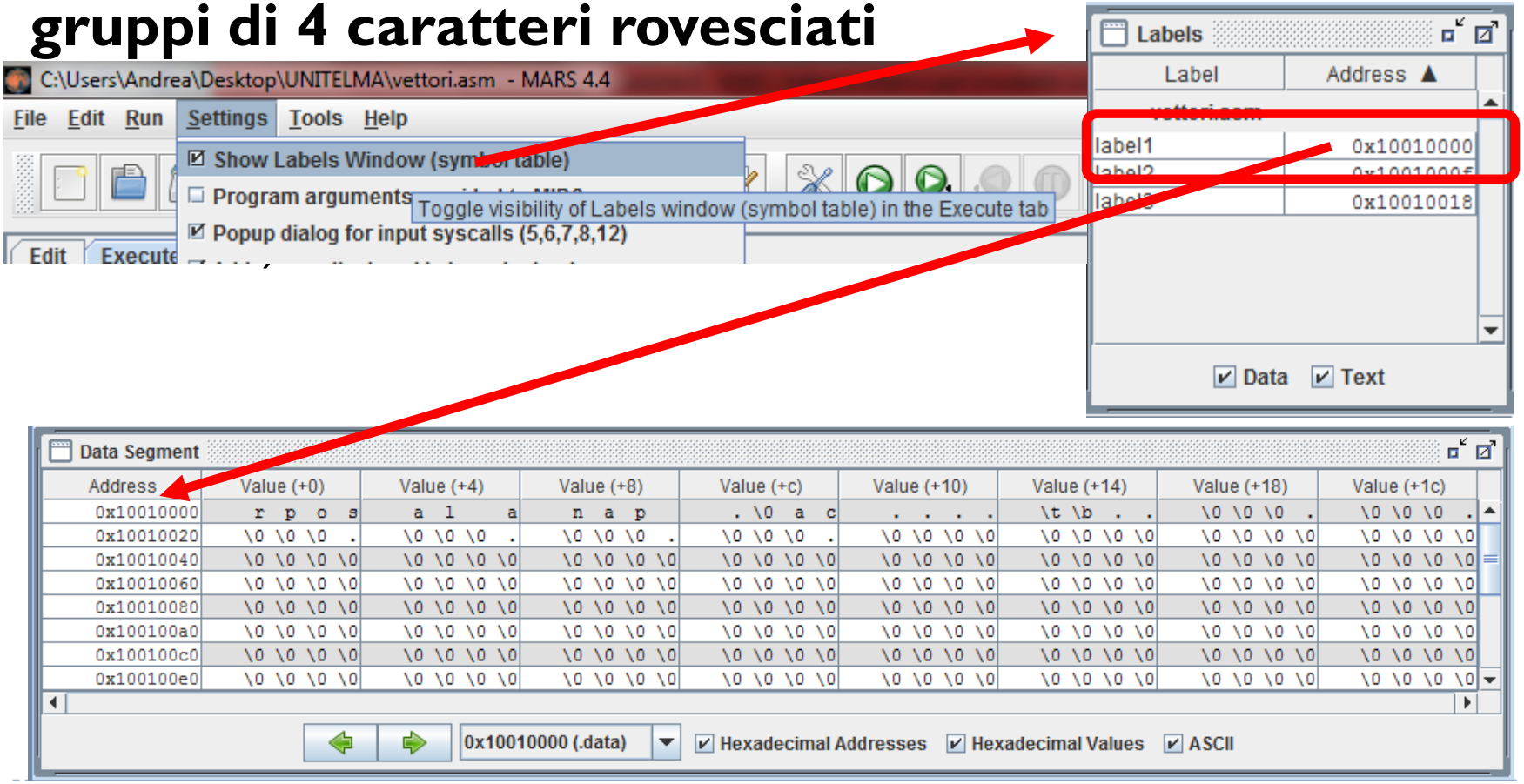



```
label1:  .asciiz    "sopra la panca"
```

MARS e i vettori

MARS è little-endian

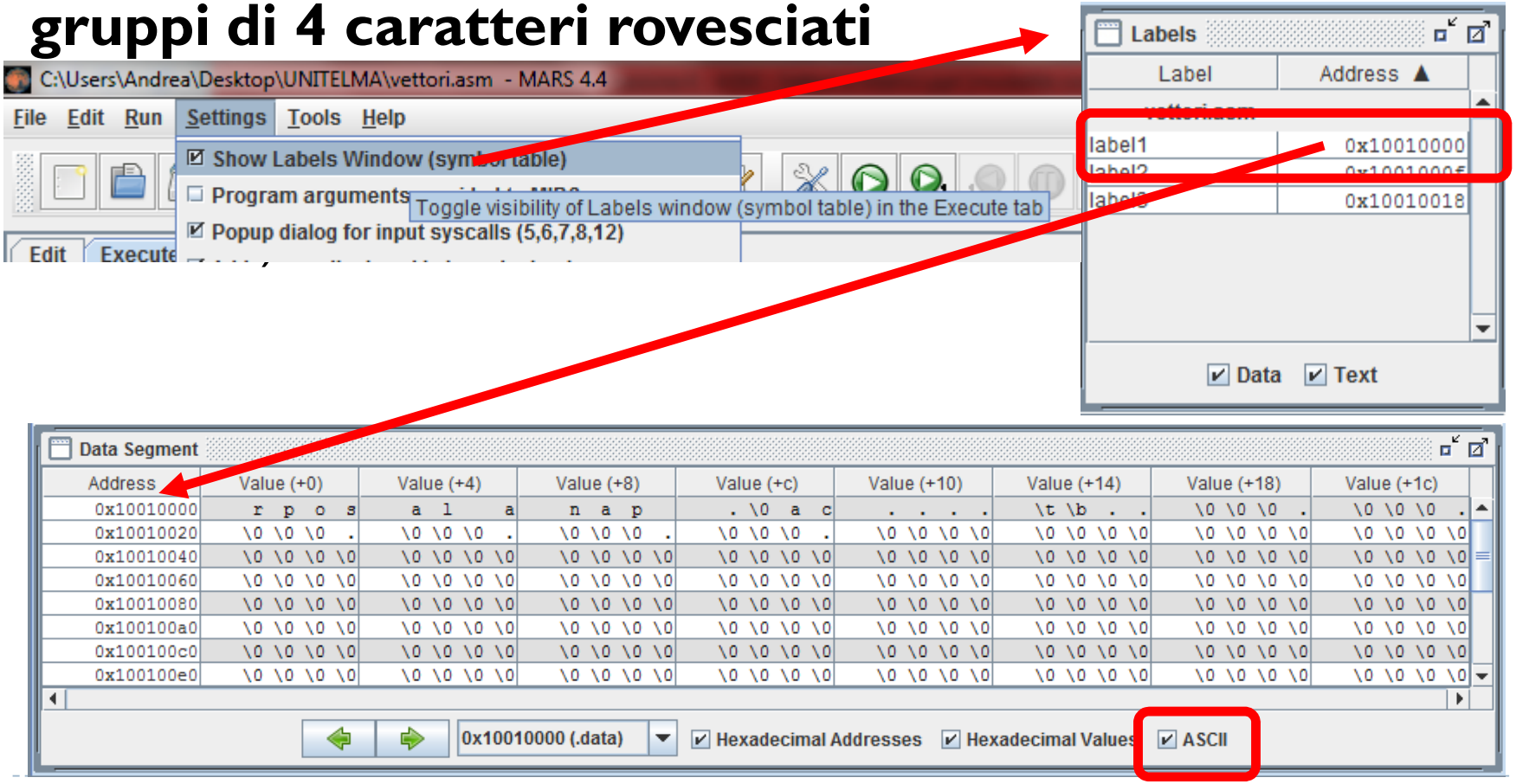
→ nella finestra dei dati le **stringhe** sono visualizzate come **gruppi di 4 caratteri rovesciati**



MARS e i vettori

MARS è little-endian

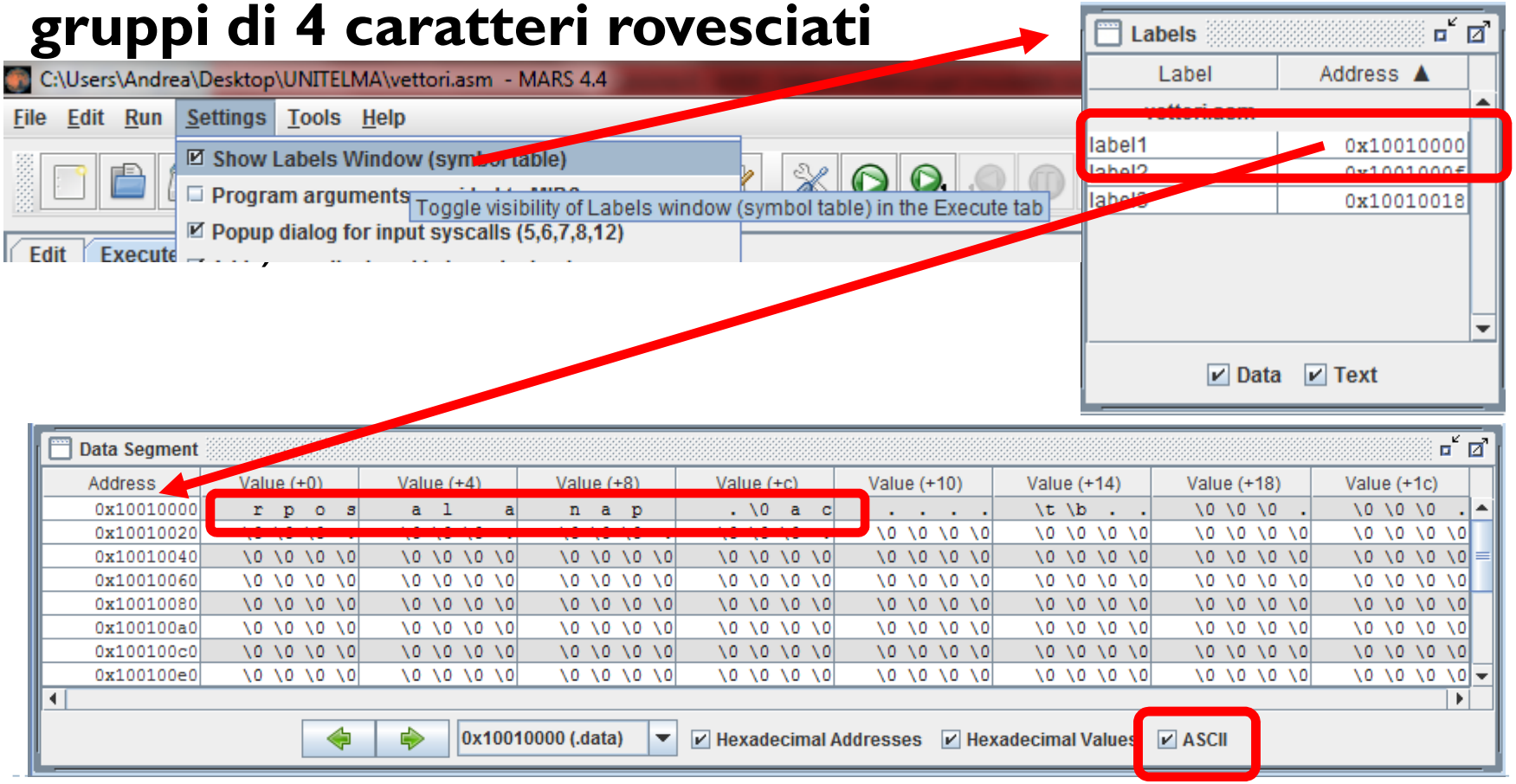
→ nella finestra dei dati le **stringhe** sono visualizzate come **gruppi di 4 caratteri rovesciati**



MARS e i vettori

MARS è little-endian

→ nella finestra dei dati le **stringhe** sono visualizzate come **gruppi di 4 caratteri rovesciati**



Accesso agli elementi per indice

Indirizzo dell'elemento i = indirizzo del vettore + i * dimensione_elemento

Esempio: (frammento che calcola l'indirizzo di un elemento in un vettore di word)

$t0 contiene l'indice dell'elemento

$t1 contiene l'indirizzo del vettore

in $t2 si ottiene l'indirizzo dell'elemento (da caricare o da scrivere in memoria)

```
sll $t2, $t0, 2      # shiftare di due bit = moltiplicare per 4
add $t2, $t2, $t1    #
```

Per elementi di dimensioni diverse (half word o byte o altro) va cambiata la prima istruzione

Se il vettore è allocato staticamente la seconda istruzione può essere direttamente sostituita dalla istruzione di accesso in memoria

```
sll $t2, $t0, 2      # offset in byte dell'el. rispetto all'inizio del vettore
lw  $s0, label1($t2) # lettura della word
```

Cicli

Nei cicli si possono usare due stili di scansione di un vettore

Scansione per indice

- Pro:
  - comoda se si deve usare l'indice dell'elemento per controlli o altro
  - l'incremento dell'indice non dipende dalla dimensione degli elementi
  - comoda se il vettore è allocato staticamente (nella parte .data)
- Contro:
  - bisogna convertire ogni volta l'indice nel corrispondente offset in byte

Scansione per puntatori (ovvero manipolando direttamente indirizzi in memoria)

- Pro:
  - si lavora direttamente su indirizzi in memoria
  - ci sono meno calcoli nel ciclo
- Contro:
  - non si ha a disposizione l'indice dell'elemento
  - l'incremento del puntatore dipende dalla dimensione degli elementi
  - bisogna calcolare l'indirizzo successivo all'ultimo elemento

Esempio (CON INDICE)

Esempio: somma degli elementi di un vettore di word a posizione **divisibile per tre**

```
.data
vettore: .word 1, 2, 3, 4, 5, 6, 7, 8, 9     # vettore da sommare
N:       .word 9                             # numero di elementi
somma:   .word 0                             # risultato
.text
main:    li    $t0, 0                        # i = 0
         lw    $t1, N                        # lettura di N
         li    $t2, 0                        # somma = 0
loop:    bge   $t0, $t1, fine                # è finito il ciclo?
         sll   $t3, $t0, 2                   # offset = i<<2
         lw    $t3, vettore($t3)             # lettura di vettore[i]
         add   $t2, $t2, $t3                 # somma += vettore[i]
         addi  $t0, $t0, 3                   # i += 3
         j     loop
fine:    sw    $t2, somma                    # memorizzo il risultato
```

Esempio (CON PUNTATORI)

```
.data
vettore: .word 1, 2, 3, 4, 5, 6, 7, 8, 9     # vettore da sommare
N:       .word  9                            # numero di elementi
somma:   .word  0                            # risultato
.text
main:    lw     $t1, N                       # lettura di N
         la     $t0, vettore                 # indirizzo di vettore
         sll    $t1, $t1, 2                  # dimensione = N * 4
         add    $t1, $t1, $t0                # fine = vettore+dimensione
         li     $t2, 0                       # somma = 0
loop:    bge    $t0, $t1, fine               # è finito il ciclo?
         lw     $t3, ($t0)                   # lettura di vettore[i]
         add    $t2, $t2, $t3                # somma += vettore[i]
         addi   $t0, $t0, 12                 # i += 3 * dim_elemento
         j      loop
fine:    sw     $t2, somma                   # memorizzo il risultato
```

Matrici = vettori di vettori

Una matrice **N x M** non è altro che una successione di N vettori, ciascuno di M elementi

- il numero di elementi totali è **N x M**
- la dimensione totale in byte è **N x M x dimensione_elemento**
- la si definisce staticamente come un vettore contenente **N x M** elementi uguali

```
matrice: .word 91:0      # spazio per una matrice 13 x 7 di word
```

Matrici = vettori di vettori

Una matrice **N x M** non è altro che una successione di N vettori, ciascuno di M elementi

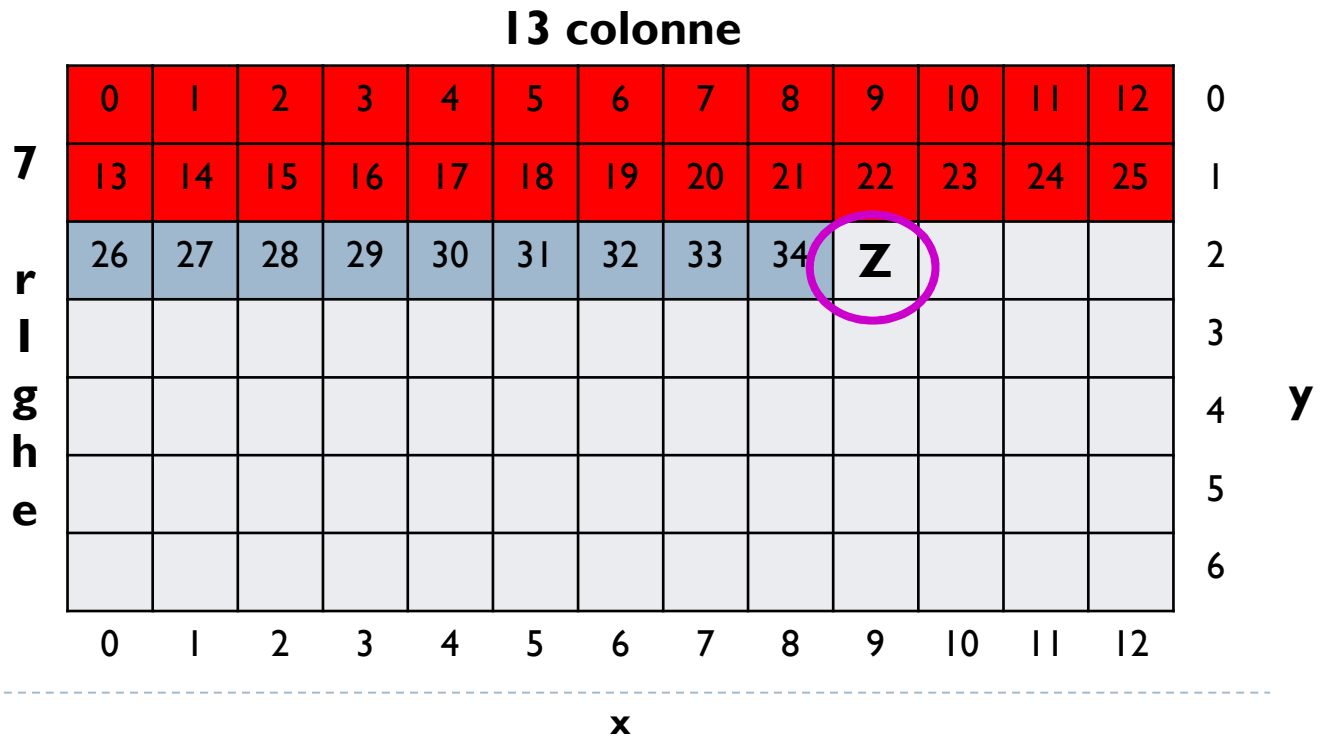
- il numero di elementi totali è **N x M**
- la dimensione totale in byte è **N x M x dimensione_elemento**
- la si definisce staticamente come un vettore contenente **N x M** elementi uguali

```
matrice: .word 91:0      # spazio per una matrice 13 x 7 di word
```

L'elemento **Z**, di coordinate **x=9, y=2** si trova ad una distanza di:

- **2** righe
- più **9** elementi

dall'inizio, ovvero ad un offset di **2*13+9 = 35 word** cioè **35*4 = 140 byte**



Matrici 3D

Questo è esattamente il calcolo svolto dal compilatore C, per esempio

scrivere	**matrice[x][y]**
per una matrice di interi definita	**int matrice[NUMCOL][NUMRIG]**
equivale a usare l'indirizzo	**matrice + (y * NUMCOL+ x) * sizeof(int)**

Matrici 3D

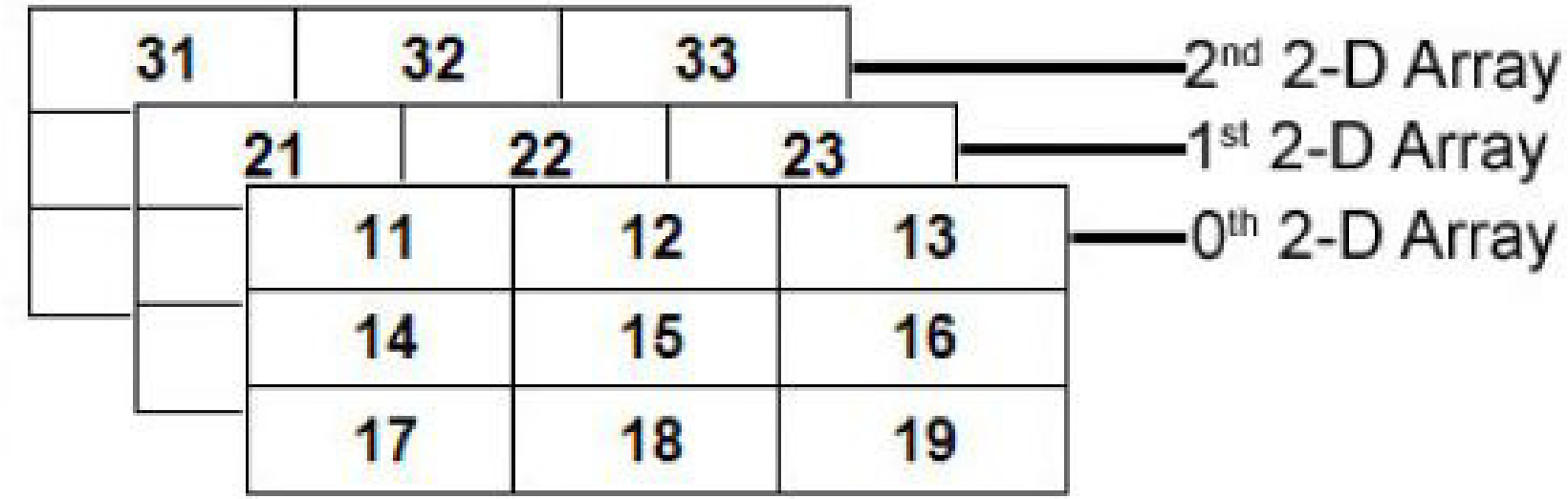
Questo è esattamente il calcolo svolto dal compilatore C, per esempio

scrivere	**matrice[x][y]**
per una matrice di interi definita	**int matrice[NUMCOL][NUMRIG]**
equivale a usare l'indirizzo	**matrice + (y * NUMCOL+ x) * sizeof(int)**

Matrici a 3 dimensioni

Una matrice 3D di **dimensioni X x Y x Z** è una successione di **Z** matrici 2D grandi **X x Y**

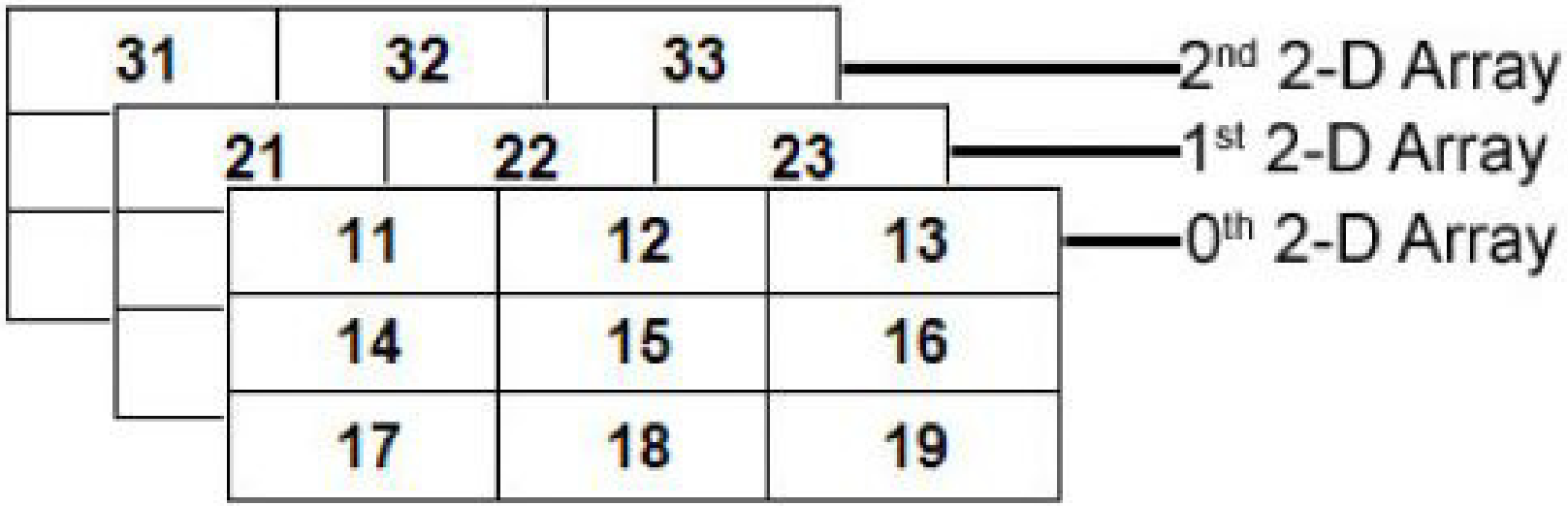
Matrici 3D



Matrici a 3 dimensioni

Una matrice 3D di **dimensioni** **X x Y x Z** è una successione di **Z** matrici 2D grandi **X x Y**

Matrici 3D



Matrici a 3 dimensioni

Una matrice 3D di **dimensioni** **X x Y x Z** è una successione di **Z** matrici 2D grandi **X x Y**

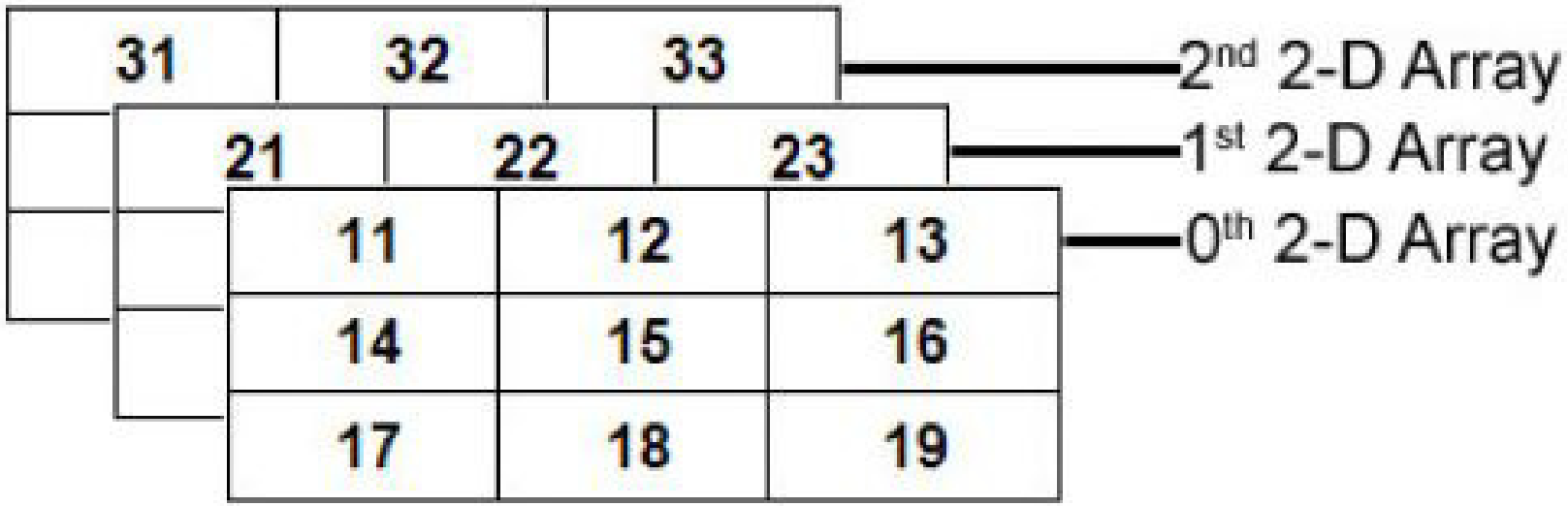
L'elemento a coordinate **x, y, z** è preceduto da:

z «strati» (matrici **X x Y** formate da XY elementi)

y «righe» di X elementi sullo stesso strato

x «elementi» sulla stessa riga e strato

Matrici 3D



Matrici a 3 dimensioni

Una matrice 3D di **dimensioni** **X x Y x Z** è una successione di **Z** matrici 2D grandi **X x Y**

L'elemento a coordinate **x, y, z** è preceduto da:

z «strati» (matrici **X x Y** formate da XY elementi)

y «righe» di X elementi sullo stesso strato

x «elementi» sulla stessa riga e strato

Quindi l'elemento si trova a **z * (X*Y) + y * X + x** elementi dall'inizio della matrice 3D

e la sua posizione in memoria è **indirizzo_matrice + (z * (X*Y) + y * X + x) * dim_el.**

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word 400:0                    # matrice quadrata di 20x20 word
DIM:       .word   20                     # lato della matrice
.text
```

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word    400:0                    # matrice quadrata di 20x20 word
DIM:       .word    20                       # lato della matrice
.text
main:      li       $t0, 0                   # coordinata x
           li       $t1, 0                   # coordinata y
           li       $t2, 0                   # somma iniziale
           lw       $t3, DIM                 # lato della matrice
```

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word    400:0                   # matrice quadrata di 20x20 word
DIM:       .word    20                      # lato della matrice
.text
main:      li       $t0, 0                  # coordinata x
           li       $t1, 0                  # coordinata y
           li       $t2, 0                  # somma iniziale
           lw       $t3, DIM                # lato della matrice
cicloRighe:
```

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word 400:0                  # matrice quadrata di 20x20 word
DIM:       .word 20                     # lato della matrice
.text
main:      li    $t0, 0                 # coordinata x
           li    $t1, 0                 # coordinata y
           li    $t2, 0                 # somma iniziale
           lw    $t3, DIM               # lato della matrice
cicloRighe:
           bge   $t1, $t3, fine         # se finite le righe
```

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word 400:0                  # matrice quadrata di 20x20 word
DIM:       .word 20                     # lato della matrice
.text
main:      li    $t0, 0                 # coordinata x
           li    $t1, 0                 # coordinata y
           li    $t2, 0                 # somma iniziale
           lw    $t3, DIM               # lato della matrice
cicloRighe:
           bge   $t1, $t3, fine         # se finite le righe
cicloColonne:
```

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word 400:0                  # matrice quadrata di 20x20 word
DIM:       .word 20                     # lato della matrice
.text
main:      li    $t0, 0                 # coordinata x
           li    $t1, 0                 # coordinata y
           li    $t2, 0                 # somma iniziale
           lw    $t3, DIM               # lato della matrice
cicloRighe:
           bge   $t1, $t3, fine         # se finite le righe
cicloColonne:
           bge   $t0, $t3, nextRiga     # se finite le colonne
```

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word 400:0                  # matrice quadrata di 20x20 word
DIM:       .word 20                     # lato della matrice
.text
main:      li    $t0, 0                 # coordinata x
           li    $t1, 0                 # coordinata y
           li    $t2, 0                 # somma iniziale
           lw    $t3, DIM               # lato della matrice
cicloRighe:
           bge   $t1, $t3, fine         # se finite le righe
cicloColonne:
           bge   $t0, $t3, nextRiga     # se finite le colonne
           bne   $t0, $t1, continua     # se x != y si continua
```

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word 400:0                  # matrice quadrata di 20x20 word
DIM:       .word 20                     # lato della matrice
.text
main:      li    $t0, 0                 # coordinata x
           li    $t1, 0                 # coordinata y
           li    $t2, 0                 # somma iniziale
           lw    $t3, DIM               # lato della matrice
cicloRighe:
           bge   $t1, $t3, fine         # se finite le righe
cicloColonne:
           bge   $t0, $t3, nextRiga     # se finite le colonne
           bne   $t0, $t1, continua     # se x != y si continua
           mul   $t4, $t1, $t3          # y*X
           add   $t4, $t4, $t0          # y*X + x
           sll   $t4, $t4, 2            # word => molt. per 4
```

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word 400:0                  # matrice quadrata di 20x20 word
DIM:       .word 20                     # lato della matrice
.text
main:      li    $t0, 0                 # coordinata x
           li    $t1, 0                 # coordinata y
           li    $t2, 0                 # somma iniziale
           lw    $t3, DIM               # lato della matrice
cicloRighe:
           bge   $t1, $t3, fine         # se finite le righe
cicloColonne:
           bge   $t0, $t3, nextRiga     # se finite le colonne
           bne   $t0, $t1, continua     # se x != y si continua
           mul   $t4, $t1, $t3          # y*X
           add   $t4, $t4, $t0          # y*X + x
           sll   $t4, $t4, 2            # word => molt. per 4
           lw    $t4, matrice2D($t4)    # carico matrice2D[x][y]
```

Somma della diagonale (INEFF.)

```
.data
matrice2D: .word 400:0                  # matrice quadrata di 20x20 word
DIM:       .word 20                     # lato della matrice
.text
main:      li    $t0, 0                 # coordinata x
           li    $t1, 0                 # coordinata y
           li    $t2, 0                 # somma iniziale
           lw    $t3, DIM               # lato della matrice
cicloRighe:
           bge   $t1, $t3, fine         # se finite le righe
cicloColonne:
           bge   $t0, $t3, nextRiga     # se finite le colonne
           bne   $t0, $t1, continua     # se x != y si continua
           mul   $t4, $t1, $t3          # y*X
           add   $t4, $t4, $t0          # y*X + x
           sll   $t4, $t4, 2            # word => molt. per 4
           lw    $t4, matrice2D($t4)    # carico matrice2D[x][y]
           add   $t2, $t4, $t2          # e lo accumulo
```

Somma della diagonale (segue)

```
continua:
        addi    $t0, $t0, 1             # x += 1
        j       cicloColonne            # alla colonna successiva
```

Somma della diagonale (segue)

```
continua:
        addi    $t0, $t0, 1             # x += 1
        j       cicloColonne            # alla colonna successiva
nextRiga:
```

Somma della diagonale (segue)

```
continua:
        addi    $t0, $t0, 1             # x += 1
        j       cicloColonne            # alla colonna successiva
nextRiga:
        li      $t0, 0                  # azzero x
        addi    $t1, $t1, 1             # y += 1
        j       cicloRighe              # alla riga successiva
```

Somma della diagonale (segue)

```
continua:
        addi    $t0, $t0, 1             # x += 1
        j       cicloColonne            # alla colonna successiva
nextRiga:
        li      $t0, 0                  # azzero x
        addi    $t1, $t1, 1             # y += 1
        j       cicloRighe              # alla riga successiva
fine:   move    $a0, $t2                # preparo la stampa
        li      $v0, 1                  # syscall 1 = print int
        syscall
        li      $v0, 10                 # syscall 10 = stop
        syscall
```

Somma della diagonale (segue)

```
continua:
        addi    $t0, $t0, 1             # x += 1
        j       cicloColonne            # alla colonna successiva
nextRiga:
        li      $t0, 0                  # azzero x
        addi    $t1, $t1, 1             # y += 1
        j       cicloRighe              # alla riga successiva
fine:   move    $a0, $t2                # preparo la stampa
        li      $v0, 1                  # syscall 1 = print int
        syscall
        li      $v0, 10                 # syscall 10 = stop
        syscall
```

NOTA: questa è una versione volutamente inefficiente che scandisce tutta la matrice. Può essere resa più efficiente:

- Usando un solo ciclo sulla coordinata x da 0 a DIM-1
- Usando i puntatori (indirizzi in memoria)
- Incrementando il puntatore di DIM+1 elementi = (DIM+1)*4 byte per passare da un elemento della diagonale al successivo

Somma diagonale (EFFICIENTE)

```
.data
matrice2D:      .word   400:0           # matrice di 20x20 word
DIM:            .word   20              # lato della matrice
```

Somma diagonale (EFFICIENTE)

```
.data
matrice2D:      .word   400:0           # matrice di 20x20 word
DIM:            .word   20              # lato della matrice
.text
```

Somma diagonale (EFFICIENTE)

```
.data
matrice2D:      .word   400:0           # matrice di 20x20 word
DIM:            .word   20              # lato della matrice
.text
        # preparazione degli indirizzi e degli incrementi
main:   # $t0 = indirizzo dell'elemento corrente
        # $t1 = incremento di una riga + 1 elemento (in byte)
        # $t2 = somma parziale
        # $t3 = indirizzo finale della matrice (byte seguente)
```

Somma diagonale (EFFICIENTE)

```
.data
matrice2D:      .word   400:0           # matrice di 20x20 word
DIM:            .word   20              # lato della matrice
.text
        # preparazione degli indirizzi e degli incrementi
main:   # $t0 = indirizzo dell'elemento corrente
        # $t1 = incremento di una riga + 1 elemento (in byte)
        # $t2 = somma parziale
        # $t3 = indirizzo finale della matrice (byte seguente)
        la      $t0, matrice2D          # indirizzo dell'inizio
```

Somma diagonale (EFFICIENTE)

```
.data
matrice2D:      .word   400:0           # matrice di 20x20 word
DIM:            .word   20              # lato della matrice
.text
        # preparazione degli indirizzi e degli incrementi
main:   # $t0 = indirizzo dell'elemento corrente
        # $t1 = incremento di una riga + 1 elemento (in byte)
        # $t2 = somma parziale
        # $t3 = indirizzo finale della matrice (byte seguente)
        la      $t0, matrice2D          # indirizzo dell'inizio
        lw      $t1, DIM                        # lato della matrice
```

Somma diagonale (EFFICIENTE)

```
.data
matrice2D:      .word   400:0           # matrice di 20x20 word
DIM:            .word   20              # lato della matrice
.text
        # preparazione degli indirizzi e degli incrementi
main:   # $t0 = indirizzo dell'elemento corrente
        # $t1 = incremento di una riga + 1 elemento (in byte)
        # $t2 = somma parziale
        # $t3 = indirizzo finale della matrice (byte seguente)
        la      $t0, matrice2D          # indirizzo dell'inizio
        lw      $t1, DIM                        # lato della matrice
        mul     $t3, $t1, $t1           # DIM * DIM elementi
        sll     $t3, $t3, 2             # totale DIM^2 * 4 byte
        add     $t3, $t3, $t0           # indirizzo finale
```

Somma diagonale (EFFICIENTE)

```
.data
matrice2D:      .word   400:0           # matrice di 20x20 word
DIM:            .word   20              # lato della matrice
.text
        # preparazione degli indirizzi e degli incrementi
main:   # $t0 = indirizzo dell'elemento corrente
        # $t1 = incremento di una riga + 1 elemento (in byte)
        # $t2 = somma parziale
        # $t3 = indirizzo finale della matrice (byte seguente)
        la      $t0, matrice2D          # indirizzo dell'inizio
        lw      $t1, DIM                        # lato della matrice
        mul     $t3, $t1, $t1           # DIM * DIM elementi
        sll     $t3, $t3, 2             # totale DIM^2 * 4 byte
        add     $t3, $t3, $t0           # indirizzo finale
        addi    $t1, $t1, 1             # DIM + 1
        sll     $t1, $t1, 2             # incremento = (DIM+1)*4
```

Somma diagonale (EFFICIENTE)

```
.data
matrice2D:      .word   400:0           # matrice di 20x20 word
DIM:            .word   20              # lato della matrice
.text
        # preparazione degli indirizzi e degli incrementi
main:   # $t0 = indirizzo dell'elemento corrente
        # $t1 = incremento di una riga + 1 elemento (in byte)
        # $t2 = somma parziale
        # $t3 = indirizzo finale della matrice (byte seguente)
        la      $t0, matrice2D          # indirizzo dell'inizio
        lw      $t1, DIM                        # lato della matrice
        mul     $t3, $t1, $t1           # DIM * DIM elementi
        sll     $t3, $t3, 2             # totale DIM^2 * 4 byte
        add     $t3, $t3, $t0           # indirizzo finale
        addi    $t1, $t1, 1             # DIM + 1
        sll     $t1, $t1, 2             # incremento = (DIM+1)*4
        li      $t2, 0                  # somma iniziale
```

Somma della diagonale (segue)

```
# $t0 = indirizzo dell'elemento corrente
# $t1 = incremento di una riga + 1 elemento
# $t2 = somma parziale
# $t3 = indirizzo finale della matrice (subito dopo)
# ciclo che scandisce un elemento ogni DIM+1
ciclo:  bge     $t0, $t3, fine          # se è finita la matrice
esco
```

Somma della diagonale (segue)

```
# $t0 = indirizzo dell'elemento corrente
# $t1 = incremento di una riga + 1 elemento
# $t2 = somma parziale
# $t3 = indirizzo finale della matrice (subito dopo)
# ciclo che scandisce un elemento ogni DIM+1
ciclo:  bge     $t0, $t3, fine          # se è finita la matrice
esco
        lw      $t4, ($t0)              # carico matrice2D[x][x]
        add     $t2, $t4, $t2           # e lo accumulo
```

Somma della diagonale (segue)

```
# $t0 = indirizzo dell'elemento corrente
# $t1 = incremento di una riga + 1 elemento
# $t2 = somma parziale
# $t3 = indirizzo finale della matrice (subito dopo)
# ciclo che scandisce un elemento ogni DIM+1
ciclo:  bge     $t0, $t3, fine          # se è finita la matrice
esco
        lw      $t4, ($t0)              # carico matrice2D[x][x]
        add     $t2, $t4, $t2           # e lo accumulo
        add     $t0, $t0, $t1           # x += (DIM+1)*4
        j       ciclo
```

Somma della diagonale (segue)

```
# $t0 = indirizzo dell'elemento corrente
# $t1 = incremento di una riga + 1 elemento
# $t2 = somma parziale
# $t3 = indirizzo finale della matrice (subito dopo)
# ciclo che scandisce un elemento ogni DIM+1
ciclo:   bge     $t0, $t3, fine          # se è finita la matrice esco
         lw      $t4, ($t0)              # carico matrice2D[x][x]
         add     $t2, $t4, $t2           # e lo accumulo
         add     $t0, $t0, $t1           # x += (DIM+1)*4
         j       ciclo
# stampa del risultato
fine:    move    $a0, $t2          # preparo la stampa
         li      $v0, 1                  # syscall 1 = print int
         syscall
         li      $v0, 10                 # syscall 10 = stop
         syscall
```

Esercizio

- Calcolare e stampare la somma delle **due** diagonali di una matrice quadrata di word
- Fate attenzione a non sommare due volte l'elemento centrale in caso di matrici di lato dispari
- Suggerimento: scandite tutta la matrice e individuate le caselle sulle due diagonali