Architettura Elaboratori Elettronici

ESERCITAZIONI

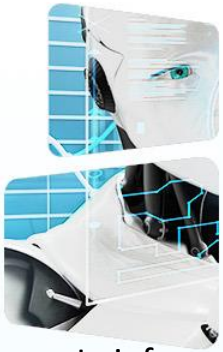
STRUTTURE DATI

Franco Liberati

liberati@di.uniroma1.it

Argomenti

- Strutture dati statiche
  - Vettore
  - Matrice
- Strutture dati dinamiche
  - Lista
  - Code
  - Stack



Struttura Dati

Definizione

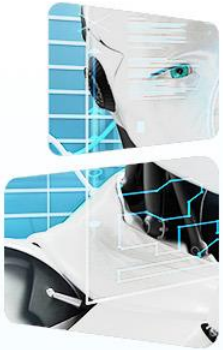
In informatica per **struttura dati** si intende un modo specifico di organizzare e memorizzare i dati (elementari) in modo che possano essere utilizzati in modo efficiente

In altre parole una struttura dati è una definizione di dati e di operazioni che possono essere eseguite su di essi. In sostanza, una struttura dati fornisce un modo per organizzare e manipolare i dati in modo efficiente e significativo all'interno di un programma informatico.

I dati sfruttati nelle strutture dati sono quelli gestiti direttamente dall’elaboratore: byte, halfword, word, float, double, ascii

Le strutture dati sono progettate per ottimizzare le operazioni di accesso, modifica e ricerca dei dati

Tra le strutture dati più note ci sono: vettori, liste, code, pile, alberi, grafi.



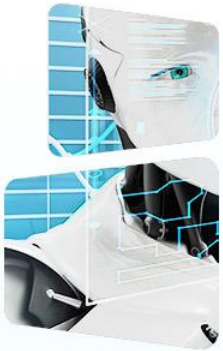
Struttura Dati

Classificazione

Le strutture dati possono essere **statiche** o **dinamiche**

- Nelle strutture dati statiche il numero dei dati è prestabilita e non muta (vettore o matrice)
- Nelle strutture dati dinamiche il numero di dati cambia nel corso dell'esecuzione del programma (pila, coda, lista, grafo). In questo ultimo caso un campo dei dati può essere usato per contenere l'indirizzo di un'ulteriore elemento della struttura dati (si dice che punta o che è un puntatore a quest'ultima)

VETTORE



VETTORE

Generalità

Un **vettore** (o **array**) è un insieme ordinato di elementi tutti dello stesso tipo disposti sequenzialmente in memoria centrale

In informatica i vettori possono essere utilizzati in diversi contesti e offrono molteplici vantaggi:

- **Memorizzazione sequenziale**: i vettori memorizzano elementi in modo sequenziale in memoria, consentendo un accesso rapido agli elementi tramite un indice
- **Accesso casuale**: Gli elementi di un vettore possono essere accessibili in modo casuale tramite l'indice corrispondente, il che rende l'accesso ai dati efficiente
- **Implementazione di strutture dati complesse**: i vettori sono spesso utilizzati come base per implementare altre strutture dati complesse come liste, code e pile.
- **Algoritmi di elaborazione dati**: Molte operazioni di elaborazione dati, come ordinamento, ricerca e manipolazione, possono essere implementate efficientemente utilizzando vettori

VETTORE

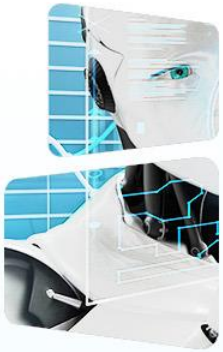
Definizione fisica

Un vettore è definito esplicitando il numero di elementi che lo compongono (**lunghezza del vettore**)

L'accesso ad un elemento del vettore è consentito specificando un **indice**

Il **primo elemento** del vettore ha **indice 0**

Indice	Indirizzo in memoria	Valore in memoria
0	**100**	**12**
1	**104**	**34**
2	**108**	**-121**
3	**112**	**0**
4	**116**	**54**



VETTORE

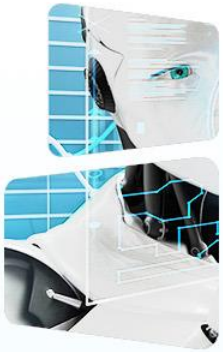
Definizione in un linguaggio ad alto livello

Un **vettore è definito**, in un linguaggio ad alto livello, dalla specifica del tipo, dall'identificatore e dalla sua lunghezza (cioè del numero di elementi)

È possibile inizializzare un elemento di un vettore specificando l'**identificativo** e l'**indice** (nel linguaggio C si usano le parentesi quadre) e il **valore**

Per prelevare un elemento è sufficiente utilizzare l'**identificativo** e l'**indice** (e una variabile per la memorizzazione del dato)

Dichiarazione	**int array[5];**	**Crea un vettore di interi in memoria (cinque elementi consecutivi)**
Inizializzazione	**array[0]=45;**	**Inserisce nella prima locazione di memoria assegnata all'array il valore 45**
Recupero dati	**int temp=array[0]**	**Nella variabile temp si trova il valore relativo alla prima locazione di memoria associata all'array (ovvero temp=45)**



VETTORE

Manipolazione usando un linguaggio ad alto livello

Effettuare la somma degli elementi di due vettori (di lunghezza 3) a medesima posizione

ESEMPIO

INPUT:

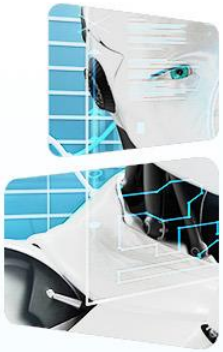
V1=(13,24,35)

V2=(47,16,-25)

OUTPUT:

V=(60,40,10)

```
#include <stdio.h>
int main()
{
   const int dim_vett=3;
   int i, a,b,addition;
   int v1[dim_vett]={13,24,35};
   int v2[dim_vett]={47,16,-25};
   int v3[dim_vett];
   for (i=0;i<dim_vett;i=i+1){
      a=v1[i];
      b=v2[i];
      addition=a+b;
      v3[i]=addition;
   }

   for (i=0;i<dim_vett;i=i+1){
      printf("%d\t", v3[i]);
   }
return 0;
}
```

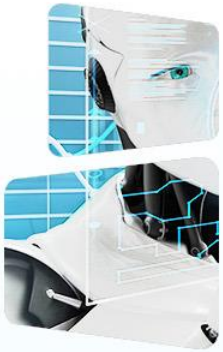


VETTORE IN MARS

Generalità

- Un **vettore** è un sequenza di dati omogenei *(dello stesso tipo)* disposti sequenzialmente in memoria
  - I dati o *elementi* di un array hanno tutti la stessa dimensione di tipo (*esize* o *dimension*) correlata al tipo elementare: **word** sono 4byte, **half** sono 2byte, **byte** sono 8bit per elemento
  - La posizione (*indice*) identifica i singoli elementi

Indice	Indirizzo in memoria	Valore in memoria
0	**100**	**12**
1	**104**	**34**
2	**108**	**-121**
3	**112**	**0**
4	**116**	**54**



VETTORE IN MARS

Definizione

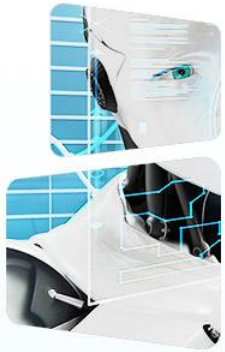
- In MARS un vettore si definisce specificando il tipo e interponendo delle virgole tra gli elementi

- Essendo una struttura statica in MARS è doveroso riportare anche il numero di elementi che lo costituiscono (**lunghezza del vettore**)

Esempio:

array8bit: .byte 12,34,-121,0,54
lunghezzaarray8bit: .byte 5

Indice	Posizione	Valore in memoria
0	100	**12**
1	101	**34**
2	102	**-121**
3	103	**0**
4	104	**54**



VETTORE IN MARS

Definizione

Esempio:

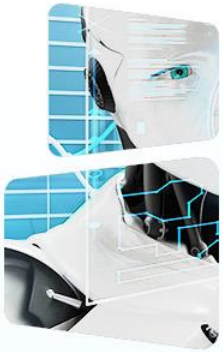
array16bit: .half 12000,-2000,-15,78
lunghezzaarray16bit: .byte 4

Indice	Posizione	Valore in memoria
0	200	**12000**
1	202	**-2000**
2	204	**-15**
3	206	**78**

Esempio:

array32bit: .word 67067,23400000,11,-785
lunghezzaarray32bit: .byte 4

Indice	Posizione	Valore in memoria
0	300	**67067**
1	304	**23400000**
2	308	**11**
3	312	**-785**



VETTORE IN MARS

Inizializzazione

- È possibile inizializzare un vettore ad un valore specifico definendo il valore e la sua lunghezza (numero di elementi) con una sintassi del tipo

etichetta: .tipo valore_di_inizializzazione:numero_elementi

- Esempio:

vettore: .word 2:10

Posizione	Valore in memoria
0	2
4	2
8	2
12	2
16	2
20	2
24	2
28	2
32	2
36	2

VETTORE IN MARS

Individuazione di un elemento

- **L'indirizzo dell'elemento** è ottenuto da:

base + index* dimension

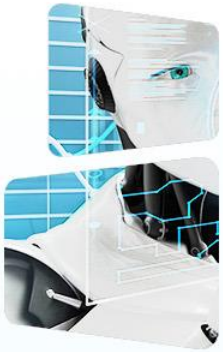
dove

base **è l'indirizzo di memoria dove inizia il vettore**

index **è l'incide dell'elemento che si vuole manipolare (inizializzare o prelevare)**

dimension **è la dimensione degli elementi (dipende dal tipo elementare)**

Posizione	Valore in memoria
0	2
4	2
8	2
12	2
16	2
20	2
24	2
28	2
32	2
36	2



VETTORE IN MARS

Individuazione di un elemento

Esempio:

array32bit: .word 123,56,44,22,-33,67
lunghezzaarray32bit: .byte 4

- L'indirizzo del quarto elemento è ottenuto da:

base* + *index* * *dimension

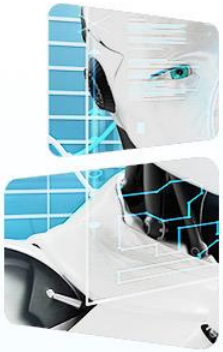
dove
base **è 100**

index **è 3**

dimension **è 4**

Cioè locazione di memoria: **112**

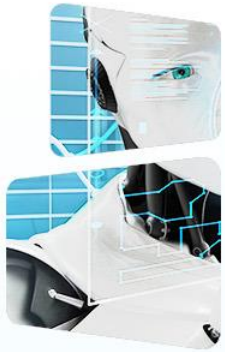
Indice		Valore in memoria
0	Array >100	123
1	104	56
2	108	44
3	112	22
4	116	-33
5	120	67



VETTORE IN MARS

Manipolazione dei dati

Per accedere agli elementi di un vettore in lettura e in scrittura si può ricorrere a due modalità



VETTORE IN MARS

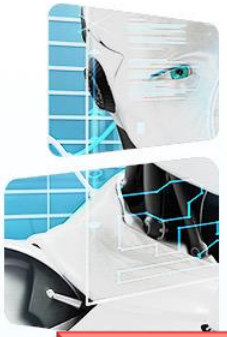
Manipolazione dei dati: a registro con spiazzamento simbolico

❑Per accedere agli elementi di un vettore in lettura e in scrittura si può ricorrere al **modo di indirizzamento a registro con spiazzamento simbolico**

vettore($a0)

Dove $a0 *deve contenere la posizione dell'elemento moltiplicato per la sua dimensione (4 in caso di word, 2 per le halfword, 1 per il byte)*

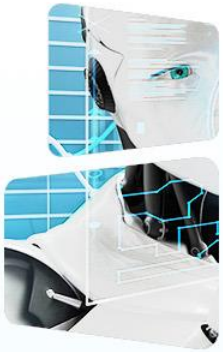
```
li $t1,0        #indice relativo al primo elemento
mul $a0,$t1,4 #indice*dimensione (posizione relativa)
lw $t0,array($a0) # copia/prelievo del primo elemento
                  #il calcolo su array($a0) riporta
                  #la posizione assoluta

li $t1,3        #indice relativo al quarto elemento
mul $a0,$t1,4 #posizione relativa del quarto elemento
lw $t3,array($a0) #prelievo del quarto elemento

.data
array: .word 45,100,-23,56
lunghezzaarray: .byte 4
```



VETTORE IN MARS

Manipolazione dei dati

Stampa di un vettore costituito da elementi di 16bit

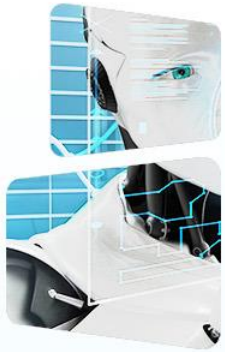


VETTORE IN MARS

Manipolazione dei dati

```
.text
.globl main
main:
        # STAMPA DI UN VETTORE
        lw $t1,nunelem        # $t1 numero elementi del vettore
        li $t2,0  # $t2 indice
loop:
        mul $t3,$t2,2         # indice*dimensione (moltiplicare x 2 perché halfword)
        lh $t4,array($t3)     # copia dell'i-esimo elemento nel registro $t4 cioè $t4=v[i]
        move $a0,$t4          #sposto l'elemento per effettuarne la stampa
        li $v0,1               #stampo i-esimo elemento
        syscall
        la $a0,tabulato       # Stampo un tabulato (per una maggiore leggibilità)
        li $v0,4
        syscall
        add $t2,$t2,1         # incremento indice
        blt $t2,$t1,loop      #confronto per determinare la fine del vettore
li $v0,10
syscall
```

```
.data
array: .half 12,43,23,54,77
nunelem: .word 5
tabulato: .asciiz "\t"
```



VETTORE IN MARS

Manipolazione dei dati: a registro con spiazzamento

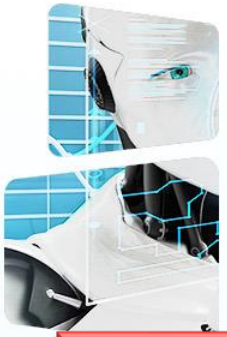
- Un'altra possibilità è quella di utilizzare come modo di indirizzamento per accedere agli elementi **a registro con spiazzamento**

($a0)

dove $a0 deve contenere un indirizzo costituito da

posizione_vettore+ (posizione_elemento)*dimensione

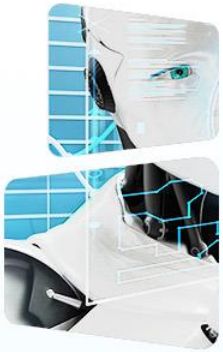
```
li $t1,0          # indice relativo del primo elemento
mul $t1,$t1,4     # indice*dimensione (posizione relativa)
la $a0,array      # indirizzo di inizio del vettore
lw $t0,($a0)      # copia del primo elemento
li $t1,3          # indice relativo del IV elemento
mul $t1,$t1,4     # indice*dimensione (posizione relativa)
add $a0,$a0,$t1   # aggiornamento  posizione fisica del IV
                  #elemento
lw $t1, ($a0)     # copia/prelievo del quarto elemento

.data
array: .word 45,100,-23,56
lunghezzaarray: .byte 4
```



VETTORE IN MARS

Manipolazione dei dati

Stampa di un vettore costituito da elementi di 16bit



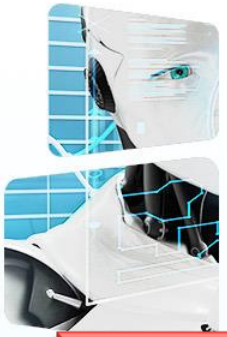
VETTORE IN MARS

Manipolazione dei dati

```
.text
.global main
main:
        la $t0,array                    # locazione di inizio del vettore
        lw $t1,nunelem                  # $t1 numero elementi del vettore
        li $t2,0                # $t2 indice
loop:
        mul $t3,$t2,4                   # indice * dimensione
        add $t3,$t0,$t3                 #posizione+(indice*dimensione)
        lh $t4,($t3)                    # copia dell'i-esimo elemento in $t4
        move $a0,$t4                    #... Elaboro elemento (es.:stampa)
        li $v0,1
        syscall
        la $a0,tabulato         # Stampo un tabulato per una maggiore leggibilità
        li $v0,4
        syscall
        addu $t2,$t2,1                  # incremento indice
        blt $t2,$t1,loop
li $v0,10
syscall
```

Si guadagna in efficienza

```
.data
array: .half 12,43,23,54,77
nunelem: .word 5
tabulato: .asciiz "\t"
```



ESERCIZIO

Definire due vettori di 5 elementi x di valore $-2^{16}<x_i<2^{15}-1$ e eseguire il **prodotto scalare (⊥)**

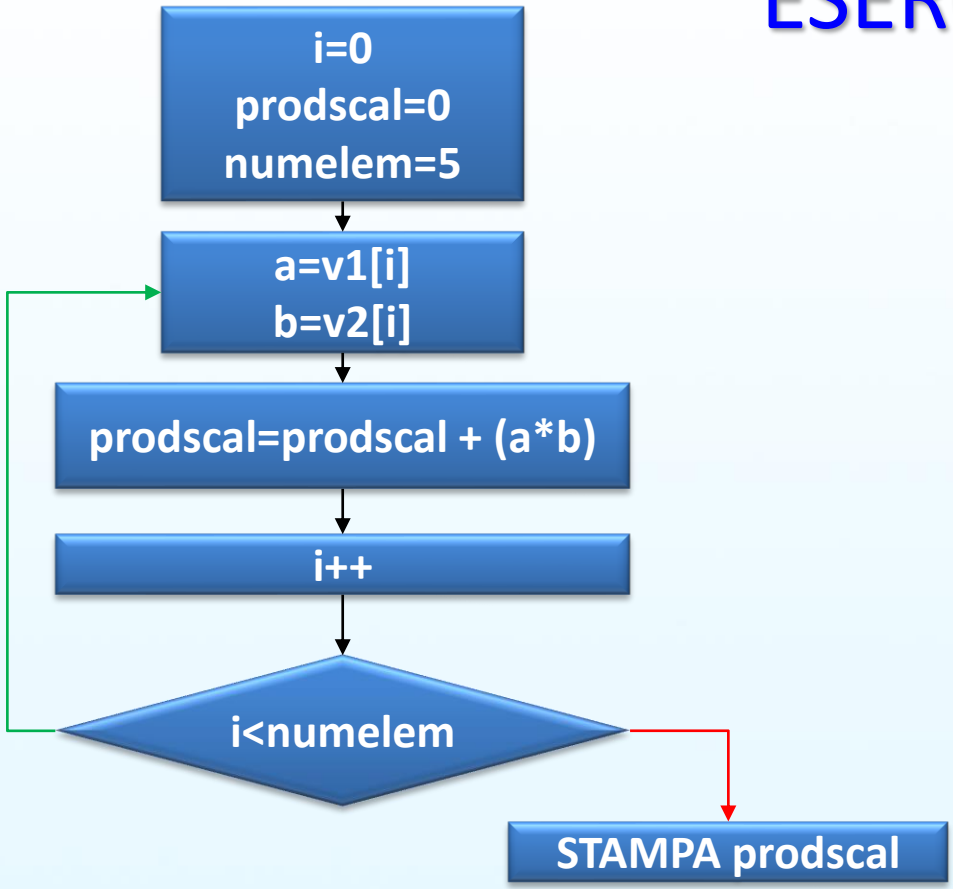
NB:

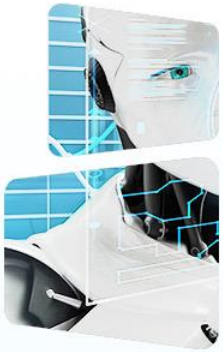
v1=(3,5,8,10,1) e v2=(1,2,3,0,13)

$v1 \perp v2 = (3\cdot1)+(5\cdot2)+(8\cdot3)+(10\cdot0)+(1\cdot13)=50$

Prodotto scalare

ESERCIZIO

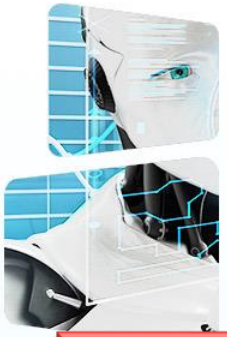




ESERCIZIO

```
.text
.globl main
main:
        xor $t0,$t0,$t0  #inizializzazione a zero del registro contenete l'indice del vettore
        li $t5,0         #inizializzatore del registro che contiene il numero di elementi
        lw $t9,numelem           #lunghezza del vettore
ciclo:
        mul $t1,$t0,2             #spiazzamento all'i-esimo elemento indice*dimensione
        lh $t2,v1($t1)                    #v1[i]
        lh $t3,v2($t1)                    #v2[i]
        mul $t4,$t2,$t3           #v1[i]*v2[i]
        add $t5,$t5,$t4           #prodotto scalare parziale
        add $t0,$t0,1                     #incremento indice
        bne $t0,$t9,ciclo         #confronto fine vettore

        move $a0,$t5              #stampa del prodotto scalare
        li $v0,1
        syscall
```

```
.data
v1: .half 3,5,8,10,1
v2: .half 1,2,3,0,13
numelem:.word 5
```



ESERCIZIO II

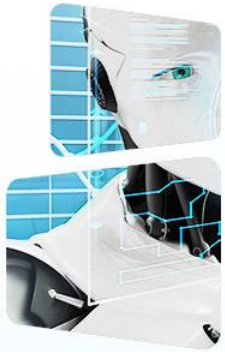
- Definire un vettore v1 di 6 elementi x di valore $-2^{32}<x_i<2^{31}-1$ e memorizzarlo in un nuovo vettore includendo solo gli elementi multipli di 5. Stampare v2

INPUT

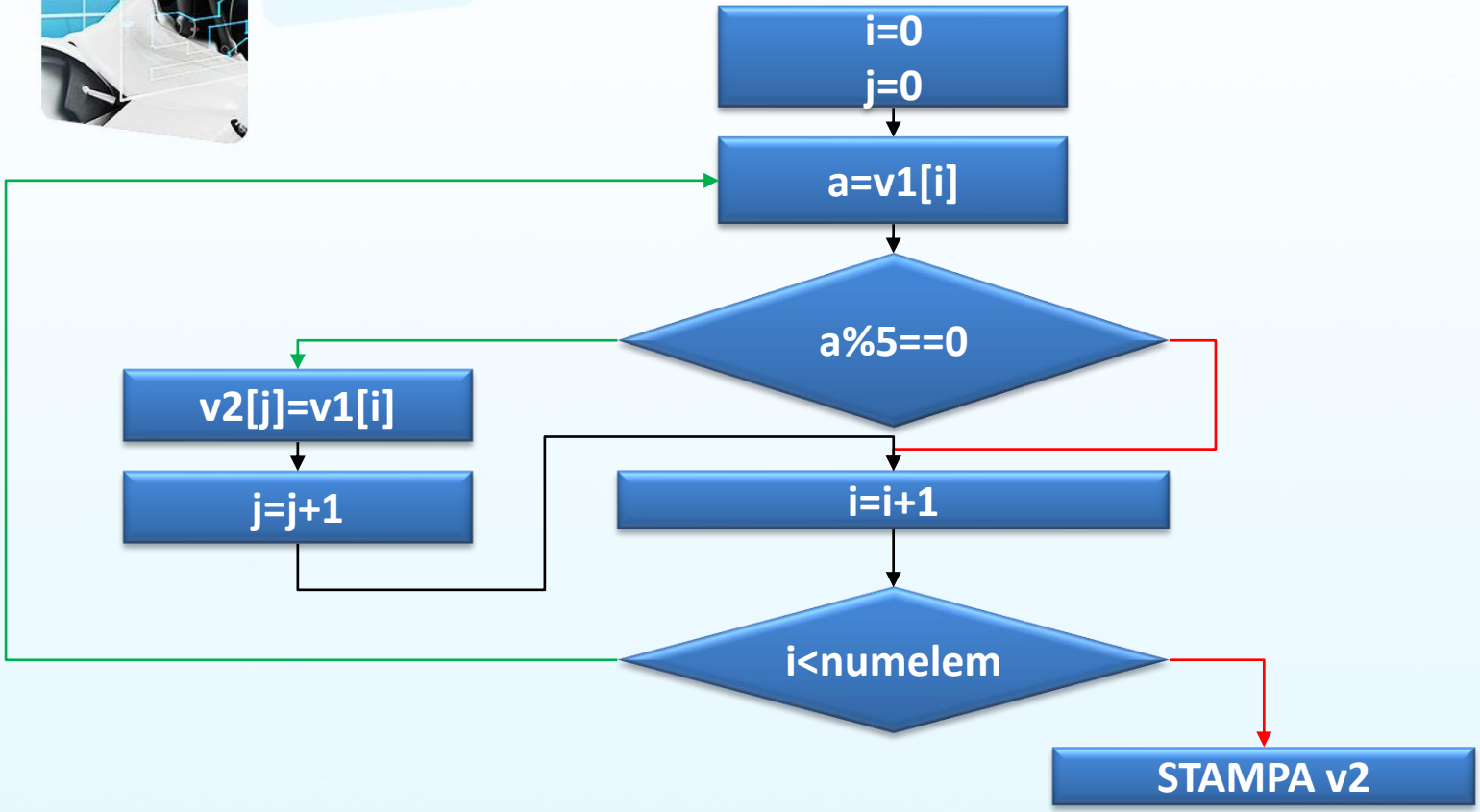
v1=(3,5,8,10,4,20)

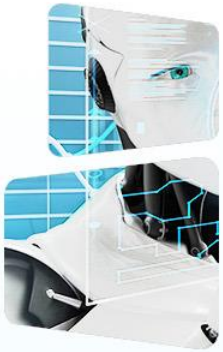
OUTPUT

v2= (5,10,20)



ESERCIZIO II



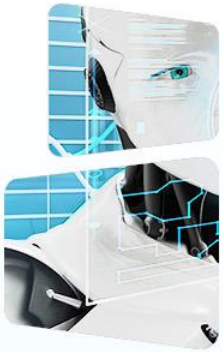


ESERCIZIO II

```
.text
.globl main
main:
        la $t0,v1        # indirizzo al primo elemento di v1
        la $t1,v2        # indirizzo al primo elemento di v2
        lw $t2,numelem   # $t2 numero elementi del vettore
        mul $t9,$t2,4    # spiazzamento dell'ultimo elemento di v1
        add $t9,$t9,$t0  # indirizzamento dell'ultimo elemento di v1

loop:   lw $t3,($t0)     #copia dell'i-esimo elemento di v1 nel registro $t3
        rem $t4,$t3,5            # vedo se v1[i] è multiplo di 5
        bnez $t4,salta
        sw $t3,($t1)             #memorizzo v2[j]=v1[i]
        addu $t1,$t1,4           # incremento indice v2

salta:  addu $t0,$t0,4           # incremento indice v1
        bgt  $t9,$t0,loop        #ciclo se non ho letto tutto  v1
        move $t8,$t1             # dimensione v2
        la $t1,v2                # lettura posizione inziale v2

stampa:
        lw $a0,($t1)
        li $v0,1
        syscall
        la $a0,separatore
        li $v0,4
        syscall
        addu $t1,$t1,4
        bgt $t8,$t1, stampa
li $v0,10
syscall

.data
v1: .word 3,5,8,10,1,7
v2:.space 24
numelem:.word 6
separatore:.asciiz "\t"
```

Si calcola
l'indirizzo relativo all'inizio del vettore
per poi svolgere il salto

STRINGA



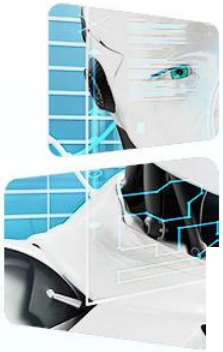
STRINGA

Generalità

Una **stringa** è una sequenza di caratteri, come lettere, numeri o simboli, che è trattata come un singolo dato

Le stringhe sono comunemente utilizzate per rappresentare testo e sono ampiamente utilizzate nei linguaggi di programmazione per manipolare e memorizzare dati di testo

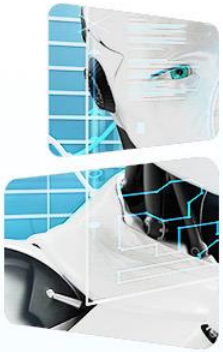
Le operazioni comuni sulle stringhe includono la ricerca, l'estrazione di sottostringhe, la concatenazione (unione) di due o più stringhe e la sostituzione di caratteri



STRINGA

Codifica utilizzata – STANDARD ASCII

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL



STRINGA

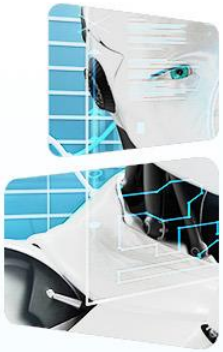
Definizione in un linguaggio ad alto livello

Una **stringa è definita**, in un linguaggio ad alto livello, dal tipo ***char***, dall'identificatore e della sua lunghezza (cioè del numero di caratteri)

È possibile inizializzare un elemento di una stringa specificando l'**identificativo** e l'**indice** (nel linguaggio C si usano le parentesi quadre) e il **simbolo**

Per prelevare un elemento è sufficiente utilizzare l'**identificativo** e l'**indice** (e una variabile per la memorizzazione del dato)

Dichiarazione	**char stringa[5];**	**Crea una stringa (cinque simboli consecutivi in memoria)**
Inizializzazione	**stringa="Ciao";**	**Inizializza con la stringa con le lettere *Ciao***
Recupero dati	**char simbolo=stringa[2]**	**Nella variabile temp si trova il carattere 'a'**



STRINGA

Manipolazione usando un linguaggio ad alto livello

Effettuare il calcolo del numero di volte in cui si ripete la lettera A in una stringa lunga 9 caratteri

ESEMPIO

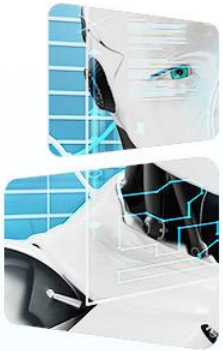
INPUT:

"Alabarda"

OUTPUT:

3

```
#include <stdio.h>

int main()
{
   char stringa[9]="Alabarda";
   int count,i;

   count=0;
   for(i=0;i<9;i++){
      if (stringa[i]=='a'){count=count+1;}
   }
   printf("Il numero di a presenti nella stringa %s è: %d",stringa,count);


   return 0;
}
```

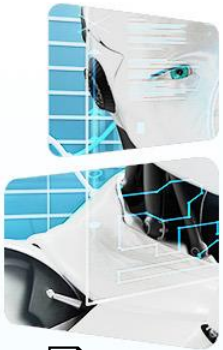


STRINGA IN MARS

Generalità

- Una stringa è un vettore di valori numerici, di un byte, che hanno il significato di carattere alfanumerico
  - Le stringhe sono sequenze numeriche (array) di caratteri con un terminatore NULL (il valore 0)
  - La stringa priva di caratteri , la stringa vuota, è quella formata dal solo terminatore NULL (tipicamente valore 0)
- I valori in codice ASCII prevedono le lettere maiuscole [A,Z] con intervallo[65,90]; minuscole [a,z] con intervallo [97,122]; e le cifre [0,9] con intervallo [48,57]

Indice	Indirizzo in memoria	Valore in memoria	Rappresentazione
0	100	67	C
1	101	65	A
2	102	83	S
3	103	65	A
4	104	0	0



STRINGA IN MARS

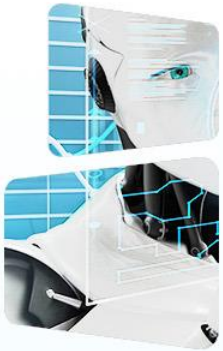
Definizione

- In MARS una stringa si definisce specificando il tipo **asciiz** o **ascii** e interponendo dei doppi apici tra la sequenza di caratteri

- Pur essendo una struttura statica in MARS NON è doveroso riportare il numero di elementi che lo costituiscono (**lunghezza del vettore**) perché se si usa la direttiva **asciiz** l'assemblatore pone il valore 0 alla fine dei caratteri tra doppi apici

Esempio:

stringa1: .asciiz "Cane"
stringa2: .ascii "Gatto"



STRINGA IN MARS

Definizione

Esempio:

stringa1: .asciiz "Cane"

Indice	Posizione	Valore in memoria	Rappresentazione
0	200	67	C
1	201	97	a
2	202	110	n
3	203	101	e
4	204	0	

Esempio:

stringa2: .ascii "Gatto"

Indice	Posizione	Valore in memoria	Rappresentazione
0	200	71	G
1	201	97	a
2	202	116	t
3	203	116	t
4	204	111	o
5	205	?	?

STRINGA IN MARS

Individuazione di un elemento

- **L'indirizzo di un carattere** è ottenuto da:

base + index* dimension

dove

base **è l'indirizzo di memoria dove inizia il vettore**

index **è l'incide dell'elemento che si vuole manipolare (inizializzare o prelevare)**

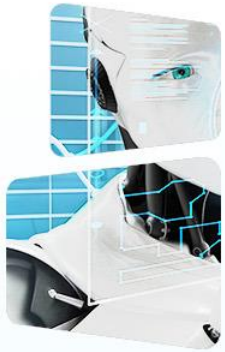
dimension **è la dimensione degli elementi cioè 1 perché si tratta di byte**

Posizione	Valore in memoria
0	2
4	2
8	2
12	2
16	2
20	2
24	2
28	2
32	2
36	2

STRINGA IN MARS

Manipolazione dei dati

Per accedere agli elementi di una stringa in lettura e in scrittura si può ricorrere a due modalità



STRINGA

ESEMPIO I

Calcolare il numero di caratteri di una strianga (str_len function)

ESEMPIO

INPUT:

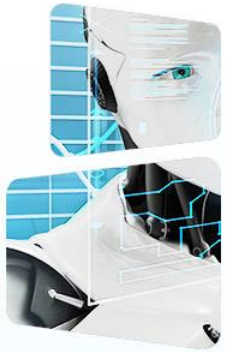
"Dedicato a chi crede che la colonna sonora quando ci sbatti contro… suoni"

OUTPUT:75

STRINGA

ESEMPIO II

```
.text
.globl main
main:
        xor $t2,$t2,$t2          #contatore delle lettere 'a'
        li $t8,97                #valore ASCII della lettera 'a'
                                 #in alternativa
                                 #lb $t8,lettera_a
        la $t0,stringa           #Indirizzo di inizio della stringa in $t0
ciclo:

        lb $t9,($t0)             #Lettura i-esimo carattere
        bne $t9 $t8, non_trovata #confronto
        add $t2,$t2,1            #incremento contatore
non_trovata:
        add $t0,$t0,1            #incremento indirizzo assoluto
        bnez $t9, ciclo          #si continua il ciclo se non si raggiunge la fine della stringa cioè il valore 0

        move $a0,$t2             #stampa del numero di 'a'
        li $v0,1                 #
        syscall                  #

        li $v0,10
        syscall

.data
stringa: .asciiz "Dedicato a chi crede che la colonna sonora quando ci sbatti contro... suoni"
```



STRINGA

ESEMPIO II

Effettuare il calcolo del numero di volte in cui si ripete la lettera *a* in una stringa lunga 9 caratteri

ESEMPIO

INPUT:

"*Dedicato a chi non ha mai letto Non Do Maho, la storia di un ricco avaro giapponese*"

OUTPUT:

10

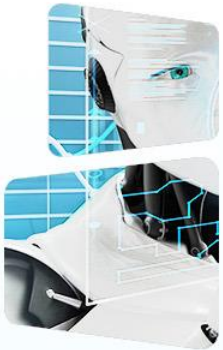
STRINGA

ESEMPIO II

```
.text
.globl main
main:
        xor $t0,$t0,$t0          #contatore delle lettere
ciclo:
        lb $t9,stringa($t0)      #Lettura i-esimo carattere
        bnez $t9 $t8, non_trovata #confronto
        add $t2,$t2,1            #incremento contatore
non_trovata:
        add $t0,$t0,1            #incremento indirizzo assoluto
        bnez $t9, ciclo          #si continua il ciclo se non si raggiunge la fine della stringa cioè il valore 0

        move $a0,$t2             #stampa del numero di 'a'
        li $v0,1                 #
        syscall                  #

        li $v0,10
        syscall
```

```
.data
stringa: .asciiz "Dedicato a chi non ha mai letto Nondo Ma Ho, la storia di un ricco avaro giappon
lettera_a:.asciiz "a"
```

MATRICE



MATRICE

(Generalità)

Una **matrice** è una tabella di dati omogenei *(dello stesso tipo)* costituita da *r* righe ed *c* colonne M_{rxc}

- I dati o *elementi* di una matrice hanno tutti la stessa dimensione (*esize*).
- La posizione (*indice riga/ indice colonna*) identifica i singoli elementi

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}
a_{51}	a_{52}	a_{53}	a_{54}

1	32	81	12
123	490	72	345
100	58	63	44
9	34	556	33
8	56	77	22

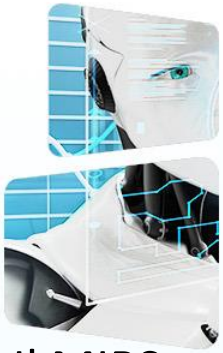
MATRICE

La matrice nel MARS

Il MIPS non consente una particolare definizione di matrice: si utilizza un vettore con lunghezza *m*x*n* e un indice per accedere alla *i*-esima posizione

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}
a_{51}	a_{52}	a_{53}	a_{54}

a_{11}
a_{12}
a_{13}
a_{14}
a_{21}
a_{22}
a_{23}
a_{24}
a_{31}
a_{32}
a_{33}
a_{34}
a_{41}
a_{42}
a_{43}
a_{44}
a_{51}
a_{52}
a_{53}
a_{54}

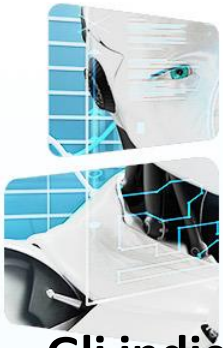


MATRICE

Definizione

Il MIPS non consente una particolare definizione di matrice: si utilizza un vettore con lunghezza *m*x*n* e un indice per accedere alla *i*-esima posizione

```
.data
nrig: .word 10                  # definizione del valore del numero di righe
ncol: .word 10                  # definizione del valore del numero di colonne
matrice: .word 0:100            # area dell'area di memoria che conterrà la matrice
                                #100 words inizializzate a zero
.text
li $t0,0      # in $t0 viene posto l'indice di riga i che assume valori da 0 a 9
li $t1,0      # in $t1 viene posto l'indice di colonna j che assume valori da 0 a 9
lw $t2,nrig             # in $t0 viene posto il numero di righe della matrice
lw $t3,ncol             # in $t1 viene posto il numero di colonne della matrice
```



MATRICE

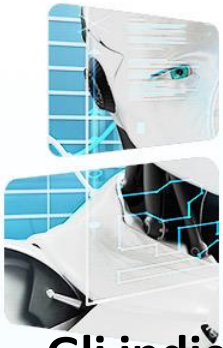
Indici

Gli indici relativi della matrice iniziano dal valore 0 che corrisponde all'elemento (0,0)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}
a_{40}	a_{41}	a_{42}	a_{43}

Gli indici relativi poi devono essere moltiplicati per la dimensione dell'elemento determinato dal tipo (4 per word, 2 per half, 1 per byte)



MATRICE

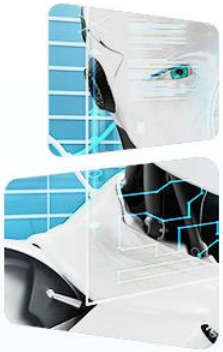
Indici

Gli indici relativi della matrice iniziano dal valore 0 che corrisponde all'elemento (0,0)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}
a_{40}	a_{41}	a_{42}	a_{43}

Gli indici relativi poi devono essere moltiplicati per la dimensione dell'elemento determinato dal tipo (4 per word, 2 per half, 1 per byte)



MATRICE

Inizializzazione in MARS

È possibile **inizializzare una matrice ad un valore specifico** definendo il valore e la sua dimensione con una sintassi del tipo

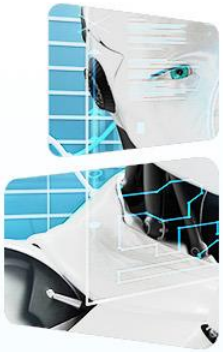
etichetta*: *.tipo valore_di_inizializzazione*:*numero_elementi

Esempio:

matrice: .word 2:20

(matrice 5x4)

2	2	2	2
2	2	2	2
2	2	2	2
2	2	2	2
2	2	2	2



MATRICE

Inizializzazione in MARS

In memoria la matrice inizializzata avrà l'organizzazione fisica in forma vettoriale

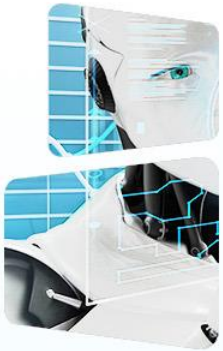
etichetta: .*tipo valore_di_inizializzazione*:*numero_elementi*

Esempio:

matrice: .word 2:20

(matrice 5x4))

a_{11}	2
a_{12}	2
a_{13}	2
a_{14}	2
a_{21}	2
a_{22}	2
a_{23}	2
a_{24}	2
a_{31}	2
a_{32}	2
a_{33}	2
a_{34}	2
a_{41}	2
a_{42}	2
a_{43}	2
a_{44}	2
a_{51}	2
a_{52}	2
a_{53}	2
a_{54}	2



MATRICE

Individuazione di un elemento della matrice

Data una matrice ad R righe e C colonne, come si fa ad individuare a quale elemento corrisponde (r,c) se si usa la notazione classica con prima cella $a_{(1,1)}$?

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}
a_{51}	a_{52}	a_{53}	a_{54}

Posizione elemento

$a_{r,c} = C(r-1)+(c-1)$

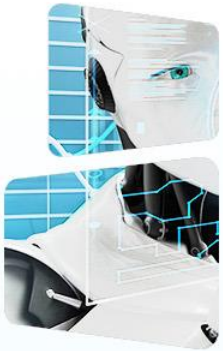
Es: $a_{43}=4(3)+2=14$

MATRICE

Esempio I

Stampa elementi di una matrice

```
for (i=1;i<R;i++)
        {
        for (j=1;j<C;j++)
                {
                elemento= M[i,j]
                print (elemento);
                print("\t")
                }
        print ("\n");
        }
```



MATRICE

Esempio I

```
.text
.globl main
main:
        li $t0,1 #indice r
        li $t1,1 #indice c
        lw $t2,R #numero righe R
        lw $t3,C #numero colonne C
analisi_riga:
        li $t1,1
analisi_colonna:
        sub $t6,$t0,1           #calcolo elemento r,c
        mul $t9,$t6,$t3 #
        sub $t7,$t1,1           #r,c= C(r-1)+(c-1)
        add $t9,$t9,$t7 #
        lb $t8,matrice($t9)     #prelievo elemento
        move $a0,$t8            #stampa elemento
        li $v0,1                #
        syscall                 #
        la $a0,tabulato #stampa tabulato
        li $v0,4
        syscall
        addi $t1,$t1,1          #incremento colonna
        ble $t1,$t3, analisi_colonna
        la $a0,riga             #stampa andata a capo (nuova riga)
        li $v0,4
        syscall
        addi $t0,$t0,1          #incremento riga
        ble $t0,$t2, analisi_riga

li $v0,10
syscall
```

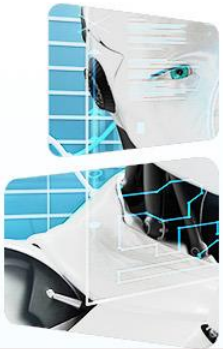
```
.data
matrice: .byte 2:20
R: .word 4
C: .word 5
tabulato:.asciiz "\t"
riga:.asciiz "\n"
```

MATRICE

Esercizio Proposto

Data una matrice 4x4 di interi (word) riportare la somma degli elementi presenti lungo le diagonali

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}



MATRICE

Esercizio Proposto

Data una matrice 4x4 di interi (word) riportare la somma degli elementi presenti lungo le diagonali

45	2534643	34453	3
56474	6	337	8232
943634	670	104	23415
60000	67324	432435	10000

10155+61010=**71165**

MATRICE

Esercizio Proposto

```
        xor $s0,$s0,$s0             #contatore

        li $t0,1                    #indice r
        li $t1,1                    #indice c
        lw $t2,R                    #numero righe R
        lw $t3,C                    #numero colonne C
ciclo1:                 #analisi diagonale principale (sin->dex)
        sub $t6,$t0,1   #calcolo elemento r,c
        mul $t9,$t6,$t3 #
        sub $t7,$t1,1   #           r,c= C(r-1)+(c-1)
        add $t9,$t9,$t7 #
        mul $t9,$t9,4   # dimensione
        lw $t8,matrice($t9)         #prelievo elemento
        add $s0,$s0,$t8             #somma al contatore

        add $t0,$t0,1   #incremento riga
        add $t1,$t1,1   #incremento colonna
        ble $t1,$t3,ciclo1 #confronto con fine colonna (i,j)=(R,C)
```

```
.data
matrice: .word 45,2534643,34453,3,56474,6,337,8232,943634,670,104,23415,60000,67324,432435,10000
R: .word 4
C: .word 4
```

```
li $t0,1 #indice r
li $t1,4 #indice c
lw $t2,R #numero righe R
lw $t3,C #numero colonne C
ciclo2:                 #analisi diagonale secondaria
        sub $t6,$t0,1   #calcolo elemento r,c
        mul $t9,$t6,$t3 #
        sub $t7,$t1,1   #           r,c= C(r-1)+(c-1)
        add $t9,$t9,$t7 #
        mul $t9,$t9,4   # dimensione
        lw $t8,matrice($t9)         #prelievo elemento
        add $s0,$s0,$t8             #somma al contatore
        add $t0,$t0,1               #incremento riga
        sub $t1,$t1,1               #decremento colonna
        ble $t0,$t2,ciclo2          #confornto con fine (i,j)=(R,1)

        move $a0,$s0
        li $v0,1
        syscall
```

FINE