esercitazione 4 - a

stesura e commento by Riccardo Galletti - © www.riccardogalletti.com/appunti_gratis
testo esercitazione prelevato dal sito del prof. Francesco Tortorella http://webuser.unicas.it/tortorella/CalcEl1/Labs/lab4.htm
Scrivere un programma in Assembly MIPS che legga da input gli elementi di un array di interi signed, salvandoli in uno spazio opportunamente allocato nel segmento .data. Il programma determini quindi l'indice del primo elemento negativo presente nell'array.
Si consideri a titolo di riferimento il seguente codice C:

```
int vet[128];
int riemp;

void main()
{ int pos;
 leggi_vet(vet,riemp);
trova_neg(vet,riemp,pos);
 cout << "Posizione primo el. negativo: << pos;
 cout << "\n";}

void leggi_vet(int v[], int &n)
{ int x; int i;
 cout << "Numero elementi: ";
cin >> n;
for(i=0;i<n;i++)
 { cout << "Elemento " << i <<": ";
 cin >> x;
vet[i]=x; }  }

void trova_neg (int v[],int n,int &p)
{ int i,int x;
 i=0;
while(i<n)
{ if(v[i]<0)
{ p=i;
 break; } //se i v[i]<0 esce fuori dal while
 i++;
 } }
```

SVOLGIMENTO

```
.data
vet: .space 512 #alloca 512 byte (128 word) consecutive
riemp: .space 4

 str_n: .asciiz "Numero di elementi: \n"
 str_elem: .asciiz "Elemento "
str_pos: .asciiz "Posizione primo elemento negativo:\n"
 str_accapo: .asciiz "\n"
str_cop: .asciiz " \n\n Created by R. Galletti"

    .text
    .globl main

  main:
addiu $s0, $0, 0  #pos
la $a0, vet  #carica l'indirizzo del primo elemento di vett in $a0
la $a1, riemp #carica l'indirizzo di riemp in $a1
jal leggi_vet #salva l'indirizzo di ritorno in $ra e chiama il sottoprogramma
sw $v1, riemp  #carica in riemp il valore immesso nel sottoprogramma leggi_vet

move $a0, $v0 #preparazione parametri per la chiamata dell'altro sottoprogramma
lw $a1, riemp #devo fare un passaggio per valore
jal trova_neg #salta a trova_neg

move $s0, $v0 # salva il valore restituito da trova_neg

 li $v0, 4
la $a0, str_pos
   syscall #stampa la stringa str_pos

li $v0, 1
move $a0, $s0
 syscall #visualizza pos

li $v0, 4
la $a0, str_cop
   syscall

li $v0,10 # serve per uscire
```

```
    syscall


#-------------------------
leggi_vet: #inizio sottoprogramma
addiu $t0, $0, 0 # x
addiu $t1, $0, 0 # i
addiu $t6, $0, 0 # n
move $t2, $a0 #salva $a0 (altrimenti verrà distrutto)

 li $v0, 4
la $a0, str_n #distruzione di $a0!!
   syscall #stampa la stringa str_n

li $v0, 5
syscall # legge n e lo mette in $v0

sw $v0, 0($a1)  # inserisce all'indirizzo presente in $a1 il valore inserito in console
move $t6, $v0 #copia il valore inserito in n
for:
li $v0, 4
la $a0, str_elem
   syscall #stampa la stringa str_elem

li $v0, 1
move $a0, $t1
 syscall #visualizza il numero i-esimo

   li $v0, 4
la $a0, str_accapo
   syscall #va a capo

li $v0, 5
syscall # legge l'elemento i-esimo e lo metto in $v0
move $t0, $v0  # sposto il valore letto nel registro x

mul $t3, $t1, 4 #fa 4*i senza modificare i
add $t4, $t2, $t3 # mette in $t4 l'indirizzo di vet (i)
sw $t0, 0($t4) #vet(i)=x

addiu $t1, $t1, 1 # i++
blt $t1, $t6, for #ricomincia il for se i<n

move $v0, $t2 # nel parametro di ritorno $v0 scrivo l'indirizzo di vet(0)
move $v1, $a1 # nel parametro di ritorno $v1 scrivo l'indirizzo di riemp
jr $ra #ritorno al main

#-------------------------
trova_neg: #inizio sottoprogramma
addiu $t0, $0, 0 # x
addiu $t1, $0, 0 # i
addiu $t2, $0, 0 # pos

while:
ble $a1, $t1, annanz #vai annanz se n<=i
mul $t3, $t1, 4 #fa 4*i senza modificare i
add $t4, $t3, $a0 #mette in $t4 l'indirizzo di vet (i)
lw $t0, 0($t4) # x=vet(i)
ble $0, $t0, oltre #vai oltre se vet(i)>=0
move $t2, $t1 #pos=i
 j annanz
oltre: addiu $t1, $t1, 1 #i++
j while

annanz:
move $v0, $t2 #nel parametro di ritorno ci metto pos
jr $ra #ritorno al main
                                        # stesura e commento by R. Galletti - © galloimmenso.com
```

esercitazione 4 - b

stesura e commento by Riccardo Galletti - © www.riccardogalletti.com/appunti_gratis
- testo esercitazione prelevato dal sito del prof. Tortorella http://webuser.unicas.it/tortorella/CalcEl1/Labs/lab4.htm

Scrivere un programma in Assembly MIPS che legga da input una stringa di max 63 caratteri e salvati in uno spazio opportunamente allocato nel segmento .data. Il programma costruisca poi una nuova stringa (anch'essa da memorizzare nel segmento .data) ottenuta convertendo in maiuscolo i caratteri alfabetici minuscoli e lasciando inalterati gli altri.
Si consideri a titolo di riferimento il seguente codice C:

```
char string1[64]; //è la stringa di partenza, con i caratteri in minuscolo
char string2[64]; //è la stringa finale, con i caratteri in maiuscolo

 void main()

 { cout << "Stringa: ";
   cin >> string1; //inserimento da console nella stringa 1
   upper(string1,string2); //fa la trasformazione dei caratteri da minuscolo in maiuscolo
   cout << "Stringa in maiuscolo: " << string2;
   cout << "\n";
 }

  void upper(char s1[],char s2[])

 { int i;
   while (s1[i] != '\0')

   {   if((s1[i] >= 'a') && (s1[i] <= 'z')) //se il codice ascii del carattere in s1[i] è compreso tra quello di 'a' e quello di 'b'

       s2[i]=s1[i]-'a'+'A'; //prende il codice ascii del carattere in s1[i], ci toglie il codice di 'a' e ci aggiunge quello di 'A'
                 //e il risultato lo mette in s2[i]: questa operazione è possibile poiché la stessa distanza che esiste
            // tra 'x' (generico carattere in minuscolo) e 'a', è la stessa tra 'X' (corrispondente carattere in maiuscolo) e 'A'

      else
       s2[i]=s1[i];

       i++;

   }

    s2[i]='\0';

 }
```

SVOLGIMENTO

```
   .data
#esercitazione 4 - b
# stesura e commento by R. Galletti - © galloimmenso.com

 .data
char1: .space 64 #alloca 64 bytes (ogni carattere=1 byte)
char2: .space 64

str_a: .ascii "a"
str_z: .ascii "z"
str_a_maiusc: .ascii "A"

 str_1: .asciiz "Stringa: \n"
 str_2: .asciiz "Stringa in maiuscolo:\n"
str_cop: .asciiz " \n\n Created by R. Galletti"

    .text
    .globl main

  main:

 li $v0, 4
la $a0, str_1
   syscall #stampa la stringa str_1

li $v0, 8
la $a0, char1 #carica l'indirizzo iniziale di char1
syscall # legge la stringa e scrive i caratteri in un buffer contenuto in $a0

la $a1, char2  # carica l'indirizzo iniziale di char2
 jal upper #salva l'indirizzo di ritorno in $ra e chiama il sottoprogramma
```

```
li $v0, 4
la $a0, str_2
syscall #visualizza la stringa

li $v0, 4
la $a0, char2
syscall #visualizza la stringa in maiuscolo

li $v0, 4
la $a0, str_cop
  syscall

li $v0,10 # serve per uscire
      syscall


#-------------------------
upper: #inizio sottoprogramma
addiu $t0, $0, 0 # x
addiu $t1, $0, 0 # i
lb $t5, str_a # carica il codice ascii di 'a' in $t5
lb $t6, str_z # carica il codice ascii di 'z' in $t6
lb $t7, str_a_maiusc # carica il codice ascii di 'A' in $t5

while: add $t4, $a0, $t1 # indirizzo di char1[i] in $t4: non è necessario fare 4*i, poichè y è un vettore di byte  e non di word
lb $t0, 0($t4) #x=char1(i)
beq $t0, $0, dopo_while #se char1(i)='\0', esci dal while

blt $t0, $t5, else #se 'a'> char1(i) salta (1° condizione dell'if)
blt $t6, $t0, else  #se 'z'< char1(i) salta (2° condizione dell'if)

add $t3, $a1, $t1 # indirizzo di char2[i] in $t3
sub $t2, $t0, $t5 # char1(i)-'a'
add $t2, $t2, $t7 #$t2= char1(i)-'a'+'A'
sb $t2, 0($t3) # carica in char2[i] il contenuto di $t8
j dopo_else #se si esegue l'if non si deve eseguire anche l'else

else:
add $t3, $a1, $t1 # indirizzo di char2[i] in $t3
sb $t0, 0($t3) #char2(i)=char1(i)

dopo_else: addi $t1, $t1, 1 #i++
j while #ritorno al while

dopo_while:
add $t3, $a1, $t1 # indirizzo di char2[i] in $t3
sb $0, 0($t3) # termina col carattere NULL la stringa

move $v0, $a0 # nel parametro di ritorno $v0 scrivo l'indirizzo di char2[i]
move $v1, $a1 # nel parametro di ritorno $v1 scrivo l'indirizzo di char2[i]
jr $ra #ritorno al main
```

stesura e commento by R. Galletti - © galloimmenso.com