Architetture 2 – Canale E-O – Primo esonero – 28 aprile 2005

ATTENZIONE: dovete scrivere molto chiaramente e commentare bene il codice.

Esercizio 1A (esercizio di sbarramento)

Si scriva un programma assembler MIPS che implementa fedelmente il seguente frammento di codice C (definizioni delle variabili comprese):

```
int i, somma = 0, vettore[10] = {1, 2, 3, 4, 5, 4, 3, 2, 1, 0};
for (i = 1 ; i<10 ; i++)
        somma += vettore[i] – vettore[i-1];
```

Esercizio 2A

Si scriva un programma assembler MIPS che:

- definisce due aree di memoria che contengono due stringhe terminate da zero, ciascuna delle quali ha i caratteri in ordine alfabetico (scegliete voi delle stringhe valide),
- definisce una terza area di memoria di nome **risultato** di dimensioni adeguate
- inserisce nell'area **risultato** una stringa terminata da zero che contiene tutti i caratteri delle due stringhe in ordine alfabetico.

Nota: si tratta del passo "**merge**" dell'algoritmo **merge-sort**.
Esempio: se le due stringhe sono "**abduxz**" e "**cdhkprtwz**" **risultato** sarà "**abcddhkprtuwxzz**"

Esercizio 3A

Si scriva un programma assembler MIPS che:

- definisce due aree di memoria di nome **nrig** ed **ncol** adatte a contenere un valore compreso tra 1 e 53, che indicano rispettivamente il numero di righe e di colonne di una matrice,
- definisce un'area di memoria di nome **matrice** adatta a contenere una matrice di half-words di nrig righe ed ncol colonne,
- riempie la matrice con i valori da 1 a nrig*ncol seguendo il cammino a spirale in senso orario che parte dal primo elemento in alto a sinistra della matrice e termina (circa) al centro.

Esempio: se **nrig** vale **5** ed **ncol** vale **7** la matrice alla fine conterrà i valori:

1	2	3	4	5	6	7
20	21	22	23	24	25	8
19	32	33	34	35	26	9
18	31	30	29	28	27	10
17	16	15	14	13	12	11

Esercizio 4A

Si scriva un programma assembler MIPS che:

- definisce una area di memoria di nome **dizionario** che contiene un vettore di 15 "struct" definite come segue (scegliete voi dei valori validi per valorizzare almeno le prime 5):

```
struct es4 { int valore;        char * chiave; }
```

- definisce una zona di nome **search** contenente una stringa terminata da zero (scelta da voi)
- cerca nel **dizionario** la struct che ha chiave uguale a **search** e:
- stampa il valore corrispondente se **search** viene trovato
- altrimenti stampa la stringa "**non c'è**"

NOTA: per stampare una stringa si mette il suo indirizzo in $a0 e si chiama la **syscall** con $v0 = 4
NOTA: per stampare un intero si mette il suo valore in $a0 e si chiama la **syscall** con $v0 = 1

Architetture 2 – Canale E-O – Primo esonero – 28 aprile 2005

ATTENZIONE: dovete scrivere molto chiaramente e commentare bene il codice.

Esercizio 1B (esercizio di sbarramento)

Si scriva un programma assembler MIPS che implementa fedelmente il seguente frammento di codice C (definizioni delle variabili comprese):

```
int k=1, massimo, vettore[10] = {1, 2, 3, 4, 5, 4, 3, 2, 1, 0};
massimo = vettore[0];
do {
        if (massimo < vettore[k])        massimo = vettore[k];
        k++;
} while (k<10);
```

Esercizio 2B

Si scriva un programma assembler MIPS che:

- definisce due aree di memoria che contengono due stringhe terminate da zero (scegliete voi delle stringhe valide),
- definisce una terza area di memoria di nome **risultato** di dimensioni adeguate
- inserisce nell'area **risultato** una stringa terminata da zero che contiene nell'ordine tutti i caratteri della seconda stringa che non appaiono nella prima stringa.

Esempio: se le due stringhe sono "**Pip po**" e "**Topo lino**" il **risultato** sarà "**Tln**"

Esercizio 3B

Si scriva un programma assembler MIPS che:

- definisce due aree di memoria di nome **nrig** ed **ncol** adatte a contenere un valore compreso tra 1 e 37, che indicano rispettivamente il numero di righe e di colonne di una matrice,
- definisce un'area di memoria di nome **matrice** adatta a contenere una matrice di words di **nrig** righe ed **ncol** colonne,
- riempie la matrice con i valori da 1 a nrig*ncol seguendo il cammino del passo del cavallo (2 a destra e 1 in basso) partendo dal primo elemento in alto a sinistra della matrice e continuando attorno alla matrice come se l'ultima colonna a destra fosse adiacente alla prima a sinistra e se sotto l'ultima riga si continuasse con la prima.

Esempio: se **nrig** vale **4** ed **ncol** vale **5** la matrice alla fine conterrà i valori:

1	9	17	5	13
6	14	2	10	18
11	19	7	15	3
16	4	12	20	8

Esercizio 4B

Si scriva un programma assembler MIPS che:

- definisce una area di memoria di nome **gente** che contiene un vettore di **13** "struct" definite come segue (scegliete voi dei valori validi per valorizzare almeno le prime 5):

  ```
  struct es4 { char * nome;        int altezza;        int peso }
  ```

- cerca nel vettore **gente** la persona più magra (che ha rapporto altezza/peso maggiore)
- stampa il nome ed il rapporto calcolato della persona trovata

NOTA: per stampare una stringa si mette il suo indirizzo in $a0 e si chiama la **syscall** con $v0 = 4
NOTA: per stampare un intero si mette il suo valore in $a0 e si chiama la **syscall** con $v0 = 1

Soluzioni

Esercizio 1A

Si scriva un programma assembler MIPS che implementa fedelmente il seguente frammento di codice C (definizioni delle variabili comprese):

```
int i, somma = 0, vettore[10] = {1, 2, 3, 4, 5, 4, 3, 2, 1, 0};
for (i = 1 ; i<10 ; i++)
        somma += vettore[i] – vettore[i-1];
```

Soluzione

Bisogna ricordarsi che il **for** fa il test PRIMA di eseguire il blocco delle istruzioni. Per ottimizzare il codice ho preferito scrivere le variabili in memoria solo alla fine del ciclo.

```
            .data
i:          .word 0                         # int i
somma:      .word 0                         # int somma = 0
vettore:    .word 1, 2, 3, 4, 5, 4, 3, 2, 1, 0   # int vettore[10] = {1, 2, 3, 4, 5, 4, 3, 2, 1, 0}

            .text
            ...                     # (questo è un frammento di codice)
            li $t0, 1               # uso $t0 per il valore della variabile i (e la inizializzo a 1)
            lw $s0, somma           # carico il valore iniziale della somma
            li $t1, 10              # metto la costante 10 in un registro per fare il confronto
for:        bge $t0, $t1, esci      # se i non è < di 10 esco dal ciclo
            mul $t2, $t0, 4         # calcolo l'offset in memoria dell'indice i (nel vettore di word)
            lw $t3, vettore($t2)    # leggo il valore di vettore[i]
            sub $t4, $t0, 1         # calcolo i-1
            mul $t5, $t4, 4         # calcolo l'offset dell'indice i-1
            lw $t6, vettore($t6)    # leggo il valore di vettore[i-1]
            sub $t3, $t3, $t6       # calcolo la differenza vettore[i]-vettore[i-1]
            add $s0, $s0, $t3       # accumulo la somma
            add $t2, $t2, 1         # incremento i
            b for                   # continuo il ciclo
esci:       sw $s0, somma           # memorizzo la somma
            sw $t0, i               # memorizzo i
            ...                     # (continua)
```

Esercizio 2A

Si scriva un programma assembler MIPS che:
- definisce due aree di memoria che contengono due stringhe terminate da zero, ciascuna delle quali ha i caratteri in ordine alfabetico (scegliete voi delle stringhe valide),
- definisce una terza area di memoria di nome **risultato** di dimensioni adeguate
- inserisce nell'area **risultato** una stringa terminata da zero che contiene tutti i caratteri delle due stringhe in ordine alfabetico.

Nota: si tratta del passo "**merge**" dell'algoritmo **merge-sort**.
Esempio: se le due stringhe sono "**abduxz**" e "**cdhkprtwz**" **risultato** sarà "**abcddhkprtuwxzz**"

Soluzione

I codici ascii crescenti dei caratteri sono in ordine alfabetico, quindi per sapere quale carattere viene prima di un altro basta confrontarne i valori. Per costruire la stringa finale basta confrontare i due caratteri iniziali delle due stringhe e copiare il minore dei due, e poi incrementare l'indice della stringa dalla quale si è preso il carattere. Se una delle stringhe è finita si copiano i caratteri dell'altra. Se entrambe sono finite si è finito il programma.

```
            .data
prima:      .asciiz "aegilpz"          # una stringa (di 7 caratteri)
seconda:    .asciiz "bgmrstyz"         # una seconda stringa (di 8 caratteri)
risultato:  .byte 0:16                 # devono esserci un numero di caratteri sufficienti +1 per lo zero

            .text
            .globl main
main:       lw $t0, 0                  # indice del carattere corrente della prima stringa
            lw $t1, 0                  # indice del carattere corrente della seconda stringa
            lw $t2, 0                  # indice del carattere corrente del risultato
ciclo:      lb $s0, prima($t0)         # carico il carattere dalla prima stringa
            lb $s1, seconda($t1)       # carico il carattere dalla seconda stringa
            bez $s0, copiaseconda      # se la prima stringa è finita copio il carattere della seconda
                                       # altrimenti la prima stringa non è finita e ...
            bez $s1, copiaprima        # se la seconda stringa è finita copio il carattere della prima
                                       # altrimenti entrambe le stringhe non sono finite
            blt $s0, $s1, copiaprima           # se il carattere della prima viene prima lo copio
                                       # altrimenti copio il carattere della seconda nel risultato
copiaseconda:                          # posso arrivare qui perchè entrambe le stringhe sono finite
            sb $s1, risultato($t2)     # metto il carattere della seconda stringa nel risultato
            bez $s1, esci              # controllo se devo uscire (anche la seconda stringa è finita)
            add $t2, $t2, 1            # altrimenti incremento l'indice del risultato
            add $t1, $t1, 1            # e incremento l'indice della seconda stringa
            b ciclo
copiaprima:                            # se arrivo qui la prima stringa sicuramente ha ancora caratteri
            sb $s0, risultato($t2)     # metto il carattere della prima stringa nel risultato
            add $t2, $t2, 1            # incremento l'indice del risultato
            add $t0, $t0, 1            # incremento l'indice della prima stringa
            b ciclo
esci:       li $v0, 10                 # fine della copia
            syscall
```

Esercizio 3A

Si scriva un programma assembler MIPS che:

- definisce due aree di memoria di nome **nrig** ed **ncol** adatte a contenere un valore compreso tra 1 e 53, che indicano rispettivamente il numero di righe e di colonne di una matrice,
- definisce un'area di memoria di nome **matrice** adatta a contenere una matrice di half-words di nrig righe ed ncol colonne,
- riempie la matrice con i valori da 1 a nrig*ncol seguendo il cammino a spirale in senso orario che parte dal primo elemento in alto a sinistra della matrice e termina (circa) al centro.

Esempio: se **nrig** vale **5** ed **ncol** vale **7** la matrice alla fine conterrà i valori:

1	2	3	4	5	6	7
20	21	22	23	24	25	8
19	32	33	34	35	26	9
18	31	30	29	28	27	10
17	16	15	14	13	12	11

Soluzione

Per fare la spirale possiamo partire dalla matrice piena di zero, poi muoverci verso DX e cambiare direzione ogni volta che si raggiunge il bordo o un elemento già riempito. Per ricordare la direzione corrente possiamo usare un indice da 0 a 3 e memorizzare in una coppia di vettori (inccol e incrig) di quanto dobbiamo incrementare (o decrementare) gli indici di riga e colonna per ciascuna delle 4 direzioni. Per evitare di calcolare ogni volta la posizione dell'elemento corrente in memoria potremmo precalcolare i 4 incrementi (decrementi) da fare in memoria come numero di byte (ottenendo inccol={2, 0, -2, 0}e incrig={0, 2*ncol, 0, -2*ncol}), ma nel programma non lo faccio.

```
            .data
nrig:       .word 30
ncol:       .word 23
matrice:    .half 0:2809            # alloco almeno 53x53 elementi inizializzati col valore zero
inccol:     .word 1, 0, -1, 0       # incrementi dell'indice di colonna nelle 4 direzioni
incrig:     .word 0, 1, 0, -1       # incrementi dell'indice di riga nelle 4 direzioni

            .text
            .globl main
main:       li $t0, 1               # metto in $t0 il primo valore da inserire
            li $t1, 0               # in $t1 metto l'indice iniziale di colonna
            li $t2, 0               # in $t2 metto l'indice iniziale di riga
            lw $t3, ncol            # carico il numero di colonne
            lw $t4, nrig            # ed il numero di righe
            mul $t5, $t3, $t4       # calcolo il valore finale (nrig*ncol)
            li $t6, 0               # all'inizio vado in direzione 0=DX
            lw $t7, inccol($t6)     # carico l'incremento verso destra
            lw $t8, incrig($t6)     # carico l'incremento verso il basso
ciclo:      bgt $t0, $t5, uscita    # se ho già inserito il valore (nrig*ncol) esco
            beq $t1, $t3, svolta    # se ho finito la riga giro a destra
            beq $t2, $t4, svolta    # se ho finito la colonna giro a destra
            blt $t1, $0, svolta     # se sono a colonna -1 giro a destra
            mul $s0, $t3, $t2       # calcolo la posizione dell'elem corrente
            add $s0, $s0, $t1       #
            mul $s0, $s0, 2         # moltiplico per la dimensione di una half
```

```
        lh $s1, matrice($s0)    # leggo il valore dell'elemento
        bgt $s1, $0, svolta     # se è già stato riempito giro a destra
        sh $t0, matrice($s0)    # altrimenti memorizzo il valore corrente
        add $t0, $t0, 1         # incremento il valore corrente
        add $t1, $t1, $t7       # passo al prossimo elemento della riga
        add $t2, $t2, $t8       # passo al prossimo elemento della colonna
        b ciclo                 # continuo
svolta:                         # per svoltare devo:
        sub $t1, $t1, $t7       # tornare un passo indietro
        sub $t2, $t2, $t8       # tornare un passo indietro
        add $t6, $t6, 1         # passare alla prossima direzione
        rem $t6, $t6, 4         # ma calcolando $t6 modulo 4
        lw $t7, inccol($t6)     # carico il nuovo incremento verso destra
        lw $t8, incrig($t6)     # carico il nuovo incremento verso il basso
        add $t1, $t1, $t7       # passo al prossimo elemento della riga
        add $t2, $t2, $t8       # passo al prossimo elemento della colonna
        b ciclo                 # continuare
uscita: li $v0, 10              # fine
        syscall
```

Esercizio 4A

Si scriva un programma assembler MIPS che:

- definisce una area di memoria di nome **dizionario** che contiene un vettore di 15 "struct" definite come segue (scegliete voi dei valori validi per valorizzare almeno le prime 5):
  struct es4 { int valore; char * chiave; }
- definisce una zona di nome **search** contenente una stringa terminata da zero (scelta da voi)
- cerca nel **dizionario** la struct che ha chiave uguale a **search** e:
- stampa il valore corrispondente se **search** viene trovato
- altrimenti stampa la stringa "**non c'è**"

NOTA: per stampare una stringa si mette il suo indirizzo in $a0 e si chiama la **syscall** con $v0 = 4
NOTA: per stampare un intero si mette il suo valore in $a0 e si chiama la **syscall** con $v0 = 1

Soluzione

Bisogna ricordare che un vettore di struct è una sequenza di gruppi di elementi e che un puntatore ad una stringa non è altro che il suo indirizzo. Per cercare la chiave esaminiamo una per volta le coppie di stringhe <search**,**chiave_corrente> scandendone i caratteri per verificare se sono uguali. Notate come ho fatto per accedere ai campi valore e chiave usando due etichette aggiuntive e calcolando lo spiazzamento corretto (indice della struct per dimensioni della struct).

```
            .data
# stringhe usate nelle struct
chiave1:    .asciiz "prima chiave"
chiave2:    .asciiz "seconda chiave"
chiave3:    .asciiz "terza chiave"
chiave4:    .asciiz "quarta chiave"
chiave5:    .asciiz "quinta chiave"
            ...                         # continua per altre 10 stringhe
dizionario:
valore:     .word 1                     # campo valore della prima struct
chiave:     .word chiave1               # campo chiave della seconda struct
            .word 2
            .word chiave2
            .word 3
            .word chiave3
            .word 4
            .word chiave4
            .word 5
            .word chiave5
            ...                         # continua per altre 10 struct
search:     .asciiz "sesta chiave"
nonce:      .asciiz "non c'è"
numelem:    .word 15

            .text
            .globl main
main:       li $t0, 0           # in $t0 tengo l'indice della struct corrente
            lw $t1, numelem     # leggo il numero di elementi del vettore
ciclostruct: beq $t0, $t1, manca # se ho raggiunto il fondo la chiave non c'è
            la $t2, search      # carico l'indirizzo del primo carattere della chiave
            mul $s0, $t0, 8     # calcolo lo spiazzamento della struct corrente
```

```
        move $s1, $s0           # lo copio per usarlo per scorrere la chiave corrente
ciclo_chiavi:
        lb $t3, ($t2)           # carico il carattere della search
        lb $t4, chiave($s1)     # carico il carattere della chiave corrente
        bneq $t3, $t4, prossima_struct      # se differiscono vado alla prossima struct
        beqz $t3, trovata       # se entrambe le stringhe sono finite l'ho trovata
        add $s1, $s1, 1         # altrimenti avanzo di un carattere della chiave
        add $t2, $t2, 1         # e avanzo di un carattere della seearch
        b ciclo_chiavi          # continuo il confronto della chiave corrente
prossima_struct:
        add $t0, $t0, 1         # passo alla prossima struct
        b ciclo_struct
trovata:
        lw $a0, valore($s0)     # carico il valore dalla struct trovata (ha lo stesso offset)
        li $v0, 1               # per stampare un intero il codice è 1
        syscall
        li $v0, 10              # fine
        syscall
manca:
        la $a0, nonce           # carico l'indirizzo della stringa “non c'è”
        li $v0, 4               # 4 = print_string
        syscall
        li $v0, 10              # fine
        syscall
```

Esercizio 1B

Si scriva un programma assembler MIPS che implementa fedelmente il seguente frammento di codice C (definizioni delle variabili comprese):

```
int k=1, massimo, vettore[10] = {1, 2, 3, 4, 5, 4, 3, 2, 1, 0};
massimo = vettore[0];
do {
        if (massimo < vettore[k])       massimo = vettore[k];
        k++;
} while (k<10);
```

Soluzione

Si noti che il do-while fa il test DOPO aver eseguito il blocco di istruzioni. Per ottimizzare il codice ho preferito scrivere le variabili in memoria solo alla fine del ciclo.

```
            .data
k:          .word 1                         # int k=1
massimo:    .word 0                         # int massimo
vettore:    .word 1, 2, 3, 4, 5, 4, 3, 2, 1, 0  # int vettore[10] = {1, 2, 3, 4, 5, 4, 3, 2, 1, 0}

            .text
            ...                 # questo è un frammento di codice (senza main)
            li $t0, 10          # carico il numero di elementi (10) che mi serve per il confronto
            lw $t1, k           # carico il valore k
            lw $t2, vettore     # carico vettore[0] in un registro che uso per massimo
do:         mul $t3, $t1, 4     # calcolo lo spiazzamento dell'elemento vettore[k]
            lw $t4, vettore($t3)# carico l'elemento corrente (vettore[k])
            bge $t2, $t4, endif # se il massimo è minore dell'elemento corrente vado avanti
            move $t2, $t4       # altrimenti aggiorno il massimo
endif:      add $t1, $t1, 1     # incremento k
            blt $t1, $t0, do    # continuo il ciclo se k < 10
            ...                 # altrimenti ho finito il ciclo e vado avanti
```

Esercizio 2B

Si scriva un programma assembler MIPS che:

- definisce due aree di memoria che contengono due stringhe terminate da zero, ~~ciascuna delle quali ha i caratteri in ordine alfabetico~~ (scegliete voi delle stringhe valide),
- definisce una terza area di memoria di nome **risultato** di dimensioni adeguate
- inserisce nell'area **risultato** una stringa terminata da zero che contiene nell'ordine tutti i caratteri della seconda stringa che non appaiono nella prima stringa.

Esempio: se le due stringhe sono "**Pip po**" e "**Topo lino**" il **risultato** sarà "**Tln**"

Soluzione

Basta usare due cicli nidificati che per ogni carattere della seconda stringa, spazzolano la prima.
Se il carattere viene trovato lo si ignora, altrimenti lo si copia nel risultato.

```
            .data
prima:      .asciiz "Pip po"
seconda:    .asciiz "Topo lino"
risultato:  .space 10               # nel caso peggiore ci vogliono tanti caratteri quanti seconda

            .text
            .globl main
main:       li $t0, 0               # indice nella seconda stringa
            li $t1, 0               # indice del risultato
ciclo_sec:  lb $t2, seconda($t0)    # sto esaminando il carattere corrente della seconda stringa
            beqz $t2, fine          # se è zero ho finito
            li $t3, 0               # altrimenti scorro la prima stringa
ciclo_prim: lb $t4, prima($t3)      # carico il carattere corrente della prima stringa
            beqz $t4, copia         # è finita, posso copiare il carattere nel risultato
            add $t3, $t3, 1         # vado avanti con la scansione
            b ciclo_prim
copia:
            sb $t2, risultato($t1)  # copio il carattere nel risultato
            add $t1, $t1, 1         # e incremento l'indice del risultato
            add $t0, $t0, 1         # e quello della seconda stringa
            b ciclo_sec
fine:       sb $0, risultato($t1)   # termino la stringa con zero
            li $v0, 10              # termino il programma
            syscall
```

Esercizio 3B

Si scriva un programma assembler MIPS che:

- definisce due aree di memoria di nome **nrig** ed **ncol** adatte a contenere un valore compreso tra 1 e 37, che indicano rispettivamente il numero di righe e di colonne di una matrice,
- definisce un'area di memoria di nome **matrice** adatta a contenere una matrice di words di **nrig** righe ed **ncol** colonne,
- riempie la matrice con i valori da 1 a nrig*ncol seguendo il cammino del passo del cavallo (2 a destra e 1 in basso) partendo dal primo elemento in alto a sinistra della matrice e continuando attorno alla matrice come se l'ultima colonna a destra fosse adiacente alla prima a sinistra e se sotto l'ultima riga si continuasse con la prima.

Esempio: se **nrig** vale **4** ed **ncol** vale **5** la matrice alla fine conterrà i valori:

1	9	17	5	13
6	14	2	10	18
11	19	7	15	3
16	4	12	20	8

Soluzione

In questo caso conviene notare che lo spostamento del cavallo corrisponde a incrementare la posizione corrente in memoria di una riga e di due elementi, ovvero di (ncol+2)*4 byte.
Il fatto che la matrice si avvolga su se stessa sia verso destra che verso il basso corrisponde a calcolare l'incremento **modulo la dimensione della matrice** (ovvero (nrig*ncol*4) byte).

```
            .data
nrig:       .word 17
ncol:       .word 21
matrice:    .word 0:1369            # alloco 37*37=1379 elementi col valore 0

            .text
            .globl main
main:
            li $t0, 1               # valore corrente da inserire
            lw $t1, nrig            # numero di righe
            lw $t2, ncol            # numero di colonne
            mul $t3, $t1, $t2       # massimo numero da inserire
            mul $t4, $t3, 4         # numero di byte con cui fare il modulo
            add $t5, $t2, 2         # numero di elementi per fare la mossa
            mul $t5, $t5, 4         # incremento della mossa in byte
            li $t6, 0               # si parte da offset 0 (il primo elemento) della matrice
ciclo:
            sw $t0, matrice($t6)    # memorizzo il valore nella locazione corrente
            add $t0, $t0, 1         # incremento il valore
            add $t6, $t6, $t5       # faccio la mossa
            rem $t6, $t6, $t4       # calcolo il modulo per rimanere nella matrice
            ble $t0, $t2, ciclo     # se non ho inserito tutti gli elementi continuo

            li $v0, 10              # altrimenti ho finito
            syscall
```

Esercizio 4B

Si scriva un programma assembler MIPS che:

- definisce una area di memoria di nome **gente** che contiene un vettore di **13** "struct" definite come segue (scegliete voi dei valori validi per valorizzare almeno le prime 5):
  struct es4 { char * nome; int altezza; int peso }
- cerca nel vettore **gente** la persona più magra (che ha rapporto altezza/peso maggiore)
- stampa il nome ed il rapporto calcolato della persona trovata

NOTA: per stampare una stringa si mette il suo indirizzo in $a0 e si chiama la **syscall** con $v0 = 4
NOTA: per stampare un intero si mette il suo valore in $a0 e si chiama la **syscall** con $v0 = 1

Soluzione

Bisogna ricordare che un vettore di struct è una sequenza di gruppi di elementi e che un puntatore ad una stringa non è altro che il suo indirizzo. Notate come ho fatto per accedere ai campi nome, altezza e peso usando tre etichette aggiuntive e calcolando lo spiazzamento corretto (indice della struct per dimensioni della struct).

```
            .data
nome1:      .asciiz "Nome1"         # stringhe usate nelle struct
nome2:      .asciiz "Nome2"
nome3:      .asciiz "Nome3"
nome4:      .asciiz "Nome4"
nome5:      .asciiz "Nome5"
            ...                     # segue per altre 8 stringhe

gente:                              # inizio del vettore di struct
nome:       .word nome1             # campo nome della prima struct
altezza:    .word 186               # campo altezza della prima struct
peso:       .word 90                # campo peso della prima struct
            .word nome2
            .word 160
            .word 44
            .word nome3
            .word 168
            .word 54
            .word nome4
            .word 190
            .word 124
            .word nome5
            .word 175
            .word 60
            ...                     # segue per altre 8 struct
nelem:      .word 13                # numero di elementi

            .text
            .globl main
main:
            li $t0, 0               # indice della prima struct
            li $s0, 0               # spazio per ricordare qual'era l'offset della struct del massimo
            li $t1, 0               # valore iniziale del massimo
            lw $t2, nelem           # carico il numero di elementi
```

```
        mul $t2, $t2, 12        # calcolo lo spiazzamento massimo che posso raggiungere
ciclo:
        lw $t3, altezza($t0)    # carico l'altezza corrente
        lw $t4, peso($t0)       # ed il peso
        div $t5, $t3, $t4       # calcolo altezza/peso
        blt $t5, $t1, minore    # se è minore del massimo corrente lo ignoro
        move $t1, $t5           # altrimenti aggiorno il massimo
        move $s0, $t0           # e ricordo il suo offset
minore: add $t0, $t0, 12        # incremento l'offset
        blt $t0, $t2, ciclo     # se non ho finito continuo

        la $a0, nome($s0)       # prendo l'indirizzo del nome del massimo
        li $v0, 4               # print_string ha il codice 4
        syscall
        move $a0, $t1           # prendo il massimo
        li $v0, 1               # print_integer ha il codice 1
        syscall                 # e lo stampo
        li $v0, 10              # fine
        syscall
```