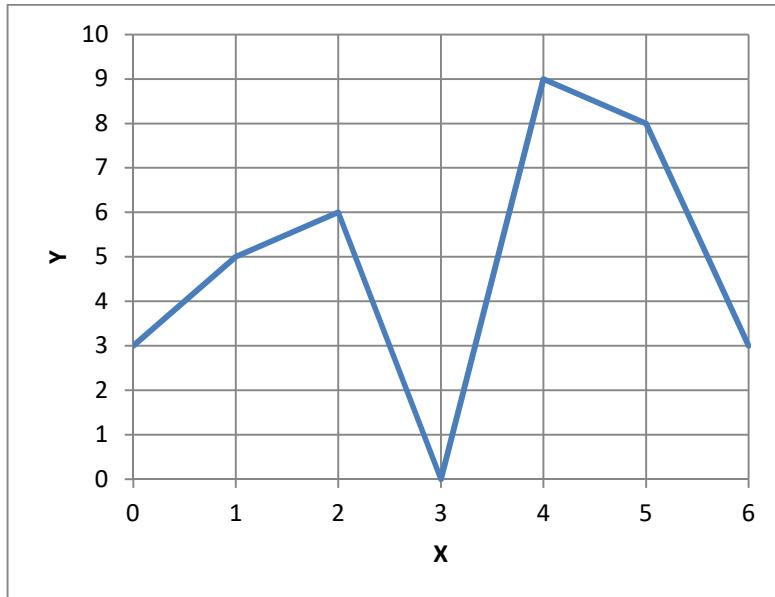


Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Si scriva una procedura **calcola** in linguaggio Assembly 8086 in grado di effettuare il calcolo dell'area sottesa da una funzione definita per punti come in figura.



Siano dati:

- un vettore di DIM elementi *word* *vet*, contenente il valore della funzione (Y) per ciascun valore intero consecutivo di X. Ciascun elemento del vettore è un intero positivo < 32768, mentre DIM è definito come costante.
- una variabile *doubleword* *res* che dovrà contenere il valore dell'integrale risultante.

Il risultato finale deve avere un'approssimazione massima di ± 0.5 . Si utilizzi il passaggio di parametri tramite *stack*, in modo che la procedura sia richiamabile dalle seguenti istruzioni:

```
[...]
LEA SI, vet
PUSH SI
SUB SP, 4
CALL calcola
POP res
POP res[2]
ADD SP, 2
[...]
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Sia data una matrice quadrata di *word* memorizzata per righe (numero di righe pari a DIM, con DIM dichiarato come costante).

Si scriva in linguaggio Assembly 8086 una **procedura valmatr** che sia in grado di valutare se la matrice quadrata è simmetrica o diagonale, ricevendone in ingresso l'indirizzo del primo elemento attraverso lo *stack*. La procedura dovrà restituire, sempre mediante lo *stack*, un valore pari a:

- 2 se la matrice è diagonale
- 1 se la matrice è simmetrica
- 0 se la matrice non è simmetrica.

Di seguito un esempio di programma chiamante:

```
[...]
LEA BX, matrix
PUSH BX
SUB SP, 2
CALL valmatr
POP AX          ; risultato in AX
ADD SP, 2
[...]
```

Si richiede specificamente che non siano utilizzate variabili aggiuntive di supporto.

Si ricorda che in una matrice diagonale solamente i valori della diagonale principale possono essere diversi da 0, mentre una matrice simmetrica ha la proprietà di essere la trasposta di se stessa.

Esempio di matrice diagonale:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{bmatrix}$$

Esempio di matrice simmetrica:

$$\begin{bmatrix} 1 & 4 & 5 & 6 & 7 \\ 4 & 2 & 8 & 6 & 4 \\ 5 & 8 & 3 & 2 & 9 \\ 6 & 6 & 2 & 4 & 4 \\ 7 & 4 & 9 & 4 & 5 \end{bmatrix}$$

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

In relazione alla misura e al modo di determinarne l'ammontare, le imposte si dividono in fisse, proporzionali, progressive e regressive. Si ha un'*imposta progressiva per scaglioni* quando il reddito imponibile viene suddiviso in parti dette *scaglioni* a ciascuna delle quali viene associata un'aliquota che cresce passando da uno scaglione a quello successivo. Si veda il seguente esempio:

Scaglioni	Aliquote
da 0 a 15.000	23%
oltre 15.000 fino a 28.000	27%
oltre 28.000 fino a 55.000	38%
oltre 55.000	41%

Un **reddito** pari a 23.000 euro sarà così suddiviso:

- 15.000 Euro rientrano nel primo scaglione, cui corrisponde un'imposta pari a 3.450 Euro
- 8.000 Euro rientrano nel secondo scaglione, cui corrisponde un'imposta pari a 2.160 Euro.

L'**imposta** sarà quindi pari a 5.610 euro.

Siano dati due vettori di *word* in memoria (variabili globali) **scaglione** e **aliquota**, che rappresentino rispettivamente la soglia bassa di ogni scaglione e la relativa aliquota, di dimensione pari alla costante **DIM**. Si scriva una procedura in Assembly 8086 in grado di calcolare l'ammontare dell'imposta dato il valore del reddito, ricevendo tale parametro mediante lo *stack*. La procedura deve restituire il risultato sempre mediante *stack*. Sia lecito supporre che il reddito massimo possa essere rappresentato mediante una *word unsigned*. Di seguito un esempio di programma chiamante, con gli stessi dati dell'esempio precedente:

```
DIM EQU 4
.model small
.stack
.data

scaglione dw 0, 15000, 28000, 55000
aliquota dw 23, 27, 38, 41

.code
.startup

    ...
    MOV AX, 23000 ; valore del reddito imponibile
    PUSH AX
    SUB SP, 2      ; spazio riservato per il valore di ritorno
    CALL calc_tax
    POP BX         ; prelevamento del valore di ritorno dallo stack
    ADD SP, 2

    ...

.exit
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Si scriva una **procedura sostituisce** in grado di espandere una stringa precedentemente inizializzata sostituendo tutte le occorrenze del carattere '%' con un'altra stringa data. Siano date quindi le seguenti tre stringhe in memoria:

- **str_orig**, corrispondente al testo compresso da espandere
- **str_sost**, contenente la il testo da sostituire in **str_orig** al posto di '%'
- **str_new**, che conterrà la stringa espansa (si supponga che abbia dimensione sufficiente a contenerla).

Di seguito un esempio di funzionamento:

str_orig db "% nella citta' dolente, % nell'eterno dolore, % tra la perduta gente"
str_sost db "per me si va"

str_new sarà quindi

"per me si va nella citta' dolente, per me si va nell'eterno dolore, per me si va tra la perduta gente"

La procedura lavora sulle stringhe come *variabili globali*, riceve mediante *stack* la lunghezza di **str_orig** e di **str_sost**, e restituisce, sempre mediante *stack*, la lunghezza della nuova stringa. Di seguito un esempio di programma chiamante:

```
[...]
PUSH 68          ; lunghezza str_orig
PUSH 12          ; lunghezza str_sost
PUSH AX          ; lunghezza stringa finale
call sostituisce
POP lung_new
ADD SP, 4
[...]
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Un test universitario prevede la somministrazione di N domande a risposta multipla a un insieme di M studenti ($N \leq 255$, $M \leq 255$). Ogni risposta è associata a una lettera alfabetica (a, b, c, d). Si supponga che ogni studente risponda a tutte le N domande.

Si scriva una **procedura valutazione** in linguaggio **Assembly 8086** in grado di calcolare il numero di risposte corrette per ogni studente, confrontandole con i valori di riferimento, e di elaborare una statistica della valutazione effettuata, tenendo conto delle seguenti strutture dati:

- **tabella** – matrice di $(N+1)*M$ byte contenente, per ciascuno studente, le N risposte, più un byte contenente il numero di risposte corrette; la matrice è memorizzata per righe in un vettore
- **res** – vettore di byte contenente le N risposte corrette del test
- **stat** – vettore di N byte che fornisce una statistica della valutazione degli studenti: in prima posizione conterrà il numero di studenti che hanno fornito 0 risposte giuste, nella seconda il numero di chi ha dato 1 risposta giusta, ecc.

La procedura riceve **stat** e **res** come variabili globali, mentre riceve l'indirizzo di **tabella** e il valore di M e N mediante **stack**. La procedura deve aggiornare il valore dell'ultimo byte di ciascuna riga di **tabella** e il vettore **stat**.

Di seguito un esempio di funzionamento (caso $N = 9$, $M = 6$):

Prima dell'esecuzione della procedura	Dopo
tabella db 'abbacbad', ?	tabella db 'abbacbad', 7
db 'abbccdbad', ?	db 'abbccdbad', 5
db 'bbbcaccdca', ?	db 'bbbcaccdca', 3
db 'abbcdbadd', ?	db 'abbcdbadd', 9
db 'bcbccbadd', ?	db 'bcbccbadd', 6
db 'acbacad', ?	db 'acbacad', 5
stat db N+1 dup (?)	stat db 0, 0, 0, 1, 0, 2, 1, 1, 0, 1
res db 'abbcdbadd'	res db 'abbcdbadd'

Di seguito un esempio di programma chiamante:

```
[...]
LEA AX, tabella
PUSH AX
PUSH M
PUSH N
CALL valutazione
ADD SP, 6
[...]
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Data una matrice di *byte* di DIMX righe e DIMY colonne, contenente valori 1 o 0, si scriva una **procedura valuta1** in grado di contare il numero di colonne di soli valori 1 presenti.

Ad esempio, nel caso

```
0,0,1,0,0,1,1,0,1,0  
0,1,1,0,0,0,1,0,1,1  
1,1,0,0,1,0,1,0,1,0  
1,0,1,0,0,1,1,0,1,0  
1,1,1,1,1,1,1,1,1,1  
1,1,1,0,0,1,1,1,1,1,1  
1,1,1,0,1,1,1,1,1,0  
1,1,0,1,1,1,0,1,0
```

sono presenti 2 colonne che soddisfano la richiesta.

La procedura riceve tramite *stack* i parametri su cui deve lavorare nel seguente modo:

- *offset* della matrice
- numero di righe (DIMX)
- numero di colonne (DIMY).

Il risultato deve essere restituito tramite *stack*. Di seguito un esempio di programma chiamante:

```
[...]  
PUSH OFFSET matrice  
PUSH DIMX  
PUSH DIMY  
PUSH 0           ; spazio per valore di ritorno  
CALL valuta1  
POP  AX  
ADD  SP, 6  
[...]
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Sia data una matrice quadrata di byte di dimensione **DIMxDIM** (**DIM** dichiarato come costante) memorizzata per righe. Gli elementi della matrice contengono il valore 0 o il valore 1.

Si vuole verificare se la matrice contiene almeno un quadrato **3x3** di 9 elementi contigui aventi valore 1.

Si scriva una procedura **cercaQuadrato** che riceve in input

- in AX l'indice progressivo di un elemento della matrice nel vettore che la memorizza
- in BX l'offset nel data segment del primo elemento della matrice.

La procedura restituisce in output (attraverso il registro DX) uno fra i due valori seguenti:

- 1, se l'elemento il cui indice è stato ricevuto in input corrisponde all'elemento centrale di un quadrato **3x3** di elementi con valore 1
- 0, altrimenti.

Si assuma che tutti gli elementi della prima e ultima colonna, nonché della prima e ultima riga della matrice contengano il valore 0.

Ad esempio, nel caso

```
0, 0, 0, 0, 0, 0, 0, 0, 0  
0, 0, 0, 0, 1, 1, 0, 0, 0  
0, 0, 0, 0, 1, 1, 0, 0, 0  
0, 0, 0, 0, 0, 0, 0, 0, 0  
0, 0, 1, 1, 1, 1, 0, 0, 0  
0, 0, 1, 1, 1, 1, 0, 0, 0  
0, 0, 1, 1, 1, 1, 0, 0, 0  
0, 0, 0, 0, 1, 0, 0, 0, 0  
0, 0, 0, 0, 0, 0, 0, 0, 0
```

la procedura **cercaQuadrato** restituisce 1 se chiamata con AX = 48, 0 in tutti gli altri casi.

Di seguito un esempio di programma che chiama la procedura **cercaQuadrato** per ogni elemento della matrice **matrice**, finché non è rilevato un quadrato:

```
[...]  
XOR AX, AX ; indice dell'elemento corrente  
LEA BX, matrice  
inizioCiclo:  
CALL cercaQuadrato  
CMP DX, 1  
JE fineCiclo  
INC AX  
CMP AX, DIM * DIM  
JL inizioCiclo  
fineCiclo:  
[...]
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Sia dato un vettore *vett_input* di interi con segno su 16 bit di dimensione N (N dichiarato come costante), che rappresenta una funzione definita in N punti. Si vuole interpolare linearmente ogni coppia (*vett_input*[j], *vett_input*[j+1]), ossia si vuole calcolare il valor medio di ogni coppia di punti.

Si scriva una procedura **interpola** che riceve tramite stack

- l'offset del vettore in input *vett_input*
- il numero di elementi N
- l'offset del vettore in output *vett_output*.

vett_output è un vettore di interi con segno su 16 bit di dimensione $2 * N - 1$, contenente sia i valori di ingresso sia quelli interpolati. Più precisamente:

$$vett_output[i] = \begin{cases} vett_input\left[\frac{i}{2}\right] & \text{se } i \text{ è pari} \\ \frac{vett_input\left[\frac{i-1}{2}\right] + vett_input\left[\frac{i+1}{2}\right]}{2} & \text{se } i \text{ è dispari} \end{cases}$$

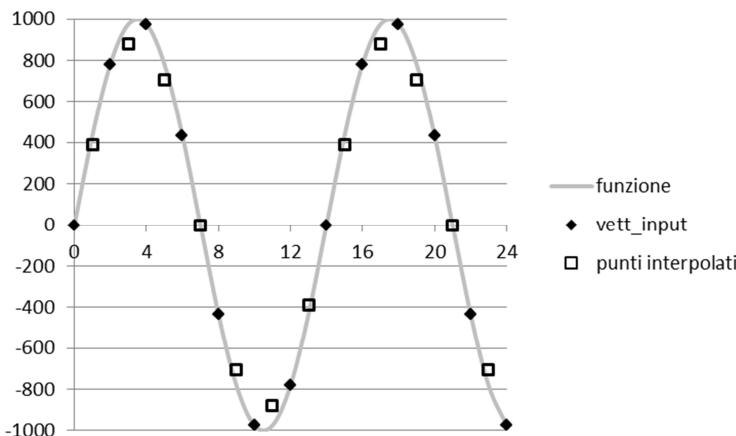
Si noti che il valore medio fra ogni coppia di elementi di *vett_input* è sicuramente rappresentabile su 16 bit, ma ciò non vale a priori per la somma di due elementi consecutivi di *vett_input*. È necessario quindi effettuare una scelta opportuna dell'ordine dei calcoli e della dimensione dei dati intermedi in modo che non si verifichi overflow nei calcoli.

Di seguito un esempio di programma chiamante:

```
N EQU 13
.MODEL small
.STACK
.DATA
vett_input DW 0, 782, 975, 434, -434, -975, -782
           DW 0, 782, 975, 434, -434, -975
vett_output DW 2 * N - 1 DUP (?)

.CODE
.STARTUP
[...]
PUSH OFFSET vett_input
PUSH N
PUSH OFFSET vett_output
CALL interpola
[...]
.EXIT
```

Di seguito si fornisce una rappresentazione grafica con i valori di *vett_input* indicati nel codice:



vett_output contiene i valori di *vett_input* alternati con i valori interpolati. Nell'esempio riportato, dopo la chiamata della funzione **interpola**, *vett_output* = [0, 391, 782, 878, 975, 704, 434, 0, -434, -705, -975, -879, -782, -391, 0, 391, 782, 878, 975, 704, 434, 0, -434, -705, -975].

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Un bacino idrico è dotato di un sensore che monitora il livello dell'acqua. Il livello dell'acqua è misurato in millimetri, ed è espresso come differenza da un riferimento posto ad un'altezza di 10 metri. Pertanto, il valore misurato è un numero intero compreso tra -10.000 (con il bacino idrico vuoto) e +30.000 (corrispondente alla massima capienza). L'elenco dei valori misurati è memorizzato nel vettore *livello*. Inoltre, ad ogni rilevamento del livello dell'acqua si memorizza l'orario. L'orario è espresso come numero di minuti trascorsi dalla mezzanotte del giorno corrente. Le misure sono saltuarie (non periodiche), ma l'intervallo minimo fra due misure è 1 minuto. L'orario di tutte le misure è memorizzato nel vettore *orario*.

Si scriva in linguaggio Assembly 8086 una procedura **livelloAcqua** che riceve tramite stack:

- l'offset del vettore *orario*
- l'offset del vettore *livello*
- un numero intero *minuti*.

La procedura copia nel registro AX il livello dell'acqua (misurato o stimato) all'orario indicato da *minuti*:

- se *minuti* è uno fra gli elementi presenti nel vettore *orario*, allora la procedura restituisce il corrispondente elemento del vettore *livello*
- altrimenti, la procedura restituisce il valore stimato, ottenuto tramite un'interpolazione lineare fra i valori misurati nei due orari più prossimi (subito prima e subito dopo).

Più precisamente:

$$AX = \begin{cases} \text{livello}[i] & \text{se } \text{minuti} = \text{orario}[i] \\ \text{livello}[j] + \frac{\text{minuti}-\text{orario}[j]}{\text{orario}[j+1]-\text{orario}[j]} (\text{livello}[j+1] - \text{livello}[j]) & \text{se } \text{orario}[j] < \text{minuti} < \text{orario}[j+1] \end{cases}$$

Si assuma che *minuti* sia sempre compreso tra l'orario del primo e dell'ultimo rilevamento giornaliero.

Di seguito un esempio di programma chiamante:

```
.MODEL small
.STACK
.DATA
orario      DW 0, 12, 51, 112, 200, 384
livello     DW -5, 46, 70, 38, -12, 49
minuti      DW 63

.CODE
.STARTUP
[...]
PUSH OFFSET orario
PUSH OFFSET livello
PUSH minuti
CALL livelloAcqua
[...]
.EXIT
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Siano date due matrici di interi positivi memorizzate per righe:

Matrice 1:

<i>riga 0:</i>	1	2	3	4
<i>riga 1:</i>	5	6	<u>7</u>	8
<i>riga 2:</i>	9	10	11	12
<i>riga 3:</i>	13	14	15	16

Matrice 2:

<i>riga 0:</i>	9	3	4	11
<i>riga 1:</i>	1	5	6	8
<i>riga 2:</i>	16	15	2	12
<i>riga 3:</i>	<u>7</u>	10	13	14

Data l'indicazione dell'indice di una riga per ciascuna matrice (ad esempio, 1 e 3, rispettivamente), si scriva in linguaggio Assembly 8086 una procedura **trova_num** in grado di determinare il valore dell'elemento presente in ciascuna delle due matrici alle righe indicate (in questo caso, 7).

La procedura riceve mediante lo *stack* gli indirizzi di due vettori di DIMxDIM byte (DIM dichiarato come costante) contenenti le due matrici, e gli indici di riga mediante AL e AH. Il risultato, ossia il valore trovato, deve essere restituito tramite lo *stack*.

Se non ci sono elementi in comune, la procedura deve restituire il valore -1. Se ci sono più elementi in comune, la procedura ne restituisca uno qualunque fra essi.

Di seguito un esempio di programma chiamante:

```
DIM      EQU 4
.model small
.stack
.data
mat1    db 1, 2, 3, 4
        db 5, 6, 7, 8
        db 9, 10, 11, 12
        db 13, 14, 15, 16
mat2    db 9, 3, 4, 11
        db 1, 5, 6, 8
        db 16, 15, 2, 12
        db 7, 10, 13, 14

.code
.startup
push offset mat1
push offset mat2
sub sp, 2
mov al, 1          ; indice riga mat1
mov ah, 3          ; indice riga mat2
call trova_num
pop ax             ; risultato
add sp, 4

.exit
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Data una matrice di interi positivi memorizzata per righe, si scriva in linguaggio Assembly 8086 una procedura **calcola_e_invia_media** che calcoli la media degli elementi di ciascuna colonna della matrice.

La procedura riceve tramite stack:

- l'offset di una matrice di N righe e M colonne, dove N e M sono dichiarati come costanti
- l'offset di un vettore di M elementi, non inizializzato.

Gli elementi della matrice sono memorizzati come word. Anche il valore medio di ciascuna colonna è memorizzato come word, ma i calcoli devono essere eseguiti su 32 bit per evitare overflow e minimizzare l'errore di approssimazione.

Per ogni colonna della matrice, la procedura deve innanzitutto calcolare la media degli elementi e salvarla nell'elemento corrispondente del vettore (ad esempio, la media della prima colonna della matrice deve essere memorizzata nel primo elemento del vettore).

Successivamente, la procedura deve inviare il contenuto del vettore ad una periferica, che si interfaccia all'8086 tramite il modulo 8255. La procedura deve quindi scrivere tutti gli elementi del vettore (un byte alla volta) sulla porta A dell'8255, che si assume precedentemente configurata in modo 0 output.

L'8086 e la periferica utilizzano la stessa modalità di memorizzazione delle word (little endian).

Di seguito un esempio di programma chiamante:

```
N EQU 5      ; numero di righe
M EQU 4      ; numero di colonne
PORTA EQU 80h
PORTB EQU PORTA+1
PORTC EQU PORTA+2
CONTROL EQU PORTA+3

#START=8255.exe#

.MODEL small
.STACK
.DATA
matrice DW 20, 35, 40, 12
        DW 26, 5, 18, 30
        DW 10, 45, 33, 58
        DW 47, 3, 35, 34
        DW 60, 45, 32, 43
media DW M DUP (?) 

.CODE
.STARTUP
PUSH OFFSET matrice
PUSH OFFSET media
CALL calcola_e_invia_media
ADD SP, 4
.EXIT
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Si scriva in linguaggio Assembly 8086 una procedura **riempizaino** che trovi una soluzione *euristica* al problema dello zaino.

Sia dato uno zaino capace di sopportare un peso W . Sono disponibili N oggetti, ciascuno caratterizzato da un peso w_i e da un valore c_i . Si vuole individuare quali oggetti inserire nello zaino al fine di massimizzare il valore totale con il vincolo del peso complessivo, inferiore a W .

La procedura riceve tramite stack:

- l'offset del vettore *valore*, che specifica il valore (strettamente positivo) di ciascun oggetto
- l'offset del vettore *peso*, che specifica il peso (strettamente positivo) di ciascun oggetto
- l'offset del vettore *stato*, che indica se l'oggetto è stato già esaminato e l'eventuale decisione presa.

Gli elementi di questo vettore possono assumere 3 valori:

- 0: l'oggetto non è stato ancora esaminato
 - 1: l'oggetto è stato inserito nello zaino
 - -1: l'oggetto non può essere inserito nello zaino perché supera la capacità residua.
- il numero *contenuto*, espresso su un byte, che indica il peso degli oggetti attualmente presenti nello zaino.

Tutti i vettori sono composti da N elementi di tipo byte (N è dichiarata come costante, così come W). Non è ammesso l'uso di altre variabili.

La procedura **riempizaino** deve:

- 1) trovare l'oggetto con il massimo *costo unitario* (definito come c_i / w_i) fra quelli il cui stato è 0. Per il calcolo del costo unitario, si consideri solo la parte intera del rapporto c_i / w_i . In caso di più oggetti con lo stesso costo unitario massimo, se ne prenda uno a scelta.
- 2) controllare se l'oggetto individuato sta nello zaino (ossia, il peso dell'oggetto sommato al peso degli oggetti già messi nello zaino è inferiore a W).
 - a. Se l'oggetto sta nello zaino, la procedura aggiorna il peso degli oggetti attualmente presenti nello zaino (modificando il valore di *contenuto* nello stack) e pone a 1 lo stato dell'oggetto.
 - b. Se l'oggetto non sta nello zaino, lo stato dell'oggetto è posto uguale a -1.

Di seguito un esempio di programma chiamante:

```
N EQU 9          ; numero di oggetti
W EQU 27          ; capacità dello zaino
.MODEL small
.STACK
.DATA
valore DB 200, 225, 250, 150, 125, 125, 130, 120, 75
peso DB 10, 12, 15, 9, 7, 6, 6, 6, 4
stato DB N DUP (0)

.CODE
.STARTUP
PUSH OFFSET valore
PUSH OFFSET peso
PUSH OFFSET stato
MOV AX, 0
PUSH AX      ; valore iniziale di contenuto
MOV CX, N
ciclo:
CALL riempizaino
LOOP ciclo
ADD SP, 8
.EXIT
```

Alla prima chiamata, l'oggetto considerato è quello in posizione 6 (con costo unitario massimo di 21) e l'oggetto viene inserito nello zaino, aggiornando il valore di *contenuto* a 6.

Al termine del programma, con i valori nell'esempio, il vettore *stato* è: 1, -1, -1, -1, -1, 1, 1, -1, 1. Gli elementi inseriti nello zaino, così come indicato nel vettore *stato*, hanno peso complessivo 26 e valore 455.

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare qualunque materiale cartaceo - tempo: 60 minuti

Un vettore *tempi* contiene una serie di *orari*, costituiti ciascuno da una coppia *ore* e *minuti*, nel seguente formato:

ore₁, minuti₁, ore₂, minuti₂, ore₃, minuti₃, ore₄, minuti₄, ...

Ogni elemento del vettore corrisponde a 1 byte.

Si scriva in linguaggio Assembly 8086 una procedura **calcolaDifferenza** che calcoli l'*intervallo* (espresso in minuti) tra ogni coppia di orari consecutivi (orario₂ – orario₁, orario₄ – orario₃, orario_{2n} – orario_{2n-1}, ecc). La procedura deve salvare il risultato in un secondo vettore *risultato*.

La procedura riceve l'offset dei due vettori tramite stack: *tempi* è un vettore di byte contenente NUM * 4 elementi, dove NUM rappresenta il numero di coppie di orari, mentre *risultato* è un vettore di word contenente NUM elementi. NUM è dichiarato come costante. Si assuma che

- ogni intervallo da calcolare abbia durata massima di 24 ore
- in ogni coppia di orari consecutivi, il secondo orario si riferisce sempre a un momento successivo al primo.

Di seguito un esempio di programma chiamante:

```
NUM EQU 3
.MODEL small
.STACK
.DATA
tempi    DB 8, 27, 17, 12, 21, 34, 9, 41, 7, 18, 15, 5
risultato DW NUM DUP (?)

.CODE
.STARTUP
...
PUSH OFFSET tempi
PUSH OFFSET risultato
CALL calcolaDifferenza
ADD SP, 4
...
.EXIT
```

Al termine del programma, con i valori nell'esempio, il vettore *risultato* è: 525, 727, 467.

Si noti che il secondo elemento del vettore *risultato* è ottenuto dalla differenza tra gli orari 21:34 e 9:41, in cui il secondo orario si riferisce al giorno successivo.

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare solamente il foglio con l'instruction set Intel fornito - tempo: 60 minuti

Un problema comune per compilatori ed editor di testo consiste nel determinare se le parentesi all'interno di una stringa sono bilanciate e correttamente annidate. Ad esempio:

- | | |
|----------------------|---|
| 1. (((1+2)*3)-(4-8)) | OK: parentesi bilanciate e correttamente annidate |
| 2. (1+2)+)4-1(| NO: parentesi non correttamente annidate |
| 3. (1+2*(5-3) | NO: parentesi non bilanciate |

Si scriva in linguaggio Assembly 8086 una procedura **verificaParentesi** che controlli la correttezza delle parentesi all'interno di una stringa. La lunghezza della stringa DIM è definita tramite costante. La procedura riceve l'offset della stringa tramite stack. Non è ammesso l'uso di variabili.

La procedura **verificaParentesi** deve restituire attraverso il registro BX:

- il valore DIM se le parentesi sono bilanciate e correttamente annidate
- altrimenti, la posizione della prima parentesi che non soddisfa i requisiti di bilanciamento e annidamento.

Negli esempi proposti, il valore restituito è:

- | | |
|----------------------|----------|
| 1. (((1+2)*3)-(4-8)) | BX = DIM |
| 2. (1+2)+)4-1(| BX = 6 |
| 3. (1+2*(5-3) | BX = 0 |

Una possibile realizzazione dell'algoritmo in pseudocodice è la seguente:

```
char s[DIM];
int par = 0;
for (i=0; i<DIM; i++)
{
    if (s[i] == '(')
    {
        par++;
        push(i);
    }
    else if (s[i] == ')')
    {
        par--;
        if (par < 0) return i;
        else pop();
    }
}
if (par!=0)
    return pop();
return DIM;
```

Di seguito un esempio di programma chiamante:

```
DIM EQU 17
.MODEL small
.STACK
.DATA
string DB "(((1+2)*3)-(4-8))"

.CODE
.STARTUP
...
PUSH OFFSET string
CALL verificaParentesi
ADD SP, 2
...
.EXIT
```

Nome, cognome, matricola

Esercizio di programmazione

Sino a 12 punti. È possibile consultare solamente l'instruction set Intel fornito. Tempo: 60 minuti

Sia data una matrice di byte contenente solo valori 0 e 1, di dimensione fissata dalle due costanti (strettamente positive) NRIGHE e NCOLONNE. Le celle contenenti il valore 1 corrispondono ai punti di una linea (o percorso) sul piano. Si scriva una procedura **seguiPercorso** in linguaggio Assembly 8086 in grado di seguire un percorso a partire da una cella data e lungo celle contigue contenenti il valore 1, finché possibile. Il percorso non presenta biforcazioni, e lo spostamento può avvenire soltanto verso destra o verso il basso. Nell'esempio seguente, la casella di partenza è (1, 0), il percorso è lungo 6 celle e la casella finale è (3, 3):

inizio	0	0	0	0	1	1
	1	1	0	0	0	1
	0	1	1	1	0	0
	0	0	0	1	0	1

fine

La procedura riceve:

- l'offset della matrice tramite stack
- l'indice della riga di partenza (compreso tra 0 e NRIGHE-1) attraverso il registro DL
- l'indice della colonna di partenza (compreso tra 0 e NCOLONNE-1) attraverso il registro DH.

La procedura restituisce la lunghezza del percorso tramite stack. Si noti che il percorso è lungo 0 se la cella di partenza ha valore 0.

Non è ammesso l'uso di variabili.

Di seguito un esempio di programma chiamante:

```
NRIGHE EQU 4
NCOLONNE EQU 6
.MODEL small
.STACK
.DATA
matrice DB 0, 0, 0, 0, 0, 0
        DB 1, 1, 0, 0, 0, 0
        DB 0, 1, 1, 1, 0, 0
        DB 0, 0, 0, 1, 0, 0

.CODE
.STARTUP
PUSH OFFSET matrice
SUB SP, 2
MOV DL, 1
MOV DH, 0
CALL seguiPercorso
POP AX
ADD SP, 2
.EXIT
```

Nome, cognome, matricola

Esercizio di programmazione

Sino a 12 punti. È possibile consultare solamente l'instruction set Intel fornito. Tempo: 60 minuti

Sia data una matrice byte contenente valori positivi, di dimensione fissata dalle due costanti (strettamente positive) NRIGHE e NCOLONNE. Si scriva una procedura **contaColonne** in linguaggio Assembly 8086 in grado di contare in quante colonne della matrice la somma degli elementi di valore pari è superiore alla somma degli elementi di valore dispari.

La procedura riceve l'offset della matrice tramite stack e restituisce nel registro AX il numero di colonne in cui la somma degli elementi pari è superiore alla somma degli elementi dispari.

Non è ammesso l'uso di variabili.

Di seguito un esempio di programma chiamante:

```
NRIGHE EQU 4
NCOLONNE EQU 6
.MODEL small
.STACK
.DATA
matrice DB 113, 10, 95, 20, 60, 10
        DB 51, 26, 120, 30, 56, 13
        DB 102, 171, 21, 111, 17, 22
        DB 204, 100, 230, 16, 76, 34

.CODE
.STARTUP
LEA SI, matrice
PUSH SI
CALL contaColonne
.EXIT
```

Nell'esempio, il valore del registro AX dopo la chiamata della procedura **contaColonne** è 4.

Nome, cognome, matricola

Esercizio di programmazione

Sino a 12 punti. È possibile consultare solamente l'instruction set Intel fornito. Tempo: 60 minuti

Una matrice di *word* di NUM righe e 3 colonne contiene informazioni riguardanti i prodotti acquistati dal cliente di un supermercato. Per ogni riga, i primi due elementi (*unsigned*) rappresentano il prezzo del prodotto, espresso in Euro e centesimi di Euro. Il terzo elemento può assumere il valore 0 o 1 e descrive se il prodotto è frutta o verdura, per cui deve essere pagato il prezzo del sacchetto biodegradabile, pari a SAC centesimi (il prezzo del sacchetto deve essere aggiunto ogni volta che il terzo elemento vale 1).

Un esempio di matrice è il seguente:

```
list      dw 4, 99, 0  
          dw 16, 49, 0  
          dw 2, 86, 1  
          dw 3, 48, 1  
          dw 8, 99, 0  
          dw 4, 21, 1
```

In questo caso, con SAC pari a 2 centesimi, il totale è di 41 Euro e 8 centesimi.

Si scriva una procedura **totale** in linguaggio Assembly 8086 in grado di calcolare il prezzo totale da pagarsi per l'elenco dei prodotti, in Euro e centesimi di Euro, tenendo conto che

- NUM è una costante di valore massimo 500
- SAC è una costante di valore massimo 5.

La procedura riceve l'offset della matrice mediante *stack* e restituisce i due valori (Euro e centesimi) sempre mediante *stack*. In caso di overflow, deve essere restituita la coppia di valori esadecimale 0FFh, 0FFh.

Di seguito un esempio di programma chiamante:

```
SAC      EQU 2           ; massimo 5  
NUM      EQU 6           ; massimo 500  
  
.model small  
.stack  
.data  
list      dw 4, 99, 0  
          dw 16, 49, 0  
          dw 2, 86, 1  
          dw 3, 48, 1  
          dw 8, 99, 0  
          dw 4, 21, 1  
  
.code  
.startup  
        SUB SP, 4  
        PUSH OFFSET list  
        call totale  
        ADD SP, 2  
        POP AX           ; Euro  
        POP DX           ; centesimi  
.exit
```

Nome, cognome, matricola.....

Esercizio di programmazione

Sino a 12 punti. È possibile consultare solamente l'instruction set Intel fornito. Tempo: 60 minuti

La distanza di Hamming tra due stringhe di ugual lunghezza è pari al numero di posizioni nelle quali i simboli corrispondenti sono diversi. In altri termini, la distanza di Hamming misura il numero di sostituzioni necessarie per convertire una stringa nell'altra. Ad esempio, si consideri la distanza di Hamming binaria tra i seguenti due interi:

11011101
11001001

Il risultato in questo caso è 2.

Si scriva una procedura **hamming** in linguaggio Assembler 8086 che calcoli la distanza di Hamming binaria tra gli elementi di indice corrispondente di due vettori di *byte* di lunghezza DIM (dichiarato come costante), e salvi il risultato in un terzo vettore.

Esempio (valori in decimale e binario):

vet1	vet2	ris
56 (0011 1000)	1 (0000 0001)	4
12 (0000 1100)	0 (0000 0000)	2
98 (0110 0010)	245 (1111 0101)	5
129 (1000 0001)	129 (1000 0001)	0
58 (0011 1010)	12 (0000 1100)	4

La procedura riceve tramite *stack* l'indirizzo dei due vettori di dati e del vettore risultato. Di seguito un esempio di programma chiamante.

```
DIM EQU 5

.model small
.stack
.data
vet1 db 56, 12, 98, 129, 58
vet2 db 1, 0, 245, 129, 12
ris db DIM dup (?)

.code
.startup
...
LEA AX, vet1
PUSH AX
LEA AX, vet2
PUSH AX
LEA AX, ris
PUSH AX
CALL hamming
ADD SP, 6
...
.exit
```

Esercizio di programmazione

sino a 12 punti – è possibile consultare solamente l'instruction set Intel - tempo: 60 minuti

Il costo di un parcheggio è pari a X Euro per ogni periodo di Y minuti. Per eventuali minuti di un periodo non completo sono addebitati comunque X Euro.

Esempio:

X: 1 Euro

Y: 40 minuti

Orario di ingresso: 12.47

Orario di uscita: 18.14

Il tempo di permanenza corrisponde a 8 periodi interi più 7 minuti. Il costo del parcheggio è 9 Euro.

Si scriva una procedura **costoParcheggio** in linguaggio Assembly 8086 in grado di calcolare il costo per il parcheggio. X e Y sono dichiarati come costanti. Gli orari di ingresso e di uscita sono memorizzati ciascuno in un vettore di 2 byte: il primo indica l'ora e il secondo i minuti. La procedura **costoParcheggio** riceve l'offset dei due vettori tramite stack e restituisce il costo del parcheggio attraverso il registro AX.

Si assuma che gli orari siano sempre consecutivi e appartenenti alla stessa giornata.

Di seguito un esempio di programma chiamante:

```
X EQU 1
Y EQU 40
.MODEL small
.STACK
.DATA
ora_in DB 12, 47
ora_out DB 18, 14

.CODE
.STARTUP
    LEA AX, ora_in
    PUSH AX
    LEA AX, ora_out
    PUSH AX
    CALL costoParcheggio
    ADD SP, 4
.EXIT
```

Nome, cognome, matricola

Esercizio di programmazione

sino a 12 punti – è possibile consultare solamente l'instruction set Intel - tempo: 60 minuti

Dati i prezzi di acquisto e di vendita di un titolo azionario, e tenendo conto dell'eventuale dividendo erogato nel periodo di possesso, si vuole calcolare il **rendimento** del titolo espresso in percentuale. Ad esempio, se il titolo è stato acquistato a 190\$ ed è stato venduto a 199\$, e se nel frattempo è stato erogato un dividendo da 2\$, il rendimento sarà dato da:

$$\frac{P_{vend} - P_{acq} + Div}{P_{acq}} = \frac{199 - 190 + 2}{190} \cdot 100 = 5.5$$

Si noti che il risultato può essere positivo o negativo.

Si scriva una procedura **calcola** in linguaggio Assembler 8086 che calcoli il rendimento di ciascun titolo in un insieme di DIM titoli (DIM dichiarato come costante), con un'approssimazione di ± 1 (non è richiesto uno specifico arrotondamento). È consentito lavorare nell'ipotesi di non avere *overflow*, ma si tenga conto dell'ordine delle operazioni per ottenere risultati significativi.

I prezzi di acquisto e di vendita di ciascun titolo, l'ammontare del dividendo e i valori di rendimento (che dovranno essere aggiornati dalla procedura) sono memorizzati in vettori di *word*. Gli indirizzi dei vettori sono passati alla procedura mediante *stack*.

Esempio (DIM = 4):

prezzo_acq	prezzo_vend	dividendo	rendimento
190	199	2	5
68	40	5	-33
71	81	0	14
84	90	1	8

Di seguito un esempio di programma chiamante.

```
DIM EQU 4
.model small
.stack
.data

prezzo_acq dw 190, 68, 71, 84
prezzo_vend dw 199, 40, 81, 90
dividendo dw 2, 5, 0, 1
rendimento dw DIM DUP(?) 

.code
.startup
LEA AX, prezzo_acq
PUSH AX
LEA AX, prezzo_vend
PUSH AX
LEA AX, dividendo
PUSH AX
LEA AX, rendimento
PUSH AX
CALL calcola
ADD SP, 8
.exit
```

Esercizio di programmazione

sino a 12 punti – è possibile consultare solamente il foglio consegnato con l'instruction set MIPS - tempo: 60 minuti

La notazione polacca inversa permette di scrivere formule matematiche senza utilizzare parentesi e regole di precedenza degli operatori. Quando si incontra un operatore all'interno dell'espressione matematica, questo si applica ai due operandi precedenti. Ad esempio, l'espressione $18 + 25 * (10 - 7) - 13$ rappresentata con la notazione polacca inversa diventa $18\ 25\ 10\ 7\ -\ *\ +\ 13\ -$

Si vuole realizzare un programma in linguaggio MIPS per calcolare il valore di un'espressione in notazione polacca inversa. L'espressione è memorizzata in un array di word, in cui gli elementi sono così codificati:

- se il primo bit è 0, l'elemento dell'array è un numero positivo nell'intervallo $(0, 2^{31} - 1)$
- se il primo bit è 1, l'elemento dell'array è un operatore (+, -, *, /). Una possibile codifica è la seguente:

somma= -1; sottrazione = -2; moltiplicazione = -3; divisione = -4

Si supponga che la sintassi dell'espressione in notazione polacca inversa sia corretta e che nel calcolo del risultato non vi sia overflow. Si scriva una procedura **calcolaPolaccaInversa** che riceva in input come primo parametro l'indirizzo dell'array contenente l'espressione e come secondo parametro la lunghezza dell'array. La procedura restituisce il valore dell'espressione.

La procedura scandisce ogni elemento dell'array. Se è un operando, ne fa il push nello stack. Se è un operatore, preleva i due elementi in cima allo stack e chiama la procedura **eseguiOperazione** per ottenere il valore dell'operazione; questo valore è poi inserito in cima allo stack. La procedura **eseguiOperazione** riceve in input l'operatore, il primo operando, il secondo operando; restituisce il valore dell'operazione.

Tutte le procedure devono essere conformi allo standard e alle specifiche per quanto riguarda il passaggio dei parametri in input, del valore di ritorno e dei registri da preservare.

Di seguito un eSEMPIO di programma chiamante e della procedura **eseguiOperazione**:

```
.data
espressione: .word 18, 25, 10, 7, -2, -3, -1, 13, -2
tabella: .word somma, sottrazione, moltiplicazione, divisione
.text
.globl main
.ent main
main: subu $sp, $sp, 4
      sw $ra, ($sp)
      la $a0, espressione
      li $a1, 9
      jal calcolaPolaccaInversa

      lw $ra, ($sp)
      addu $sp, $sp, 4
      jr $ra
.end main

eseguiOperazione:
      subu $t0, $zero, $a0
      subu $t0, $t0, 1
      sll $t0, $t0, 2
      lw $t1, tabella($t0)
      jr $t1
somma: addu $v0, $a1, $a2
      b fine
sottrazione: subu $v0, $a1, $a2
      b fine
moltiplicazione: mulou $v0, $a1, $a2
      b fine
divisione: divu $v0, $a1, $a2
      b fine
fine: jr $ra
```

Nome, Cognome, Matricola:.....

Esercizio di programmazione

sino a 12 punti – tempo: 60 minuti

è possibile consultare solamente il foglio consegnato contenente l'instruction set MIPS
il codice va scritto in stampatello – eventuali operazioni sullo stack vanno adeguatamente commentate

In matematica, la trasposta di una matrice è la matrice ottenuta scambiandone le righe con le colonne.
Ad esempio, per una matrice 4x4

$$A = \begin{pmatrix} 126 & -988 & 65 & 52 \\ 7 & 0 & 2 & 643 \\ 66 & 532 & 43 & 9254 \\ 5 & -51 & 4352 & -452 \end{pmatrix} \quad A^T = \begin{pmatrix} 126 & 7 & 66 & 5 \\ -988 & 0 & 532 & -51 \\ 65 & 2 & 43 & 4352 \\ 52 & 643 & 9254 & -452 \end{pmatrix}$$

Si scriva una procedura `calcolaTrasp` in grado di trasformare una matrice quadrata di *word* con segno memorizzata per righe, calcolandone la trasposta e aggiornando i valori memorizzati. La procedura non deve utilizzare altre variabili in memoria.

L'indirizzo della matrice è passato tramite `$a0`, mentre il numero di elementi di una riga è passato mediante `$a1`. Di seguito un esempio di programma chiamante.

```
DIM = 4
.data
matrice: .word 126, -988, 65, 52
          .word 7, 0, 2, 643
          .word 66, 532, 43, 9254
          .word 5, -51, 4352, -452

.text

.globl main
.ent main

main:    subu $sp, $sp, 4
          sw $ra, ($sp)
          la $a0, matrice
          li $a1, DIM
          jal calcolaTrasp
          lw $ra, ($sp)
          addiu $sp, $sp, 4
          jr $ra
```

Nome, Cognome, Matricola:.....

Esercizio di programmazione

sino a 12 punti – è possibile consultare solamente il foglio consegnato con l'Instruction set MIPS - tempo: 60 minuti

Data una sequenza di interi con segno, rappresentati come *word* in memoria, si scriva una procedura **monotono** in grado di determinare la posizione della più lunga sottosequenza non decrescente nel vettore e il numero di elementi che la compongono.

Il vettore su cui la procedura lavora è già inizializzato. La procedura riceve in **\$a0** l'indirizzo del vettore e in **\$a1** la sua lunghezza, mentre restituisce nel registro **\$v0** il numero di elementi della sottosequenza e nel registro **\$v1** l'indice del primo elemento di tale sottosequenza.

Si lavori nell'ipotesi per cui esista una singola sottosequenza della dimensione massima.

Esempio (vettore di 12 elementi):

vet: .word 15, 64, 9, 2, 4, 5, 9, 1, 294, 52, -4, 5

La procedura dovrà fornire (si assuma che gli elementi del vettore abbiano indice variabile tra 0 e 11):
\$v0 = 4, \$v1 = 3.

Di seguito un esempio di programma chiamante:

```
.data
vet: .word 15, 64, 9, 2, 4, 5, 9, 1, 294, 52, -4, 5

.text
.globl main
.ent main
main: subu $sp, $sp, 4
sw $ra, 0($sp)

la $a0, vet          # indirizzo di vet
li $a1, 12            # dimensione di vet
jal monotono

lw $ra, 0($sp)
addiu $sp, $sp, 4
jr $ra
```

Nome, Cognome, Matricola:.....

Esercizio di programmazione

sino a 12 punti – è possibile consultare solamente il foglio consegnato con l'instruction set MIPS - tempo: 60 minuti – scrivere in stampatello

Si vuole realizzare un programma in linguaggio MIPS per comprimere un vettore di byte contenente numeri interi senza segno, utilizzando il seguente algoritmo di compressione:

- gli elementi del vettore di partenza sono quantizzati
- una sequenza di elementi uguali (dopo la quantizzazione) è sostituita con un unico elemento.

Esempio: si supponga che l'intervallo di quantizzazione sia 10, ossia il vettore quantizzato contenga solo multipli di 10. I vettori dopo la quantizzazione e finale, rispettivamente, sono riportati di seguito.

vettore iniziale	14	16	18	134	24	22	23	149	140	141	145	146
vettore quantizzato	10	10	10	130	20	20	20	140	140	140	140	140
vettore finale	10	130	20	140								

Si richiede di implementare la procedura **comprimi** che implementa l'algoritmo di compressione descritto. La procedura riceve in input:

1. indirizzo del vettore di byte senza segno da comprimere
2. numero di elementi nel vettore da comprimere
3. indirizzo di un vettore di byte non inizializzato, che conterrà i valori compressi.

La procedura restituisce il numero di elementi nel vettore di byte compresso.

La procedura scandisce ogni elemento del vettore. Ad ogni iterazione del ciclo, richiama la procedura **quantizza** (il cui codice è fornito sotto) per ottenere il valore quantizzato dell'elemento corrente. Dopo aver ottenuto il valore quantizzato, la procedura **comprimi** lo confronta con l'ultimo elemento inserito nel vettore compresso. Il nuovo valore quantizzato è inserito nel vettore compresso solo se diverso.

La procedura **quantizza** riceve come unico argomento l'elemento corrente (l'intervallo di quantizzazione è definito tramite costante) e restituisce il valore quantizzato; la sua implementazione è fornita sotto.

Di seguito un eSEMPIO di programma chiamante e il codice della procedura **quantizza**:

```
.data
vettore:      .byte 14, 16, 18, 134, 24, 22, 23, 149, 140, 141, 145, 146
vettoreCompresso: .space 12
INTERVALLO_QUANT = 10
```

```
.text
.globl main
.ent main
main:
    subu $sp, $sp, 4
    sw $ra, ($sp)
    la $a0, vettore
    li $a1, 12
    la $a2, vettoreCompresso
    jal comprimi

    lw $ra, ($sp)
    addu $sp, $sp, 4
    jr $ra
.end main
```

```
quantizza:
    divu $t0, $a0, INTERVALLO_QUANT
    mulou $v0, $t0, INTERVALLO_QUANT
    jr $ra
```