

Soluzioni ulteriori Esercizi per laboratorio Assembly

Scrivere in Assembler per Intel 80x86 la funzione TrovaMax che cerca il massimo tra due vettori non ordinati di byte (numeri senza segno) e ritorna in AL 0 se il massimo si trova nel primo vettore, 1 se il massimo si trova nel secondo. **Viene passato anche il numero N di elementi dei due vettori.** Di conseguenza il programma chiamante stampa a video una stringa diversa, come riportato di seguito (si supponga che la funzione ScriviStringa visualizzi a video la stringa passata come parametro):

```
SECTION data
    Vett1: db 13,15,22,7,5,3,21,2,0,10
    Vett2: db 1,7,3,2,22,21,3,28,7,11
    Stringa1: db 'Il massimo è nel vettore 1',0
    Stringa2: db 'Il massimo è nel vettore 2',0

SECTION text
..start:
    mov ax, data
    mov ds, ax
    mov es, ax
    mov ax, Vett2
    push ax
    mov ax, Vett1
    push ax
mov ax, 10
push ax
    call TrovaMax
    add sp, 6
    cmp al, 1
    je Max_in_2
    mov bx, Stringa1
    jmp fine
Max_in_2:
    mov bx, Stringa2
fine:
    push bx
    call ScriviStringa
    add sp, 2
    ret

TrovaMax:
    push bp
    mov bp, sp
    mov si,[bp+6] ; Vett1
    mov di,[bp+4] ; Vett2
    mov bl,0      ; bl contiene il massimo attuale
    mov cx,[bp+8] ; N = 10
Ciclo:
    mov ah,[si]
    cmp ah, bl
    ja NuovoMassimo1
CicloRef1:
    inc si
    mov ah,[di]
```

```

        cmp ah,bl
        ja NuovoMassimo2
CicloRef2:
        inc di
        loop Ciclo
        jmp FineFunz
NuovoMassimo1:
        mov bl,ah
        mov al, 0
        jmp CicloRef1
NuovoMassimo2:
        mov bl,ah
        mov al, 1
        jmp CicloRef2
FineFunz:
        pop bp
        ret

```

Scrivere in Assembler per Intel 80x86 la funzione `MirrorStringa` che riceve in ingresso una stringa zero-terminata stringa (non passata tramite lo stack). La stringa è composta da 2 parole di lunghezza qualsiasi (anche diverse tra loro) separate da un solo spazio. La funzione deve scambiare di posto le due parole e scrivere il risultato nella stringa (zero-terminata) mirror. Ad esempio, se la stringa in ingresso vale 'Grande Pennello', la stringa risultante dal mirror deve valere 'Pennello Grande'.

Le variabili del programma sono le seguenti:

```

stringa: db 'Buona Pasqua',0
mirror: resb 100

```

Si scriva anche il programma main che chiama la funzione.

CPU 8086

SECTION data

```

stringa: db 'Buona Pasqua',0
mirror: resb 100

```

SECTION text

..start:

```

        mov ax, data
        mov ds, ax
        mov es, ax
        call MirrorStringa
        mov bx, mirror
        push bx
        call ScriviStringa
        add sp, 2
        mov ax, 4c00h
        int 21h

```

MirrorStringa:

```

        mov si, stringa

```

```

        mov di, mirror
Ciclo:  lodsb
        cmp al, ' '
        je Ciclo2
        cmp al, 0
        jne Ciclo
Ciclo2: lodsb
        cmp al, 0
        je ricomincia
        stosb
        jmp Ciclo2
ricomincia:
        mov al, ' '
        stosb                ; metto lo spazio in mirror
        mov si, stringa ; mi riposiziono all'inizio di stringa
Ciclo3: lodsb
        cmp al, ' '
        je Fine
        stosb
        jmp Ciclo3
Fine:   mov al, 0
        stosb                ; metto il terminatore in mirror
        ret

ScriviStringa:
        push bp
        mov bp, sp
        mov si, [bp+4]
        mov  ah, 0eh
        mov  bx, 0000h
Stampa: lodsb
        cmp al, 0
        je fineStampa
        int  10h
        jmp Stampa
fineStampa:
        pop bp
        ret                ;Ritorno alla procedura chiamante

```

Scrivere in Assembler per Intel 80x86 la funzione `Filtro` che copia un vettore di N byte dall'indirizzo DS:SI all'indirizzo ES:DI. I dati vengono copiati all'indirizzo destinazione se e solo se il bit meno significativo del dato è a 1.

Le variabili del programma sono le seguenti:

`N: db 100`

`Vett1: times 25 db 3, times 25 db 4, times 25 db 5, times 25 db 6`

`Vett2: resb 100`

Si scriva anche il programma main che chiama la funzione.

CPU 8086

SECTION data

`N: db 100`

`Vett1: times 25 db 3`

`times 25 db 4`

`times 25 db 5`

`times 25 db 6`

`Vett2: resb 100`

SECTION text

`..start:`

`mov ax, data`

`mov ds, ax`

`mov es, ax`

`call Filtro`

`mov ax, 4c00h`

`int 21h`

Filtro:

`mov si, Vett1`

`mov di, Vett2`

`xor ch, ch`

`mov cl, [N]`

Ciclo:

`lodsb`

`test al, 00000001b`

`je salta`

`stosb`

salta:

`loop Ciclo`

`ret`

Scrivere in Assembler per Intel 80x86 un programma che richiama la funzione EliminaElementi la cui definizione C è la seguente:

```
void EliminaElementi (int *vettore, int X, int Y)
```

Tale funzione deve eliminare tutti gli elementi con valore compreso tra X e Y (inclusi) dal vettore vettore. Il vettore non è ordinato e contiene valori byte positivi ed è terminato con il valore -1 (che non può essere mai presente come valore di un elemento).

La funzione deve essere richiamata dal programma principale con passaggio di parametri tramite lo stack e deve rimuovere dal vettore tutte le occorrenze di valori compresi tra X e Y.

Le variabili del programma sono le seguenti:

```
vettore: db 12, 33, 22, 11, 55, 23, 12, 120, 1, -1
```

Si scriva anche il programma main che chiama la funzione.

CPU 8086

SECTION data

```
vettore: db 12, 33, 22, 11, 55, 23, 12, 120, 1, -1
```

SECTION text

..start:

```
    mov ax, data
    mov ds, ax
    mov es, ax
    mov ax, vettore
    push ax
    mov ax, 23          ; valore X
    push ax
    mov ax, 55          ; valore Y
    push ax
    call EliminaElementi
    add sp, 6
    mov ax, 4c00h
    int 21h
```

EliminaElementi:

```
    push bp
    mov bp, sp
    mov si, [bp+8] ; vettore
    mov bl, [bp+6] ; X
    mov bh, [bp+4]; Y
```

Ciclo:

```
    lodsb
    cmp al, -1          ; ho finito il vettore
    je Fine
    cmp al, bl
    jb Ciclo
    cmp al, bh
```

```

        ja Ciclo
        call RimuoviElemento
        jmp Ciclo
Fine:
        pop bp
        ret

RimuoviElemento:
        ; si punta sull'elemento da rimuovere
        ; devo copiare indietro gli elementi successivi usando di
        push si          ; salvo si per poter ricominciare da dove mi ero fermato
        mov di, si
        dec di           ; torno indietro di 1 per puntare sull'elemento da sostituire/cancellare
CicloFunzione:
        lodsb
        stosb
        cmp al, -1       ; lo faccio dopo in modo da copiare anche il -11
        je FineFunzione
        jmp CicloFunzione
FineFunzione:
        pop si
        dec si           ; devo tornare indietro di una posizione
        ret

```

Scrivere in Assembler per Intel 80x86 un programma che richiama la funzione `SaturaElementi` la cui definizione C è la seguente:

```
void SaturaElementi (unsigned char *vettore, int N)
```

Tale funzione deve controllare ciascun elemento del vettore e se maggiore di N, modificarlo con il valore di N. Il vettore non è ordinato e contiene valori byte positivi ed è terminato con il valore -1 (che non può essere mai presente come valore di un elemento).

La funzione deve essere richiamata dal programma principale con passaggio di parametri tramite lo stack.

Le variabili del programma sono le seguenti:

```
vettore: db 12, 33, 22, 11, 55, 23, 12, 120, 1, -1
```

Si scriva anche il programma main che chiama la funzione.

CPU 8086

SECTION data

```
vettore: db 12, 33, 22, 11, 55, 23, 12, 120, 1, -1
```

SECTION text

```

..start:
        mov ax, data
        mov ds, ax
        mov es, ax
        mov ax, vettore

```

```

push ax
mov ax, 50          ; valore N
push ax
call SaturaElementi
add sp, 4
mov ax, 4c00h
int 21h

```

SaturaElementi:

```

push bp
mov bp, sp
mov si, [bp+6] ; vettore
mov bl, [bp+4] ; N

```

Ciclo:

```

lodsb
cmp al, -1          ; ho finito il vettore
je Fine
cmp al, bl
jb Ciclo
mov [si-1], bl     ; attenzione al -1 perchè lodsb ha già incrementato si
jmp Ciclo

```

Fine:

```

pop bp
ret

```

Scrivere in Assembler per Intel 80x86 un programma che riceve in ingresso (tramite variabili definite nel segmento dati) tre stringhe ASCIIZ zero-terminate (secondo la convenzione C) *Stringa1*, *Stringa2* e *Stringa3*. Il programma deve riportare in *Stringa3* una parola di *Stringa1* seguita da una di *Stringa2*, seguita da una *Stringa1*, e così via. Le parole sono delimitate da singoli spazi.

A titolo di esempio, considerando il seguente segmento dati:

```

Stringa1: db "Io sto bene.",0
Stringa2: db "oggi molto", 0
Stringa3: resb 255

```

il programma deve riportare in *Stringa3* la stringa ASCIIZ "Io oggi sto molto bene."

CPU 8086

SECTION data

```

Stringa1: db "Io sto bene.",0
Stringa2: db "oggi molto", 0
Stringa3: resb 255

```

SECTION text

..start:

```

mov ax, data
mov ds, ax
mov es, ax
call FondiStringhe
mov ax, Stringa3
push ax

```

```
call ScriviStringa
add sp, 2
mov ax, 4c00h
int 21h
```

FondiStringhe:

```
mov si, Stringa1
mov bx, Stringa2
mov di, Stringa3
jmp CicloStringa1
```

SiamoSuStringa1:

```
mov al, ''
stosb
```

CicloStringa1:

```
lods b
cmp al, 0
je FineStringa1
cmp al, ''
je SiamoSuStringa2
stos b
jmp CicloStringa1
```

SiamoSuStringa2:

```
mov al, ''
stosb
```

CicloStringa2:

```
mov al, [bx]
cmp al, 0
je FineStringa2
inc bx
cmp al, ''
je SiamoSuStringa1
stos b
jmp CicloStringa2
```

FineStringa1:

```
; controllo se ci sono ancora caratteri su Stringa2 e li copio tutti su Stringa3
mov al, ''
stosb
```

CicloFineStringa1:

```
mov al, [bx]
cmp al, 0
je Fine
stosb
inc bx
jmp CicloFineStringa1
```

FineStringa2:

```
; controllo se ci sono ancora caratteri su Stringa1 e li copio tutti su Stringa3
mov al, ''
stosb
```

CicloFineStringa2:

```
lods b
cmp al, 0
je Fine
```



```

        stosb
        jmp CicloFineStringa2
Fine:
        mov al, 0
        stosb      ; metto il terminatore in Stringa3
        ret

ScriviStringa:
        push bp
        mov bp, sp
        mov si,[bp+4]
        mov  ah,0eh
        mov  bx,0000h
Stampa:
        lodsb
        cmp al, 0
        je fineStampa
        int   10h
        jmp  Stampa
fineStampa:
        pop bp
        ret      ;Ritorno alla procedura chiamante

```

Scrivere in Assembler per Intel 80x86 un programma che riceve in ingresso (tramite variabili definite nel segmento dati) due stringhe ASCII zero-terminate (secondo la convenzione C) *Stringa1* e *Stringa2*. Il programma deve riportare nel registro AX il numero di caratteri di *Stringa1* presenti anche (almeno una volta) in *Stringa2*. Ad esempio, se *Stringa1* vale "città" e *Stringa2* vale "tà" il programma deve restituire 3 in AX, se valgono invece "ciminiera" e "italia" il programma deve restituire 4, ecc.

CPU 8086

SECTION data

```
Stringa1: db 'citta',0
```

```
Stringa2: db 'ta', 0
```

SECTION text

```
..start:
```

```

        mov ax, data
        mov ds, ax
        mov bx, 0
        call ContaCaratteriRipetuti
        mov ax, 4c00h
        int 21h

```

```
ContaCaratteriRipetuti:
```

```

        mov si, Stringa1
        mov di, Stringa2

```

```
Ciclo:
```

```

        lodsb
        cmp al, 0

```

```

        je Fine
CicloInterno:
        mov ah, [di]
        inc di
        cmp ah, 0
        je FineCicloInterno
        cmp ah, al
        jne CicloInterno
        inc bx
        mov di, Stringa2
        jmp Ciclo
FineCicloInterno:
        mov di, Stringa2
        jmp Ciclo
Fine:
        mov ax, bx
        ret

```

Scrivere in Assembler per Intel 80x86 la funzione `periodispari` che analizza un vettore `Vett` di lunghezza `N` (**con `Vett` e `N` parametri della funzione passati mediante lo stack**) e riporta in `AX` il valore -1 se in `Vett` ci sono più elementi dispari che pari, 1 se invece ci sono più valori pari che dispari e 0 se sono in egual numero. Il programma chiamante ha il seguente codice Assembler (si supponga che la funzione `ScriviStringa` visualizzi a video la stringa passata come parametro):

```

SECTION data
Vett: db 13,15,22,7,5,3,21,2,0,10
N: db 10
StringaDispari: db 'Vett contiene più elementi dispari',0
StringaPari: db 'Vett contiene più elementi pari',0
StringaUguale: db 'Vett contiene tanti elementi dispari quanti pari',0

SECTION text
..start:
        mov ax, data
        mov ds, ax
        mov es, ax
        mov ax, Vett
        push ax
        mov ax, N
        push ax
        call periodispari
        add sp, 4
        cmp ax, 0
        je uguale
        jg pari
        mov bx, StringaDispari
        jmp fine
pari:
        mov bx, StringaPari
        jmp fine
uguale:
        mov bx, StringaUguale
fine:
        push bx

```

```

call ScriviStringa
add sp, 2
mov ax, 4c00h
int 21h

```

pariodispari:

```

push bp
mov bp, sp
mov si, [bp+6] ; Vett
mov cx, [bp+4] ; N
xor bx, bx          ; in bl conto i pari, in bh i dispari

```

Ciclo:

```

lodsb
test al, 00000001b ; se pari il test dà 0 altrimenti 1 (diverso da 0)
je incpari
inc bh
jmp salta

```

incpari:

```
inc bl
```

salta:

```

loop Ciclo
cmp bl, bh
ja piupari
je uguali
mov ax, -1 ; ho più valori dispari
jmp Fine

```

piupari:

```

mov ax, 1
jmp Fine

```

uguali:

```
mov ax, 0
```

Fine:

```

pop bp
ret

```

Scrivere in Assembler per Intel 80x86 la funzione `contacaratteri` che conta quante volte si presenta nella stringa ASCII zero-terminata (secondo la convenzione C) `Stringa` il carattere `ch` (con `Stringa` e `ch` parametri della funzione passati mediante lo stack) e riporta in `AX` tale numero.

Le variabili del programma sono le seguenti:

```

Stringa: db "Evviva la pappa con il pomodoro",0
Ch: db 'p'

```

Si scriva anche il programma main che chiama la funzione.

contacaratteri:

```

push bp
mov bp, sp
mov si, [bp+6] ; Stringa
mov cl, [bp+4] ; C
xor bx, bx          ; uso bx per contare i caratteri

```

Ciclo:

```

lodsb
cmp al, cl

```

```
        jne salta
        inc bx
salta:   loop Ciclo
        mov ax, bx
        pop bp
        ret
```

Scrivere in Assembler per Intel 80x86 la funzione `confrontaVettori` che riceve in ingresso due vettori di byte Vett1 e Vett2, entrambi di lunghezza N, anch'esso fornito come parametro (i parametri della funzione sono passati mediante lo stack). La funzione deve riportare il AX il numero di volte in cui gli elementi della stessa posizione in Vett1 e Vett2 sono uno pari e l'altro dispari, o viceversa.

Ad esempio, le variabili del programma sono le seguenti:

```
Vett1: db 2, 5, 7, 4, 55, 22
Vett2: db 10, 9, 4, 22, 24, 3
N: db 6
```

Per l'esempio riportato, la funzione deve riportare in AX il valore 3.
Si scriva anche il programma main che chiama la funzione.

SECTION data

```
Vett1: db 2, 5, 7, 4, 55, 22
Vett2: db 10, 9, 4, 22, 24, 3
N: db 6
```

SECTION text

..start:

```
        mov ax, data
        mov ds, ax
        mov es, ax
        mov ax, Vett1
        push ax
        mov ax, Vett2
        push ax
        xor ah, ah
        mov al, byte [N]
        push ax
        call confrontaVettori
        add sp, 6
        mov ax, 4c00h
        int 21h
```

confrontaVettori:

```
        push bp
        mov bp, sp
        mov si, [bp+8] ; Vett1
        mov di, [bp+6] ; Vett2
        xor ch, ch
        mov cl, [bp+4] ; C
        xor bx, bx          ; uso bx per contare i caratteri
```

Ciclo:

```
lods b
and al, 00000001b
and ah, [di] ; uso direttamente l'and per poi confrontare al e ah
inc di
cmp al, ah ; se sono diversi significa che uno è pari e l'altro dispari
jne salta
inc bx
```

salta:

```
loop Ciclo
mov ax, bx
pop bp
ret
```

Scrivere in Assembler per Intel 80x86 la funzione `decrementaStringa` che riceve in ingresso una stringa ASCIIZ (zero terminata) `Stringa` e un vettore di byte `Vett`. Si supponga che il vettore contenga tanti elementi quanti sono i caratteri della stringa. I parametri della funzione sono passati mediante lo stack. La funzione deve sostituire ogni carattere della stringa decrementandone il valore ASCII del corrispondente valore del vettore, eccetto nel caso di carattere ' ' (spazio).

Ad esempio, le variabili del programma sono le seguenti:

```
Stringa: db "Buon Natale",0
Vett: db 1, 2, 1, 0, 5, 1, 0, 3, 0, 2, 1
```

Per l'esempio riportato, la stringa alla fine deve valere "Asnn Maqajd".
Si scriva anche il programma main che chiama la funzione.

SECTION data

```
Stringa: db 'Buon Natale',0
Vett: db 1, 2, 1, 0, 5, 1, 0, 3, 0, 2, 1
```

SECTION text

..start:

```
mov ax, data
mov ds, ax
mov es, ax
mov ax, Stringa
push ax
mov ax, Vett
push ax
call decrementaStringa
add sp, 4
mov ax, Stringa
push ax
call ScriviStringa
add sp, 2
mov ax, 4c00h
int 21h
```

`decrementaStringa:`

```
push bp
mov bp, sp
```

```
mov si, [bp+6] ; Stringa
mov di, [bp+4] ; Vett
```

Ciclo:

```
lodsb
cmp al, 0
je Fine
cmp al, ''
je salta
mov bl, [di]
inc di
sub al, bl
mov [si-1], al
jmp Ciclo
```

salta:

```
inc di
jmp Ciclo
```

Fine:

```
pop bp
ret
```

ScriviStringa:

```
push bp
mov bp, sp
mov si,[bp+4]
mov ah,0eh
mov bx,0000h
```

Stampa:

```
lodsb
cmp al, 0
je fineStampa
int 10h
jmp Stampa
```

fineStampa:

```
pop bp
ret ;Ritorno alla procedura chiamante
```

Scrivere in Assembler per Intel 80x86 la funzione `CopiaCaratteri` che riceve in ingresso una stringa `Sorg`, una stringa `Dest`, un vettore di byte `Vett`, e un intero (byte) `N`. `N` rappresenta la lunghezza delle stringhe (che non sono zero terminate) e `Vett` contiene `N` elementi byte positivi. I parametri della funzione sono passati mediante lo stack. La funzione deve copiare nella stringa `Dest` in posizione `i` il carattere della stringa `Sorg` in posizione `Vett[i]`. Nel caso `Vett[i]` sia maggiore di `N` si copia in `Dest` l'ultimo carattere di `Sorg` (cioè in posizione `N`).

Ad esempio, le variabili del programma sono le seguenti:

```
Sorg: db "Buon Anno a tutti"
```

```
N: db 17
```

```
Dest: resb 17
```

```
Vett: db 1, 7, 25, 2, 10, 3, 2, 15, 2, 1, 6, 11, 16, 17, 5, 4, 12
```

Per l'esempio riportato, `Dest` alla fine conterrà "Bniu outuBAati n". (**considerando di partire da 1 come indice**).

Si scriva anche il programma main che chiama la funzione.

SECTION data

```
Sorg: db 'Buon Anno a tutti'
```

```
N: db 17
```

```
Dest: resb 17
```

```
Vett: db 1, 7, 25, 2, 10, 3, 2, 15, 2, 1, 6, 11, 16, 17, 5, 4, 12
```

SECTION text

```
..start:
```

```
    mov ax, data
```

```
    mov ds, ax
```

```
    mov ax, Sorg
```

```
    push ax
```

```
    mov ax, Dest
```

```
    push ax
```

```
    xor ah, ah
```

```
    mov al, [N]
```

```
    push ax
```

```
    mov ax, Vett
```

```
    push ax
```

```
    call CopiaCaratteri
```

```
    add sp, 8
```

```
    mov ax, Dest
```

```
    push ax
```

```
    ; Non essendo zero terminato devo modificare ScriviStringa e passare anche la lunghezza
```

```
    xor ah, ah
```

```
    mov al, [N]
```

```
    push ax
```

```
    call ScriviStringa2
```

```
    add sp, 4
```

```
    mov ax, 4c00h
```

```
    int 21h
```

```
CopiaCaratteri:
```

```
    push bp
```

```
    mov bp, sp
```

```
    mov di, [bp+8] ; Dest
```

```
xor ch, ch
mov cl, [bp+6] ; N
mov bx, [bp+4] ; Vett
xor dh, dh
```

Ciclo:

mov si, [bp+10]; Sorg -- lo metto qui per ripristinare sempre il valore iniziale a cui sommare
successivamente

```
mov dl, [bx]
cmp dl, [bp+6] ; se Vett[i] > N metto Sorg[N]
jae maggiore
add si, dx
dec si ; considero che comincio da 1 e non da 0
```

salta:

```
mov al, [si]
mov [di], al
inc di
inc bx
loop Ciclo
jmp Fine
```

maggiore:

```
add si, [bp+6]
dec si ; considero che comincio da 1 e non da 0
jmp salta
```

Fine:

```
pop bp
ret
```

ScriviStringa2:

```
push bp
mov bp, sp
mov si, [bp+6]
mov cx, [bp+4]
mov ah, 0eh
mov bx, 0000h
```

Stampa:

```
lods b
int 10h
loop Stampa
```

fineStampa:

```
pop bp
ret ; Ritorno alla procedura chiamante
```

Scrivere in Assembler per Intel 80x86 la funzione `copiaVettore` che riceve in ingresso due vettori di byte Vett1 e Vett2, e un intero N. Il primo vettore Vett1 è di lunghezza non nota, ma la fine del vettore è identificata dal valore -1. Gli elementi di Vett1 devono essere copiati in Vett2 se e solo se sono maggiori o uguali al valore N. Si abbia cura di terminare (con il valore -1) anche il vettore Vett2.

I parametri della funzione sono passati mediante lo stack. La funzione deve riportare in AX il numero di valori copiati.

Ad esempio, le variabili del programma sono le seguenti:

Vett1: db 2, 5, 7, 4, 55, 22, -1

Vett2: **resb 256**

N: db 6

Per l'esempio riportato, la funzione deve riportare in AX il valore 3 e il vettore Vett2 vale: 7, 55, 22, -1

Si scriva anche il programma main che chiama la funzione.

SECTION data

Vett1: db 2, 5, 7, 4, 55, 22, -1

Vett2: resb 256

N: db 6

SECTION text

..start:

```
    mov ax, data
    mov ds, ax
    mov es, ax
    mov ax, Vett1
    push ax
    mov ax, Vett2
    push ax
    xor ah, ah
    mov al, [N]
    push ax
    call copiaVettori
    add sp, 6
    mov ax, 4c00h
    int 21h
```

copiaVettori:

```
    push bp
    mov bp, sp
    mov si, [bp+8] ; Vett1
    mov di, [bp+6] ; Vett2
    mov cl, [bp+4] ; N
    xor bx, bx           ; uso temporaneamente per contare
```

Ciclo:

```
    lodsb
    cmp al, -1
    je Fine
    cmp al, cl
    jae copia
    jmp Ciclo
```

copia:

```
    stosb
```

```
    inc bx
    jmp Ciclo
Fine:
    ; ho già -1 in AX
    stosb
    mov ax,bx
    pop bp
    ret
```

Scrivere in Assembler per Intel 80x86 la funzione `fondiStringhe` che riceve in ingresso tre stringhe `StringaS1`, `StringaS2` e `StringaD`, e un intero a 16 bit `N` (passati mediante lo stack). Le due stringhe sorgenti `StringaS1` e `StringaS2` sono di lunghezza fissa indicata dal parametro `N`. La funzione deve ricopiare nella stringa destinazione `StringaD` un carattere dalla stringa sorgente con il codice ASCII inferiore. Si tenga presente che le lettere maiuscole hanno codice ASCII inferiore delle lettere minuscole e che lo spazio ha codice ASCII inferiore alle lettere maiuscole. Quindi, ad esempio, se le due stringhe sorgenti sono **“Bzon Pazame”** e **“Cutu Netolo”** (`N=11`), la stringa destinazione deve essere **“Buon Natale”**.

Il programma chiamante ha il seguente codice Assembler:

```
    mov ax, N
    push ax
    mov ax, StringaS1
    push ax
    mov ax, StringaS2
    push ax
    mov ax, StringaD
    push ax
    call fondiStringhe
    add sp, 8
    ret
```

SECTION data

StringaS1: db 'Bzon Pazame'

StringaS2: db 'Cutu Netolo'

StringaD: resb 11

N: db 11

SECTION text

..start:

```
    mov ax, data
    mov ds, ax
    mov es, ax
    mov ax, N
    push ax
    mov ax, StringaS1
    push ax
    mov ax, StringaS2
    push ax
    mov ax, StringaD
    push ax
    call fondiStringhe
    add sp, 8
    mov ax, StringaD
```

```
push ax
mov ax, N
push ax
call ScriviStringa2
mov ax, 4c00h
int 21h
```

fondiStringhe:

```
push bp
mov bp, sp
mov bx, [bp+10]      ; indirizzo di N
xor ch, ch
mov cl, [bx]         ; N
mov si, [bp+8]       ; StringaS1
mov bx, [bp+6]       ; StringaS2
mov di, [bp+4]       ; StringaD
```

Ciclo:

```
lodsb
mov ah, [bx]
cmp al, ah
ja mettoS2
stosb
inc bx
```

salta:

```
loop Ciclo
jmp Fine
```

mettoS2:

```
xchg al, ah
stosb
inc bx
jmp salta
```

Fine:

```
pop bp
ret
```

ScriviStringa2:

```
push bp
mov bp, sp
mov si, [bp+6]
mov bx, [bp+4]
xor ch, ch
mov cl, [bx]
mov ah, 0eh
mov bx, 0000h
```

Stampa:

```
lodsb
int 10h
loop Stampa
```

fineStampa:

```
pop bp
ret      ;Ritorno alla procedura chiamante
```

Scrivere in Assembler per Intel 80x86 la funzione `calcolaSAD` che riceve in ingresso un vettore di word (dati a 16 bit) `Vett` di lunghezza `N` nota (entrambi i parametri sono passati mediante lo stack). Ogni elemento a 16 bit del vettore è in realtà composto da due valori a 8 bit (il più significativo MSB e il meno significativo LSB). La funzione deve scorrere il vettore e ogni qualvolta il MSB è di valore inferiore al LSB, li deve scambiare nel vettore. Inoltre, la funzione deve calcolare la somma delle differenze in valore assoluto tra MSB e LSB ($SAD = \text{Sum of Absolute Differences}$) e riportarla in uscita nel registro `AX`.

Il programma chiamante ha il seguente codice Assembler:

```
mov ax, Vett
push ax
mov ax, [N]
push ax
call calcolaSAD
add sp, 4
ret
```

Se, ad esempio, i valori di `Vett` e `N` fossero i seguenti:

`Vett: dw 1234h, 1144h, 4412h, 2323h, 2324h, 2423h`

`N: dw 6`

il registro `AX` dovrebbe riportare il valore `89h` ($=22h+33h+32h+0h+1h+1h$). Inoltre il vettore dovrebbe diventare il seguente (notare gli scambi di MSB con LSB):

`Vett: dw 3412h, 4411h, 4412h, 2323h, 2423h, 2423h`

SECTION data

`Vett: dw 1234h, 1144h, 4412h, 2323h, 2324h, 2423h`

`N: dw 6`

SECTION text

`..start:`

```
mov ax, data
mov ds, ax
mov ax, Vett
push ax
mov ax, [N]
push ax
call calcolaSAD
add sp, 4
mov ax, 4c00h
int 21h
```

calcolaSAD:

```
push bp
mov bp, sp
mov cx, [bp+4] ; N (gia word)
mov si, [bp+6] ; Vett
mov dx, 0 ; accumulo temporaneamente la SAD in DX
```

`Ciclo:`

```
lodsw
cmp ah, al ; confronto MSB (ah) con LSB (al)
ja MSBmaggiore
xchg al, ah
mov [si-2], ax ; con il lodsw SI è stato incrementato di 2
```

MSBmaggiore:

```
sub ah,al
xchg al, ah
xor ah, ah
add dx, ax
loop Ciclo
jmp Fine
```

Fine:

```
mov ax, dx
pop bp
ret
```

Scrivere in Assembler per Intel 80x86 la funzione `mischia_stringhe` che riceve in ingresso due stringhe zero-terminate Sorg1 e Sorg2 e una stringa Dest (sempre zero terminata). La funzione deve copiare in Dest alternativamente un carattere da Sorg1 e uno da Sorg2, partendo da Sorg1. Se una delle due stringhe termina prima, i rimanenti caratteri dell'altra stringa vanno ricopiato in Dest. Tutti i parametri sono passati mediante lo stack.

Esempio:

```
Sorg1:      db 'Evviva',0
Sorg2:      db 'Carnevale',0
Dest:       resb 100
```

La funzione deve copiare in Dest la seguente stringa: 'ECvavrinveavale',0
Si scriva anche il programma main che chiama la funzione.

SECTION data

```
Sorg1: db 'Evviva',0
Sorg2: db 'Carnevale',0
Dest:  resb 100
```

SECTION text

..start:

```
mov ax, data
mov ds, ax
mov es, ax
mov ax, Sorg1
push ax
mov ax, Sorg2
push ax
mov ax, Dest
push ax
call mischia_stringhe
add sp, 6
mov ax, Dest
push ax
call ScriviStringa
add sp, 2
mov ax, 4c00h
int 21h
```

mischia_stringhe:

```
push bp
```

```
mov bp, sp
mov si, [bp+8] ; Sorg1
mov bx, [bp+6] ; Sorg2
mov di, [bp+4] ; Dest
```

Ciclo:

```
lodsb
cmp al, 0
je FineSorg1
stosb
mov al, [bx]
cmp al, 0
je FineSorg2
stosb
inc bx
jmp Ciclo
```

FineSorg1:

```
; finisco di copiare Sorg2
mov al, [bx]
cmp al, 0
je Fine
stosb
inc bx
jmp FineSorg1
```

FineSorg2:

```
; finisco di copiare Sorg1
lodsb
cmp al, 0
je Fine
stosb
jmp FineSorg2
```

Fine:

```
stosb ; in ogni caso in al ho 0 e devo terminare Dest
pop bp
ret
```

ScriviStringa:

```
push bp
mov bp, sp
mov si,[bp+4]
mov ah,0eh
mov bx,0000h
```

Stampa:

```
lodsb
cmp al, 0
je fineStampa
int 10h
jmp Stampa
```

fineStampa:

```
pop bp
ret ;Ritorno alla procedura chiamante
```

Scrivere in Assembler per Intel 80x86 la funzione `calcolalunghezzastringa` che riceve in ingresso una stringa ASCIIZ (stringa zero terminata, come in C) `Stringa` (il cui indirizzo è passato mediante lo stack). La funzione deve calcolare la lunghezza della stringa e riportarla in `AX`. Inoltre deve riportare in `CX` il valore 1 se tale lunghezza è un numero pari e 0 se è un numero dispari.

Si scriva anche il programma `main` che chiama la funzione.

SECTION data

`Stringa:db 'Carnevale',0`

SECTION text

`..start:`

```
    mov ax, data
    mov ds, ax
    mov ax, Stringa
    push ax
    call calcolalunghezzastringa
    add sp, 2
    mov ax, 4c00h
    int 21h
```

`calcolalunghezzastringa:`

```
    push bp
    mov bp, sp
    mov si, [bp+4] ; Stringa
    xor bx, bx          ; temporaneo per conteggio caratteri
```

`Ciclo:`

```
    lodsb
    cmp al, 0
    je Fine
    inc bx
    jmp Ciclo
```

`Fine:`

```
    mov ax, bx
    test ax, 1b
    je pari
    mov cx, 0
    jmp salta
```

`pari:`

```
    mov cx, 1
```

`salta:`

```
    pop bp
    ret
```

Scrivere in Assembler per Intel 80x86 la funzione `estrai_sottostringa` che riceve in ingresso una stringa ASCIIZ (stringa zero terminata, come in C) `Stringa1`, un carattere `Car` e un intero `N` (tutti passati mediante lo stack). La funzione deve restituire nella stringa ASCIIZ `Stringa2` (questa volta inserita come variabile) gli `N` caratteri di `Stringa1` successivi alla prima occorrenza di `Car`. Se `Car` non è presente in `Stringa1`, `Stringa2` rimarrà stringa nulla (vuota). Se ci sono meno di `N` carattere tra la prima occorrenza di `Car` e la fine di `Stringa1`, in `Stringa2` vengono copiati solo i caratteri presenti. Esempi:

```
Stringa1= "Pomodoro" Car='o' N=4 --> Stringa2="modo"
Stringa1= "Pomodoro" Car='m' N=3 --> Stringa2="odo"
Stringa1= "Pomodoro" Car='a' N=5 --> Stringa2=""
Stringa1= "Pomodoro" Car='d' N=5 --> Stringa2="oro"
```

Ricordare di zero-terminare la stringa `Stringa2`.

Si scriva anche il programma main che chiama la funzione.

SECTION data

`Stringa1: db 'Pomodoro',0`

`Stringa2: resb 100`

`Car: db '0'`

`N: db 4`

SECTION text

`..start:`

```
mov ax, data
mov ds, ax
mov es, ax
mov ax, Stringa1
push ax
xor ah, ah
mov al, [Car]
push ax
mov al, [N]
push ax
call estrai_sottostringa
add sp, 6
mov ax, Stringa2
push ax
call ScriviStringa
add sp, 2
mov ax, 4c00h
int 21h
```

`estrai_sottostringa:`

```
push bp
mov bp, sp
mov si, [bp+8] ; Stringa1
mov di, Stringa2
mov cx, [bp+4] ; N
mov dl, [bp+6] ; Car
```

`Ciclo:`

```
lodsb
cmp al, 0
je Fine
cmp al, dl
```

```

        je Trovato
        jmp Ciclo
Trovato:
        lodsb
        cmp al, 0
        je Fine
        stosb
        loop Trovato
        jmp Fine
Fine:
        mov al, 0
        stosb
        pop bp
        ret

ScriviStringa:
        push bp
        mov bp, sp
        mov si,[bp+4]
        mov  ah,0eh
        mov  bx,0000h
Stampa:
        lodsb
        cmp al, 0
        je fineStampa
        int  10h
        jmp Stampa
fineStampa:
        pop bp
        ret          ;Ritorno alla procedura chiamante

```

Scrivere in Assembler per Intel 80x86 la funzione `seleziona_caratteri` che riceve in ingresso 4 parametri: un valore byte N, un vettore di byte V contenente N elementi, una stringa di caratteri Sorg memorizzata secondo convenzione Pascal (il primo byte contiene la lunghezza della stringa – si veda esempio sotto) e una stringa di caratteri Dest sempre con convenzione Pascal e lunga N-1 caratteri. Tutti i parametri sono passati mediante lo stack.

La funzione deve scorrere V sommando due valori consecutivi. Il risultato della somma indica l'indice del carattere di Sorg che deve essere copiato in Dest. Se tale somma supera la lunghezza di Sorg, in Dest deve essere copiato l'ultimo carattere di Sorg. Esempio:

```

N:      db 6
V:      db 1, 26, 6, 2, 6, 7
Sorg: db 31, "Buon Natale e Felice Anno Nuovo"
Dest: db 5, resb 5

```

Notare che Sorg e Dest NON sono zero-terminated e che il primo byte di Sorg e Dest indica la loro lunghezza. Si realizzi una soluzione che funziona sempre, non solo con i dati riportati come esempio! Suggerimento: se i registri a disposizione non dovessero bastare, usare lo stack per memorizzare temporaneamente i registri, usarli e poi recuperarli dallo stack una volta finito di usarli. Si scriva anche il programma main che chiama la funzione.

SECTION data

```

N:      db 6

```

```
V:      db 1, 26, 6, 2, 6, 7
Sorg:   db 31, 'Buon Natale e Felice Anno Nuovo'
Dest:   db 5
        resb 5
```

SECTION text

..start:

```
    mov ax, data
    mov ds, ax
    mov es, ax
    mov ax, Sorg
    push ax
    mov ax, V
    push ax
    xor ah, ah
    mov al, [N]
    push ax
    mov ax, Dest
    push ax
    call seleziona_caratteri
    add sp, 8
    mov ax, Dest
    push ax
    call ScriviStringa3          ; da modificare perchè stringa stile Pascal
    add sp, 2
    mov ax, 4c00h
    int 21h
```

seleziona_caratteri:

```
    push bp
    mov bp, sp
    mov si, [bp+10]; Sorg
    mov di, [bp+4] ; Dest
    inc di          ; mi sposto sul primo carattere corretto
    mov cx, [bp+6] ; N - dato in cl
    mov bx, [bp+8] ; V
    mov dl, [si]    ; lunghezza di Sorg
```

Ciclo:

```
    mov al, [bx]
    mov ah, [bx+1]
    add al, ah
    cmp al, dl
    jbe ok
    mov al, dl      ; maggiore della lunghezza, metto N
```

ok:

```
    push bx
    xor ah, ah
    mov bx, ax
    mov al, [si+bx]
    stosb
    pop bx
    inc bx
    loop Ciclo
```

```
pop bp
ret
```

ScriviStringa3:

```
push bp
mov bp, sp
mov si,[bp+4]
xor ch, ch
mov cl, [si]
inc si
mov ah,0eh
mov bx,0000h
```

Stampa:

```
lods b
int 10h
loop Stampa
pop bp
```

```
ret ;Ritorno alla procedura chiamante
```

Scrivere in Assembler per Intel 80x86 la funzione `correggi_stringa` che riceve in ingresso 2 parametri: una stringa di caratteri zero-terminata `Sorg` e un singolo carattere `Car`. Tutti i parametri sono passati mediante lo stack.

La funzione deve scorrere la stringa `Sorg` e cercare il carattere `Car`. Dove lo trova, deve sostituire il carattere `Car` con il carattere precedente nel codice ASCII.

Esempio:

```
Sorg: db "Buon Natale e Felice Anno Nuovo",0
Car: db 'n'
```

In questo caso alla fine in `Sorg` ci dovrebbe essere la stringa zero-terminata "Buom Natale e Felice Ammo Nuovo".

Si scriva anche il programma `main` che chiama la funzione.

SECTION data

```
Sorg: db 'Buon Natale e Felice Anno Nuovo',0
Car: db 'n'
```

SECTION text

..start:

```
mov ax, data
mov ds, ax
mov ax, Sorg
push ax
xor ah, ah
mov al, [Car]
push ax
call correggi_stringa
add sp, 4
mov ax, Sorg
push ax
call ScriviStringa
add sp, 2
mov ax, 4c00h
```

int 21h

correggi_stringa:

```
push bp
mov bp, sp
mov si, [bp+6] ; Sorg
mov cl, [bp+4] ; Car
```

Ciclo:

```
lodsb
cmp al, 0
je Fine
cmp al, cl
je Trovato
jmp Ciclo
```

Trovato:

```
dec al
mov [si-1], al ; -1 perchè lodsb ha già incrementato SI
jmp Ciclo
```

Fine:

```
pop bp
ret
```

ScriviStringa:

```
push bp
mov bp, sp
mov si,[bp+4]
mov ah,0eh
mov bx,0000h
```

Stampa:

```
lodsb
cmp al, 0
je fineStampa
int 10h
jmp Stampa
```

fineStampa:

```
pop bp
ret ;Ritorno alla procedura chiamante
```

Scrivere in Assembler per Intel 80x86 la funzione `concatena_stringhe` che riceve in ingresso 3 parametri: due stringhe sorgenti di caratteri zero-terminate `Sorg1` e `Sorg2`, e una stringa destinazione (che deve essere zero terminata) `Dest`. Tutti i parametri sono passati mediante lo stack.

La funzione deve copiare in `Dest` le due stringhe sorgenti, una di seguito all'altra, mettendo prima quella più corta (nel caso siano lunghe uguali, l'ordine non importa). Per calcolare la lunghezza delle stringhe si consiglia di definire una funzione apposita (che va comunque riportata!).

Esempio:

```
Sorg1:      db "Vi auguro ",0
Sorg2:      db "un felice 2009",0
Dest:       resb 100
```

In questo caso alla fine in `Dest` ci dovrebbe essere la stringa zero-terminata "Vi auguro un felice 2009".

Si scriva anche il programma main che chiama la funzione.

SECTION data

```
Sorg1: db 'Vi auguro ',0
Sorg2: db 'un felice 2009',0
Dest:  resb 100
```

SECTION text

..start:

```
mov ax, data
mov ds, ax
mov es, ax
mov ax, Sorg1
push ax
mov ax, Sorg2
push ax
mov ax, Dest
push ax
call concatena_stringhe
add sp, 6
mov ax, Dest
push ax
call ScriviStringa
add sp, 2
mov ax, 4c00h
int 21h
```

`concatena_stringhe`:

```
push bp
mov bp, sp
mov si, [bp+8] ; Sorg1
mov bx, [bp+6] ; Sorg2
mov di, [bp+4] ; Dest
push si
call LungStringa
add sp, 2
mov dx, ax
push bx
call LungStringa
add sp, 2
mov cx, ax
```

```

        cmp dx, cx
        je Maggiore1
Maggiore2:
        mov al, [bx]
        cmp al, 0
        je FineSorg2
        stosb
        inc bx
        jmp Maggiore2
Maggiore1:
        lodsb
        cmp al, 0
        je FineSorg1
        stosb
        jmp Maggiore1
FineSorg2:
        lodsb
        cmp al, 0
        je Fine
        stosb
        jmp FineSorg2
FineSorg1:
        mov al, [bx]
        cmp al, 0
        je Fine
        stosb
        inc bx
        jmp FineSorg1
Fine:
        stosb          ; ho già in al il terminatore che uso per terminare Dest
        pop bp
        ret

LungStringa:
        push bp
        mov bp, sp
        push si
        xor cx, cx          ; temporaneo per il calcolo lunghezza -- devo usare cx perchè nella chiamata
lo uso per secondo (dopo DX)
        mov si, [bp+6] ; +6 e non +4 perchè ho fatto anche push si per salvare il valore di si
CicloFunz:
        lodsb
        cmp al, 0
        je FineFunz
        inc cx
        jmp CicloFunz
FineFunz:
        mov ax, cx
        pop si
        pop bp
        ret

```

ScriviStringa:

```

push bp
mov bp, sp
mov si,[bp+4]
mov  ah,0eh
mov  bx,0000h

```

Stampa:

```

lodsb
cmp al, 0
je fineStampa
int 10h
jmp Stampa

```

fineStampa:

```

pop bp
ret ;Ritorno alla procedura chiamante

```

Scrivere in Assembler per Intel 80x86 la funzione `trova_max` che riceve in ingresso un vettore `Vett` di byte (dati a 8 bit) e un valore intero `N` (che rappresenta il numero di elementi del vettore `Vett`). La funzione deve trovare il valore massimo in `Vett` e riportare in `AX` l'indice (posizione) corrispondente in `Vett` (nel caso di più elementi con il valore massimo si deve riportare l'ultimo). Tutti i parametri sono passati mediante lo stack.

Esempio:

```

Vett: db 4, 7, 2, 7, 5
N:    db 5

```

La funzione deve ritornare in `AX` il **valore 3** (si ricorda che il primo elemento del vettore ha indice 0).

Si scriva anche il programma `main` che chiama la funzione.

SECTION data

```

Vett: db 4, 7, 2, 7, 5
N:    db 5

```

SECTION text

..start:

```

mov ax, data
mov ds, ax
mov ax, Vett
push ax
xor ah, ah
mov al, [N]
push ax
call trova_max
add sp, 4
mov ax, 4c00h
int 21h

```

trova_max:

```

push bp
mov bp, sp
mov si, [bp+6] ; Vett
mov cx, [bp+4] ; N
mov dh, -127 ; attuale massimo (valore minimo possibile con 8 bit)
mov dl, -1 ; indice attuale
mov bl, -1 ; contatore dell'indice nel vettore

```

```

        xor ah, ah
Ciclo:   lodsb
        inc bl
        cmp al, dh
        jge NuovoMassimo ; per riportare l'indice dell'ultimo massimo in caso di valore uguali
devo usare jge invece di jg
salta:   loop Ciclo
        jmp Fine
NuovoMassimo:
        mov dh, al
        mov dl, bl
        jmp salta
Fine:    xor dh, dh
        mov ax, dx
        pop bp
        ret

```

Scrivere in Assembler per Intel 80x86 la funzione `elimina_lettera` che riceve in ingresso una stringa Sorg (zero terminata), una stringa Dest (sempre zero terminata) e un carattere (byte) C. La funzione deve copiare in Dest la stringa Sorg eccetto per i caratteri C che si trovano in posizione pari (il primo carattere è in posizione 0 da considerare come posizione pari). Tutti i parametri sono passati mediante lo stack.

Esempio:

```

Sorg: db 'Ormai inizia l'estate',0
Dest: resb 100
C:    db 'i'

```

La funzione deve copiare in Dest la seguente stringa: 'Ormai inizia l'estate',0
 Si scriva anche il programma main che chiama la funzione.

SECTION data

```

Sorg: db "Ormai inizia l'estate",0
Dest: resb 100
C:    db 'i'

```

SECTION text

```

..start:
        mov ax, data
        mov ds, ax
        mov es, ax
        mov ax, Sorg
        push ax
        mov ax, Dest
        push ax
        xor ah, ah
        mov al, [C]
        push ax
        call elimina_lettera
        add sp, 6
        mov ax, Dest

```

```
push ax
call ScriviStringa
add sp, 2
mov ax, 4c00h
int 21h
```

elimina_lettera:

```
push bp
mov bp, sp
mov si, [bp+8] ; Sorg
mov di, [bp+6] ; Dest
mov dl, [bp+4] ; C
mov dh, 0 ; contatore nella stringa Sorg
```

Ciclo:

```
lodsb
cmp al, 0
je Fine
cmp al, dl
jne copia
test dh, 00000001b
jne copia
```

salta:

```
inc dh
jmp Ciclo
```

copia:

```
stosb
jmp salta
```

Fine:

```
stosb ; termino la stringa Dest
pop bp
ret
```

ScriviStringa:

```
push bp
mov bp, sp
mov si, [bp+4]
mov ah, 0eh
mov bx, 0000h
```

Stampa:

```
lodsb
cmp al, 0
je fineStampa
int 10h
jmp Stampa
```

fineStampa:

```
pop bp
ret ; Ritorno alla procedura chiamante
```

Scrivere in Assembler per Intel 80x86 la funzione `trova_min` che riceve in ingresso un vettore `Vett` di byte (dati a 8 bit) e un valore intero `N` (che rappresenta il numero di elementi del vettore `Vett`). La funzione deve trovare il valore minimo in `Vett` e riportare in `AX` l'indice (posizione) corrispondente in `Vett` (nel caso di più elementi con il valore minimo si deve riportare l'ultimo). Tutti i parametri sono passati mediante lo stack.

Esempio:

```
Vett: db 4, 7, 2, 2, 5
N:    db 5
```

La funzione deve ritornare in `AX` il valore 3 (si ricorda che il primo elemento del vettore ha indice 0). Si scriva anche il programma `main` che chiama la funzione.

SECTION data

```
Vett: db 4, 7, 2, 2, 5
N:    db 5
```

SECTION text

..start:

```
mov ax, data
mov ds, ax
mov ax, Vett
push ax
xor ah, ah
mov al, [N]
push ax
call trova_min
add sp, 4
mov ax, 4c00h
int 21h
```

trova_min:

```
push bp
mov bp, sp
mov si, [bp+6] ; Vett
mov cx, [bp+4] ; N
mov dh, 127 ; attuale minimo (valore massimo possibile con 8 bit)
mov dl, -1 ; indice attuale
mov bl, -1 ; contatore dell'indice nel vettore
xor ah, ah
```

Ciclo:

```
lodsb
inc bl
cmp al, dh
jle NuovoMinimo ; per riportare l'indice dell'ultimo massimo in caso di valore uguali devo
```

usare `jle` invece di `jl`

salta:

```
loop Ciclo
jmp Fine
```

NuovoMinimo:

```
mov dh, al
mov dl, bl
jmp salta
```

Fine:

```
xor dh, dh
```

```
mov ax, dx
pop bp
ret
```

Scrivere in Assembler per Intel 80x86 la funzione `sommaparidispari` che analizza un vettore `Vett` di lunghezza `N` (con `Vett` e `N` parametri della funzione passati mediante lo stack) e calcola la somma dei valori pari e la somma dei valori dispari. La funzione riporta in `AX` la differenza tra questi due valori.

Dati questi valori:

```
Vett: db 13,15,22,7,5,3,21,2,0,10
N: db 10
```

la funzione riporta in `AX` il **valore 24 (somma pari = 61, somma dispari = 37)**.

Si scriva anche il programma main che chiama la funzione.

SECTION data

```
Vett: db 13,15,22,7,5,3,21,2,0,10
```

```
N: db 10
```

SECTION text

`..start:`

```
    mov ax, data
    mov ds, ax
    mov ax, Vett
    push ax
    xor ah, ah
    mov al, [N]
    push ax
    call sommaparidispari
    add sp, 4
    mov ax, 4c00h
    int 21h
```

`sommaparidispari:`

```
    push bp
    mov bp, sp
    mov si, [bp+6] ; Vett
    mov cl, [bp+4] ; N
    dec cl          ; da 0 a N-1
    mov bx, 0       ; somma pari
    mov dx, 0       ; somma dispari
    mov ch, -1      ; contatore dell'indice nel vettore
    xor ah, ah
```

`Ciclo:`

```
    lodsb
    inc ch
    test ch, 00000001b
    je pari
    add dx, ax
```

`salta:`

```
    cmp ch, cl      ; fine vettore?
    je Fine
```

```

        jmp Ciclo
pari:
        add bx, ax
        jmp salta
Fine:
        sub bx, dx
        mov ax, bx
        pop bp
        ret

```

Scrivere in Assembler per Intel 80x86 la funzione `InvertiStringa` che riceve in ingresso due stringhe `StringaS` e `StringaD`, e un intero a 16 bit `N` (passati mediante lo stack). La stringa sorgente `StringaS` è di lunghezza fissa indicata dal parametro `N`. La funzione deve ricopiare nella stringa destinazione `StringaD` i caratteri della stringa sorgente, ma in ordine inverso e un carattere sì e uno no. Quindi, ad esempio, se la stringa sorgente è “Andrea Prati” (`N=12`), la stringa destinazione deve essere “iaParn”.

Si scriva anche il programma main che chiama la funzione.

```

SECTION data
StringaS: db 'Andrea Prati'
StringaD: resb 100
N: db 12

```

SECTION text

```

..start:
        mov ax, data
        mov ds, ax
        mov es, ax
        mov ax, StringaS
        push ax
        mov ax, StringaD
        push ax
        xor ah, ah
        mov al, [N]
        push ax
        call InvertiStringa
        add sp, 6
        mov ax, StringaD
        push ax
        mov al, ch                ; InvertiStringa mi ha lasciato in ch i caratteri scritti in StringaD
        push ax
        call ScriviStringa2
        add sp, 4
        mov ax, 4c00h
        int 21h

```

```

InvertiStringa:
        push bp
        mov bp, sp
        mov si, [bp+8] ; StringaS
        mov di, [bp+6] ; StringaD

```

```

    mov bx, [bp+4] ; N
    mov cl, 0FFh   ; booleano per decidere se scrivere (FF=si, 00=no)
    xor ch, ch      ; conto i caratteri scritti in StringaD

Ciclo:
    mov al, [si+bx-1] ; ultimo carattere di StringaS
    cmp cl, 0FFh
    je scrivi

salta:
    not cl           ; nego cl (FF diventa 00 e viceversa)
    dec bx
    je Fine
    jmp Ciclo

scrivi:
    stosb
    inc ch
    jmp salta

Fine:
    pop bp
    ret

ScriviStringa2:
    push bp
    mov bp, sp
    mov si, [bp+6]
    mov cx, [bp+4]
    mov ah, 0eh
    mov bx, 0000h

Stampa:
    lodsb
    int 10h
    loop Stampa

fineStampa:
    pop bp
    ret ;Ritorno alla procedura chiamante

```

Implementare una funzione assembly che:

1. prenda in ingresso l'indirizzo in memoria di una stringa, ovvero di una sequenza di caratteri terminata da 0
2. restituisca
 - 1 se tutti i caratteri contenuti sono dei numeri
 - 2 se tutti i caratteri sono lettere minuscole o maiuscole
 - 0 altrimenti (cioè se ci sono sia numeri che lettere o se ci sono altri caratteri non alfanumerici o stringa vuota, o tutti gli altri casi)

Si scriva anche il programma main che chiama la funzione.

SECTION data

Stringa: db '12124',0

SECTION text

..start:

mov ax, data

```
mov ds, ax
mov ax, Stringa
push ax
call valutaStringa
add sp, 2
mov ax, 4c00h
int 21h
```

valutaStringa:

```
push bp
mov bp, sp
mov si, [bp+4] ; Stringa
mov cl, 0 ; conta numeri
mov ch, 0 ; conta lettere minuscole o maiuscole
mov dl, 0 ; conta lunghezza stringa
xor ah, ah
```

Ciclo:

```
lodsb
cmp al, 0
je Fine
push ax
call ENumero
add sp, 2
cmp bl, 1 ; numero
je numero
push ax
call ELettera
add sp, 2
cmp bl, 1 ; lettera
je lettera
```

salta:

```
inc dl
jmp Ciclo
```

numero:

```
inc cl
jmp salta
```

lettera:

```
inc ch
jmp salta
```

Fine:

```
cmp ch, dl
je SoloLettere
cmp cl, dl
je SoloNumeri
mov ax, 0
jmp FineFine
```

SoloLettere:

```
mov ax, 2
jmp FineFine
```

SoloNumeri:

```
mov ax, 1
```

FineFine:

```
pop bp
```

ret

ENumero:

```
push bp
mov bp, sp
mov bl, 0
mov al, [bp+4]
cmp al, '0'
jb FineENumero
cmp al, '9'
ja FineENumero
mov bl, 1
```

FineENumero:

```
pop bp
ret
```

ELettera:

```
push bp
mov bp, sp
mov bl, 0
mov al, [bp+4]
cmp al, 'A'
jb FineELettera
cmp al, 'z'
ja FineELettera
; qui siamo sicuramente tra 'A' e 'z'
cmp al, 'Z'
jbe okLettera
; qui siamo tra dopo 'Z' e prima di 'z'
cmp al, 'a'
jb FineELettera
```

okLettera:

```
mov bl, 1
```

FineELettera:

```
pop bp
ret
```

Implementare una funzione assembly che:

1. prenda in ingresso
 - l'indirizzo in memoria di un vettore di interi positivi (32 bit)
 - la lunghezza del vettore (sempre inferiore a 1000 elementi) espressa come intero a 16 bit
2. restituisca la media aritmetica del vettore espressa come un intero a 16 bit

Successivamente implementare una funzione assembly che:

1. prenda in ingresso
 - l'indirizzo in memoria di un vettore di interi positivi (32 bit)
 - la lunghezza del vettore (sempre inferiore a 1000 elementi) espressa come intero a 16 bit
 - la media aritmetica del vettore espressa come un intero a 16 bit
2. restituisca la percentuale (come valore intero compreso tra 0 e 100 approssimato per difetto) dei valori strettamente superiori alla media

Si scriva anche il programma main che chiama la funzione.

SECTION data

Vett: dw 23, 21, 45, 43

N: dw 4

SECTION text

..start:

```
mov ax, data
mov ds, ax
mov ax, Vett
push ax
mov ax, [N]
push ax
call calcolaMedia
add sp, 4
; in AX ho la media
push ax
mov ax, Vett
push ax
mov ax, [N]
push ax
call calcolaPercentuale
add sp, 6
; in AX ho la percentuale 0-100
mov ax, 4c00h
int 21h
```

calcolaMedia:

```
push bp
mov bp, sp
mov si, [bp+6] ; Vett
mov cx, [bp+4] ; N
mov dx, 0
```

Ciclo:

```
lodsw
add dx, ax
loop Ciclo
xor ax, ax
xchg dx, ax ; ora in dx:ax ho la somma a 32 bit
```

```

    div word [bp+4]      ; divido per N per avere la media (risultato in AX, resto in DX)
    pop bp
    ret

```

calcolaPercentuale:

```

    push bp
    mov bp, sp
    mov si, [bp+6] ; Vett
    mov dx, [bp+8] ; media
    mov cx, [bp+4] ; N
    xor bx, bx      ; numero valori superiori alla media

```

Ciclo2:

```

    lodsw
    cmp ax, dx
    jbe nonsuperiore
    inc bx

```

nonsuperiore:

```

    loop Ciclo2
    ; in bx ho il numero di valori superiori alla media
    ; per avere la percentuale devo moltiplicare per 100 e dividere per N
    mov ax, bx
    mov dx, 100
    mul dx ; risultato in DX:AX
    div word [bp+4] ; risultato in AX e resto in DX
    pop bp
    ret

```

Implementare una funzione assembly che:

- prenda in ingresso:
 - l'indirizzo in memoria di una stringa sorgente (ovvero di una sequenza di caratteri terminata da 0 o da \$)
 - l'indirizzo in memoria di una stringa destinazione
- scriva nell'indirizzo destinazione una sequenza di caratteri terminata da 0 o da \$ che contenga tutti i caratteri della stringa sorgente escluse le vocali **maiuscole e minuscole**.

Si scriva anche il programma main che chiama la funzione.

SECTION data

Sorg: db 'Andrea Prati',0

Dest: resb 100

SECTION text

..start:

```

    mov ax, data
    mov ds, ax
    mov es, ax
    mov ax, Sorg
    push ax
    mov ax, Dest
    push ax
    call copiaStringaSpeciale
    add sp, 4

```

```
mov ax, Dest
push ax
call ScriviStringa
add sp, 2
mov ax, 4c00h
int 21h
```

copiaStringaSpeciale:

```
push bp
mov bp, sp
mov si, [bp+6] ; Sorg
mov di, [bp+4] ; Dest
xor ah, ah
```

Ciclo:

```
lodsb
cmp al, 0 ; eventualmente modificare con $
je Fine
push ax
call EVocale
add sp, 2
cmp bl, 1
je noncopiare
stosb
```

noncopiare:

```
jmp Ciclo
```

Fine:

```
pop bp
ret
```

EVocale:

```
push bp
mov bp, sp
mov bl, 0
mov al, [bp+4]
cmp al, 'a'
je vocale
cmp al, 'e'
je vocale
cmp al, 'i'
je vocale
cmp al, 'o'
je vocale
cmp al, 'u'
je vocale
cmp al, 'A'
je vocale
cmp al, 'E'
je vocale
cmp al, 'I'
je vocale
cmp al, 'O'
je vocale
cmp al, 'U'
```

```

        je vocale
        jmp FineFunz
vocale:
        mov bl, 1
FineFunz:
        pop bp
        ret

ScriviStringa:
        push bp
        mov bp, sp
        mov si,[bp+4]
        mov ah,0eh
        mov bx,0000h

Stampa:
        lodsb
        cmp al, 0
        je fineStampa
        int 10h
        jmp Stampa
fineStampa:
        pop bp
        ret                ;Ritorno alla procedura chiamante

```

Implementare una funzione assembly che:

- prenda in ingresso:
 - l'indirizzo in memoria di una stringa (ovvero di una sequenza non vuota di caratteri terminata da 0 o da \$, a vostra scelta)
 - l'indirizzo di un'area di memoria sufficientemente ampia
- copi nell'area di memoria tutti i caratteri numerici della stringa
- restituisca in uscita un intero pari alla somma dei valori numerici letti

ESEMPI:

Avendo in ingresso la stringa "Stavo andando a 100 all'ora", l'area di memoria conterrà la stringa "100", ed in uscita ci sarà 1

Avendo in ingresso la stringa "44 gatti in fila per 6 col resto di 2", l'area di memoria conterrà la stringa "4462", ed in uscita ci sarà 16

Si scriva anche il programma main che chiama la funzione.

SECTION data

Sorg: db "44 gatti in fila per 6 col resto di 2",0

Dest: resb 100

SECTION text

..start:

```

        mov ax, data
        mov ds, ax
        mov es, ax
        mov ax, Sorg

```

```

push ax
mov ax, Dest
push ax
call copiaconta
add sp, 4
; in dx ho il numero sommato
mov ax, Dest
push ax
call ScriviStringa
add sp, 2
mov ax, 4c00h
int 21h

```

copiaconta:

```

push bp
mov bp, sp
mov si, [bp+6] ; Sorg
mov di, [bp+4] ; Dest
xor dx, dx      ; somma dei numeri
xor ah, ah

```

Ciclo:

```

lodsb
cmp al, 0 ; eventualmente modificare con $
je Fine
push ax
call ENumero
add sp, 2
cmp bl, 1
jne noncopiare
stosb
sub al, '0' ; trasformo il carattere nel suo numero corrispondente
add dx, ax

```

noncopiare:

```

jmp Ciclo

```

Fine:

```

pop bp
ret

```

ENumero:

```

push bp
mov bp, sp
mov bl, 0
mov al, [bp+4]
cmp al, '0'
jb FineENumero
cmp al, '9'
ja FineENumero
mov bl, 1

```

FineENumero:

```

pop bp
ret

```

ScriviStringa:

```
push bp
mov bp, sp
mov si,[bp+4]
mov  ah,0eh
mov  bx,0000h
```

Stampa:

```
lodsb
cmp al, 0
je fineStampa
int  10h
jmp Stampa
```

fineStampa:

```
pop bp
ret      ;Ritorno alla procedura chiamante
```
