1.1.1 Esercizio – conta le occorrenze di un carattere in una stringa

Scrivere un programma che legge una stringa di memoria lunga un numero arbitrario di caratteri (ma terminata da \0), inserita in un buffer di memoria di indirizzo noto, e conta le volte che appare il carattere specificato dentro un'altra locazione di memoria. Il risultato viene messo in una terza locazione di memoria.

```
.GLOBAL _main

.DATA
stringa:     .ASCIZ "Stringa di caratteri ASCII che usiamo come esempio"

lettera:     .BYTE 'e'
conteggio:   .BYTE 0x00

.TEXT
_main:       NOP
             MOV $0x00, %CL
             LEA stringa, %ESI
             MOV lettera, %AL
comp:        CMPB $0x00, (%ESI)
             JE fine
             CMP (%ESI),%AL
             JNE poi
             INC %CL
poi:         INC %ESI
             JMP comp
fine:        MOV %CL, conteggio
             RET
```

Variazione sul tema per l'indirizzamento (uso di displacement + registro di modifica):

```
             MOV $0, %ESI
             MOV lettera, %AL
comp:        CMPB $0x00, stringa(%ESI)
             JE fine
             CMP stringa(%ESI),%AL
```

1.1.2 Esercizio – calcolo del fattoriale

Si scriva un programma che calcola il fattoriale di un numero naturale (da 0 a 9) contenuto nella variabile `dato`, di tipo byte. Il risultato deve essere inserito in una variabile `risultato`, di dimensione opportuna. Si controlli che `dato` non ecceda 9. Prestare attenzione al dimensionamento della moltiplicazione.

```
.GLOBAL _main

.DATA
numero:      .BYTE 9
risultato:   .LONG 1

.TEXT
_main:       NOP

             MOV $0, %ECX
             MOV $1, %EAX
             MOV numero, %CL
             CMP $9, %CL
             JA fine
             CMP $1, %CL
             JBE fine

ciclo_f:     MUL %ECX
             DEC %CL
             JNZ ciclo_f

fine:        MOV %EAX, risultato
             RET
```

1.1.3 Esercizio: test se una stringa di bit è palindroma

Scrivere un programma che si comporta come segue:

1. prende in ingresso un numero a 16 bit, contenuto in memoria nella variabile "numero".

2. controlla se "numero" è o meno una stringa di 16 bit palindroma (cioé se la sequenza di 16 bit letta da sx a dx è uguale alla sequenza letta da dx a sx).

3. Se X è (non è) palindromo, il programma inserisce 1 (0) nella variabile a 8 bit "palindromo", che si trova in memoria

```
.GLOBAL _main
.DATA
numero:      .WORD 0xF18F
palindromo:  .BYTE 1       # scommetto sul risultato positivo.

.TEXT

_main:       NOP

             MOV numero, %AX
             MOV $8, %CL
             MOV $0, %BL
ciclo:       RCL %AH       # metto i bit di AH in BL in ordine inverso
             RCR %BL       # usando il carry come appoggio
             DEC %CL
             JNZ ciclo

             CMP %AL, %BL
             JE termina
             MOVB $0, palindromo
termina:     RET
```

1.1.4 Esempio: test di primalità

Scrivere un programma che si comporta come segue:

1. prende in ingresso un numero a 16 bit, contenuto in memoria nella variabile `numero`.

2. controlla se `numero` è o meno un numero primo. Se lo è, mette in `primo` il numero 1. Altrimenti mette in `primo` il numero 0.

```
.GLOBAL _main

.DATA
numero:         .WORD 39971
primo:          .BYTE 1

.TEXT
_main:       NOP
             MOV  numero, %AX
             CMP  $2,%AX
             JBE  termina

# AX contiene il numero N su 16 bit. BX contiene il divisore. BX va
# inizializzato a 2 e portato, al piu', fino a N-1.

             MOV  $2,%BX

ciclo:       MOV  $0,%DX       #[DX,AX] contiene il numero N su 32 bit
             PUSH %AX          # salvo AX viene sporcato dalla divisione
             DIV  %BX
             POP  %AX          # si ripristina AX

             CMP  $0,%DX       # DX contiene il resto della divisione
             JE   nonprimo     # il numero ha un divisore

             INC  %BX
             CMP  %AX,%BX
             JAE  termina

# Una finezza: visto che il numero da dividere sta su 16 bit, se non
# è primo ha un divisore che sta su 8 bit (teorema di Gauss).
# Quindi, quando BH è diverso da 0, posso terminare il ciclo.

             CMP  $0, %BH
             JNE  termina
             JMP  ciclo

nonprimo:    MOVB $0,primo

termina:     RET
```

Altra versione dello stesso programma, più efficiente: si testa la divisione per due fuori dal ciclo (basta guardare il LSB di AX) , si parte testando 3 come primo divisore, e si saltano tutti i divisori pari sommando due al divisore.

```
# test di primalita' (2)

.GLOBAL _main

.DATA

numero:         .WORD 39971
primo:          .BYTE 1

.TEXT

_main:       NOP

             MOV  numero, %AX
             CMP  $2,%AX
             JBE  termina

# testo subito la divisibilità per due

             RCR  %AX
             JNC  nonprimo
             RCL  %AX

# AX contiene il numero N su 16 bit. BX contiene il divisore. BX va
# inizializzato a 3 e portato, al piu', fino a N-1.

             MOV  $3,%BX

ciclo:       MOV  $0,%DX       #[DX,AX] contiene il numero N su 32 bit
             PUSH %AX          # salvo AX (viene sporcato dalla DIV)
             DIV  %BX
             POP  %AX          # ripristino AX

             CMP  $0,%DX       # DX contiene il resto della divisione
             JE   nonprimo     # il numero ha un divisore

             ADD  $2,%BL       # sfrutto il teorema di Gauss: se il
             JC   termina      # divisore non sta su 8 bit, ho finito.
             CMP  %AX,%BX
             JAE  termina
             JMP  ciclo

nonprimo:    MOVB $0,primo

termina:     RET
```

1.1.5 Esercizio: conteggio bit a 1 in un vettore (con sottoprogramma)

Scrivere un programma che:

- definisce un vettore `numeri` di `enne` numeri naturali a 16 bit in memoria (`enne` sia una costante simbolica)
- definisce un sottoprogramma per contare il numero di bit a 1 di un numero a 16 bit. Tale sottoprogramma ha come parametro di ingresso il numero da analizzare (in AX), e restituisce il numero di bit a 1 in CL.
- utilizzando il sottoprogramma appena descritto, calcola il numero totale di bit a 1 nel vettore ed inserisce il risultato in una variabile `conteggio` di tipo word.

```
.GLOBAL _main

.DATA
.SET enne, 10
numeri:      .WORD 0,0,0,0,0,0,0,0,0,1
conteggio:   .WORD 0x00

.TEXT
_main:  NOP
             MOV $0, %ESI
             MOV $0, %CX
             MOV $0, %DX

ciclo:       MOV numeri(, %ESI, 2), %AX
             CALL conta
             INC %ESI
             ADD %CX, %DX
             CMP $enne, %ESI
             JB ciclo
             MOV %DX, conteggio
             XOR %EAX, %EAX
             RET

#--------------------------
# sottoprogramma "conta"
# conta il n. di bit a 1 in una word
# par. ingresso: AX, word da analizzare
# par. uscita: CL, conto dei bit a 1

conta:       PUSH %AX
             MOVB $0x00,%CL
comp:        CMP $0x00,%AX
             JE fine
             SHR %AX
             ADCB $0x0, %CL
             JMP comp
fine:        POP %AX
             RET
#--------------------------
```

1.1.6 Esercizio: calcolo dei coefficienti binomiali

Si scriva un programma che calcola e mette nella variabile di memoria `risultato` il coefficiente binomiale $\binom{A}{B}$, calcolato come $\frac{A!}{B!(A-B)!}$. Si assuma che A e B siano due numeri naturali minori di 10, con $A \geq B$, contenuti in memoria. Si ponga particolare attenzione nel dimensionare correttamente le variabili in memoria (a partire da `risultato`), le moltiplicazioni e le divisioni. Si faccia uso di un sottoprogramma per il calcolo del fattoriale di un numero.

```
.GLOBAL _main

.DATA
A:          .BYTE 9
B:          .BYTE 5
A_fatt:     .LONG 0
B_fatt:     .LONG 0
AB_fatt:    .LONG 0
den:        .LONG 0
risultato:  .WORD 0

.TEXT
_main:       NOP
             MOV B, %AL
             CMP %AL, A              #2. Se A<B, termina.
             JB fine_prog

#3. Calcola il coefficiente binomiale (A B), pari a A!/(B!*(A-B)!), lo
#   stampa e ritorna al punto 1.
#   (si tenga presente che 0!=1, e che 9!=362000).

             MOV B, %CL
             CALL fatt
             MOV %EAX, B_fatt
             MOV A, %CL
             CALL fatt
             MOV %EAX, A_fatt
             SUB B, %CL
             CALL fatt
             MOV %EAX, AB_fatt

             MOV B_fatt, %EDX    #   Calcola il denominatore B!*(A-B)!
             MUL %EDX
             MOV %EAX, den

             MOV $0,%EDX         #   Calcola A!/(B!*(A-B)!)
             MOV A_fatt, %EAX
             DIVL den

#   In EAX c'e' il quoziente della divisione, che e' il risultato che ci
#   interessa. Sta sicuramente su 16 bit, in quanto il max e' (9 4),
#   che fa 362880 / (24 * 120) = 126.

             MOV %AX, risultato
fine_prog:   XOR %EAX, %EAX
             RET
```

```
#-------------------------------------------------
# sottoprogramma "fatt"
# calcola in EAX il fattoriale del numero passato in CL, ammesso
# che stia su 32 bit.

fatt:       MOV $1, %EAX
            CMP $1, %CL
            JBE fine_f
            PUSH %ECX                 #possono essere spostate qui
            PUSH %EDX
            AND $0x000000FF, %ECX

ciclo_f:    MUL %ECX
            DEC %CL
            JNZ ciclo_f

            POP %EDX
            POP %ECX
fine_f:     RET
#-------------------------------------------------
```