IL DEBUG

Il DEBUG è un'utility di uso frequente dell'MS-DOS e serve per la ricerca e l' eliminazione degli errori nei programmi; Esso ci consente di visualizzare passo-passo tutte le istruzioni che vengono eseguite tramite il comando T; ci permette di scrivere dei programmi in assembler ed eseguirli; inoltre si possono leggere i registri e vedere i valori delle celle di memoria.

Per iniziare a scrivere con il DEBUG andare sul sistema operativo MS-DOS.
Al prompt dell'MS-DOS scrivere DEBUG senza estensione:

c:>DEBUG (invio)
-
(Nota: Il debug risponde con un trattino il quale indica che ora non bisogna inserire comandi del DOS ma comandi conosciuti dal DEBUG ; in altre parole indica che sta aspettando un comando dall'utente)
Se dopo il trattino inseriamo il simbolo ? cioè

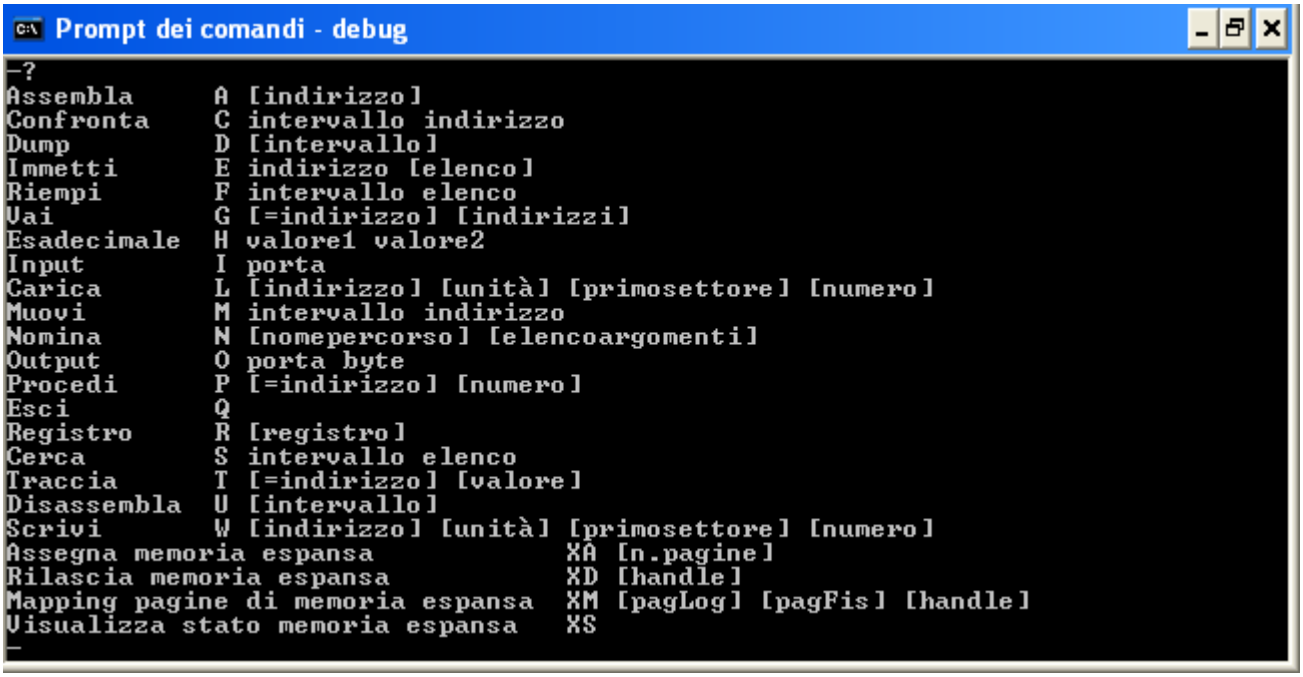
-? (invio)

il sistema ci permette di vedere tutti i comandi del DEBUG

PROMPT DEI COMANDI DEL DEBUG

Dopo essere entrati in debug inserire ? per visualizzare i comandi del debug e la loro sintassi:

-? (invio)



Comandi del debug:

Assembla A [indirizzo]
Confronta C intervallo indirizzo
Dump D [intervallo]
Immetti E indirizzo [elenco]
Riempi F intervallo elenco
Vai G [=indirizzo] [indirizzi]
Esadecimale H valore1 valore2
Input I porta
Carica L [indirizzo] [unità] [primosettore] [numero]
Muovi M intervallo indirizzo
Nomina N [nomepercorso] [elencoargomenti]
Output O porta byte
Procedi P [=indirizzo] [numero]
Esci Q
Registro R [registro]
Cerca S intervallo elenco
Traccia T [=indirizzo] [valore]
Disassembla U [intervallo]
Scrivi W [indirizzo] [unità] [primosettore] [numero]
Assegna memoria espansa XA [n.pagine]
Rilascia memoria espansa XD [handle]
Mapping pagine di memoria espansa XM [pagLog] [pagFis] [handle]
Visualizza stato memoria espansa XS
-

Visualizzazione dei registri e sua modifica.

IL COMANDO R:

Dopo essere entrati nel debug se dopo il trattino inseriamo il simbolo R cioè

-R (invio) il sistema ci permette di vedere i registri con i loro contenuti.

I registri sono piccole aree di memoria; si trovano nella CPU, ma non devono essere confusi con la memoria. Infatti i registri, come la memoria, servono a memorizzare un'informazione, ma sono anche, a differenza della memoria, piccole aree di lavoro privilegiate sulle quali è possibile effettuare operazioni di tipo specifico. Infatti tra tutte le istruzioni che ogni microprocessore è in grado di fare sul contenuto dei registri, non di tutte le istruzioni ne esiste la corrispondente sulla memoria. Da questo segue che per poter eseguire tali istruzioni è necessario prima effettuare il trasferimento dei dati dalla memoria ai registri. Da tutto questo discorso si evince che **le operazioni sui registri sono più veloci di quelle sulla memoria.** Infatti lavorando con i registri le informazioni non devono essere più indirizzate in nessun posto e non devono arrivare all'Unità Centrale attraverso il BUS DATI.
A differenza della memoria, in ogni microprocessore il numero dei registri è fisso.
Nel caso **dell'8086/8088 i registri sono 14 ed ognuno ha la lunghezza di una parola cioè di 16 bit**.

Nel DEBUG si usa solo il sistema ESADECIMALE, non c'è quindi la necessità di inserire la lettera H finale, ma se si vuole mettere è indifferente.

Il comando R del DEBUG non si limita a visualizzare i registri, ma se aggiungiamo il nome del registro, il comando indicherà al DEBUG che desideriamo visualizzare il registro e poi modificarlo.
ESEMPIO:

-R CX (invio) permette di vedere i contenuti del registro CX;
mentre
-R AX (invio)
AX 0000 ;risposta del DEBUG
: 3A7 (invio) ; se dopo i due punti scriviamo 3A7 vuol dire che vogliamo modificare AX e che al posto del valore che prima aveva cioè 0000 vogliamo inserire 3A7.
A questo punto per vedere se il DEBUG ha effettuato il cambiamento che noi volevamo, basta digita re di nuovo il comando che ci permette di vedere i registri, cioè R
-R (invio)
avremo
AX 03A7 BX=..... eccetera.

Il comando D

- D (invio) permette di leggere un blocco di memoria; compaiono 8 righe organizzate in tre colonne. Nella colonna di sinistra vi sono due numeri esadecimali separati da **:** che non è un segno di divisione; E' un modo contorto per scrivere gli indirizzi di memoria, le cui motivazioni sono ormai prevalentemente storiche ma che è di uso universale e quindi intoccabile.
L'**indirizzo si calcola così:** si prende il primo numero, lo si moltiplica per 16 decimale (cioè si aggiunge uno zero in fondo a destra: è esadecimale (shift a sinistra)) e si somma il secondo.

Esempio:

0E23:0100
significa E230+100=E330.
Questo è l'indirizzo della prima posizione di RAM considerata.
Nella colonna centrale appaiono, su ogni riga, 16 coppie di cifre esadecimali. Ciascuna coppia corrisponde a un byte e descrive l'attuale contenuto di un particolare byte in memoria. Quello più a sinistra è il contenuto del byte il cui indirizzo è dato, nel modo indicato nella colonna di sinistra; quello successivo è il contenuto del byte successivo (quindi nell'esempio fatto, di indirizzo E331), e così via fino all'ultimo, corrispondente all'indirizzo E33F.
Alla riga successiva, all'estremità sinistra, troviamo l'indirizzo E33F.
Alla riga successiva, all'estremità sinistra, troviamo l'indirizzo E340, e il discorso ricomincia.

Il comando **D ci permette quindi di esaminare il contenuto di 8*16=128 byte per volta.**
La colonna di destra contiene in ciascuna 16 caratteri alfanumerici, che costituiscono una rappresentazione figurativa dei corrispondenti byte della colonna centrale, secondo la codifica, detta ASCII, di uso pressoché universale. Se i byte in memoria rappresentano un testo scritto questa codifica ci permette di "leggere" facilmente il testo; se rappresenta un programma o dati numerici codificati in altra forma, essa non serve assolutamente a nulla.
Avendo dato soltanto il comando D il programma DEBUG ha scelto di sua iniziativa un'area di RAM disponibile per scrivere nuovi programmi, ma è possibile fargli visualizzare qualunque area della memoria compresa entro i primi 1048576 byte facendo seguire il **D** dall'indirizzo iniziale (scritto sempre nel solito modo contorto che gli piace tanto).

Provate ad esempio a dare il comando

-D FFFF:0
così facendo leggete un pezzo di ROM. Il suo contenuto è generalmente incomprensibile (è scritto in linguaggio macchina), tuttavia nella prima riga, colonna di destra **leggete una data**: essa è la data in cui è stata scritta la ROM, che serve per identificazione e che è leggibile in codice ASCII.

Diamo adesso il comando
-D 0100 (invio)
permette di leggere un blocco di memoria a partire dall'indirizzo 0100 indicato.

Il comando A

-A (invio) ;converte le istruzioni in codice macchina (A=assembla)

-A 0100 (invio) ;assembla il programma dall'indirizzo 0100

A questo punto si può iniziare a scrivere un programmma :

-A 0100 (invio)
MOV AL,10 (invio)
..... ; altre istruzioni
INT 3 ; questa istruzione indica che il programma è terminato;

L'istruzione INT 3 , è un'interruzione di tipo tre e ci permette di tornare al DOS. Alla fine di ogni programma scritto in assembler bisogna mettere tale interruzione.

Il comando U

-U (invio) ;Il comando U disassembla le istruzioni, ci permette di vedere l'ultimo indirizzo e possiamo quindi stabilire il numero di celle occupate dal programma.
Nell'esempio precedente supponiamo di avere:
110 INT 3
allora per vedere quanto è lungo il nostro programma dobbiamo fare la differenza tra l'ultima cella occupata dall'istruzione INT 3 e la prima cella del nostro programma che nel nostro esempio iniziava con 0100 cioè
(0110-0100)+1 sono le celle occupate dal nostro programma.

Per uscire dal DEBUG

-Q (invio) ; indica quit cioè comando per uscire dal DEBUG

Nota :

dati= byte sui quali facciamo le operazioni

indirizzo= numero che identifica la cella contenente un byte, che rappresentano dei dati.

IL DEBUG

STRUTTURA DI UN PROGRAMMA

Quando si scrive un programma con il DEBUG si può iniziare con il comando

-A 0100
……. istruzioni

-int 3

Il comando int 3 (cioè interruzione di tipo 3) deve essere inserito alla fine del nostro programma e consente di ritornare al sistema operativo. Per disassemblare il programma si utilizza

-U (invio) che disassembla le istruzioni e ci permette di vedere l'ultimo indirizzzo.

Per stabilire il numero di celle occupate dal nostro programma bisogna vedere l'indirizzo in cui abbiamo inserito il valore di int 3. Ad esempio se il programma è arrivato all'indirizzo

110 int 3

vuol dire che le celle occupate sono

(110-100)+1=11 ove 100 è l'indirizzo di inizio, 110 è l'indirizzo finale, +1 perché è quello attuale.

Per salvare il programma da noi scritto in un floppy-disk si usa la seguente procedura:

N a:\nome.com	**(invio)**	comando per salvare il file nome.com in assembler
R BX	**(invio)**	registro da cui si inizia a salvare il file
0	**(invio)**	comando per azzerare il contenuto di BX
R CX	**(invio)**	registro in cui si termina il salvataggio
11	**(invio)**	numero di celle occupate dal programma nell' esempio 11
w	**(invio)**	significa write e consente di salvare il programma

Per recuperare il programma sul disco:
uscire con il comando:
-Q (invio)

c:\DEBUG a:\nome.com (invio)
-u 100 (invio) ;per visualizzare il programma salvato dalla posizione 100
-r (invio) ;per vedere la prima riga del comando
-t (invio) ; per vedere passo-passo il programma

Il comando G

G=100 (invio) G sta per go ,100 sta per indirizzo da cui partire

Cioè G=100 è un comando che ci permette di eseguire le istruzioni successive a partire dall'indirizzo dato (100) e le esegue contemporaneamente.

Il comando T

Invece il comando

T=100 (invio) T è un comando per eseguire le istruzioni una per volta a partiredall'indirizzo voluto, ad esempio 100 fino a quando il programma non incontra int 3

Il comando MOV

La sintassi del comando MOV è la seguente:

MOV destinazione, sorgente

questa istruzione indica al programma di prendere il dato rappresentato da "sorgente" e di copiarlo in "destinazione"; destinazione può essere un registro (a 16 o ad 8 bit) o una locazione di memoria; sorgente può essere un registro, una locazione o un dato immediato.
Se destinazione è un registrodi segmento allora sorgente non può essere un dato immediato; il registro CS non può mai essere usato come destinazione. Non è possibile spostare il contenuto di un registro di segmento in un altro registro di segmento; se si desidera effettuare questa operazione bisogna utilizzare uno dei registri di uso generale, per esempio AX con due successive istruzioni MOV:

```
MOV AX, CS  (invio)  ;copia CS in AX
MOV DS, AX (invio)  ;trasferisci il contenuto di AX in DS
```

Proviamo a fare adesso un piccolo programma che utilizza il comando MOV.

ESEMPIO:

Programma scambia1.com

```
A 100
MOV AL, 01
MOV AH, 02
MOV BL, AL
MOV BH, AH
INT 3
```

Per salvare questo programma di nome scambia1.com in un floppy disk digitare dal programma DEBUG

N a:\scambia1.com (invio)
R BX (invio)
BX 0000 (invio)
:0 (invio)
R CX (invio)
: 10 (invio); In questo modo diciamo quanti byte vogliamo salvare del programma
W (invio); in questo modo il nostro programma è già salvato nel dischetto.

Per richiamarlo digitare:

c:>debug a:\scambia1.com

ESEMPIO:

Programma scambia2.com

Questo esempio serve a far vedere come vengono scambiat i dati da un registro all'altro per cominciare a prendere familiarità con i primi comandi. Proviamo a scambiare i dati dal registro AX al registro BX. Scriviamo:

```
c:>DEBUG (invio)
-A 100
MOV AX, 123              ;mette 0123 in AX
MOV BX, 0                ;mette 0000 in BX
MOV BL, AL               ;mette 23 che si trova in AL e lo mette in BL
MOV BH, AH               ;mette 01 che si trova in AH e lo mette in BH
MOV AX, 0                ;mette 0000 in AX
MOV BX, AX               ; adeso in AX c'è zero quindi mette 0000 in BX
INT 3
```

Per salvare questo programma di nome scambia2.com in un floppy disk digitare dal programma DEBUG

```
N a:\scambia2.com (invio)
R BX (invio)
BX 0000 (invio)
:0 (invio)
R CX (invio)
: 10 (invio);  In questo modo diciamo quanti byte vogliamo salvare del programma
W (invio); in questo modo il nostro programma è già salvato nel dischetto.
```

Per richiamarlo digitare:

```
c:>debug a:\scambia2.com
```

Modi di indirizzamento dell'8086

Il software dell'8086 prevede diversi modi per indirizzare gli operandi;
L' **indirizzamento diretto** con un offset a 16 bit, oppure l' **indirizzamento indiretto** con base (tramite i registri BX o BP); l'**indirizzamento con indice** (tramite SI o DI) più uno spiazzamento costante opzionale a 8 0 16 bit. questo spiazzamento costante può essere il nome di una variabile o direttamente un numero.

Esempio di indirizzamento:

Scriviamo adesso il semplice programma che chiameremo **a:\indir.com**

PROGRAMMA A:\INDIR.COM

```
A 500              ;partiamo dall'indirizzo 500 per inserire i dati
DB 11, 22, 0       ; DB= define byte cioè definiamo i byte per inserire i byte in indirizzi successivi
                    ;ad esempio nell'indirizzo 500  mettiamo  11      (F9)
                   ;              "          501     "        22      (D4)
                   ;              "          502     "        0       (02)
                   ;cioè F9+D4=1CD ove  CD va in AL mentre 1 è il carry CY
A 100              ; a partire dall'indirizzo 100 inseriamo le istruzioni
MOV AL, [500]      ; indirizzamento diretto: il contenuto dell'indirizzo 500 nel nostro caso contiene il
;valore 11 il quale viene copiato nel registro AL
ADD AL, [501]      ;aggiunge il valore 22 che si trova nell'indirizo 501 e lo copia in AL cioè
                   ;AL <= 11+22=33, cioè in AL si avrà il valore 33
INT 3              fine delle istruzioni del programma.
```

Adesso proviamo a salvarlo nel dischetto:

senza uscire dal debug dopo INT 3 (invio) digitiamo:

```
N a:\nome. com   ; nel nostro esempio possiamo chiamarlo a:\indir.com
R BX
BX 0000
: 0                ;cioè cancelliamo il valore di BX
R CX
CX 0000
: B              ;numero di celle occupate fino ad INT 3 cioè (A+1)
W                ; la sigla W sta per Write
```

In questo modo il debug ci dice quanti byte vengono salvati nel nostro dischetto.

Per richiamare il file salvato sul dischetto dal prompt di c:> digitare:

C:>debug a:\indir.com
-g 100 ; questo comando ci permette di eseguire le istruzioni a partire dall'indirizzo
; dato cioè dall'indirizzo 100 e le esegue contemporaneamente
; il debug a questo comando risponde così:

```
AX=0000   BX=00000 CX=010B  DX= 0000.....
.....NC.....
                                    MOV AL,[0500]          DS:05000=F9
```

-t ; per seguire passo passo il programma digitamo t ed il debug
;ci risponde così

```
AX=00F9  BX=0000 CX=010B DX=0000.....
.....NC.....
                                      ADD AL,[0501]   DS:0501=D4
```

-t

```
AX=00CD  BX=0000 CX=010B   DX=0000.....
.....CY.....
                                      MOV [0502],AL    DS:0502=CD
```

-t

```
AX=00CD   BX=0000  CX=010B  DX=0000.....
.....CY.....
                                                INT 3
```

-Q ;per uscire dal debug

ISTRUZIONI ARITMETICHE

1) L'istruzione somma.

Per eseguire la somma con il DEBUG si usa il simbolo ADD=istruzione somma

ADD destinazione, sorgente

Questa istruzione fa in modo che venga eseguita la:
somma tra sorgente a destinazione e mette il risultato in destinazione.

destinazione <---- destinazione + sorgente

Esempio:

ADD AL, 10H

questa espressione fa in modo che il dato 10H venga sommato al contenuto del registro AL ed infine riporta il risultato finale della somma nel registro AL.
Destinatario e sorgente devono avere la stessa grandezza, cioè lo stesso numero di bit.

La somma si può eseguire anche fra due registri della stessa capacità.

2) L'istruzione differenza.

Per eseguire la differenza si utilizza il comando:
SUB= istruzione della differenza

SUB destinazione, sorgente

questa istruzione fa in modo che venga eseguita la sottrazione tra il valore contenuto nel registro destinazione e il valore contenuto nel registro sorgente e mette il risultato della differenza nel registro destinazione:

Destinazione <--- Destinazione - Sorgente

Programma somma

Supponiamo adesso di volere eseguire la somma con il DEBUG della seguente formula matematica:

[8 - (4+1)]

```
A 100 (invio)
MOV Al, 8 (invio)              ;mette 8 in AL
MOV BL, 4 (invio)               ;mette 4 in BL
MOV BH, 1 (invio)               ;mette 1 in BH
ADD BH, BL (invio)              ;somma BL con BH e mette il risultato in BH
SUB AL, BH (invio)              ; sottrai AL con BH e metti il risultato in AL
INT 3 (invio)                   ;interruzione software di tipo 3
```

Per salvare il programma con il nome somma.com scriviamo:

```
-N a:\nome. com   ; nel nostro esempio possiamo chiamarlo a:\somma.com
R BX
BX 0000
: 0                ;cioè cancelliamo il valore di BX
R CX
CX 0000
: B              ;numero di celle occupate fino ad INT 3 cioè (A+1)
W                ; la sigla W sta per Write
```

In questo modo il debug ci dice quanti byte vengono salvati nel nostro dischetto.

Per recuperare il programma salvato sul disco:
uscire con il comando:
-Q (invio)

```
c:\DEBUG a:\nome.com      (invio)
-u 100 (invio)    ;per visualizzare il programma salvato dalla posizione 100
-r (invio)        ;per vedere la prima riga del comando
-t (invio)        ; per vedere passo-passo il programma
```

Istruzione INC

ESEMPIO:

File A:\INCRE.com

```
A  100
MOV AL,26          ;mette 26 in AL
INC AL             ;incrementa AL, adesso in AL viene posto 26+1=27
ADD AL,76          ;aggiunge 76 a 27 e lo mette in AL
MOV AH, AL         ;sposta il contenuto di AL e lo mette in AH
ADD AL, AH         ;aggiune AL+AH e il risultato va in AL
INT 3
```

……
……

ESEMPIO:

File a:\incre2.com

```
A 100
MOV AH,0           ;azzera AH
MOV AL, AH         ;azzera AL
INC AL             ;incrementa AL; mette 1 in AL
MOV AX,0
MOV DS, AX
MOV BX,9
MOV AX, [BX]
INC AX
INT 3
```

……
……

ESEMPIO:

File a:\indir.com

```
A 500
DB 11,22,0
A 100
MOV AL, [500]
ADD AL, [501]
MOV [502], AL
INT 3
```

UTILIZZIAMO GLI INDIRIZZAMENTI CON REGISTRI:

ESEMPIO:

FILE A:\REG.COM

```
A 100
MOV AX, 1
MOV DS, AX
MOV BX, 9
MOV AX, [BX]          ;indirizzamento indiretto a registro (muove il contenuto del registro BX
MOV AL, BL            ;e lo mette nel registro AX
ADD AX, 2
SUB AX, BX
MOV BX, AX
INT 3
```

……
……

UTILIZZIAMO GLI INDIRIZZAMENTI CON INDICE:

Esercizio:

Inserire i byte 1, 2,3,4,5,6, all'indirizzo 400 , moltiplicarli per 2 e trasferire il prodotto all'indirizzo 500.
Visualizzare i risultati con i comandi d 400 e d 500

Svolgimento:

```
a 400
db 1,2,3,4,5,6
a 500
db 0,0,0,0,0,0

-a 100
0D00:0100 BE0004      MOV    SI,0400
0D00:0103 BF0005      MOV    DI,0500
0D00:0106 B302        MOV    BL,02
0D00:0108 B90600      MOV    CX,0006
0D00:010B 8A04        MOV    AL,[SI]
0D00:010D F6E3        MUL    BL
0D00:010F 8905        MOV    [DI],AX
0D00:0111 46          INC    SI
0D00:0112 47          INC    DI
0D00:0113 49          DEC    CX
0D00:0114 75F5        JNZ    010B
0D00:0116 CC          INT    3
-r
-t
t……
per visualizzare il risultato utilizzare il comando dump cioè  D    indirizzo
-d 400
0D00:0400  01 02 03 04 05 06 20 67-69 85 20 65 73 69 73 74   ..... gi. esist
0D00:0410  65 6E 74 65 0D 0A 09 25-31 20 62 79 74 65 0D 0A   ente...%1 byte..
0D00:0420  1B 54 6F 74 61 6C 65 20-64 65 69 20 66 69 6C 65   .Totale dei file
0D00:0430  20 65 6C 65 6E 63 61 74-69 3A 0D 0A 38 28 53 69    elencati:..8(Si
0D00:0440  20 8A 20 76 65 72 69 66-69 63 61 74 6F 20 75 6E    . verificato un
0D00:0450  20 65 72 72 6F 72 65 20-6E 65 6C 6C 61 20 76 61    errore nella va
0D00:0460  72 69 61 62 69 6C 65 20-64 27 61 6D 62 69 65 6E   riabile d'ambien
0D00:0470  74 65 29 0D 0A 0D 28 63-6F 6E 74 69 6E 75 61 20   te)...(continua
-d 500
0D00:0500  02 04 06 08 0A 0C 00 6F-6D 65 20 64 69 20 66 69   .......ome di fi
0D00:0510  6C 65 20 6E 6F 6E 20 76-61 6C 69 64 6F 0D 0A 3F   le non valido..?
0D00:0520  49 6D 70 6F 73 73 69 62-69 6C 65 20 61 70 72 69   Impossibile apri
0D00:0530  72 65 20 69 6C 20 66 69-6C 65 20 64 69 20 69 6E   re il file di in
```

```
0D00:0540  66 6F 72 6D 61 7A 69 6F-6E 69 20 64 65 6C 20 50   formazioni del P
0D00:0550  61 65 73 65 20 69 6E 64-69 63 61 74 6F 0D 0A 00   aese indicato...
0D00:0560  95 41 74 74 69 76 61 20-6F 20 64 69 73 61 74 74   .Attiva o disatt
0D00:0570  69 76 61 20 75 6E 20 63-6F 6E 74 72 6F 6C 6C 6F   iva un controllo
-
```

ESERCIZIO DI TRASFERIMENTO DI BYTE:

Dati i seguenti byte
0,1,2,3,4,5,6,7,8,9 all'indirizzo 400h
2,3,4,5,6,7,8,9,a,b all'indirizzo 410h
eseguire la somma e trasferirli alii'indirizzo 420h

Svolgimento:

```
-a 400
0D01:0400 db 0,1,2,3,4,5,6,7,8,9
0D01:040A
-a 410
0D01:0410 db 2,3,4,5,6,7,8,9,a,b
0D01:041A
-a 420
0D01:0420 db 0,0,0,0,0,0,0,0,0,0

-A 100
0D01:0100 BE0004       MOV     SI,0400
0D01:0103 BD1004       MOV     BP,0410
0D01:0106 BF2004       MOV     DI,0420
0D01:0109 B90A00       MOV     CX,000A
0D01:010C 8A04         MOV     AL,[SI]
0D01:010E 8A6600       MOV     AH,[BP+00]
0D01:0111 00E0         ADD     AL,AH
0D01:0113 8805         MOV     [DI],AL
0D01:0115 46           INC     SI
0D01:0116 45           INC     BP
0D01:0117 47           INC     DI
0D01:0118 49           DEC     CX
0D01:0119 75F1         JNZ     010C
0D01:011B CC           INT     3
```

alla fine dell'esecuzione del programma con il commando d 400 possiamo visualizzare il trasferimento e la somma dei byte all'indirizzo 420h.

```
- d 400
0D01:0400  00 01 02 03 04 05 06 07-08 09 62 79 74 65 0D 0A   ..........byte..
0D01:0410  02 03 04 05 06 07 08 09-0A 0B 69 20 66 69 6C 65   ..........i file
0D01:0420  02 04 06 08 0A 0C 0E 10-12 14 0D 0A 38 28 53 69   ............8(Si
0D01:0430  20 8A 20 76 65 72 69 66-69 63 61 74 6F 20 75 6E    . verificato un
0D01:0440  20 65 72 72 6F 72 65 20-6E 65 6C 6C 61 20 76 61    errore nella va
0D01:0450  72 69 61 62 69 6C 65 20-64 27 61 6D 62 69 65 6E   riabile d'ambien
0D01:0460  74 65 29 0D 0A 0D 28 63-6F 6E 74 69 6E 75 61 20   te)...(continua
0D01:0470  25 31 29 0E 52 65 76 69-73 69 6F 6E 65 20 25 31   %1).Revisione %
```