MINI CORSO DI *ASSEMBLER 8086*

(tratto dal sito:http://www.giobe2000.it/Tutorial/Indice/Home.asp)
(versione dell' 8 gennaio 2010 - rev: 03/07/2012)

IL DEBUG DEL DOS

Nel linguaggio comune un **bug** è una cosa fastidiosa, un pidocchio, una cimice, una calamità. Gli anglosassoni sono molto immediati nel definire i concetti e spesso associano ad eventi tecnici un'immagine forte e molto colorata; così un errore di programmazione diventa un bug e lo strumento che consente di localizzare ed eliminare i bug è diventato **Debugger**, lo "spulciatore".

Se l'assemblatore ha ritenuto corretta la sintassi del sorgente e il linker ha generato senza errore l'eseguibile non è detto che il nostro programma "*giri*" come ci aspettiamo. Al contrario, sono spesso gli errori di concetto (semantici) che si manifestano di solito con devastante arroganza, bloccando tutto il computer e costringendoci a resettarlo (premere il pulsante di reset o chiudendo l'applicazione con Ctrl-Alt-Canc). Sono i cosiddetti errori di run-time.
In questi casi il modo migliore per capire dove abbiamo sbagliato è ricompilare il sorgente inserendo dei **break-point**, cioè dei punti in cui il programma è costretto a fermarsi.

I break-point di solito si fissano inserendo nel punto desiderato l'istruzione **`INT 03H`** (ricorda bene questo metodo!!); la loro presenza è del tutto irrilevante nell'esecuzione normale di un programma, ma se l'eseguibile è fatto girare sotto **DEBUG** esso si fermerà proprio nel punto previsto, consentendo l'esecuzione passo-passo delle successive istruzioni, consentendo la localizzazione di quella che ha creato l'errore run-time.

Dunque nella fase di collaudo e messa a punto (tuning) di un programma il Debug è uno strumento di lavoro particolarmente utile. Il debug(ger) si occupa del caricamento in memoria dell'eseguibile, in modo paragonabile a quello del **loader** del sistema operativo e, dopo aver preparato i puntatori (**CS**:**IP**) sulla prima istruzione e (SS:SP) sulla prima locazione dello stack cede il controllo a noi, manifestando la sua disponibilità con un trattino posto sulla riga successiva all'intestazione.

Con il Debugger è possibile aprire ed eseguire qualunque programma eseguibile. E' in grado di mostrarne il codice dissassemblato (molto vicino a quello sorgente originale) ed è facile scorrerne il contenuto, per cercare di capire o scoprire i trucchi del suo autore. (è quello che fanno i "crackatori" di professione).

Comando ? - Help

Dopo essere entrati in Debug il comando **?** mostra l'elenco dei comandi disponibili (tutti i comandi si danno digitandone la lettera corrispondente dopo il prompt del debug, cioè dopo il trattino).
Il Debug.exe in dotazione con il Dos (generalmente in c:\windows\system32) mostra questo help:

RICHIAMO HELP DEL DEBUG

```
C:\WINDOWS\Desktop>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
Nome         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]
Alloca memoria espansa              XA [n. pagine]
Rilascia memoria espansa            XD [handle]
Mapping pagine di memoria espansa   XM [pagLog] [pagFis] [handle]
Visualizza stato memoria espansa    XS
```

Il programma Debug è particolarmente utile quando apre e gestisce un determinato file. Per far questo basta digitare nella finestra DOS la scritta **debug <nomeeseguibile>**. Vediamo un esempio:

CARICAMENTO DI UN FILE DIRETTAMENTE COME ARGOMENTO DEL COMANDO DEBUG

```
C:\trans>debug prova.com
-d
168C:0100 E9 23 01 00 DA C4 C4 C4-C4 C4 C4 C4 C4 C4 C4 C4 .#..............
168C:0110 C4 C4 C4 C4 C4 C4 C4 C4-C4 C4 C4 C4 C4 C4 C4 C4 ................
168C:0120 C4 C4 C4 C4 C4 C4 C4 C4-C4 C4 C4 C4 C4 C4 C4 C4 ................
168C:0130 C4 C4 C4 C4 C4 C4 C4 C4-C4 C4 C4 C4 BF 00 7C 20 ..............|
168C:0140 20 20 42 65 6E 76 65 6E-75 74 6F 20 74 72 61 20   Benvenuto tra
168C:0150 69 20 70 72 6F 67 72 61-6D 6D 61 74 6F 72 69 20 i programmatori
168C:0160 64 65 6C 20 43 6C 75 62-20 7E 41 72 63 68 2D 4C del Club ~Arch-L
168C:0170 61 62 7E 20 20 20 7C 00-7C 20 20 20 C4 C4 C4 C4 ab~   |.|    ....
-q

C:\trans>
```

Comando N - Name - sintassi: N <PercorsoFile>

Va sottolineato che è possibile caricare un file anche senza specificarne il nome subito dopo il comando **debug**. Lo si può fare digitando 2 comandi (**L** e **N** <nomefile>) sul prompt del debug stesso:

CARICAMENTO DI UN FILE MEDIANTE COMANDI **N** <NomeFile> ED **L**

```
C:\trans>debug
-n prova.com
-l
-d
168C:0100 E9 23 01 00 DA C4 C4 C4-C4 C4 C4 C4 C4 C4 C4 C4 .#..............
168C:0110 C4 C4 C4 C4 C4 C4 C4 C4-C4 C4 C4 C4 C4 C4 C4 C4 ................
168C:0120 C4 C4 C4 C4 C4 C4 C4 C4-C4 C4 C4 C4 C4 C4 C4 C4 ................
168C:0130 C4 C4 C4 C4 C4 C4 C4 C4-C4 C4 C4 C4 BF 00 7C 20 ..............|
168C:0140 20 20 42 65 6E 76 65 6E-75 74 6F 20 74 72 61 20   Benvenuto tra
168C:0150 69 20 70 72 6F 67 72 61-6D 6D 61 74 6F 72 69 20 i programmatori
168C:0160 64 65 6C 20 43 6C 75 62-20 7E 41 72 63 68 2D 4C del Club ~Arch-L
168C:0170 61 62 7E 20 20 20 7C 00-7C 20 20 20 C4 C4 C4 C4 ab~   |.|    ....
-q
```

```
C:\trans>
```

Dopo aver digitato **N nomefile.ext** ed aver confermato con Invio non sembra succedere niente; in realtà il gestore viene avvisato che dovrà cercare un programma con nome **nomefile** ed estensione **ext**.
Tramite il comando **L** (load), confermato da Invio, il debug è costretto a cercare il file e a caricarlo in memoria; solo se non lo trova segnalerà errore (File not found o Impossibile trovare il file).

ERRORE DI CARICAMENTO RELATIVO AD UN FILE NON TROVATO

```
C:\trans>debug
-n pippo.txt
-l
File not found
```

Il DEBUG è uno strumento dedicato all'analisi del processore (i suoi registri) e della memoria e quindi ogni numero da esso presentato va inteso nella lingua del processore cioè in esadecimale e quindi è importante ricordare fin d'ora che i numeri esadecimali si possono trattare senza la "h" con DEBUG.

Comando Q - Quit - sintassi: Q

Il comando per uscire è **Q** (quit). Ogni operazione svolta fino a quel momento viene persa irrimediabilmente (dopotutto siamo in ram) e il controllo viene ceduto di nuovo al sistema operativo.

Comando D - Dump - sintassi: D [[Segm:]OffSet]

Cominciamo con il comando **D** (dump)**.** Scriviamo quindi debug e poi il comando d

ANALISI MEMORIA RAM

```
C:\trans>debug
-d
166C:0100 4D 00 00 0E 00 FE BF 81-00 8B 36 92 DE 8B 44 FE M.........6...D.
166C:0110 54 4D 50 3D 43 3A 5C 57-49 4E 44 4F 34 00 5B 16 TMP=C:\WINDO4.[.
166C:0120 45 4D 50 00 54 45 4D 50-3D 43 3A 5C 57 49 4E 44 EMP.TEMP=C:\WIND
166C:0130 4F 57 53 5C 54 45 4D 50-00 50 52 4F 4D 50 54 3D OWS\TEMP.PROMPT=
166C:0140 24 70 24 67 00 77 69 6E-62 6F 6F 74 64 69 72 3D $p$g.winbootdir=
166C:0150 43 3A 5C 57 49 4E 44 4F-57 53 00 50 41 54 48 3D C:\WINDOWS.PATH=
166C:0160 43 3A 5C 57 49 4E 44 4F-57 53 3B 43 3A 5C 57 49 C:\WINDOWS;C:\WI
166C:0170 4E 44 4F 57 53 5C 43 4F-4D 4D 41 4E 44 00 43 4F NDOWS\COMMAND.CO

C:\trans>
```

Analizziamo la videata ottenuta: sono 8 righe così strutturate:
- la parte di sinistra (in verde) mostra l'indirizzo segmentato della prima locazione di memoria che andremo ad ispezionare: il puntatore è in esadecimale.
- la parte centrale (in bianco) elenca il contenuto di 16 locazioni (byte) consecutive, a partire da quella indirizzata nella parte a sinistra espresso in esadecimale.
- la parte di destra (in giallo) cerca di tradurre in forma Ascii il contenuto delle medesime locazioni.

Nella **zona verde** abbiamo:
- l'indirizzo di segmento che è, per tutte le righe visualizzate, sempre lo stesso (nell'esempio 166C): il suo valore dipende esclusivamente da quello della prima zona di ram libera al momento del caricamento del debugger in memoria; questo

ultimo riserva ai nostri esperimenti un intero segmento che comincia proprio da 0166C:0000.
- l'indirizzo di offset iniziale è 0100: il suo valore risente del fatto che Debug ha riservato per se stesso le prime 256 locazioni, per ospitare, nella logica dei programmi gestiti dal Dos, il proprio PSP [acronino di prefisso di segmento di programma]).
La **zona bianca** contiene la traduzione esadecimale di quello che debug ha effettivamente trovato: il loro valore è assolutamente imprevedibile e generalmente può essere cambiato in ogni momento, senza provocare alcun danno, come vedremo nelle prossime pagine.
La **zona gialla** contiene i caratteri Ascii corrispondenti ai valori esadecimali visualizzati nella parte centrale. Tutti i bytes che corrispondono a caratteri Ascii stampabili sono mostrati in chiaro per quello che sono, mentre tutti gli altri (da 00H a 1FH: caratteri di controllo - da 80H a FFH: caratteri ascii estesi) sono visualizzati con un punto.

Se il comando **D** viene dato di nuovo, viene mostrata a video la successiva "cestinata" di 128 bytes, e così ad oltranza.

Specificando l'indirizzo desiderato ci viene offerta la possibilità di verificare concetti ed idee maturate in altri ambiti di studio; per esempio possiamo verificare se la ram del video (modo testo) è effettivamente costituita da bytes alternati di carattere (ascii) e di colore (attributo).
Per verificarlo occorre pulire il video con il comando dos **CLS** e successivamente aprire il programma debug. Digitare il comando **D** seguito dall'indirizzo dell'area dedicata alla modalità testo ovvero B800:0000:

ANALISI MEMORIA VIDEO

```
C:\trans>debug
-d B800:0000
B800:0000 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:0010 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:0020 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:0030 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:0040 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:0050 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:0060 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:0070 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
-d
B800:0080 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:0090 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:00A0 43 07 3A 07 5C 07 74 07-72 07 61 07 6E 07 73 07 C.:.\.t.r.a.n.s.
B800:00B0 3E 07 64 07 65 07 62 07-75 07 67 07 20 07 20 07 >.d.e.b.u.g. .
B800:00C0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:00D0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:00E0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
B800:00F0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . .
-
```

La prima sequenza di 8 righe ci dice che le prime 128 locazioni contengono 64 coppie consecutive di bytes 20h/07h. I primi 160 bytes (in figura su sfondo grigio) sono relativi alla prima riga del monitor che come si nota è vuota [in realtà sono spazi (=20h) con colore bianco su sfondo nero (=07h)]. Le prime coppie diverse da 20h/07h sono visibili a partire dal 161° bytes (indirizzo B800:00A0) relativi alla 2° riga. La sequenza esadecimale 43 07 3A 07 5C 07 74 07-72 07 61 07 6E 07 73

07 3E 07 64 07 65 07 62 07-75 07 67 07 corrisponde alla stringa a video **C:\trans>debug**.

Un altro esempio di analisi della memoria è quello sottostante che mostra come visualizzare la data del BIOS.

LETTURA DELLA DATA DEL BIOS

```
-d FFFF:5 L8
FFFF:0000 30 32 2F-32 32 2F 30 36                         02/22/06

B800:00F0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07  . . . . . . . .
-
```

Comando F - Fill - sintassi: F <IndirizzoIniziale> <IndirizzoFinale o lunghezza> <sequenza hex o ascii>

Il comando **F** (FILL, riempi) nella prima sintassi si aspetta l'intervallo di indirizzi (espresso come Segmento:Offset) iniziale e finale (nell'esempio sottostante 100h e 17Fh) e la sequenza di valori da inserire. Bisogna stare attenti a non esagerare: se si cancellano le locazioni sotto l'indirizzo 005C oppure sopra E000 (valore indicativo) il debug potrebbe "piantarsi".

RESET ZONA DI MEMORIA MEDIANTE RIEMPIMENTO DI CARATTERI '\0'

```
C:\trans>debug
-d
166C:0100 BA 42 86 E9 65 FE BF 81-00 8B 36 92 DE 8B 44 FE .B..e.....6...D.
166C:0110 BE C6 DB 8B 74 09 03 C6-50 E8 0D FA 34 00 5B 16 ....t...P...4.[.
166C:0120 03 F1 2B C6 8B C8 E8 7B-F4 83 F9 7F 72 0B B9 7E ..+....{....r..~
166C:0130 00 F3 A4 B0 0D AA 47 EB-08 AC AA 3C 0D 74 02 EB ......G....<.t..
166C:0140 F8 8B CF 81 E9 82 00 26-88 0E 80 00 C3 8B 1E 92 .......&........
166C:0150 DE BE 1A D4 BA FF FF B8-00 AE CD 2F 3C 00 C3 A0 .........../<...
166C:0160 DB E2 0A C0 74 09 56 57-E8 2A 21 5F 5E 73 0A B9 ....t.VW.*!_^s..
166C:0170 04 01 FC 56 57 F3 A4 5F-5E C3 50 56 33 C9 33 DB ...VW._^.PV3.3.
-f 100 17F 00
-d 100
166C:0100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-
```

Il comando **F** è particolarmente versatile: il suo terzo parametro può essere di diverso tipo. Infatti la memoria può essere riempita con:
- singoli bytes,
- sequenze di bytes,
- caratteri Ascii (delimitati da '),
- stringhe Ascii di qualunque lunghezza (delimitate da ').

ESEMPI USO DEL COMANDO FILL

```
-f 100 10F 41,42,43,44,45,46
-d 100
166C:0100 41 42 43 44 45 46 41 42-43 44 45 46 41 42 43 44 ABCDEFABCDEFABCD
166C:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-f 110 11F '*'
-d 100
166C:0100 41 42 43 44 45 46 41 42-43 44 45 46 41 42 43 44 ABCDEFABCDEFABCD
```

```
166C:0110 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A **********
166C:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-f 120 12F 'Ciao'
-d 100
166C:0100 41 42 43 44 45 46 41 42-43 44 45 46 41 42 43 44 ABCDEFABCDEFABCD
166C:0110 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A **********
166C:0120 43 69 61 6F 43 69 61 6F-43 69 61 6F 43 69 61 6F CiaoCiaoCiaoCiao
166C:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-
```

Come in altri comandi del debug, il range degli indirizzi da coinvolgere può essere passato anche sotto forma di indirizzo iniziale e lunghezza (espressa con una **L** seguita dal valore esadecimale corrispondente al numero di locazioni da riempire *[quindi occorre scrivere lA se si desiderano sovrascrivere 10 byte]*). Vediamone un esempio:

ALTRO ESEMPIO USO DEL COMANDO FILL

```
-f 130 lB 'Marco Sechi'
-d 100
166C:0100 41 42 43 44 45 46 41 42-43 44 45 46 41 42 43 44 ABCDEFABCDEFABCD
166C:0110 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A **********
166C:0120 43 69 61 6F 43 69 61 6F-43 69 61 6F 43 69 61 6F CiaoCiaoCiaoCiao
166C:0130 4D 61 72 63 6F 20 53 65-63 68 69 00 00 00 00 00 Marco Sechi.....
166C:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Comando C - Compare - sintassi: C <IndirizzoIniziale Area1> <IndirizzoFinale o lunghezza Area1> <IndirizzoIniziale Area2>

Se i valori presenti in memoria sono particolarmente significativi può essere necessario confrontarli tra loro, sebbene questo evento sia poco probabile.

Il comando **C** (COMPARE, confronta) si aspetta 3 indirizzi (espressi come segmento:offset): quello iniziale e finale relativo alla prima area e quello iniziale della seconda area. Se le 2 aree di memoria contengono gli stessi bytes non viene mostrato alcun valore;

CONFRONTO TRA DUE ZONE DI MEMORIA (zona1: 0100 a 010A - zona2: 0200 a 020A)

```
-c 100 10A 110
166C:0100 41 2A 166C:0110
166C:0101 42 2A 166C:0111
166C:0102 43 2A 166C:0112
166C:0103 44 2A 166C:0113
166C:0104 45 2A 166C:0114
166C:0105 46 2A 166C:0115
166C:0106 41 2A 166C:0116
166C:0107 42 2A 166C:0117
166C:0108 43 2A 166C:0118
166C:0109 44 2A 166C:0119
166C:010A 45 2A 166C:011A
-c 150 15A 160
-
-d 100
166C:0100 41 42 43 44 45 46 41 42-43 44 45 46 41 42 43 44 ABCDEFABCDEFABCD
166C:0110 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A **********
166C:0120 43 69 61 6F 43 69 61 6F-43 69 61 6F 43 69 61 6F CiaoCiaoCiaoCiao
166C:0130 4D 61 72 63 6F 20 53 65-63 68 69 00 00 00 00 00 Marco Sechi.....
166C:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

```
166C:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Naturalmente è possibile passare anche degli indirizzi completi (cioè con il valore del segmento). Se l'indirizzo specificato contiene solo l'offset i dati visualizzati si ritengono appartenenti al segmento puntato da DS (data segment);

CONFRONTO TRA DUE ZONE DI MEMORIA CON DIFFERENTI SEGMENTI

```
-c B800:0100 10A 200
B800:0100 31 4E 166C:0200
B800:0101 07 32 166C:0201
B800:0102 20 C0 166C:0202
B800:0103 07 86 166C:0203
B800:0104 30 04 166C:0204
B800:0105 07 46 166C:0205
B800:0106 30 3C 166C:0206
B800:0107 07 0D 166C:0207
B800:0108 20 75 166C:0208
B800:0109 07 02 166C:0209
B800:010A 34 88 166C:020A
```

La sintassi del comando **C** prevede come al solito 2 modi, sostanzialmente uguali. Il comando può essere dato passando l'indirizzo iniziale della prima area e la lunghezza (espressa con una **L** seguita dal valore esadecimale corrispondente al numero di locazioni da confrontare) seguito dall'indirizzo iniziale della seconda area:

CONFRONTO TRA DUE ZONE DI MEMORIA UTILIZZANDO LA 2° SINTASSI

```
-c 100 L10 200
166C:0100 41 4E 166C:0200
166C:0101 42 32 166C:0201
166C:0102 43 C0 166C:0202
166C:0103 44 86 166C:0203
166C:0104 45 04 166C:0204
166C:0106 41 3C 166C:0206
166C:0107 42 0D 166C:0207
166C:0108 43 75 166C:0208
166C:0109 44 02 166C:0209
166C:010A 45 88 166C:020A
166C:010B 46 04 166C:020B
166C:010C 41 89 166C:020C
166C:010D 42 36 166C:020D
166C:010E 43 E3 166C:020E
166C:010F 44 D7 166C:020F
```

Comando M - COPIA - sintassi: M <IndirizzoIniziale Sorgente> <IndirizzoFinale o lunghezza Sorgente> <IndirizzoIniziale Destinazione>

Il comando **M** (MOVE, muovi) si aspetta 3 indirizzi (espressi come segmento:offset): quello iniziale e finale dell'area in cui sono contenuti i bytes da copiare e quello iniziale dell'area in cui si desidera copiarli.

Quindi più che di uno spostamento si tratta di una copia, dato che i bytes coinvolti saranno ancora presenti nelle locazioni sorgente. Vediamo un esempio

1 * ESEMPIO DI COPIA

```
-d 100
166C:0100 41 42 43 44 45 46 41 42-43 44 45 46 41 42 43 44 ABCDEFABCDEFABCD
166C:0110 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A ****************
166C:0120 43 69 61 6F 43 69 61 6F-43 69 61 6F 43 69 61 6F CiaoCiaoCiaoCiao
166C:0130 4D 61 72 63 6F 20 53 65-63 68 69 00 00 00 00 00 Marco Sechi.....
166C:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-m 110 11F 160
-d 100
166C:0100 41 42 43 44 45 46 41 42-43 44 45 46 41 42 43 44 ABCDEFABCDEFABCD
166C:0110 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A ****************
166C:0120 43 69 61 6F 43 69 61 6F-43 69 61 6F 43 69 61 6F CiaoCiaoCiaoCiao
166C:0130 4D 61 72 63 6F 20 53 65-63 68 69 00 00 00 00 00 Marco Sechi.....
```

```
166C:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0160 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A **********
166C:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Il comando può essere anche dato passando l'indirizzo iniziale della prima area e la lunghezza (espressa con una **L** seguita dal numero esadecimale delle locazioni da copiare - ad esempio **L10** per copiare 16 byte) seguito dall'indirizzo iniziale della seconda area:

2 * ESEMPIO DI COPIA

```
-m 100 L10 170
-d 100
166C:0100 41 42 43 44 45 46 41 42-43 44 45 46 41 42 43 44 ABCDEFABCDEFABCD
166C:0110 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A **********
166C:0120 43 69 61 6F 43 69 61 6F-43 69 61 6F 43 69 61 6F CiaoCiaoCiaoCiao
166C:0130 4D 61 72 63 6F 20 53 65-63 68 69 00 00 00 00 00 Marco Sechi.....
166C:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:0160 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A **********
166C:0170 41 42 43 44 45 46 41 42-43 44 45 46 41 42 43 44 ABCDEFABCDEFABCD
```

Comando S - SEARCH - sintassi: S <IndirizzoIniziale Area1> <IndirizzoFinale o lunghezza Area1> <IndirizzoIniziale Area2>

Il comando **S** (SEARCH, cerca) si dimostra spesso utile, non tanto per cercare stringhe, per le quali è perfettamente abilitato, ma soprattutto per cercare determinate sequenze di bytes; l'esperienza dimostra che con esso è possibile trovare tutti i punti di un programma in cui viene, per esempio, chiamata una certa procedura di sistema.
Come è noto ogni istruzione è codificata con uno o più bytes: di solito un codice operativo ed un operando; a forza di ficcare il naso nei programmi eseguibili con il buon vecchio debug non ci vuole molto tempo per imparare i principali accoppiamenti tra istruzioni e relativi bytes.
Così, per esempio, può essere utile sapere in quali punti di un dato programma abbiamo l'istruzione **INT 21H** (servizi DOS) corrispondente alla coppia di byte: **CD 21**. Basta quindi utilizzare questo comando seguito dall'indirizzo iniziale e finale (espressi come segmento:offset) e dalla sequenza di bytes da cercare.
L'immagine seguente mostra il risultato della ricerca in una zona di memoria in cui è stato caricato il programma **putciao.com** che usa questa istruzione 4 volte:

ESEMPIO DI RICERCA CON IL COMANDO S

```
-C:\trans>debug putciao.com
-u 100 113
168E:0100 B402 MOV AH,02
168E:0102 B243 MOV DL,43
168E:0104 CD21 INT 21
168E:0106 B249 MOV DL,49
168E:0108 CD21 INT 21
168E:010A B241 MOV DL,41
168E:010C CD21 INT 21
168E:010E B24F MOV DL,4F
168E:0110 CD21 INT 21
168E:0112 C3 RET
-s 100 112 cd,21
168E:0104
168E:0108
168E:010C
168E:0110
-s 100 112 cd,10
-
```

CREA CON: DEBUG<PUTCIAO.TXT

```
a
MOV AH, 2
MOV DL, 43
INT 21
MOV DL, 49
INT 21
MOV DL, 41
INT 21
MOV DL, 4F
INT 21
RET

n putciao.com
r bx
0
r cx
13
w
q
```

(mettere un invio dopo q)

Se la sequenza di bytes cercata non viene trovata il comando non produce effetto: (vedi ultimo comando del riquadro precedente).
Il comando **S** è particolarmente versatile: il suo terzo parametro può essere di diverso tipo: infatti oltre alle sequenze di bytes possono essere cercate anche stringhe Ascii di qualunque lunghezza. La cosa è particolarmente utile quando si deve trovare, per esempio, la zona dei messaggi di un dato programma. Dagli esempi si evince che la ricerca per stringhe non è case sensitive e che non si può effettuare la ricerca in una zona di memoria superiore a 64K

ESEMPI RICERCA STRINGHE

```
-s 0:0 LFFFF 'sechi'
0000:0E98
-s 0:0 LFFFF 'SECHI'
0000:0E98
-s 0:0 L10000 'sechi'
       ^ Errore
-d 0:e98
0000:0E90                         27 73 65 63 68 69 27 0D 'sechi'.
0000:0EA0 0D 0D 0D 2C 34 36 0D 00-00 00 00 00 00 00 00 00 ...,46..........
0000:0EB0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:0EC0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:0ED0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:0EE0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:0EF0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:0F00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0000:0F10 00 00 00 00 00 00 00 00 ........
```

Comando E - ENTER - sintassi: E [[Indirizzo]:Offset] <sequenza hex o ascii>

Il comando **E** (ENTER, inserisci) consente di modificare il contenuto di una o più locazioni di memoria a partire dall'indirizzo (espresso come segmento:offset) passato come primo parametro; dopo di esso possono essere scritti uno o più valori esadecimali, separati da spazi o da virgole. Non appena il comando così strutturato viene confermato con Invio il primo valore va a sostituire quello della locazione indicata nel comando, il secondo sostituisce il contenuto della locazione successiva, e così via...

ESEMPI DI SCRITTURA SULLA MEMORIA VIDEO - DIGITARE CLS PRIMA DI AVVIARE IL DEBUG
SI NOTI LA SCRITTA **ABCD** CHE APPARE DOPO AVER DIGITATO IL COMANDO e B800:BE 41 0E 42 0E 43 0E 44 0e

```
-C:\trans>debug ABCD
-d B800:A0
B800:00A0 43 07 3A 07 5C 07 74 07-72 07 61 07 6E 07 73 07 C.:.\.t.r.a.n.s.
B800:00B0 3E 07 64 07 65 07 62 07-75 07 67 07 20 07 20 07 >.d.e.b.u.g. . .
B800:00C0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:00D0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:00E0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:00F0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:0100 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:0110 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
-e B800:BE 'A' 0E 42 0E 43 0E 44 0e
-
```

Il comando **E** può essere dato anche senza la sequenza di bytes. Questo metodo consente di modificare il contenuto della ram in modo più sicuro del precedente. Infatti digitando **E** <indirizzo> e confermando con Invio viene mostrato l'indirizzo specificato seguito dal suo contento originale in esadecimale e da un punto. Se si digita un valore esadecimale esso andrà a sostituire quello visualizzato a sinistra. A questo punto possiamo seguire una delle due modalità possibili:
- se si preme il tasto Invio il processo di modifica viene confermato e riappare il prompt del debug (il trattino).

- se invece si preme lo spazio appare il contenuto della locazione successiva seguito ancora da un punto e il processo può continuare con le stesse modalità; i nuovi valori digitati andranno a sostituire i vecchi non appena si deciderà di dare la conferma finale (sempre con la pressione del tasto Invio).

Da notare che, se si conferma il byte mostrato con spazio senza aver digitato alcun numero sostitutivo, il byte originale non subisce variazioni, ma è ancora possibile intervenire sui valori successivi (sempre in attesa di confermare le eventuali modifiche con la pressione finale del tasto Invio):

ESEMPI DI SCRITTURA SULLA MEMORIA VIDEO - DIGITARE CLS PRIMA DI AVVIARE IL DEBUG
SI NOTI LA SCRITTA CDEFHJ CHE APPARE DOPO AVER DIGITATO LA SEQUENZA DI CARATTERI IN AZZURRO

```
C:\trans>debug CDEFHJ
-d B800:A0
B800:00A0 43 07 3A 07 5C 07 74 07-72 07 61 07 6E 07 73 07 C.:.\.t.r.a.n.s.
B800:00B0 3E 07 64 07 65 07 62 07-75 07 67 07 20 07 20 07 >.d.e.b.u.g. . .
B800:00C0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:00D0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:00E0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:00F0 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:0100 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
B800:0110 20 07 20 07 20 07 20 07-20 07 20 07 20 07 20 07 . . . . . .
-e B800:BE
B800:00BE 20.43  07.04
B800:00C0 20.44  07.03  20.45  07.    20.46  07.02  20.48  07.50
B800:00C8 20.4a  07.08  20.
```

Comando R - Register - sintassi: R [registro]

Il comando **R** (Register, registri) senza parametri mostra il contenuto dei registri (a 16 bit) del processore; naturalmente si tratta di una simulazione o meglio del valore che avrebbero i registri CPU prima di cominciare l'esecuzione del programma.

Facciamo delle considerazioni in merito ai valori iniziali nei registri:

- I registri comuni (quelli utente AX, DX, e quelli indice DI e SI, con BP) hanno sempre un valore iniziale pari a 0000H. La cosa è ragionevole poiché il DEBUG simula le condizioni ottimali. Sebbene non sia indispensabile è bene pensare di iniziare l'esecuzione di un programma con i registri "puliti".

STATO DEI REGISTRI COMUNI

```
C:\trans>debug
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=17B0 ES=17B0 SS=17B0 CS=17B0 IP=0100 NV UP EI PL NZ NA PO NC
17B0:0100 5A       POP DX
-q

C:\trans>debug video.com
-r
AX=0000 BX=0000 CX=0025 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17C1 ES=17C1 SS=17C1 CS=17C1 IP=0100 NV UP EI PL NZ NA PO NC
17C1:0100 E81B00 CALL 011E
-q

C:\trans>debug ml.exe
-r
AX=0000 BX=0005 CX=DE30 DX=0000 SP=00B8 BP=0000 SI=0000 DI=0000
DS=17C1 ES=17C1 SS=17D1 CS=17D1 IP=0000 NV UP EI PL NZ NA PO NC
17D1:0000 0E      PUSH CS
```

- Quando il comando DEBUG è eseguito senza argomenti abbiamo che:
> i registri CX e BX contengono il valore 0000h.
> i registri di segmento (CS, DS, SS e ES) hanno tutti lo stesso valore. Questo

valore non è prevedibile e dipende esclusivamente dall'indirizzo della prima locazione libera in memoria nel momento in cui debug è stato allocato dal loader del dos.
> il puntatore di stack (SP) ha sempre un valore piuttosto elevato FFEEH.
> il puntatore di istruzione (IP) sarà lasciato sempre a 0100H. Abbiamo più volte sottolineato che il loader del Dos vuole per sè le prime 256 locazioni, per il Prefisso di Segmento di Programma (PSP)

STATO DEI REGISTRI CON IL DEBUG SENZA ARGOMENTI

```
C:\trans>debug
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=17B0 ES=17B0 SS=17B0 CS=17B0 IP=0100 NV UP EI PL NZ NA PO NC
17B0:0100 5A       POP DX
```

- Quando il comando DEBUG è eseguito con un argomento relativo ad un eseguibile di tipo **.com** abbiamo che:
> i registri BX e CX vengono impostati con la dimensione in byte del file caricato. Il file di esempio (vedi figura successiva) **video.com** occupa 37 byte (25h in esadecimale) ed infatti BX=0000h e CX=0025h.
> i registri di segmento (CS, DS, SS e ES) hanno tutti lo stesso valore. Del resto i file **.com** sono monosegmento;
> il puntatore di stack (SP) ha sempre un valore piuttosto elevato FFFEH, solo 2 locazioni sopra la fine del segmento.
> il puntatore di istruzione (IP) è impostato a 0100H, esattamente come avviene nella realtà quando il programma sotto test è eseguito da solo senza la tutela del debugger.

STATO DEI REGISTRI CON IL DEBUG CON ARGOMENTO UN ESEGUIBILE .COM

```
C:\trans>debug video.com
-r
AX=0000 BX=0000 CX=0025 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17C1 ES=17C1 SS=17C1 CS=17C1 IP=0100 NV UP EI PL NZ NA PO NC
17C1:0100 E81B00 CALL 011E
C:\trans>dir video.com
 Il volume nell'unità C è HDD
 Numero di serie del volume: 3454-6C6B

Directory di C:\trans

04/01/2010  00.31              37 video.com
               1 File 37 byte
               0 Directory 15.469.608.960 byte disponibili
```

- Nel caso il comando DEBUG presenti come argomento il nome di un eseguibile di tipo **.exe** abbiamo che:
> i registri BX e CX vengono impostati ad un valore correlato con la dimensione in byte del file caricato. Occorre ricordare che la parte di intestazione (**header**: pari a 512byte) di un exe (è usata dal loader per il caricamento del programma stesso) non contiene codice eseguibile e pertanto non viene caricata dal debugger. Prendendo come esempio il file **ml.exe** (vedi figura successiva) notiamo che la quantità di byte effettivamente caricata è la dimensione del file meno quella relativa all'header. Calcoli alla mano abbiamo quindi: 385.072 byte - 512 byte = 384.560 che in esadecimale è 5 DE30h e che corrisponde ai valori visibile nei registri BX e CX).
> I registri di segmento (CS, DS, SS e ES), il puntatore di stack (SP) e di istruzione

(IP) sono impostati in base alle direttive indicate nell'header. L'intestazione di un programma EXE occupa almeno 512 bytes del programma EXE. Esistono alcune utility in grado di metterne in chiaro il contenuto dell'header. Una di queste è **ExeMod** distribuita con il pacchetto MASM.

1° ESEMPIO: STATO DEI REGISTRI CON IL DEBUG CON ARGOMENTO UN ESEGUIBILE .EXE	
C:\trans>**debug ml.exe** -**r** AX=0000 BX=0005 CX=DE30 DX=0000 SP=**00B8** BP=0000 SI=0000 DI=0000 DS=**17C1** ES=**17C1** SS=**17D1** CS=**17D1** IP=**0000** NV UP EI PL NZ NA PO NC 17D1:0000 0E PUSH CS -**q** C:\trans>**exemod ml.exe** Microsoft (R) EXE File Header Utility Version 4.02 Copyright (C) Microsoft Corp 1985-1987. All rights reserved. ml.exe (hex) (dec) .EXE size (bytes) 5E030 385072 Minimum load size (bytes) 450 1104 Overlay number 0 0 Initial CS:IP 0000:**0000** Initial SS:SP 0000:**00B8** 184 Minimum allocation (para) 0 0 Maximum allocation (para) FFFF 65535 Header size (para) 4 4 Relocation table offset 40 64 Relocation entries 0 0	Dimensione FILE: 385.072byte-512byte = 384.560byte => 5DE30h => AX=0005 e CX=DE30 Primo segmento disponibile 17C1: 17C1 ==> CS successivo segmento disponibile: 17D1 17D1+ 0000 = 17D1 ==> CS 17D1+ 0000 = 17D1 ==> SS IP e SP come indicato nell'header

Vediamo ulteriori esempi:

2° ESEMPIO: STATO DEI REGISTRI CON IL DEBUG CON ARGOMENTO UN ESEGUIBILE .EXE	
C:\trans>**exemod exehdr** Microsoft (R) EXE File Header Utility Version 4.02 Copyright (C) Microsoft Corp 1985-1987. All rights reserved. exehdr (hex) (dec) .EXE size (bytes) 10600 67072 Minimum load size (bytes) 8810 34832 Overlay number 0 0 Initial CS:IP 0020:2118 Initial SS:SP 0661:2000 8192 Minimum allocation (para) 241 577 Maximum allocation (para) FFFF 65535 Header size (para) 8 8 Relocation table offset 40 64 Relocation entries E 14 C:\trans>**debug exehdr.exe** -**r** AX=0000 BX=0001 CX=0400 DX=0000 SP=2000 BP=0000 SI=0000 DI=0000 DS=**17C1** ES=17C1 SS=**1E32** CS=17F1 IP=2118 NV UP EI PL NZ NA PO NC 17F1:2118 B430 MOV AH,30	Dimensione FILE: 67.072byte-512byte = 66560byte => 10400h => AX=0001 e CX=0400 Primo segmento disponibile 17C1: 17C1 ==> CS successivo segmento disponibile: 17D1 17D1+ 0020 = 17F1 ==> CS 17D1+ 0661 = 1E32 ==> SS IP e SP come indicato nell'header

3° ESEMPIO: STATO DEI REGISTRI CON IL DEBUG CON ARGOMENTO UN ESEGUIBILE .EXE	
C:\trans>**exemod beep** Microsoft (R) EXE File Header Utility Version 4.02 Copyright (C) Microsoft Corp 1985-1987. All rights reserved. beep.exe (hex) (dec) .EXE size (bytes) 14CA 5322 Minimum load size (bytes) 227A 8826 Overlay number 0 0 Initial CS:IP 0000:0054 Initial SS:SP 012D:0FA0 4000 Minimum allocation (para) FB 251 Maximum allocation (para) FFFF 65535 Header size (para) 20 32 Relocation table offset 1E 30 Relocation entries 2 2 C:\trans>**debug beep.exe** -**r**	Dimensione FILE: 5.322byte-512byte = 4810byte => 12CAh => AX=0000 e CX=12CA Primo segmento disponibile 17C1: 17C1 ==> CS successivo segmento disponibile: 17D1 17D1+ 0000 = 17D1 ==> CS 17D1+ 012D = 18FE ==> SS IP e SP come indicato nell'header

```
AX=0000 BX=0000 CX=12CA DX=0000 SP=0FA0 BP=0000 SI=0000 DI=0000
DS=17C1 ES=17C1 SS=18FE CS=17D1 IP=0054 NV UP EI PL NZ NA PO NC
17D1:0054 BFD818                MOV DI,18D8
```

4° ESEMPIO: STATO DEI REGISTRI CON IL DEBUG CON ARGOMENTO UN ESEGUIBILE .EXE

```
C:\trans>exemod E:\Arch-Lab\_Help\FreeDOS\comandi\asm\asm.exe

Microsoft (R) EXE File Header Utility Version 4.02
Copyright (C) Microsoft Corp 1985-1987. All rights reserved.

E:\Arch-Lab\_Help\FreeDOS\comandi\asm\asm.exe (hex)    (dec)

.EXE size (bytes)                           D05A    53338
Minimum load size (bytes)                  16AAA    92842
Overlay number                                 0        0
Initial CS:IP                           0CCF:000E
Initial SS:SP                           14AB:0080      128
Minimum allocation (para)                     9A7     2471
Maximum allocation (para)                    FFFF    65535
Header size (para)                              2        2
Relocation table offset                        1C       28
Relocation entries                              0        0
C:\trans>debug E:\Arch-Lab\_Help\FreeDOS\comandi\asm\asm.exe
-r
AX=0000 BX=0000 CX=CE5A DX=0000 SP=0080 BP=0000 SI=0000 DI=0000
DS=1818 ES=1818 SS=2CD3 CS=24F7 IP=000E NV UP EI PL NZ NA PO NC
24F7:000E 0E                    PUSH CS
```

Dimensione FILE:
53.338byte-512byte = 52826byte =>
CE5Ah => AX=0000 e CX=CE5A

Primo segmento disponibile 1818:
1818 ==> CS
successivo segmento disponibile: 1828
1828+ 0CCF= 24F7 ==> CS
1828+ 14AB = 2CD3 ==> SS

IP e SP come indicato nell'header

Di solito il comando **R** è utile per modificare il valore di un registro; per questo è necessario far seguire al comando la sigla del registro desiderato. Non appena si conferma con Invio viene proposto il vecchio valore e, sulla linea successiva, appare il segno dei 2 punti, dopo il quale è possibile digitare il nuovo valore, da confermare ancora con Invio:

MODIFICA DEL CONTENUTO DEI REGISTRI

```
-r ax
AX 0000
:1234
-r
AX=1234 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=166C ES=166C SS=166C CS=166C IP=0100 NV UP EI PL NZ NA PO NC
166C:0100 4D       DEC   BP
```

Il comando **R** è usato ogni volta che si desidera ricominciare l'analisi e il collaudo di un programma. Infatti è necessario *obbligare il processore a puntare sulla prima istruzione*: **non è dunque possibile fare a meno** del *comando* **R IP** *seguito dal valore* **0100** (nel caso di un eseguibile COM)!

Comando RF - Registro Flag

Una trattazione a parte merita la visualizzazione del registro dei flag. E' visualizzato in forma interpretata (zona gialla della figura precedente), cioè al posto dei bit significativi viene mostrata la sigla del valore attuale del flag corrispondente, in accordo con la seguente tabella:

FLAG		SET		RESET	
Overflow	Overflow	OV	si	NV	no
Direction	Direzione	DN	decremento	UP	incremento
Interrupt	Interruzione	EI	abilitato	DI	disabilitato
Sign	Segno	NG	negativo	PL	positivo
Zero	Zero	ZR	si	NZ	no
Auxiliary Carry	Riporto ausiliario	AC	si	NA	no
Parity	Parità	PE	pari	PO	dispari
Carry	Riporto	CY	si	NC	no

Il comando **RF** mostra lo stato corrente dei flag documentati: se si vuole è possibile modificarne uno o più, semplicemente specificando la sigla del nuovo stato, anche in ordine casuale:

MODIFICA DEL CONTENUTO DEL REGISTRO DEI FLAG

```
-rf
NV UP EI PL NZ NA PO NC -CY NG
-rf
NV UP EI NG NZ NA PO CY -
```

Comando L - Load File - sintassi: L [[CS]:Offset] [idDisco NrPrimoSettore QuantiSettori]

Il comando **L** (LOAD, carica) carica in memoria un file di qualunque tipo; di solito viene dato senza parametri, dopo aver specificato il nome (con l'eventuale percorso) mediante il comando **N**, già visto in precedenza. Il file viene sistemato in memoria rigorosamente a partire dall'indirizzo CS:0100H (ad esclusione dei file EXE che vengono rilocati all'indirizzo indicato nel suo Header).

CARICAMENTO FILE - SI NOTI CHE CX CONTIENE IL NUMERO 407h CHE CORRISPONDE IN DECIMALE A 1031

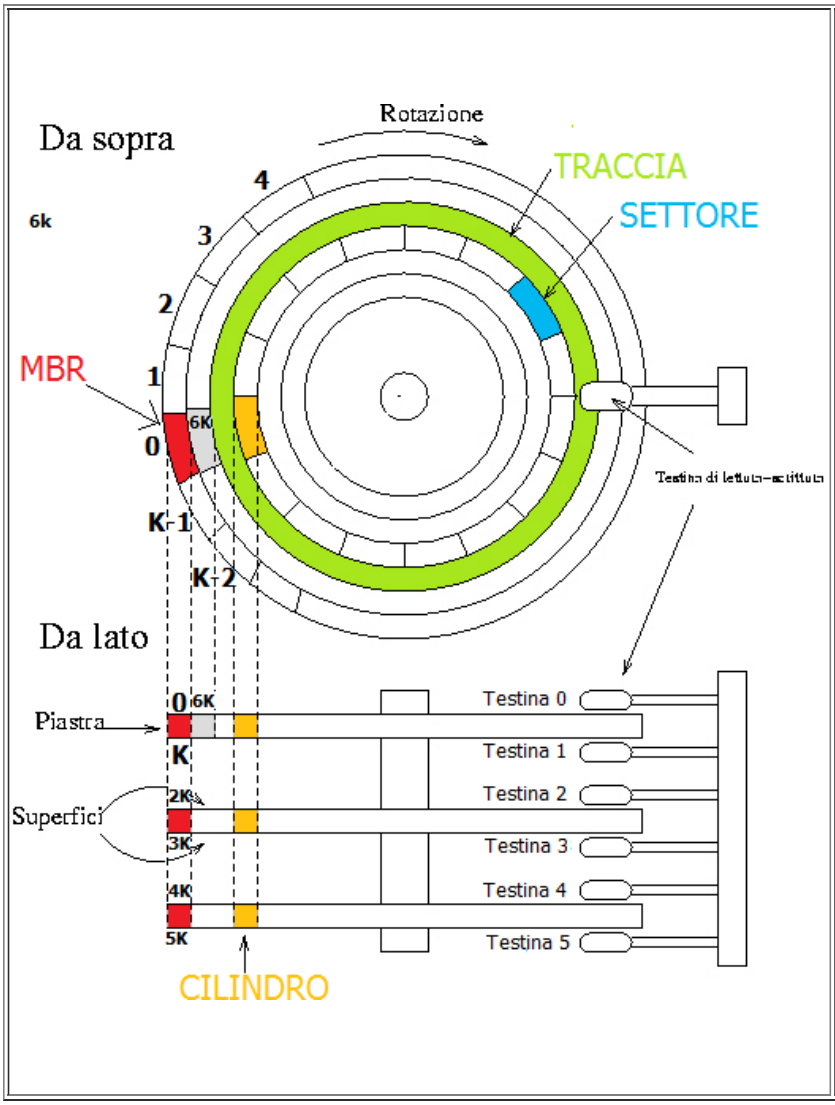
```
C:\trans>debug
-n prova.com
-l
-r
AX=0000 BX=0000 CX=0407 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=168C ES=168C SS=168C CS=168C IP=0100 NV UP EI PL NZ NA PO NC
168C:0100 E92301                JMP 0226
-q

C:\trans>dir

Il volume nell'unità C non ha etichetta
Numero di serie del volume: 141E-1AF3
Directory di C:\trans

.            <DIR>    16/05/05    1.11 .
..           <DIR>    16/05/05    1.11 ..
PROVA COM    1.031    24/05/09   11.08 PROVA.COM
               2 file        29.779 byte
               2 dir    346.394.624 byte disponibili
```

Il file caricato in memoria può essere di qualunque tipo (di testo TXT, di dati DAT, eseguibile COM o EXE) sebbene la potenza di debug sia meglio evidenziata nella gestione dei files eseguibili.
Se, dopo il comando **L**, è specificato un indirizzo (offset) esso rappresenta la locazione a partire dalla quale viene caricato il file; naturalmente questo ha senso solo se il file è diverso da EXE: infatti questa tipologia di file viene sempre rilocata all'indirizzo specificato nel suo header!
Se l'indirizzo specificato è solo quello di offset i dati visualizzati si ritengono appartenenti al segmento **CS**.
Non dimenticare che, dopo la copia del file è disponibile un numero a 32 bit che esprime la quantità di bytes caricati in memoria, lasciato nella coppia di registri **BX**,**CX**, rispettivamente parte alte e parte bassa. Si noti che nell'esempio **CX** vale 407h che corrisponde alla dimensione del file prova.com pari a 1031 byte.

Il comando **L** (LOAD, carica) è spesso utile anche per caricare in memoria uno o più settori di una memoria di massa (hard disk o floppy disk).
In questo caso la sintassi è :

```
L [[CS:]Offset] IDDisco NrPrimoSettore QuantiSettori
```

Dove:
- l'indirizzo **[[CS]:offset]** è dove travasare i bytes letti sul disco (le [] indicano l'opzionalità)
- **IDDisco** è un numero che indica il disco da cui prendere i dati:
  0 per il drive A,
  1 per il drive B,
  2 per il drive C, e cosi via...
- **NrPrimoSettore** indica il numero logico di settore del primo settore da caricare dal disco appena indicato;
- **QuantiSettori** indica il numero di settori da caricare; non è possibile leggere più di 80 settori, pari a 40k di memoria.



Vediamo che cosa si intende per numero logico di settore (LBA). Precisiamo che il settore è l'unità minima di informazione che può essere letta o scritta e varia, a seconda della capacità del disco, da 32 a 4096 byte (di solito sono 512).
Si tratta di una numerazione progressiva che parte da 0 e raggiunge un valore massimo dato dalla seguente espressione:

(N°Settori per Traccia) x (N°Tracce) x (N°Lati).

La numerazione dei settori logici parte dal primo settore fisico (master boot record) e prosegue in sequenza sulla medesima traccia (Cylinder) del lato (Head) corrente. Si prosegue poi lungo tutte le tracce dello stesso cilindro passando prima sul lato opposto e poi su quelli successivi fino all'esaurimento di tutti i settori del cilindro più esterno. Si continua spostandosi sul cilindro interno adiacente e si ripete così il procedimento fino all'ultimo settore dell'ultima traccia dell'ultimo cilindro.
Questo modo logico di numerare i settori di un disco è tipico del DOS ed è diverso da quello assoluto/fisico usato dal BIOS e costituito dalla terna di numeri detta sinteticamente CHS (Cylinder, Head, Sector).
Così, per esempio, il primo settore fisico, il boot record, identificato dal numero CHS 0,0,1, corrisponde al settore logico 0.
La numerazione logica dei settori da parte del DOS è estremamente efficace perché consente al Sistema Operativo di accedere a qualunque disco, indipendentemente dalle sue caratteristiche hardware.

La formula che lega la numerazione fisica (CHS (cylinder-head-sector) o BIOS) a quella logica (o DOS) è la seguente:

LBA = (S-1)+(H*k)+ C*k*f dove **f** è il numero di testine e **k** il nr di settori per traccia

La numerazione logica dei settori da parte del DOS è, per altro, estremamente efficace perché consente al Sistema Operativo di accedere a qualunque disco, indipendentemente dalle sue caratteristiche hardware.

La possibilità di leggere interi settori di un disco è senza dubbio una funzionalità di straordinaria potenza. Sebbene non comporti grandi rischi conviene agire con discrezione, sicurezza e competenza.

CARICAMENTO DEL PRIMO SETTORE DEL DISCO C (512 BYTE) - WINDOWS 98

```
C:\trans>debug
-l 1000 2 0 1
-d 1000
166C:1000 EB 58 90 4D 53 57 49 4E-34 2E 31 00 02 08 20 00 .X.MSWIN4.1... .
166C:1010 02 00 00 00 00 F8 00 00-3F 00 40 00 3F 00 00 00 ........?.@.?...
166C:1020 81 77 25 00 5C 09 00 00-00 00 00 00 02 00 00 00 .w%.\...........
166C:1030 01 00 06 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
166C:1040 80 00 29 F3 1A 1E 14 4E-4F 20 4E 41 4D 45 20 20 ..)....NO NAME
166C:1050 20 20 46 41 54 33 32 20-20 20 FA 33 C9 8E D1 BC FAT32 .3....
166C:1060 F8 7B 8E C1 BD 78 00 C5-76 00 1E 56 16 55 BF 22 .{...x..v..V.U."
166C:1070 05 89 7E 00 89 4E 02 B1-0B FC F3 A4 8E D9 BD 00 ..~..N..........
```

Il comando **L**, della figura, carica a partire dall'indirizzo 1000h, dal disco 2 (driver C) partendo dal settore n°0 (il primo) 1 solo settore (cioè 512 bytes). Nella figura successiva si può notare che la lettura diretta dell'Hard Disk non è permessa nei sistemi WIN recenti.

Comando W - Scrive su disco un file - sintassi: W [[CS]:Offset] [idDisco NrPrimoSettore QuantiSettori]

Il comando **W** (WRITE, scrivi) permette di creare un file (o di sovrascriverlo, se già esiste) nella cartella corrente del disco; di solito viene dato dopo aver specificato il nome (e l'eventuale percorso) del file con il comando **N**, già visto in precedenza. La sintassi in questo caso è la seguente:

```
W [[CS:]Offset]
```

Se l'indirizzo specificato è solo quello di offset i dati da scrivere si ritengono appartenenti al segmento CS.

Quando si usa il comando **W** per salvare il nostro lavoro in un file sul disco non bisogna dimenticare di caricare nei registri BX e CX la parte alta e bassa del numero a 32 bytes che indica la quantità di bytes da salvare!!

Il modo più semplice per calcolare il numero da mettere in **BX**,**CX** consiste nel

prendere l'indirizzo di offset della locazione successiva all'ultima istruzione digitata e sottrarre da esso il numero 0100h.

CREAZIONE DI UN PICCOLO PROGRAMMA

```
C:\trans>debug
-a
1679:0100 mov ah,9
1679:0102 mov dx,108
1679:0105 int 21
1679:0107 ret
1679:0108 db 'Hello, World!',0d,0a,'$'
1679:0118
-r bx
BX 0000
:0
-r cx
CX 0000
:18
-n saluta.com
-w
Scrittura di 00018 byte in corso
-q

C:\trans>saluta
Hello, World!
```

Ad esempio se, abbiamo prodotto una codice eseguibile di 24 bytes, quando salviamo su disco dobbiamo eseguire, in sequenza, tutte queste operazioni:
- dare un nome al file da creare (o sovrascrivere) tramite il comando **N**
- scrivere in **BX** il valore 0000H e in **CX** il valore 0018H (esadecimale di 24)
- finalmente confermare il salvataggio con il comando **W**:

Il comando **W** (WRITE, scrivi) è complementare al comando **L**, usato per caricare in memoria uno o più settori di una memoria di massa (hard disk o floppy disk); Quindi **W** serve anche per fare l'operazione opposta ovvero per scrivere direttamente un settore di disco.

La sintassi in questo caso è:

```
W [[CS:]Offset] IDDisco NrPrimoSettore QuantiSettori
```

Dove:
- l'indirizzo **[[CS]:offset]** è dove prelevare i bytes da riversare sul disco (le [] indicano l'opzionalità)
- **IDDisco** è un numero che indica il disco da cui prendere i dati:
  0 per il drive A,
  1 per il drive B,
  2 per il drive C, e cosi via...
- **NrPrimoSettore** indica il numero logico di settore del primo settore da sovrascrivere del disco appena indicato;
- **QuantiSettori** indica il numero di settori da scrivere; non è possibile scrivere più di 80 settori, pari a 40k di memoria.

L'esempio successivo mostra come scrivere su un floppy 2Kb (4 settori da 512 byte) attualmente presenti a partire dall'indirizzo CS:1000H; il primo settore da sostituire è il 64h (valore esadecimale pari al decimale 100d) :

SCRITTURA DI DUE KBYTE SU FLOPPY

```
C:\trans>debug
-w 1000 0 64 4
```

Comando U - Disassembla eseguibile - sintassi: U <IndirizzoIniziale Area1> <IndirizzoFinale o lunghezza Area1>

Il comando **U** (UNASSEMBLE, disassembla) ha senso solo dopo aver specificato il nome di un file eseguibile (con il comando **N**) e averlo caricato in memoria (con il comando **L**), magari dopo aver pulito la memoria chiamata ad ospitarlo (con il comando **F**). Se viene dato senza parametri inizia a tradurre i bytes presenti in memoria a partire dall'indirizzo CS:0100H; il segmento di default è ovviamente quello del codice: **CS**.
E' facile capire la potenza di questo comando: con esso si possono analizzare le istruzioni di qualunque file eseguibile, in primis quelli da noi prodotti, in fase di messa a punto o di ricerca delle rogne.
Naturalmente la "traduzione" in chiaro di un sorgente non è priva di intoppi e malizie: è il prezzo che bisogna pagare all'esperienza e al colpo d'occhio.

Il comando **U** prova comunque a tradurre in assembly i bytes che trova; se per qualche ragione il programma contiene tabelle, dati o messaggi (cosa estremamente probabile!) il risultato sarà del tutto illogico, cioè verranno mostrate sequenze di istruzioni assurde e scorrelate. Bisogna quindi saper localizzare le aree dati di un programma, escludendole dall'indagine con **U** a beneficio di quella effettuata con il comando **D**.

Consideriamo a titolo d'esempio il seguente file sorgente:

FILE: CLSKEYP.ASM

```
.MODEL tiny
.code
   ORG 100H
INIZIO:  MOV AH,00H  ; Pulisci lo schermo
         MOV AL,03H  ; (ClearScreen)
         INT 10H
         MOV AH,00H  ; Aspetta la pressione
         INT 16H     ; di un tasto
         INT 20H     ; Torna al dos
END INIZIO
```

Creiamo il file eseguibile utilizzando il macro assembler ML con il comando **`ML /AT CLSKEYP.ASM`**.

Disassembliamo il file .com così ottenuto:

DISASSEMBLAGGIO DEL FILE CLSKEYP.COM

```
C:\trans>dir clskeyp.com
 Il volume nell'unità C è HDD
 Numero di serie del volume: 3454-6C6B

Directory di C:\arch_def_1\Scuola\sitoBrescianet\Prove\assembler

06/01/2010   19.41            12 clskeyp.com
               1 File 12 byte
               0 Directory 15.425.859.584 byte disponibili

C:\trans>debug clskeyp.com
-u
17C1:0100 B400      MOV AH,00
17C1:0102 B003      MOV AL,03
17C1:0104 CD10      INT 10
17C1:0106 B400      MOV AH,00
17C1:0108 CD16      INT 16
17C1:010A CD20      INT 20
17C1:010C 7A69      JPE 0177
```

```
17C1:010E 6F        DB  6F
17C1:010F 3A20      CMP AH,[BX+SI]
17C1:0111 2020      AND [BX+SI],AH
17C1:0113 2020      AND [BX+SI],AH
17C1:0115 2020      AND [BX+SI],AH
17C1:0117 206D6F    AND [DI+6F],CH
17C1:011A 7620      JBE 013C
17C1:011C 2020      AND [BX+SI],AH
17C1:011E 2020      AND [BX+SI],AH
-r
AX=0000 BX=0000 CX=000C DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17C1 ES=17C1 SS=17C1 CS=17C1 IP=0100 NV UP EI PL NZ NA PO NC
17C1:0100 B400      MOV AH,00
-u 100 LC
17C1:0100 B400      MOV AH,00
17C1:0102 B003      MOV AL,03
17C1:0104 CD10      INT 10
17C1:0106 B400      MOV AH,00
17C1:0108 CD16      INT 16
17C1:010A CD20      INT 20
```

Osserviamo con calma l'output del comando **U**:
- la pressione di **U** (ed ogni altra successiva) coinvolge almeno 16 bytes di memoria (nell'esempio al primo **U** 32 byte). L'indirizzo della locazione (se non specificato) è sempre dato dal valore corrente di **CS**:**IP**.
- I bytes di ciascuna riga visualizzata appartengono ad una ben precisa istruzione, ad insindacabile giudizio del debugger. Per ogni riga mostrata sono resi disponibili:
  > l'indirizzo logico (CS:Offset corrente) del primo dei bytes coinvolti.
  > la sequenza dei bytes associati all'istruzione riconosciuta; di essi il primo è certamente un codice operativo.
  > la loro interpretazione mnemonica:
- Il processo continua per tutti i bytes coinvolti, anche per quelli che non appartengono più al programma caricato in memoria; lo si può notare nelle ultime righe ottenute con la prima esecuzione del comando **U** dell'esempio precedente dove il codice, del tutto improbabile e illogico, non corrisponde più al sorgente che abbiamo compilato.

In certi casi è conveniente passare 2 parametri, l'indirizzo iniziale e il numero (preceduto da una **L**) di byte, in esadecimale, da dissassemblare. Conoscendo la lunghezza del programma è possibile quindi evitare la visualizzazione di istruzioni fasulle.
Vediamo un esempio di utilizzo mediante un file batch. Si consideri il seguente file di testo disasm.txt e l'eseguibile clskeyp.com che abbiamo appena creato. Poiché l'eseguibile è composto da 12 byte inserisco, dopo il parametro **L,** la rappresentazione esadecimale di 12 ovvero: C.

FILE: DISASM.TXT

```
u 100 Lc
q
```

(ricordarsi l'invio finale)

Eseguiamo la sequenza di comandi qui sotto

CREAZIONE DEL SORGENTE ASSEMBLER

```
C:\trans>debug clskeyp.com<disasm.txt|find ":">clskeyp.src


C:\trans>type clskeyp.src
17C1:0100 B400 MOV AH,00
17C1:0102 B003 MOV AL,03
17C1:0104 CD10 INT 10
```

```
17C1:0106 B400 MOV AH,00
17C1:0108 CD16 INT 16
17C1:010A CD20 INT 20

C:\trans>
```

Il file testuale ottenuto contiene la sequenza di istruzioni assembler del nostro programma. Chiaramente per poter svolgere questa operazione correttamente è necessario modificare DISASM.TXT in modo che sappia distinguere l'area codice (su cui applico **U**) dall'area dati dove devo usare il comando **D**.

Comando A - ASSEMBLA - Crea un programma eseguibile -

sintassi: A [[CS]:offset]

Il comando **A** (ASSEMBLY, assembla) permette la creazione di programmi eseguibili anche in ambiente debug; si tratta di un esercizio senza futuro, al limite della paranoia, poco pratico e piuttosto complesso; la logica dei debugger (per esempio quella di richiedere numeri esadecimali senza la H finale) rende questo esercizio addirittura controproducente.
L'avvertimento appena evidenziato è per sottolineare di evitare l'uso proprio di questo comando per creare programmi. Anche l'uso per effettuare piccole correzioni locali è assurdo: si fa prima a tornare in editor, correggere, ricompilare e rientrare in debug con la nuova versione dell'eseguibile.
Volendo trovare per forza una occasione utile possiamo pensare all'uso in una esercitazione sulla scrittura di istruzioni poiché l'assemblatore del debug (cioè il comando **A**) è in grado di riconoscere gli errori, se la stringa digitata non è corretta viene segnalato errore.

Vediamo dunque come si usa Assembly: non appena si conferma con Invio il comando **A**, sulla riga successiva appare l'indirizzo puntato da **CS**:**IP**. Nella posizione dove lampeggia il cursore dovremo scrivere il codice mnemonico di una istruzione, confermando ogni volta con Invio.
Solo quando riteniamo di aver finito bisogna premere l'invio 2 volte di fila: si torna così al prompt del debug (il segno meno, a sinistra).

CREAZIONE DI UN PICCOLO PROGRAMMA

```
C:\trans>debug
-a
179E:0100 mov ah,9
179E:0102 mov dx,108
179E:0105 int 21
179E:0107 ret
179E:0108 db 'Ciao, 3N e 3M Itis',0d,0a,'$'
179E:011D
-r bx
BX 0000
:0
-r cx
CX 0000
:1D
-n saluta.com
-w
Scrittura di 0001D byte in corso
-u 100 L8
179E:0100 B409   MOV AH,09
179E:0102 BA0801 MOV DX,0108
179E:0105 CD21   INT 21
179E:0107 C3     RET
-d 108 11C
179E:0100                43 69 61 6F 2C 20 33 4E      Ciao, 3N
```

```
179E:0110 20 65 20 33 4D 20 49 74-69 73 0D 0A 24 e 3M Itis..$
-q

C:\trans>saluta
Ciao, 3N e 3M Itis

C:\trans>
```

La verifica con il comando **U 100 L8** (figura qui sopra) dimostra che in memoria sono stati effettivamente allocati i bytes corrispondenti alle istruzioni che abbiamo digitato:

I comandi più importanti del debugger sono certamente quelli illustrati qui di seguito: la loro azione è rivolta decisamente al collaudo di un programma, cioè alla sua esecuzione controllata.
Se il tuo programma fa i capricci, basta forzare l'esecuzione fino ad un certo punto sicuro (comando **G**) e poi procedere a piccoli passi (comando **T**) evitando di eseguire (comando **P**) le procedure personali (riconoscibili dalla mnemonica **CALL**) o di sistema (attivate dall'istruzione **INT**).

Comando T - TRACE - sintassi: T [=IndirizzoDaEseguire oppure NrIstruzioni]

Il comando **T** (TRACE, traccia) esegue una o più istruzioni; di solito viene usato senza parametri, al fine di eseguire l'istruzione corrente (cioè quella puntata da **CS**:**IP**); subito dopo visualizza automaticamente la nuova situazione dei registri e le informazioni relative all'istruzione cha verrà eseguita al prossimo step (indirizzo logico, codici associati e stringa mnemonica).
La tecnica utilizzata da questo comando è nota come esecuzione passo-passo.
Questo comando consente l'uso di 2 parametri, entrambi poco pratici:
- il primo consente di specificare l'indirizzo della prima istruzione da eseguire (esempio: **T = 0112**, notare il segno "uguale", mai usato finora); anche volendo far partire l'esecuzione da un certo punto intermedio del programma l'effetto potrebbe essere diverso da quello ottenibile nell'esecuzione passo-passo dall'inizio: verrebbe a mancare infatti l'influenza sui registri di tutte le precedenti istruzioni.
- il secondo parametro è rischioso, se usato con sufficienza o scarsa attenzione; consente di eseguire di seguito più istruzioni, semplicemente indicandone il numero dopo il comando **T** (esempio: **T 12**). Il rischio è di entrare dentro le procedure e perdere il filo del programma, specialmente se si entra in una **INT** di sistema.

DEBUG PASSO PASSO DI UN PICCOLO PROGRAMMA

```
C:\trans>debug saluta.com
-r
AX=0000 BX=0000 CX=001D DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17CB ES=17CB SS=17CB CS=17CB IP=0100 NV UP EI PL NZ NA PO NC
168D:0100 B409       MOV AH,09
-u 100 18
17CB:0100 B409       MOV AH,09
17CB:0102 BA0801     MOV DX,0108
17CB:0105 CD21       INT 21
17CB:0107 C3         RET
-t

AX=0900 BX=0000 CX=001D DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17CB ES=17CB SS=17CB CS=17CB IP=0102 NV UP EI PL NZ NA PO NC
17CB:0102 BA0801     MOV DX,0108
-t
```

```
AX=0900 BX=0000 CX=001D DX=0108 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17CB ES=17CB SS=17CB CS=17CB IP=0105 NV UP EI PL NZ NA PO NC
17CB:0105 CD21        INT 21
-t

AX=0900 BX=0000 CX=001D DX=0108 SP=FFF8 BP=0000 SI=0000 DI=0000
DS=17CB ES=17CB SS=17CB CS=0C52 IP=0445 NV UP DI PL NZ NA PO NC
0C52:0445 EAA004B507 JMP 07B5:04A0
```

Osserviamo con calma cosa ci fornisce il comando **T.** In particolare possiamo seguire i cambiamenti dei registri dopo l'esecuzione dell'istruzione corrente; per esempio l'azione della prima **T** (esegue **MOV AH,09**) cambia il valore nel registro **AH** (si vede chiaramente che AX diventa 0900h)
l'azione della terza **T** è una cosa inutile e sbagliata perché ci obbliga a entrare nel codice di una procedura di sistema (**INT 10**) di cui diamo per scontata l'esattezza; in questi casi bisogna dare il comando **P**.

Notare come l'azione della terza **T** abbia cambiato drasticamente il registro **IP** rispetto ai prevedibili cambiamenti imposti nei due primi trace. Anche il valore del registro di segmento ha assunto un valore diverso rimarcando il fatto che non siamo più dentro il nostro programma (o meglio nel segmento ad esso riservato dal debugger). L'indirizzo fisico è ora 0C965H (**CS**=0C52h **IP**=0445h => C520h+445h=C965h) , molto diverso da quel 17DB5h da cui siamo partiti (**CS**=17CBh **IP**=0105h => 17CB0h+105h=17DB5h).
Se per errore entriamo all'interno del codice di una **INT** è poi laborioso ritornare nel nostro programma: bisogna riportare sia **CS** che **IP** ai valori originali, sperando di non essere andati troppo avanti, cioè di non averli persi di vista irrimediabilmente. Pertanto in presenza di un istruzione **INT** occorre usare il comando **P**

DEBUG PASSO PASSO DI UN PICCOLO PROGRAMMA

```
C:\trans>debug saluta.com
-u 100 18
17CB:0100 B409        MOV AH,09
17CB:0102 BA0801      MOV DX,0108
17CB:0105 CD21        INT 21
17CB:0107 C3          RET
-t

AX=0900 BX=0000 CX=001D DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17CB ES=17CB SS=17CB CS=17CB IP=0102 NV UP EI PL NZ NA PO NC
17CB:0102 BA0801      MOV DX,0108
-t

AX=0900 BX=0000 CX=001D DX=0108 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17CB ES=17CB SS=17CB CS=17CB IP=0105 NV UP EI PL NZ NA PO NC
17CB:0105 CD21        INT 21
-p
Ciao, 3N e 3M Itis

AX=0924 BX=0000 CX=001D DX=0108 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17CB ES=17CB SS=17CB CS=17CB IP=0107 NV UP EI PL NZ NA PO NC
17CB:0107 C3 RET
```

Comando P - Esegue delle procedure - sintassi: P [=IndirizzoDaEseguire oppure NrIstruzioni]

Il comando **P** (PROCEED, procedi) esegue un'intera procedura, di solito rappresentata dall'istruzione **CALL xxxx** o **INT xyH**: nel primo caso **xxxx** è un

sottoprogramma sviluppato con il codice eseguibile sotto test, normalmente allocato nel medesimo segmento di codice dell'istruzione **CALL**. **INT** rappresenta la chiamata ad una procedura di sistema, sempre esterna al segmento del nostro codice.

Vediamo alcune allocazioni di sistema standard:
- una funzione BIOS trova posto nella parte più alta del primo mega di memoria, nell'area fisica che va dall'indirizzo F000H, a meno che non sia stata "rilocata" da programmi di gestione, come drivers o applicativi residenti in memoria. Capita spesso che la scheda video si appropri del vettore INT 10h, mappandone i servizi dentro la sua rom, a partire dall'indirizzo C0000H
- una funzione DOS è collocata nella parte bassa della memoria: un indirizzo verosimile è per esempio 0E000H
- una funzione speciale (per esempio quelle del mouse, attivate con INT 33H) è allocata in posizioni non molto distanti da quelle del DOS, per esempio a 11000H

Quindi le istruzioni **INT xyH** non vanno mai tracciate con il comando **T**, ma eseguite in blocco con il comando **P**; il loro debug può essere interessante solo a fini didattici o di auto-istruzione, per vedere come sono fatte.

Comando G - Esegue il programma - sintassi: G [IndirizzoDoveArrivare]

Il comando **G** (GO, vai!) esegue tutto il programma; se viene dato senza parametri inizia l'esecuzione di tutto il programma presente in memoria a partire dall'indirizzo **CS**:**IP**; il segmento di default è ovviamente quello di codice, **CS**. Nei file .com **IP** = 100h
Naturalmente l'esecuzione "diretta" (senza parametri) del programma non ha senso: per fare questo basta lanciare l'eseguibile dal prompt del Dos, senza scomodare il debugger.
Lo scopo di **G** è quello di eseguire in blocco un certo numero di istruzioni, per poi proseguire passo-passo con il comando **T** o con il comando **P**; per fare questo è sufficiente aggiungere un indirizzo dopo la **G**, confermando con Invio; il programma sarà eseguito fino al punto indicato.

La sequenza di azioni consigliata per la ricerca di malfunzionamenti è la seguente:
- scorro con il comando **U** il codice disassemblato e dopo aver annotato l'indirizzo **xyH** dell'istruzione successiva a quella fino alla quale voglio eseguire il programma digito il comando **G xyH** e confermo con Invio. Vediamo un esempio:

ESECUZIONE FINO AD UN DETERMINATO INDIRIZZO

```
C:\trans>debug saluta.com
-u 100 18
17CB:0100 B409 MOV AH,09
17CB:0102 BA0801 MOV DX,0108
17CB:0105 CD21 INT 21
17CB:0107 C3 RET
-g 105

AX=0900 BX=0000 CX=001D DX=0108 SP=FFFE BP=0000 SI=0000 DI=0000
DS=17CB ES=17CB SS=17CB CS=17CB IP=0105 NV UP EI PL NZ NA PO NC
17CB:0105 CD21 INT 21
```

Questo comando consente anche di eseguire in blocco le istruzioni ripetute (quelle con prefisso **REP**) e le istruzioni gestite da **LOOP**: in questo caso può essere conveniente alternare questa possibilità a quella offerta dal comando **T**, più adatto per cercare e trovare imperfezioni durante l'esecuzione di queste procedure iterative.

Quando si utilizza il comando **G** occorre fare molta attenzione a specificare l'indirizzo corretto, altrimenti si rischia di mandare in crash il computer!!

Comando H - SOMMA E DIFFERENZA TRA 2 HEX - sintassi: H <valoreh1> <valoreh2>

Il comando **H** (HEX, esadecimale) calcola e visualizza contemporaneamente prima la somma e poi la differenza dei 2 numeri esadecimali forniti dopo il comando; entrambi non possono essere più grandi di 4 cifre (e vanno specificati ovviamente senza la **H** finale). I risultati negativi sono espressi in complemento a 2. Vediamo alcuni esempi:

ESECUZIONE FINO AD UN DETERMINATO INDIRIZZO

```
C:\trans>debug
-h 100 200
0300 FF00
-h 23A 100
033A 013A
-h 9876 1234
AAAA 8642
```

Comando I - Input da una porta - sintassi: I <IndirizzoPorta>

Il comando **I** (INPUT, ingresso) esegue la lettura della porta specificata dal parametro e mostra, sulla riga successiva, il valore esadecimale letto.

LETTURA REGISTRO DI STATO DELLA PORTA PARALLELA LPT1 E LPT2

```
-i 379
DE
-i 279
00
```

Comando O - Ouput su una porta - sintassi: O <IndirizzoPorta> <Codice>

Il comando **O** (OUTPUT, uscita) spedisce un byte alla porta specificata dal primo parametro; il byte da trasmettere è passato come secondo parametro.

SCRITTURA NEL REGISTRO DI USCITA DELLA PORTA PARALLELA LPT1:

```
-o 378 ff
```