Classe 3a INF - Esecuzione di un interrupt: ricostruzione delle operazioni al calcolatore con Turbo Debugger

Turbo debugger è un potente strumento di sviluppo pensato, anni fa, come complemento agli strumenti di programmazione di casa Borland, quali Turbo Pascal, Turbo C, ecc.

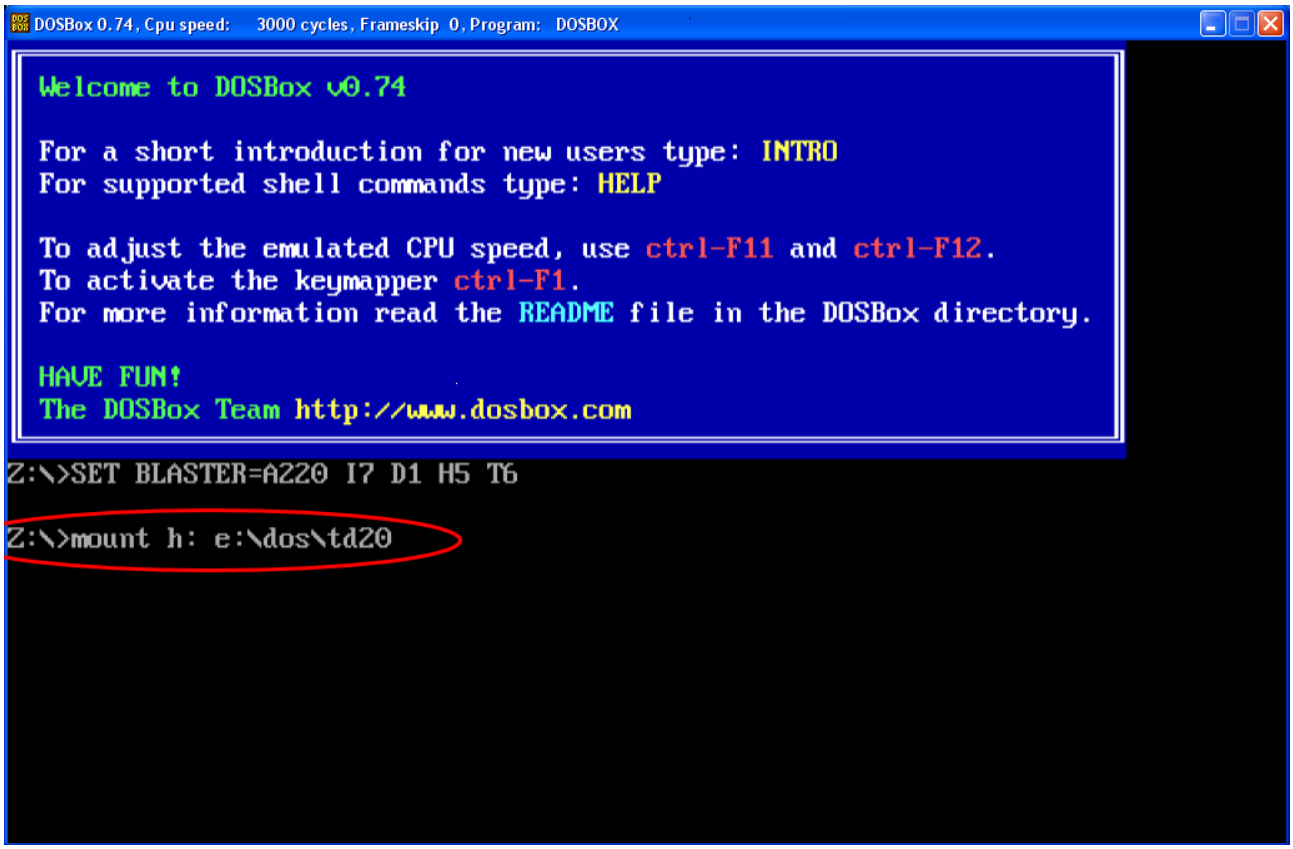
Il programma permette di operare con un potente, ma nello stesso tempo intuitivo, strumento di debugging sotto il sistema operativo DOS, supportando il debug completo dei tre linguaggi principali di Borland vale a dire Pascal, C / C++ e Assembly.

Nonostante quella di debuggare programmi compilati con appositi compilatori e opzioni di compilazione sia il motivo perchè questo programma esiste, è possibile utilizzarlo anche per piccoli test estemporanei del linguaggio assembly, quale quello che andremo ad effettuare oggi.

Ci poniamo l'obbiettivo di riprodurre in modo chiaro le operazioni effettuate dall'accadere di un interrupt, ossia verificare effettivamente sulla macchina che quanto studiato in teoria accade effettivamente.

Iniziamo a dire che per questa prova abbiamo dovuto utilizzare il programma DosBox. In realtà infatti il sistema operativo del PC della LIM della classe III Inf, essendo a 64 bit, non permette più di utilizzare direttamente sul sistema operativo i programmi DOS.

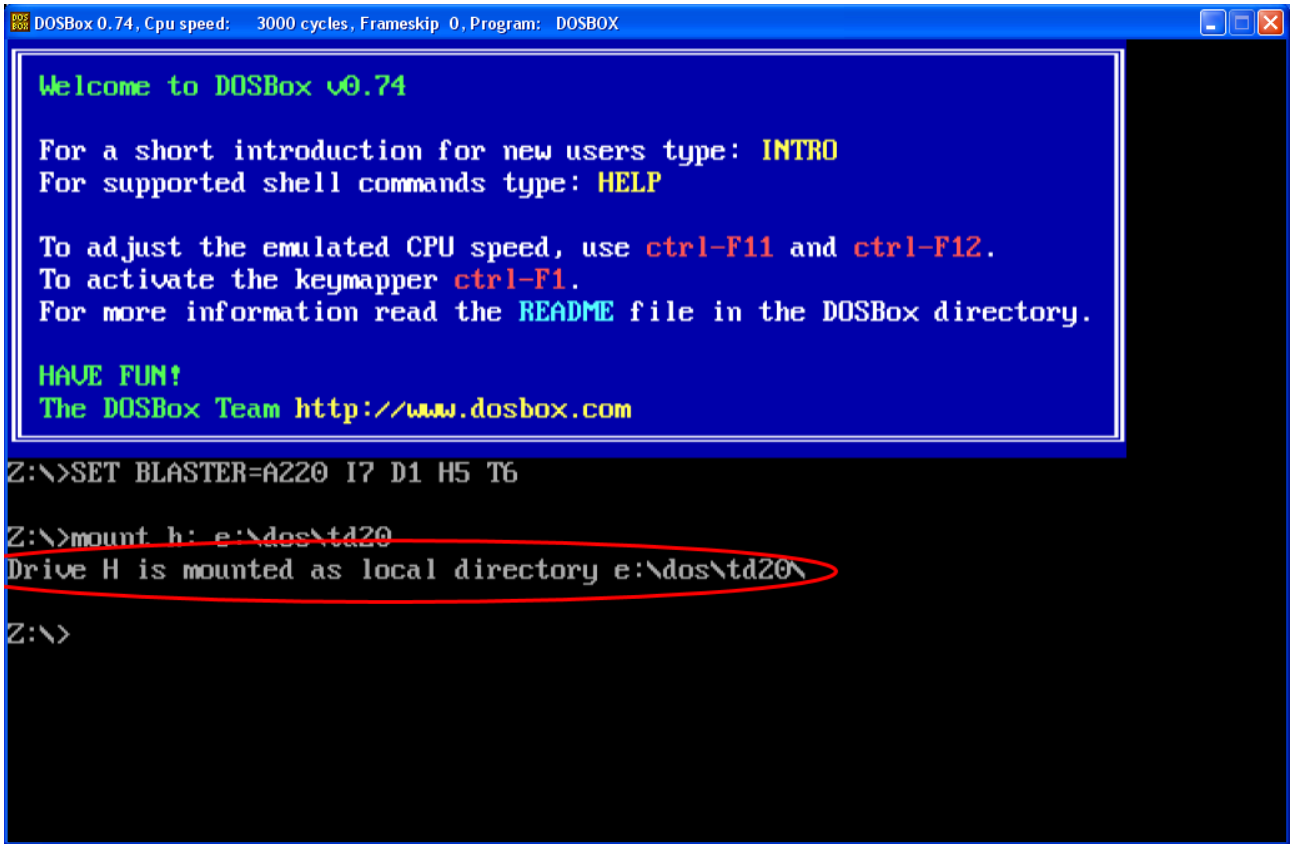
Quella sotto è la schermata iniziale del programma DosBox, un emulatore molto accurato del sistema DOS, capace di girare anche su piattaforma a 64 bit.



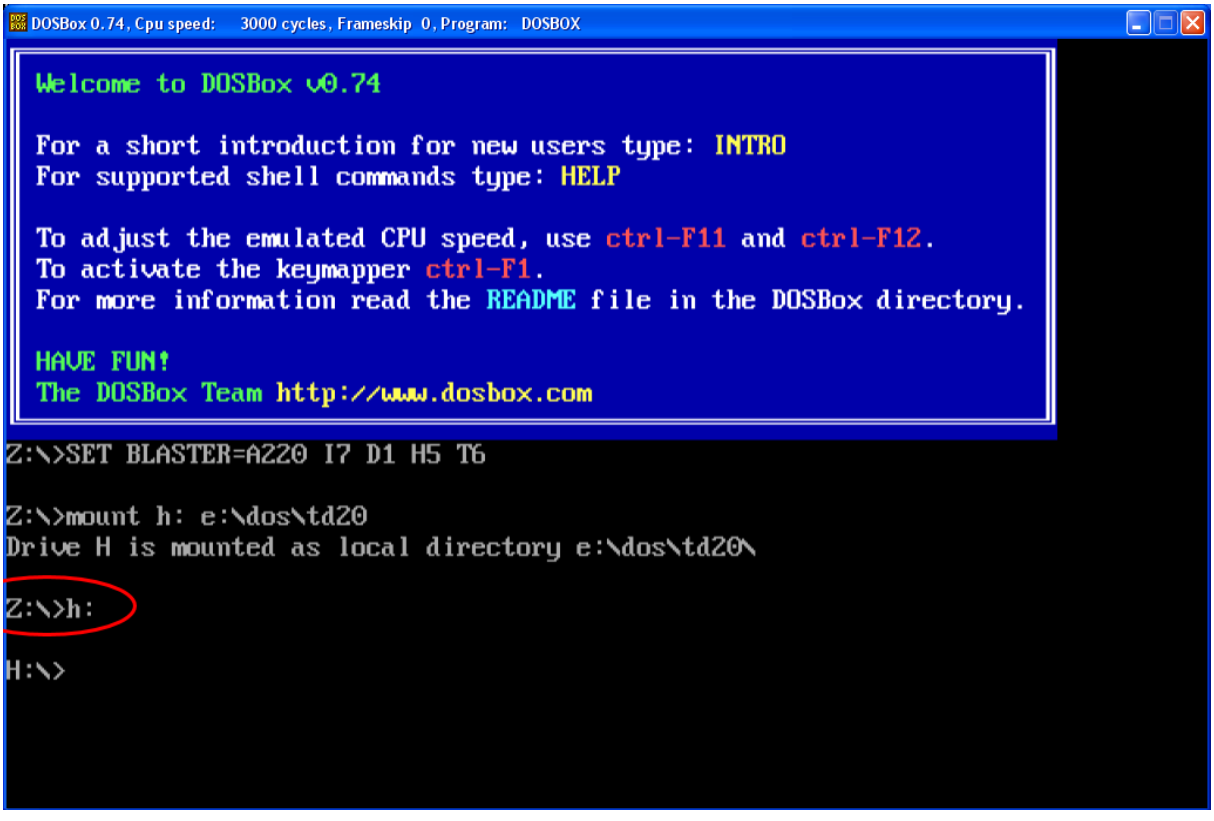
Il comando cerchiato si riferisce al montaggio su un drive DOS (virtuale), di una directory (reale, ossia presente sulla macchina su cui gira DosBox), per pemrettere a DosBox stesso

di vederla (di default nessuna risorsa sul computer ospite è visibile).

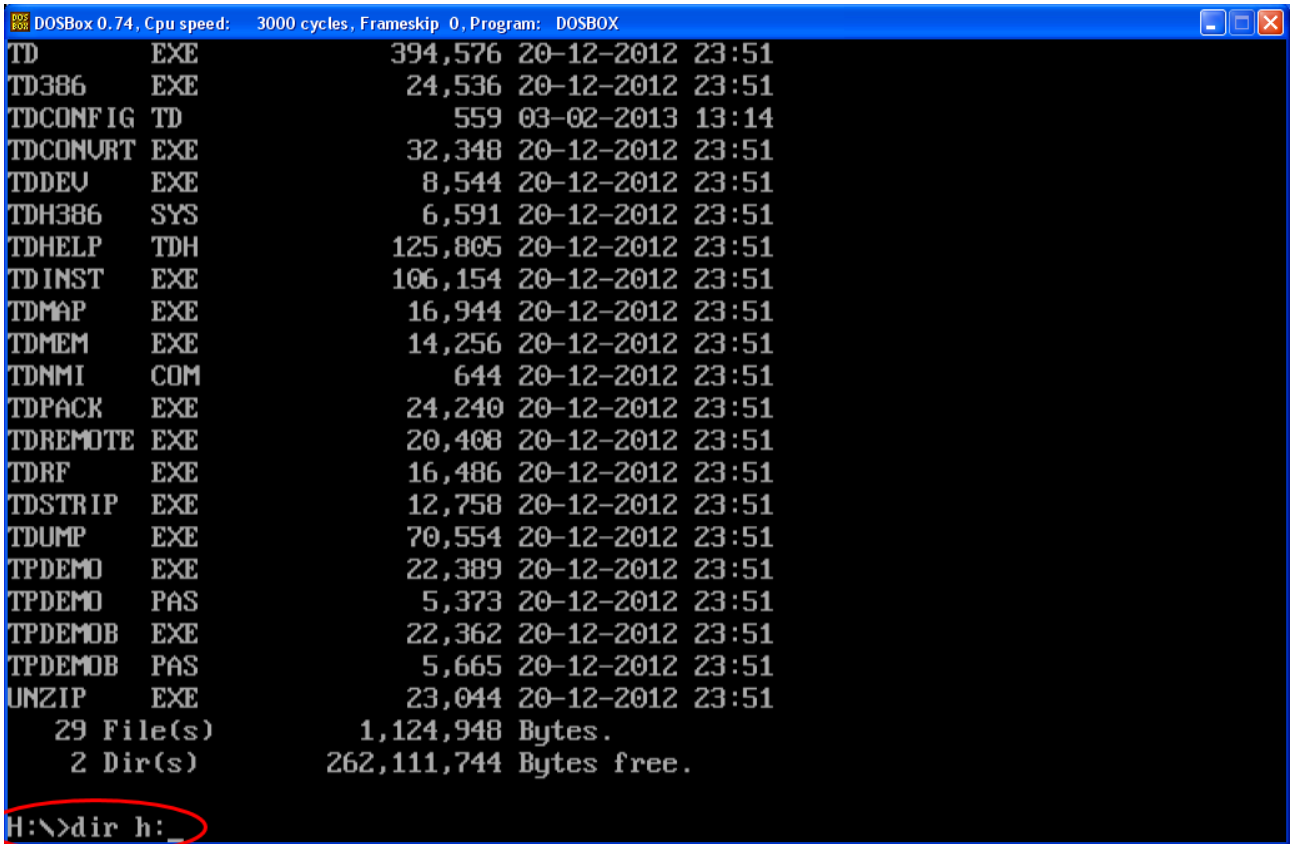
Per confermare l'avvenuto, corretto, montaggio della direcotry indicata, il programma risponde con il messaggio cerchiato nell'immagine sottostante:



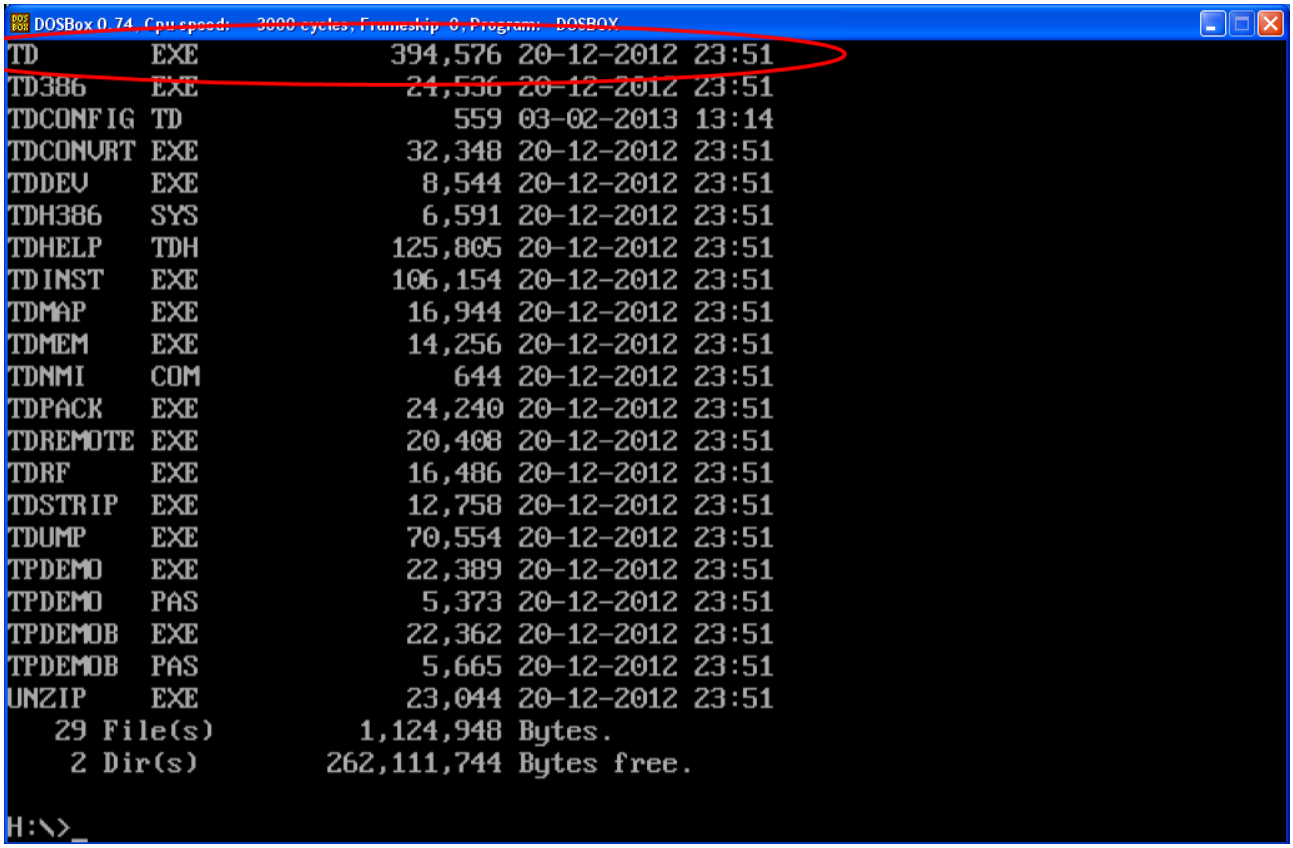
a questo punto, come in un normale sistema DOS, siccome esiste il drive h: è possibile selezionarlo, indicandolo, come riportato nella figura qui sotto:

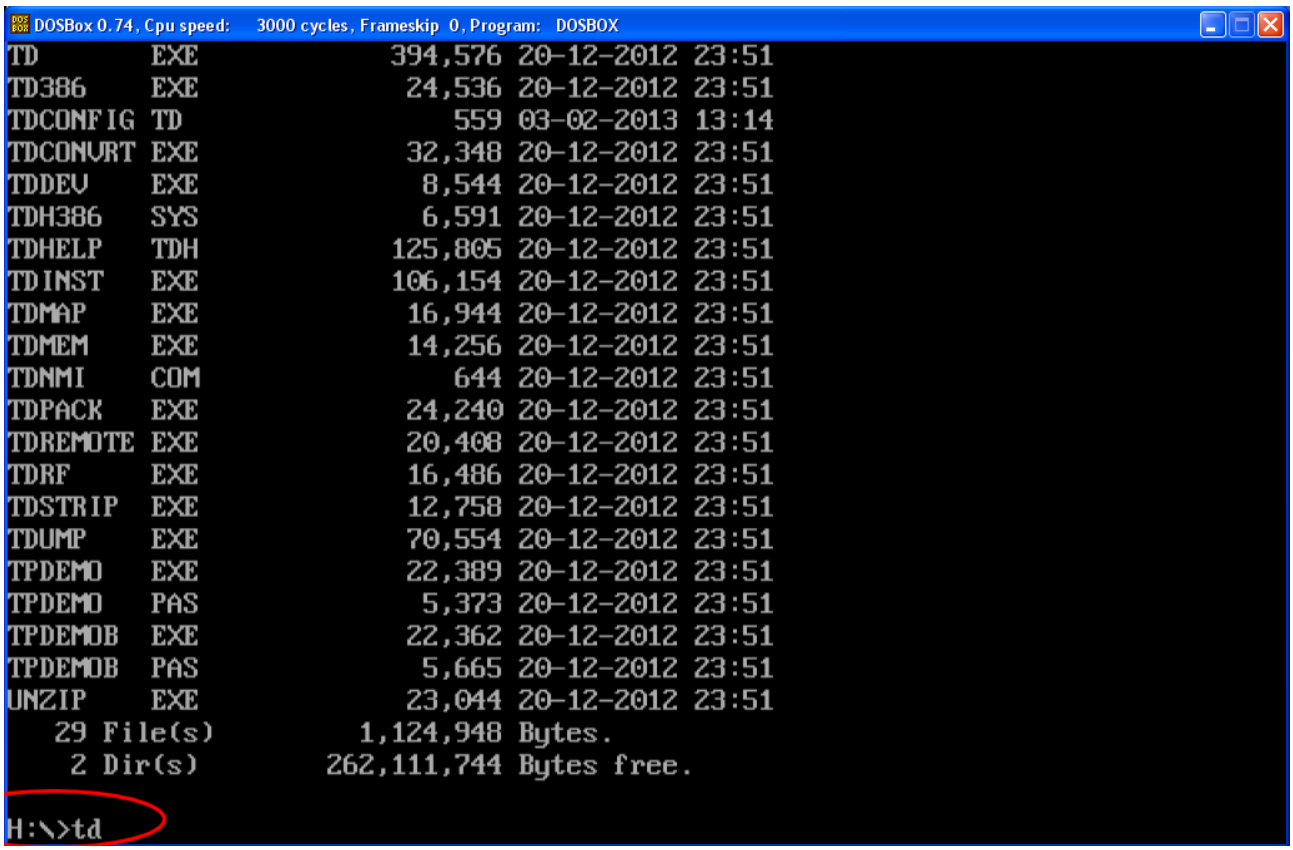


in seguito si può operare su tutti i file della directory puntata da h: in DosBox, ovviamente anche eseguendoli (semprechè si tratti di eseguibili DOS a 16 bit).

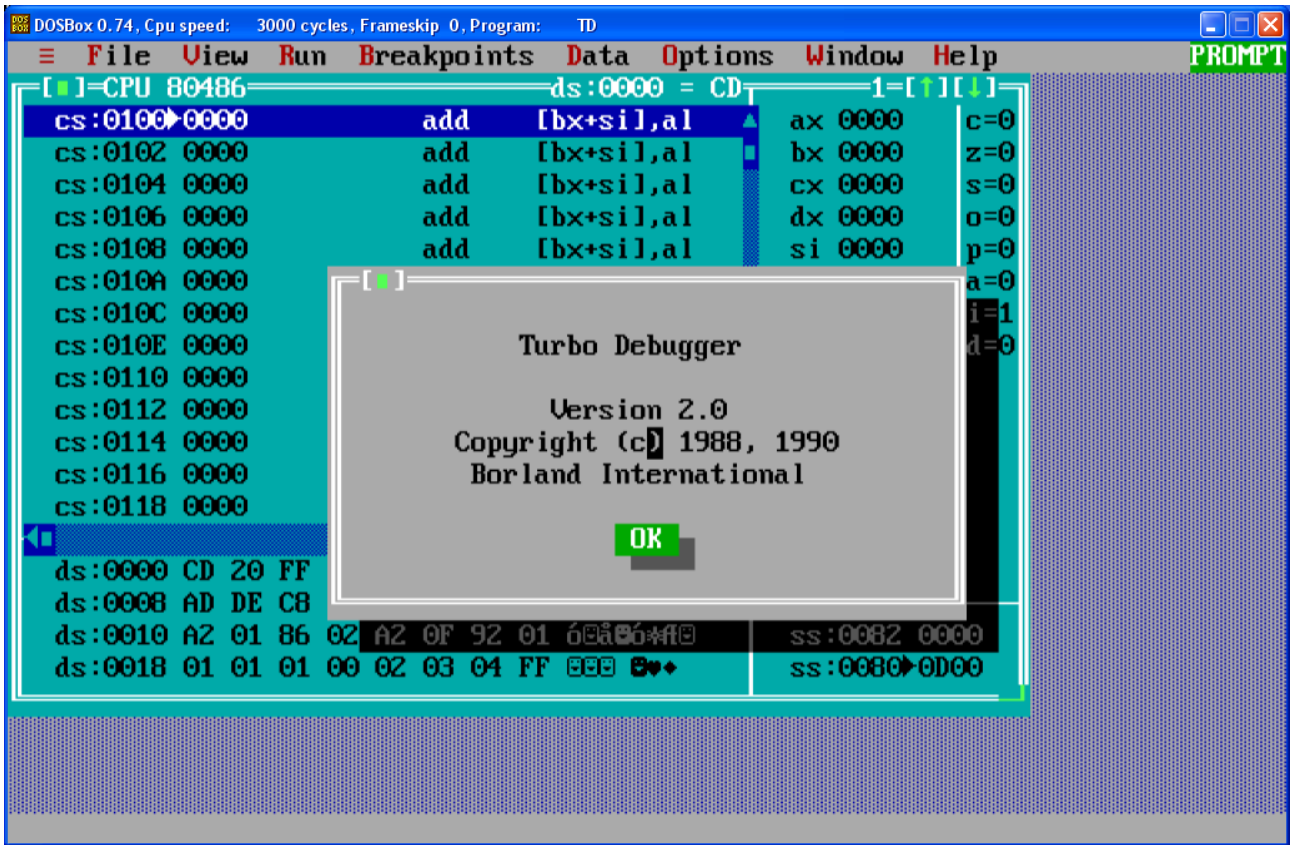


A questo punto siccome abbiamo montato proprio la directory contenente il programma Turbo Debuger, ecco che è possibile lanciare il programma stesso, avviando il file td.exe:



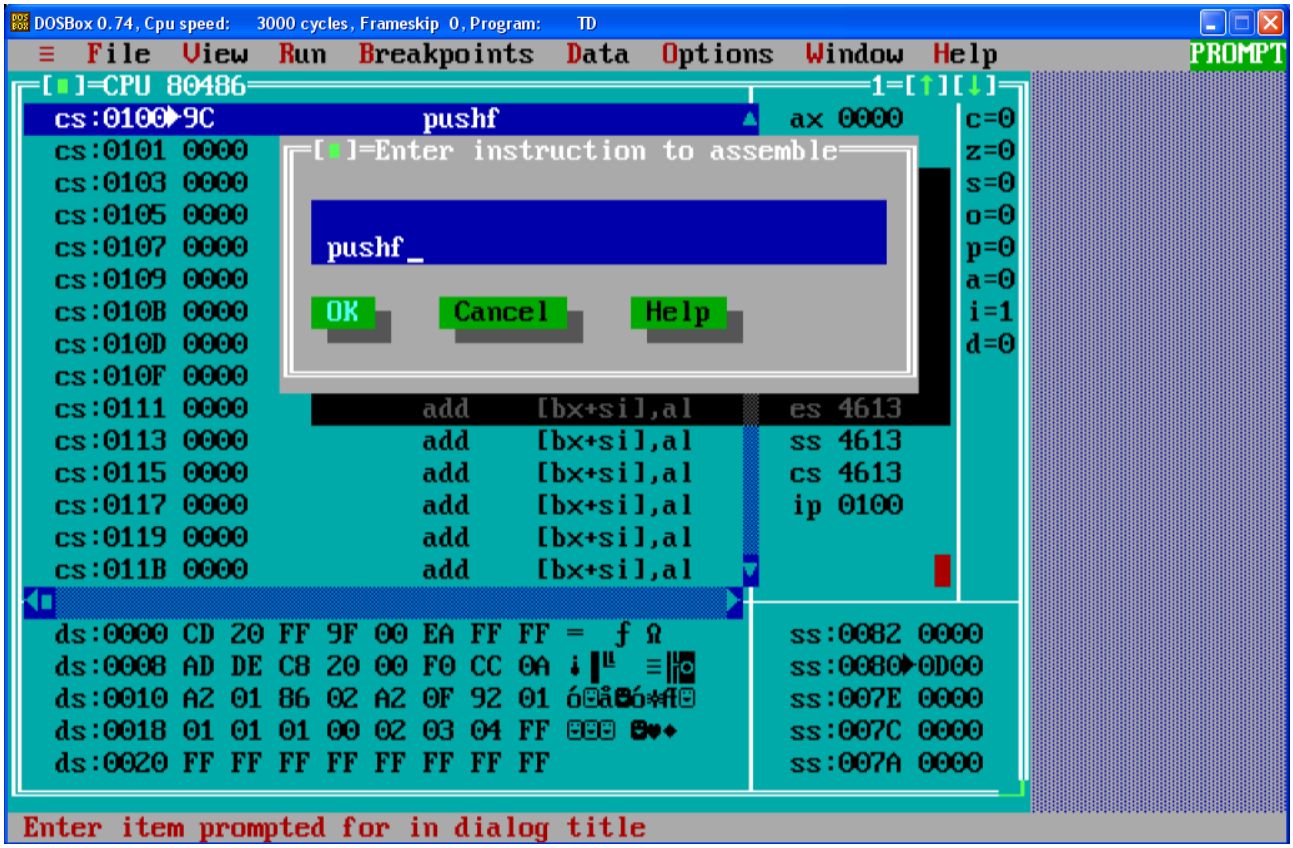


dato il comando e confermato appare la schermata iniziale del programma che è tipicamente basata su un impostazione semigrafica (grafica formata da caratteri):



dobbiamo ora assemblare alcune istruzioni per il nostro test. Questo è possibile perchè

Turbo Debugger possiede anche una sezione che permette di assemblare al volo istruzioni assembly:



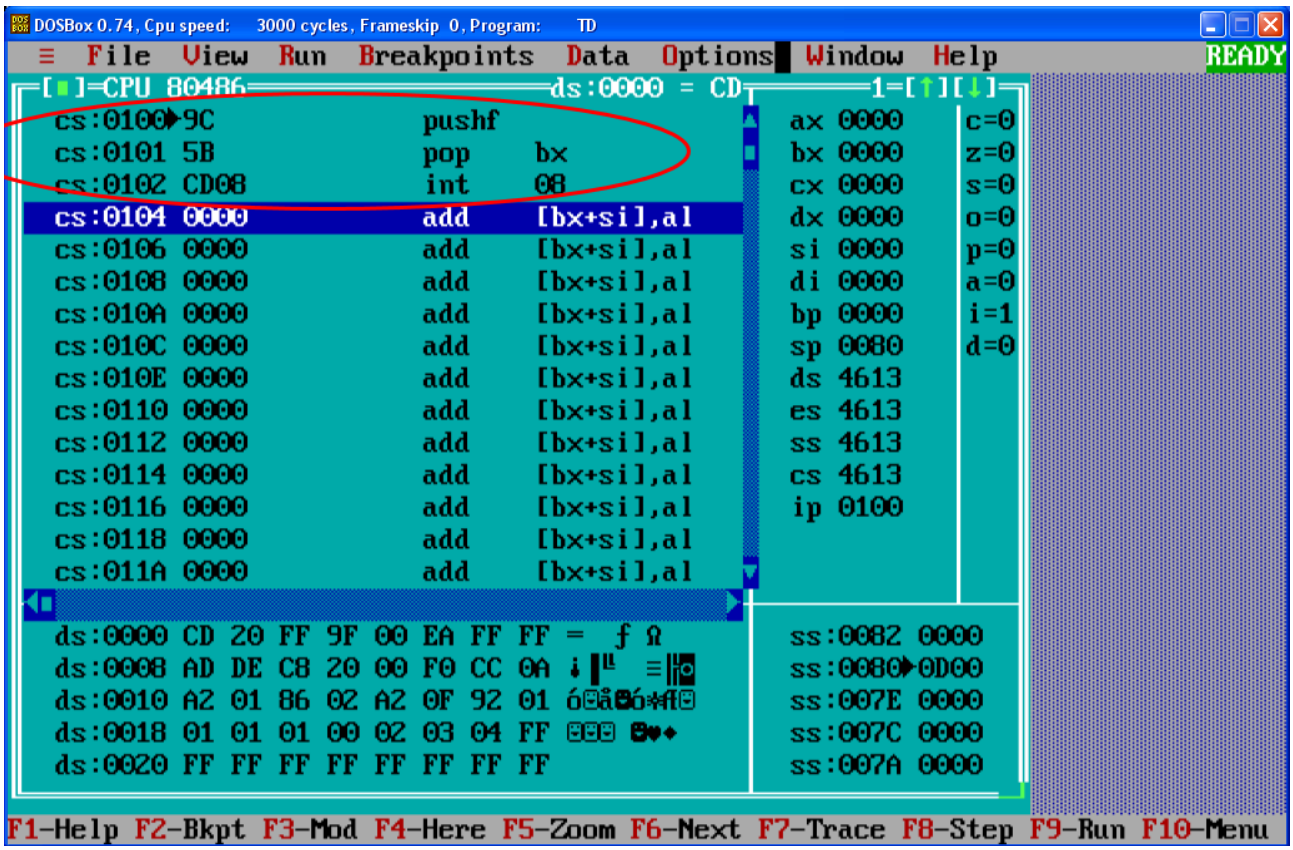
per far ciò basta posizionarsi nella apposita finestra delle istruzioni (riquadro grande in alto a sinistra della finestra generale detta della CPU). Posizionatici su una linea qualunque, o la linea voluta, iniziare a scrivere il nome dell'istruzione da assemblare. Nel nostro caso le istruzioni che andremo ad assemblare sono 3. Due per rilevare il valore del registro flag e una è l'istruzione di interrupt (software) vera e propria. Si ricorda infatti che all'accadere dell'interrupt la CPU inizia una procedura che effettua il salvataggio del registro flag e poi successivamente dei registri IP e CS.

Siccome rilevare il contenuto preciso (come valore) del registro flag è piuttosto complesso, è più comodo salvare lo stesso nello stack tramite l'apposita istruzione PUSHF e subito dopo estrarlo tramite una istruzione POP BX, che permetta il suo immagazzinamento in uno dei registri di uso generale. In questo modo il valore dei flag al momento dell'interrupt è noto e visibile.

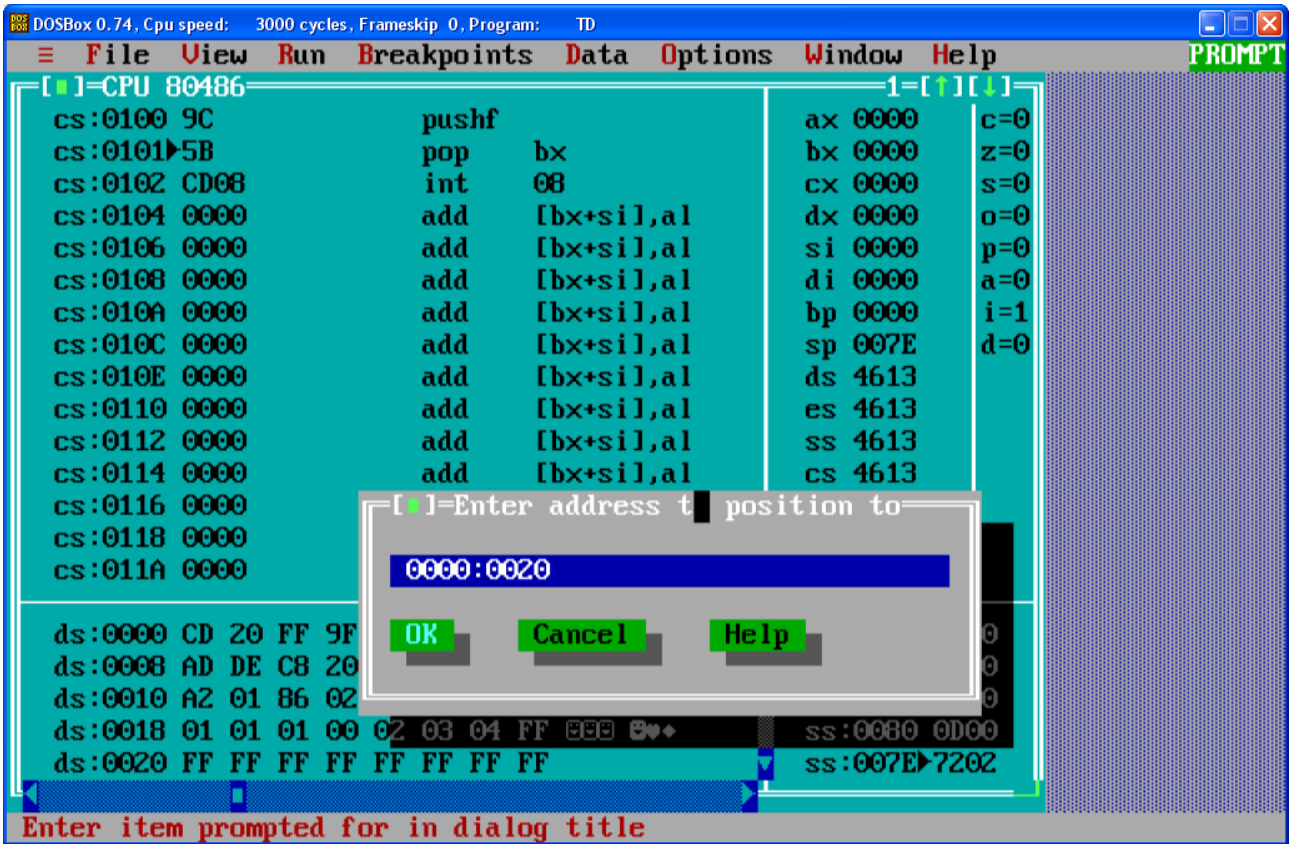
Successivamente si inserisce l'istruzione di interrupt vera e proria quale ad esempio INT 8h.

Quindi il nostro programmino assembly diventa:

```
PUSHF
POP BX
INT 8h
```

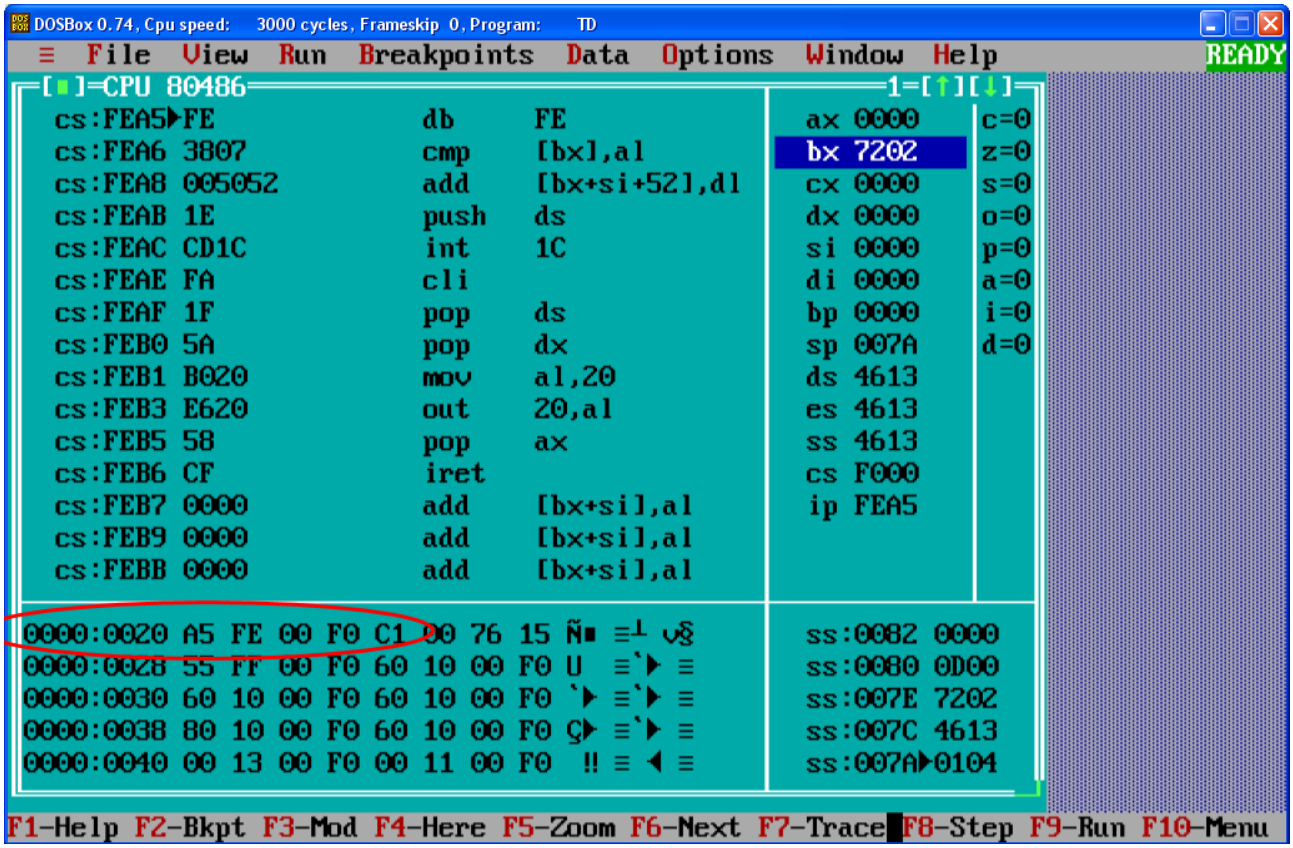


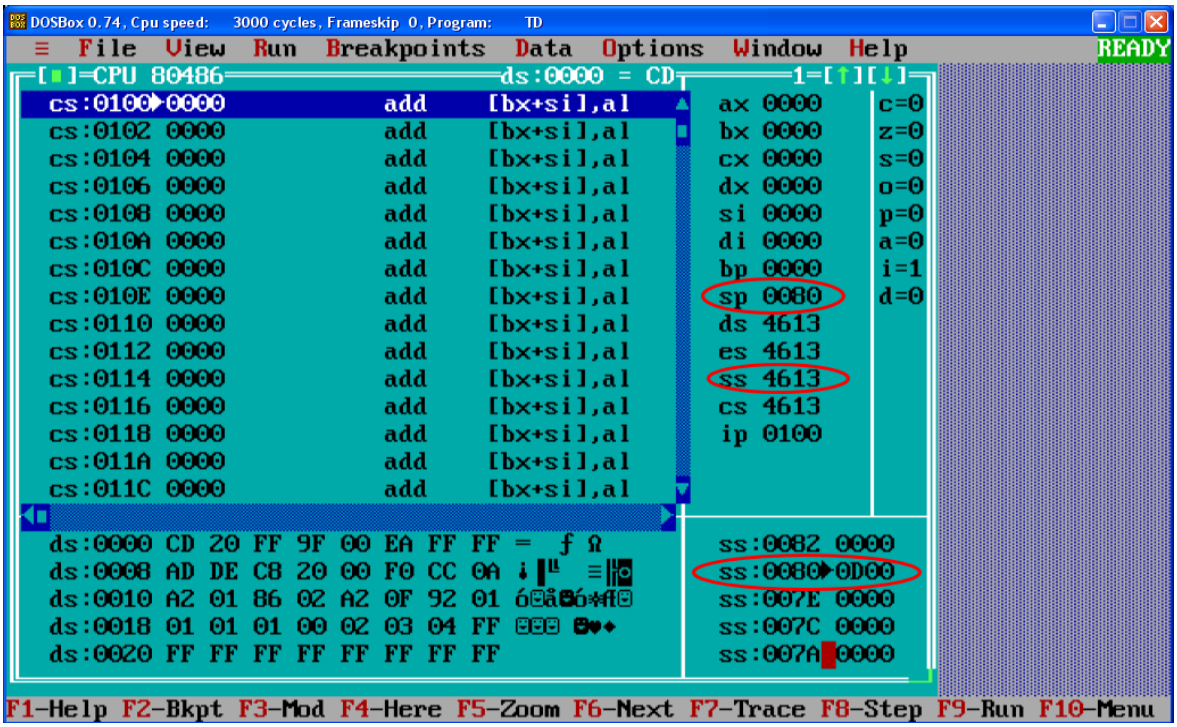
Per far funzionare tale programma dovremo poi spostare il punto di esecuzione (CS:IP) sull'inizio di tale programma, ossia il cursore a freccia dovrà essere affiancato alla prima istruzione del nostro programma.



Successivamente dovremo capire in quale posizione andrà a saltare la routine di interrupt,

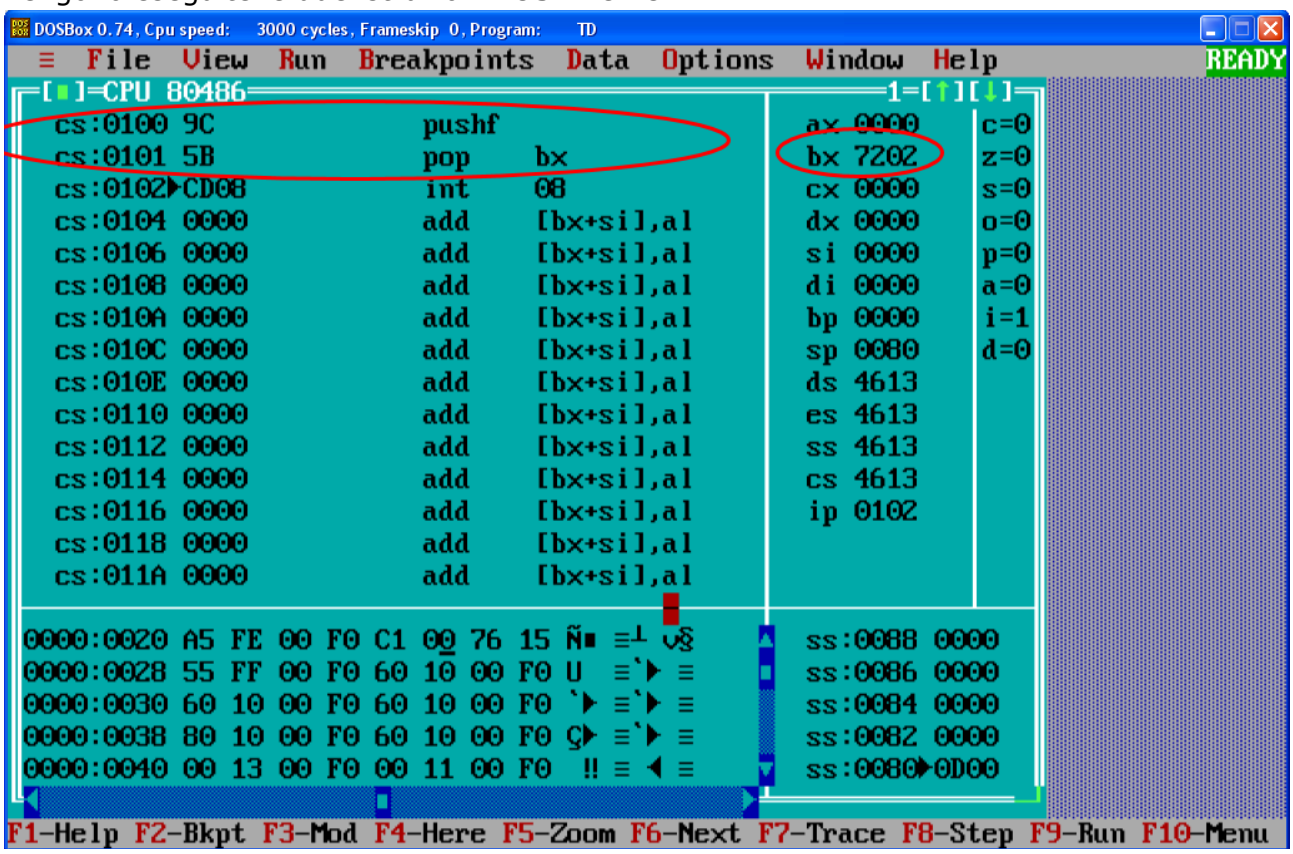
ossia visionare l'opportuna cella del vettore (che si ricorda è nella parte più alta della memoria). Per far ciò, nel nostro caso dovremo consultare il contenuto delle 4 locazioni seguenti a 0000:0020, considerato che 8 x 4 fa 32 e che 32 decimale corrisponde a 20 esadecimale. Tramite la finestra di dump (quella sottostante a quella di codice) andiamo ad analizzare il contenuto delle locazioni indicate, utilizzando un menù (GoTo..) attivabile con tasto destro del mouse che permette di spostarsi all'interno della memoria RAM (vedi figura pagina precedente).



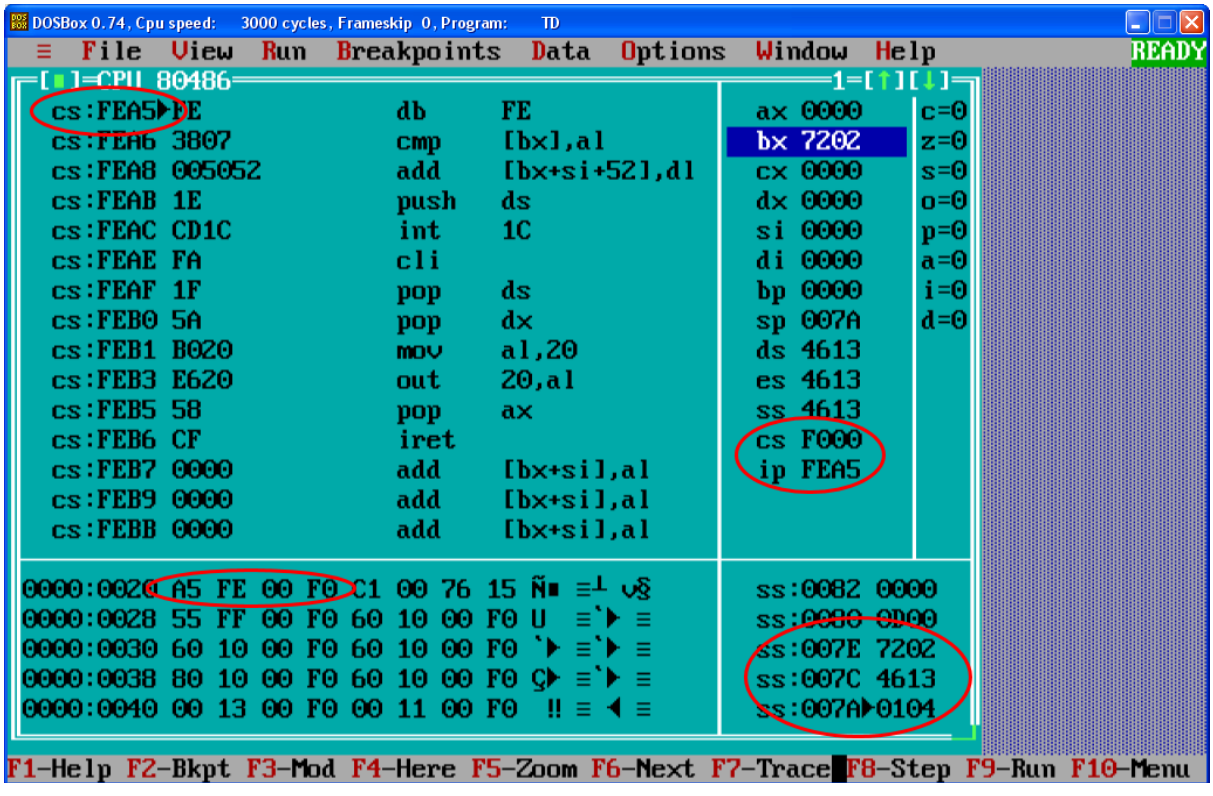


Ovviamente si deve anche prendere nota della situazione dello stack prima della esecuzione dell'interrupt, per poterne poi apprezzare la differenza.

Successivamente si dà il via al programma, premendo due volte F8. In questo modo vengono eseguite le due istruzioni PUSHF e POP BX.

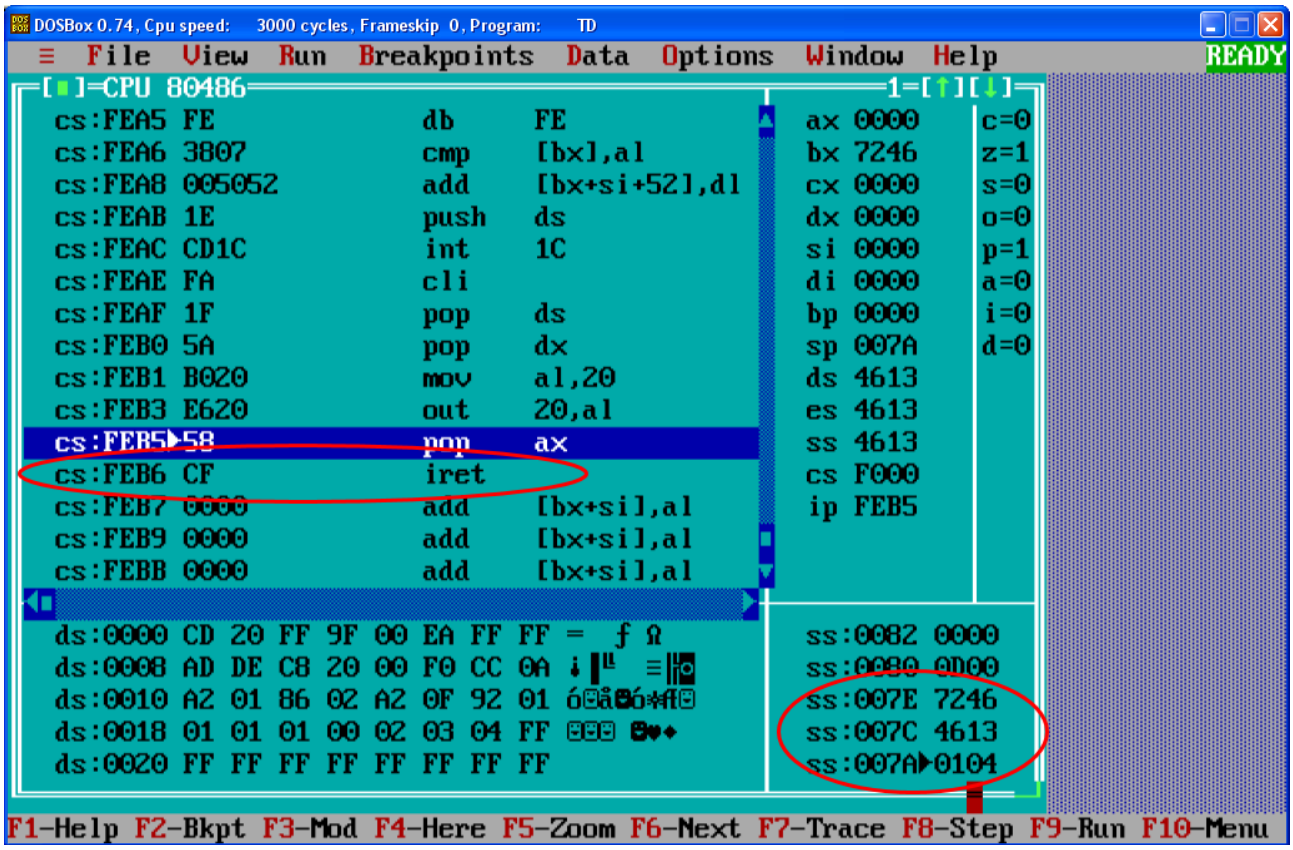


Il cursore del punto di esecuzione ovviamente si muoverà. Arrivati all'istruzione INT 8h invece è necessario eseguirla con un comando speciale, vale a dire Run / Istruction Trace:



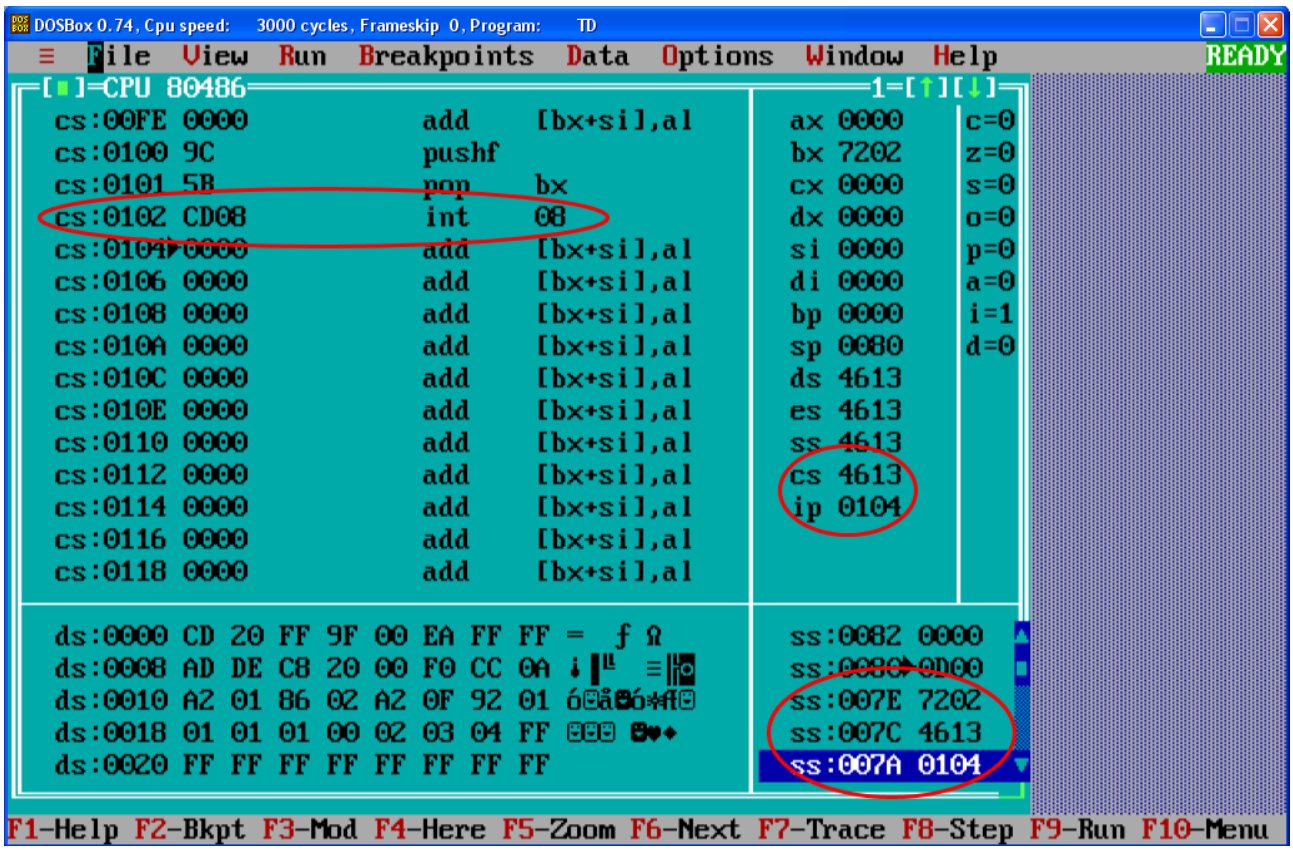
come visibile nella figura qui sopra il controllo di esecuzione salta allora improvvisamente ad un punto del tutto diverso pari a quello indicato nel vettore di interrupt alla locazione 0000: 0020 e seguenti. Ovviamente i valori di CS e IP cambiano anch'essi e tre nuovi valori a 16 bit sono inseriti in modo trasperente nello stack, il cui puntatore (registro stack pointer – SP), si muove verso l'alto della memoria (nell'interfaccia verso il basso, in quanto l'ordine convenzionale è invertito).

A questo punto viene eseguito il codice della routine di interrupt e questa routine si interrompe solo quando l'esecuzione raggiunge l'istruzione IRET preposta a terminare la routine di interrupt. A questo punto sono riposizionati i valori salvati all'inizio della chiamata ad interrupt, ed ovviamente il controllo ritorna al punto di chiamata.



Come infatti mostra la figura ancora successiva, il punto di esecuzione è ritornato alla locazione successiva al punto della chiamata di interrupt (in questo caso la locazione con offset 0104), mentre il valore originario di CS viene ristabilito. I valori presenti nello stack non sono necessariamente cancellati, ma semplicemente essi sono esclusi dai contenuti correnti dello stack, siccome lo stack pointer viene spostato.

Termina così l'esecuzione della routine di interrupt, ed il programma procede lungo il flusso logico precedentemente abbandonato.


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TD
File View Run Breakpoints Data Options Window Help
READY
CPU 80486
cs:00FE 0000 add [bx+si],al
cs:0100 9C pushf
cs:0101 5B pop bx
cs:0102 CD08 int 08
cs:0104 0000 add [bx+si],al
cs:0106 0000 add [bx+si],al
cs:0108 0000 add [bx+si],al
cs:010A 0000 add [bx+si],al
cs:010C 0000 add [bx+si],al
cs:010E 0000 add [bx+si],al
cs:0110 0000 add [bx+si],al
cs:0112 0000 add [bx+si],al
cs:0114 0000 add [bx+si],al
cs:0116 0000 add [bx+si],al
cs:0118 0000 add [bx+si],al
ax 0000
bx 7202
cx 0000
dx 0000
si 0000
di 0000
bp 0000
sp 0080
ds 4613
es 4613
ss 4613
cs 4613
ip 0104
c=0
z=0
s=0
o=0
p=0
a=0
i=1
d=0
ds:0000 CD 20 FF 9F 00 EA FF FF
ds:0008 AD DE C8 20 00 F0 CC 0A
ds:0010 A2 01 86 02 A2 0F 92 01
ds:0018 01 01 01 00 02 03 04 FF
ds:0020 FF FF FF FF FF FF FF FF
ss:0082 0000
ss:007E 7202
ss:007C 4613
ss:007A 0104
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu