Classe 3 A Inf - Alcune informazioni sull'uso del programma Debug

Un po' di storia

Debug è un programma che da lunghissimo tempo è presente nei sistemi Microsoft. Fin dall'epoca dei primi dischetti DOS, il programma debug è sempre stato presente nei dischi del sistemo operativo, facendo quindi parte di esso come applicativo / utility complementare.

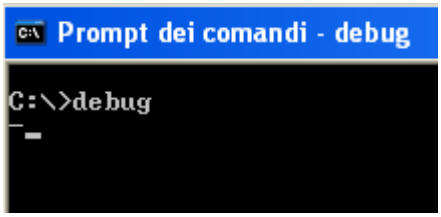
Il programma è una utility piuttosto rudimentale per assemblare, disassemblare semplici programmi, visionare parti della memoria RAM, analizzare lo stato dei registri, fare il dump (ossia la visualizzazione dettagliata dei contenuti) della meoria ed altre operazioni tutte tipiche della programmazione a basso livello (programmazione assembly). Ovviamente per programmare in assembly, esistono altri tools più completi e potenti di debug, che tra l'altro non ha avuto praticamente alcuna evoluzione dala versione di 30 anni fa. Primi fra tutti l'accoppiata Turbo Assembler + Turbo Debugger che permette di sviluppare completi programmi assembly, con strumenti professionali, seguendo (almeno nelle prime versioni di tali progammi) le convenzioni e gli standard dei programmi a 16 bit (per DOS).

Attualmente, è bene sottolinearlo, debug non funziona più in una modalità reale, ossia riportando, come accadeva sull'originale PC IBM, lo stato effettivo della CPU e della memoria, ma, ormai da anni, sotto Windows, su macchine DOS virtuali, il cui stato non ha collegamento diretto con quello della macchina reale sottostante, vale a dire il valore dei registri, i contenuti della memoria e altri dettagli non rappresentano puntualmente quelli della macchina reale che si stà utilizzando[1].

Vedremo in queste brevi note di utilizzo, alcune operazioni concrete effettuate con debug.

Uso di Debug

Il programma debug può essere lanciato dalla linea di comando di DOS (finestra prompt dei comandi) digitando il suo nome per esteso, ossia **debug**. Non appaiono a questo punto interfacce particolarmente vistose come quelle che si possono vedere in Turbo Debugger (vd. Figure sul libro di testo a pagg.), ma semplicemente il prompt (l'indicazione) che precede il cursore cambia immediatamente, dal classico prompt del DOS a un prompt con lineetta. Questo indica che siamo entrati nel programma debug e siamo pronti ad utilizzarlo.

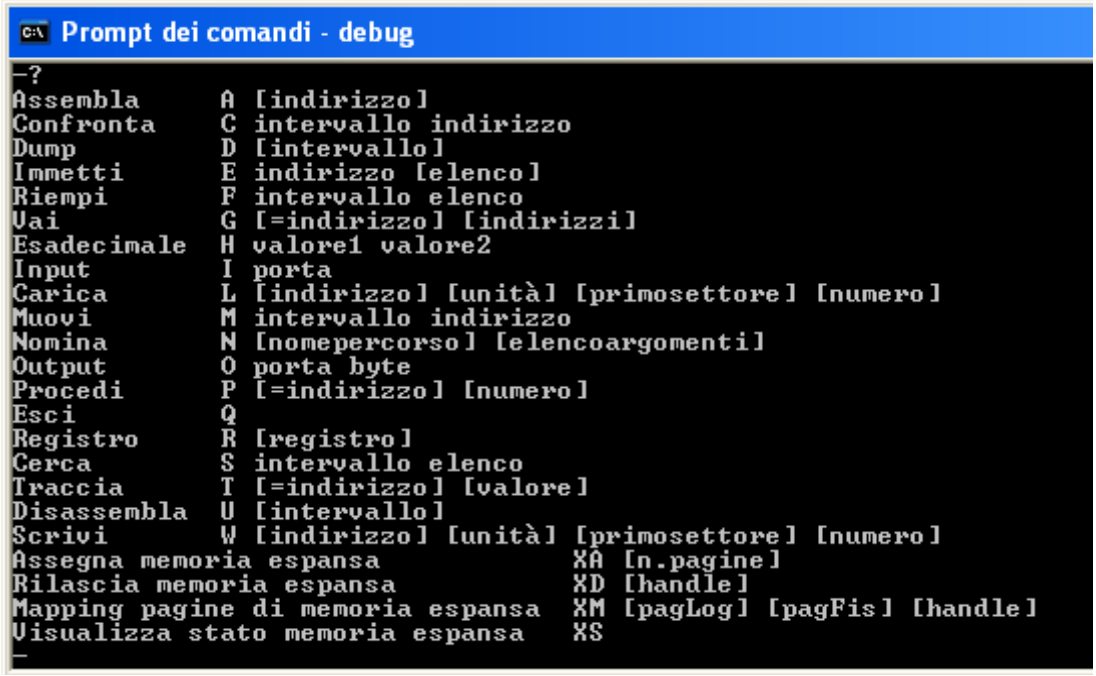


Per conoscere i comandi disponibili nel programma possiamo digitare al prompt il carattere **?;** appariranno a questo punto una lista di tutti i comandi disponibili in questo ambiente.

Precisamente a sinistra appare il nome ossia la funzione che il comando ha, subito dopo l'effettivo comando da dare al prompt per lanciare la funzionalità richiesta ed a seguire, in quasi tutti i casi, alcuni parametri spesso opzionali (ossia che possono essere inseriti o no a seconda dei casi). Iniziamo col dire che per uscire nuovamente dal programma si deve digitare la

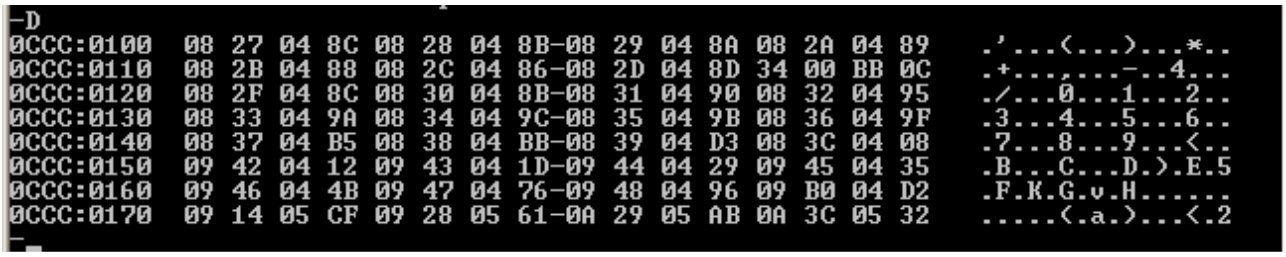
1 Si pensi a questo proposito che il massimo spazio di indirizzamento per DOS era 1 Mbyte, mentre oggi abbiamo macchine con Gigabyte di memoria reale e CPU a 64 bit. Questo rende evidente che quanto vediamo con debug non è la parte fisica della macchina.

lettera Q (debug non fa distinzione tra maiuscole e minuscole).



Per vedere una zona di memoria in forma di dati basta digitare D ed il programma effettua una operazione di dump (visualizzazione dei contenuti della memoria centrale).

La parte di memoria visualizzata per default è quella a cui punta il registro DS (data segment, ossia registro dei dati), e dalla posizione puntata dal registro IP. Di default vengono disassemblate 16 locazioni x 8 righe = 128 byte di memoria, ma volendo fare il dump di una parte più ampia di memoria, ciò è possibile inserendo dopo il d il range di indirizzi dell'offset separato da spazio, ossia



l'indicazione dell'area da stampare in questo caso, indicando la locazione di offset di inizio stampa e quella di fine.

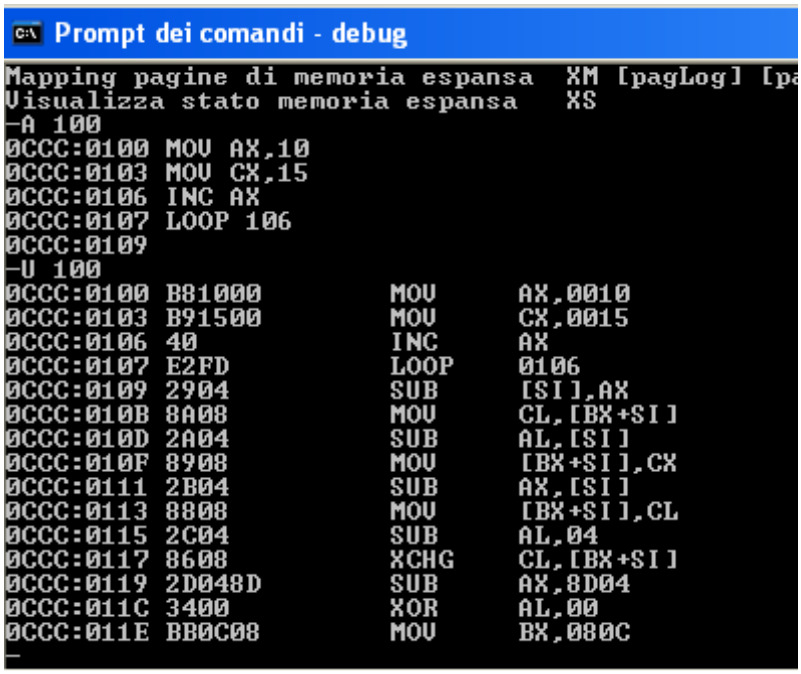
I valori indicati nel dump, per motivi di leggibilità e brevità sono espressi sempre con coppie di cifre esadecimali (2 cifre esadecimali = 1 byte = 8 bit). All'estremo lato destro possiamo trovare i caratteri ASCII corrispondenti. Nel caso il codice all'interno della memoria non sia tale da produrre caratteri, si può evitare questo problema, sarà indicato il simbolo di punto (.) che indica la presenza di tali caratteri speciali. Uno dei vantaggi di questo ulteriore parte per analizzare, è quello di poterla utilizzare per individuare frasi già presenti in memoria o rese tali, in pratica se esiste una stringa intellegibile inserita in memoria.

Passiamo ora a vedere due istruzioni estremamente utili per le operazioni di assemblazione, vale a

dire appunto il comando *assembla* (ossia il comando che permette di passare da codici mnemonici delle istruzioni assembly, ossia dall'assembly al linguaggio macchina), esso viene lanciato dal comando A. Nel caso si voglia assemblare da una certa locazione di memoria basta indicarla subito dopo il comando A (ad es. A 300). Il segmento a cui fa riferimento questo comando è quello di codice (CS), in quanto si stà lavorando con del codice di programma.

Dopo aver dato il comando appare l'indirizzo (indirizzo di segmento + offset) all'inizio del quale sarà assemblata l'istruzione. L'utente dovrà allora inserire una linea di assembly valida, ossia con regole sintattiche accettabili da debug. Se la sintassi sarà errata il programma darà una indicazione di errore. Continuerà a venire proposta la stessa locazione per ritentare un inserimento di codice macchina assemblato.

Per indicare di voler interrompere il processo di assemblazione, nel senso che si sono inserite le istruzioni volute, si deve dare semplicemente Invio senza scrivere nulla. A quel punto viene interrotta la richiesta l'input delle istruzioni assembly. E' importante anche notare che l'assemblare programmi in questo modo comporta alcune difficoltà e svantaggi. Per primo non avremo con questo programma la possibilità di definire vere e proprie etichette e quindi i riferimenti per i salti saranno direttamente gli indirizzi di offset o segmento e offset per i salti cosiddetti "lunghi". Secondariamente l'inserire istruzioni in un punto di memoria sovrappone i dati assemblati in precedenza e quindi questo comporta che non è putroppo possibile fare facilmente ed in modo indolore correzioni al codice digitato[2]. In pratica usalmente è necessario avere in mano tutto il codice assebmly che si vuole digitare e inserirlo in modo univoco e progressivo. In caso contrario le nuove istruzioni si sovrapporranno a quelle vecchie, ossia a codice già precedentemente assemblato. In ogni caso per programmi con qualche decina di istruzioni anche un tool così rudimentale può essere funzionale e adatto.



A fianco della istruzione da assemblare viene continuamente indicata la locazione dalla quale si sta andando ad inserire il codice assembly indicato.

Ora che avremo assemblato un piccolo programma assembler, potremmo decidere di rivedere il suo

2 Tutto questo non accade con i veri e propri programmi assemblatori in quanto essi ricostruiscono totalmente il programma ogni volta che esso viene assemblato, mentre debug opera purtroppo solo in modalità interattiva, ossia localmente.

codice. Questo può essere fatto solo con il compando unassembly, ossia disassembla.
Per lanciare questo comando si preme U eventualmente seguito dalla indicazione dell' offset. Da notare che il processo in questo caso è del tutto automatico. Se l'inserimento di istruzioni infatti presume un pensiero logico, il leggerle già scritte e trasfromare il codice binario in assembly può essere fatto solo tramite un automatismo. Indichiamo nell'immagine sopra un esempio di questi procedimenti. Nella schermata sopra possiamo notare sulla estrema sinistra una serie di indirizzi di memoria suddivisi in segmento (prima dei due punti) e offset (dopo i due punti). Seguono su ogni linea[3] i codici operativi e gli eventuali operandi. Nello specifico:
MOV AX, 10 ha codice operativo B8 (relativo all'operazione movimento (MOV) AX, <costante>), e operando di 2 byte[4], in questo caso costituito da 10 00.
Successivamente l'istruzione **MOV BX, 15** che in modo simile ha una sua codifica per il codice operativo (ossia il codice che identifica univocamente il tipo di istruzione[5]). Esso risulta (dalla figura sopra), B5 con operandi al seguito 15 00, derivati dal valore 15 della costante da caricare nel registro BX.
INC AX; è un istruzione che non ha operandi variabili, ma tutto è fissato sia operazione che registro sulla quale si svolge. Quindi essa è codificata con il solo codice operativo vale a dire 40.
Ovviamente l'istruzione INC BX o INC CX[6] (che esistono entrambe) hanno codici operativi diversi.

Successivamente si pone il problema di far funzionare il programma, ma soprattutto di farlo funzionare di modo che sia possibile verificare la sua correttezza in un procedimento passo-passo.
In questo caso il comando di fondamentale importanza è il comando trace (o traccia nella versione italiana del programma). Tale comando permette di vedere l'evoluzione del valore dei registri via via che viene eseguito il programma con interruzioni ad ogni istruzione. Si noti la sintassi

T [= indirizzo [valore]]

Per attivare trace in questa modalità bisogna dare il comando:
T = 100 (nel nostro caso avendo assemblato dall'indirizzo 100). Successivamente dare ripetuti comandi T ai passi successivi. Volendo eseguire n singole istruzioni in modo tracciato si può anche dare il comando:

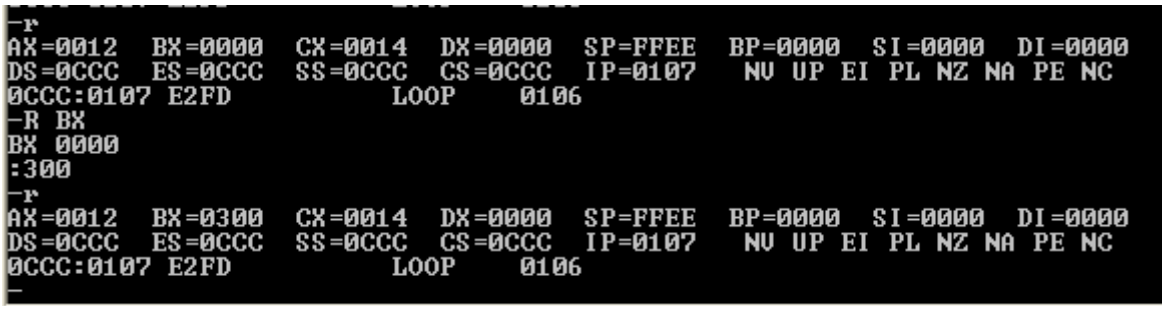
T = 100 5

che farà eseguire partendo dal codice della locazione CS:100, il trace di 5 istruzioni.
Infine è importante poter visualizzare i registri in qualunque momento essi servano. Questo è

3 Si noti che le linee di nostro interesse non sono tutte quelle visualizzate, ma solo quelle dalla locazione con offset 100 a quella con offset 107 (o meglio 108, dato che il codice dell'istruzione LOOP in questo caso è costituito da due byte)
4 Come al solito gli operandi di più byte appaiono nella codifica in posizione invertita rispetto alla normale lettura. Questa è una regola generale della codifica macchina delle CPU.
5 Seppur si tratta sempre di un movimento di dati, muovere una costante in AX o in BX non è la stessa esatta operazione, così come è diverso muovere un dato da un registro ad un altro. Per questo l'istruzione MOV possiede molti codici operativi diversi.
6 Si noti che il risultato di queste istruzioni ritorna non all'accumulatore AX, ma nel registro stesso su cui si opera ossia si ha BX = BX + 1 o CX = CX +1.

sempre possibile tramite il comando R, che mostra tutti i registri. E' possibile, in caso vi sia bisogno visualizzare i singoli registri dare invece il comando R nome registro

In questo modo viene visualizzato solo il valore del registro richiesto assieme alla possibilità di alterarne il valore, cosa assai importante.



Si può notare come la modifica al registro BX, poi sia effettivamente stata effettuata, nella successiva visualizzazione tramite il comando R. Il comando R <registro>, visualizza quindi il valore corrente del registro e successivamente da modo di inserire un valore diverso. Si deve premere Invio (Enter nelle tastiere americane) per confermare semplicemente il valore letto.

L'istruzione che si vede sotto alle indicazioni dei registri è esattamente quella a cui punta la coppia di registri CS:IP, che indicano il punto di esecuzione del programma, ossia l'istruzione correntemente da eseguire[7].

7 Tale concetto corrisponde al punto di esecuzione corrente di un programma in un linguaggio ad alto/medio livello quale C o C++.