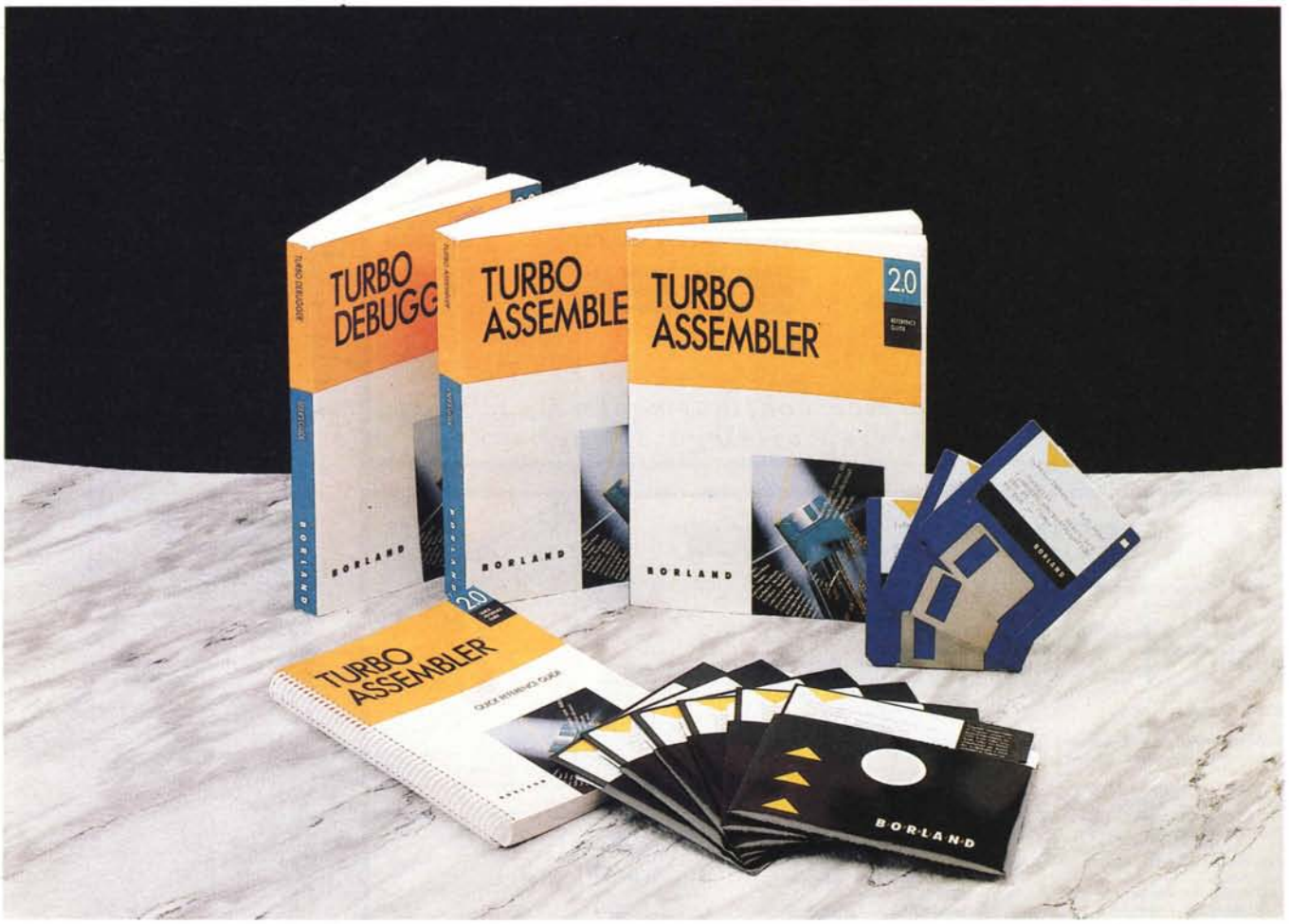
# Borland Turbo Debugger 2.0 & Assembler 2.0

di Sergio Polini

*Secondo Confucio, dare consigli è la cosa più facile del mondo; il senso di una così autorevole opinione è che facciamo meglio ad ascoltare la nostra testa piuttosto che le ugole altrui. Ma Jon Louis Bentley fa eccezione. I suoi consigli su come scrivere programmi più efficienti (Writing Efficient Programs, Prentice-Hall, 1982) non sono le solite ricette del tipo «come avere successo nella vita», ma fanno tesoro di esperienze maturate nei più importanti laboratori software del mondo, dall'IBM alla Bell; il suo libro, come racconta Jerome Feder (The Evolution of Unix System Performance, Unix System Readings and Applications, Vol. II, Prentice-Hall, 1987) è stato utilizzato anche per migliorare le prestazioni di Unix. Bentley sottolinea l'importanza di definire con cura le strutture dei dati e di mettere a punto algoritmi efficaci, avvalendosi di un linguaggio di alto livello; durante questa fase, si deve*



*in primo luogo badare a che il programma funzioni correttamente, ma poi si deve passare ad esaminarne le prestazioni globali; se queste non sono soddisfacenti, non si può ottimizzare «tutto» né ci si deve fidare del proprio intuito circa le parti del programma su cui può convenire intervenire: occorre un «profiling» che ci dica quali istruzioni del programma vengono eseguite più spesso e quanto tempo viene speso in ogni procedura. Individuati così i segmenti di codice su cui lavorare, si possono provare diverse strutture di dati e algoritmi, si possono riscrivere in modo più efficiente alcune parti; se ciò ancora non basta, si può infine fare qua e là ricorso all'Assembler.*

*Come avremo modo di verificare quando esamineremo il Turbo Profiler, la Borland ha fatto propri tali consigli.*

*Soprattutto ha messo a disposizione dei propri utenti i «tools» necessari per metterli in atto. Abbiamo esaminato lo scorso anno le prime versioni del Turbo Debugger e del Turbo Assembler, di cui ora abbiamo le versioni 2.0; in un prossimo numero proveremo il nuovissimo Turbo Profiler 1.0. Il Debugger, completo ed eccellente fin dalla sua prima uscita, viene ora arricchito di una più moderna interfaccia utente, analoga a quella del Turbo C++ 1.0 visto a luglio, nonché di meccanismi per rendere ancora più comoda la caccia al bug (come la «esecuzione a ritroso») e per estenderla a situazioni fino ad ora un po' disperate, come il debugging di un programma residente o di un device driver.*

*Il Profiler, a differenza di tanti altri prodotti similari, offre non solo un comodissimo ambiente interattivo, nel tradizionale stile Borland, ma soprattutto quella possibilità di tener conto del numero di volte che viene eseguita ogni singola istruzione che, necessaria per un adeguato profiling, risulta utilissima anche per il debugging (una istruzione eseguita zero volte è una istruzione non «testata»). L'Assembler, pur non tradendo l'impostazione originaria, è stato ora dotato della possibilità di produrre codice più efficiente mediante un'opzione che lo trasforma in «multi-pass»; viene invece rispettata la caratteristica che già avevamo sottolineato nella prova del gennaio dello scorso anno: unico linguaggio Borland privo di un ambiente interattivo, si propone soprattutto come tool di ottimizzazione, come strumento per realizzare moduli OBJ da linkare a programmi realizzati con linguaggi di alto livello; alle originarie caratteristiche progettate a questo scopo se ne aggiungono ora altre che ne rendono ancora più comodo l'uso. Come anticipato a luglio, non sarebbe stato possibile trovare in un solo numero della rivista lo spazio necessario per illustrare tutte le ultime novità della Borland. Per quanto, quindi, il Turbo Debugger & Tools si presenti come un unico prodotto (non è possibile acquistarne separatamente i diversi componenti), ci limiteremo questo mese a mettere alla prova i due prodotti «rinnovati», l'Assembler e il Debugger.*

*Le versioni 1.0 di questi sono già state esaminate con il dovuto dettaglio, rispettivamente, nel gennaio e nel febbraio dello scorso anno; ci concentreremo quindi ora prevalentemente su quanto offrono di nuovo le versioni 2.0. La prova del Profiler vi verrà invece proposta in un prossimo numero*

**Turbo Debugger & Tools**

**Produttore:**
*Borland International, Inc.*
*1800 Green Hills Road P.O. Box 660001*
*Scotts Valley, CA 95066-0001*
**Distributore:**
*Borland Italia Srl - Via Cavalcanti, 5*
*20127 Milano - Tel. 02-2610102*
**Prezzi** *(IVA esclusa):*
*Turbo Debugger & Tools* L. 299.000
*Upgrade dal Turbo Assembler/ Debugger 1.0 o 1.5* L. 149.000

## Installazione e documentazione

Il Turbo Debugger richiede una versione DOS 2.0 o successiva e un computer IBM o compatibile con almeno 384K di RAM, come le versioni precedenti. Non è però più possibile l'installazione su macchine dotate unicamente di dischetti da 360K: occorrono o dischetti da 3½ (anche quelli da 720K) o dischetti da 5¼, ad alta densità (1,2 Mega). È comunque consigliabile il disco rigido. La confezione comprende anche due versioni adatte a sfruttare le caratteristiche di configurazioni hardware avanzate: su macchine con un 80286 o 80386 e almeno 640K di memoria estesa è possibile usare TD286, ovvero un debugger che, ponendo il processore in modo protetto e riservando a sé la memoria estesa, lascia più spazio per il programma su cui lavorare (nella memoria normale risiede unicamente un «loader» di pochi Kbyte); su macchine con un 80386 e almeno 640K di memoria estesa è peraltro possibile usare TD386, un debugger che, sfruttando il modo virtuale del processore, lascia interi 640K al programma da spulciare. Per il resto, anche la versione base fa uso della memoria espansa (EMS) eventualmente presente. Il Turbo Assembler, grazie all'assenza di una sofisticata interfaccia utente, può essere comodamente usato anche su dischetti da 360K. Le procedure d'installazione, al solito, sono molto semplici e completamente guidate: vi provvedono appositi programmi INSTALL, analoghi a quelli forniti con altri prodotti Borland. L'occupazione di spazio su disco è modesta: bastano circa 500K per il Debugger (compreso il file di help) e appena 100K per l'Assembler, ai quali si possono aggiungere quelli richiesti da diversi programmi di utilità, peraltro poco ingombranti.

I manuali (dei quali è disponibile la traduzione in italiano) conservano l'elevato livello qualitativo delle versioni pre-

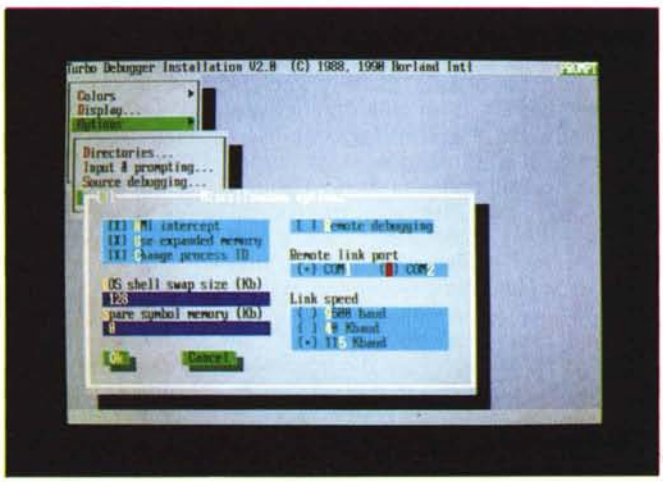



*Figura 1 — Anche il programma di installazione del debugger (TDINST) è stato dotato di una interfaccia utente analoga a quella del Turbo C++ e del Turbo Debugger (mouse, input box, check box, radio button, ecc.).*

*Figura 2 — Il Turbo Debugger consente di ripercorrere all'indietro l'esecuzione di un programma, almeno fino ad un punto da dove si sono usati solo i comandi Trace (F7) o Instruction Trace (Alt-F7). Se si usa l'opzione -k, la parte inferiore della finestra Execution history contiene la serie dei comandi dati; posizionandosi su uno di questi e scegliendo l'opzione Keystroke restore dal menu locale, il programma viene ricaricato e rieseguito, con gli stessi comandi.*

*Figura 3 — Il Turbo Debugger 1.5 comprendeva una opzione Hierarchy nel menu View, dedicata alla illustrazione della gerarchia di oggetti nei programmi Turbo Pascal 5.5. L'opzione è stata ora non solo estesa alle classi del Turbo C++ 1.0, ma arricchita della possibilità di mostrare le diverse classi «madri» di una classe C++ con multiple inheritance.*

cedenti: avevamo apprezzato a suo tempo la chiarezza e la completezza di quello del Debugger, comprendente anche efficaci sezioni didattiche circa i bug più comuni per chi programma in C, Pascal o Assembler, nonché l'estrema utilità, per chi si accosti alla programmazione in assembler, dei manuali del Turbo Assembler, ricchi di preziosi suggerimenti nonché di una guida completa alla realizzazione di moduli da linkare a programmi scritti in altri linguaggi Borland (mancano naturalmente i riferimenti agli ormai abbandonati Turbo Basic e Turbo Prolog). Oltre alle tradizionali *User's Guide* e *Reference Guide,* il Turbo Assembler è ora dotato anche di una utile *Quick Reference Guide* che, oltre a contenere in forma sintetica le informazioni del *Reference,* riporta anche una rapida guida alle istruzioni del processore e del coprocessore numerico in ordine alfabetico, con indicazioni quali i flag modificati e le diverse possibili codifiche esadecimali. Curiosamente, tuttavia, qualcosa manca e qualcos'altro è stato trasferito su disco. Solo in un file UPDATE.DOC, ad esempio, sono documentate le istruzioni e le direttive dell'80486 che ora l'Assembler riconosce; l'illustrazione di tutti i programmi di utilità che accompagnano il Debugger, come le specifiche per la realizzazione di un device driver per usare con questo un debugger hardware, si trovano invece in MANUAL.DOC, mentre un file UTIL.DOC contiene le informazioni relative ad alcuni dei programmi di utilità forniti con l'Assembler. Mistero. Speriamo solo che non sia l'indizio di una nuova tendenza.

### Esecuzione a ritroso

Il nuovo Debugger si lascia usare ora anche con il mouse. L'organizzazione generale dei menu e delle finestre è rimasta pressoché immutata, con poche varianti e alcune notevoli innovazioni. Rimane il ricco repertorio di finestre per l'esame del sorgente del programma su cui si sta lavorando, per tenere sotto controllo il valore corrente delle sue variabili ed eventualmente modificarlo, per esaminare e impostare *breakpoint,* per aprire, chiudere ed analizzare un *log* della sessione di debugging, per guardare alla sequenza di chiamate di procedura che si sono succedute (con possibilità di controllare i valori delle variabili locali ad ogni chiamata), per visualizzare le istruzioni Assembler generate dal compilatore e lo stato dei registri, per il *dump* di una qualsiasi area di memoria. Rimane anche la folta popolazione dei piccoli menu associati ad ogni finestra che, grazie sia alla loro struttura variabile in funzione della finestra su cui sono stati aperti sia alla coerenza dei comandi pur nella eterogeneità delle diverse situazioni, aiutano non poco a rendere facile e veloce l'uso di uno strumento estremamente ricco e articolato.

Rimangono anche altre caratteristiche già esaminate nel febbraio dello scorso anno, quali il potente meccanismo delle macro per rendere automatiche complesse sequenze di comandi, o la ricchezza delle opzioni per seguire passo passo l'esecuzione di un programma: una istruzione per volta o fino ad un punto scelto dall'utente, saltando subito alla istruzione successiva alla chiamata di una procedura (Step) o percorrendo anche le singole istruzioni di questa (Trace), e così via.

È stata peraltro potenziata *Hierarchy* introdotta con la versione 1.5: allora si trattava di consentire all'utente di esaminare la gerarchia degli «oggetti» definita in un programma Turbo Pascal 5.5; ora non solo si può accedere nello stesso modo alla gerarchia di classi di un programma Turbo C++, ma, tenendo conto della possibilità di classi con *multiple inheritance,* la finestra comprende anche un pannello dedicato alla illustrazione dei possibili diversi antenati di una classe.

È invece del tutto nuova la finestra *Execution history,* che si accompagna ad una delle principali novità della versione 2.0: non solo è possibile eseguire passo passo un programma, ma anche ripercorrere a ritroso i vari passi (purché compiuti con l'istruzione Trace); basta premere il tasto Alt-F4. La nuova finestra amplia questa possibilità: nel pannello superiore sono mostrate le istruzioni fino a quel momento eseguite, tra le quali è possibile scegliere quella alla quale si vuole ritornare; in quello inferiore (se si è data l'opzione -k nella riga comando o se l'opzione è stata resa permanente con TDINST) sono mostrati tutti i tasti premuti dall'utente fino a

quel momento; selezionando una riga del pannello si può provocare l'automatico nuovo caricamento in memoria del programma e la sua esecuzione fino a quel punto. Si tratta, come si vede, di due modi estremamente comodi per rimediare a quelle situazioni (così frequenti!) in cui si è dato un comando Trace di troppo. La necessità di tenere memoria degli «stati» attraverso i quali un programma passa durante la sua esecuzione ha ovviamente un costo: il *tracing* risulta quindi spesso un po' lento. Se lo si desidera, tuttavia, il menu del primo pannello della finestra *Execution history* offre una opzione per disabilitare il *back tracing* e tornare così a ritmi più sostenuti.

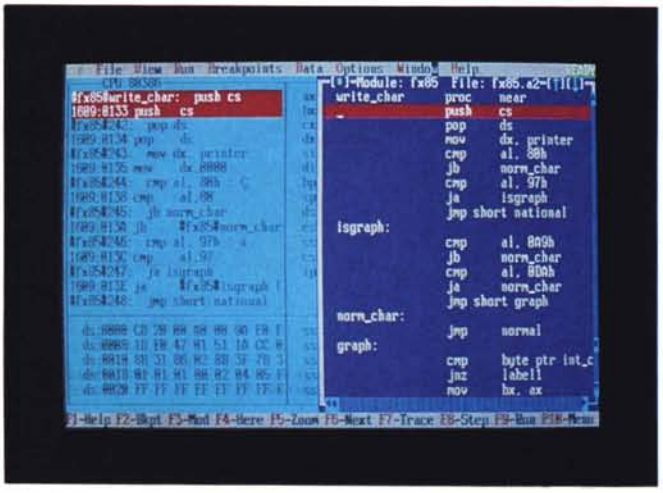


*Figura 4 — Il Turbo Debugger 2.0 consente anche il debugging di programmi residenti. Nella figura, la routine associata all'INT 28h nella unit TSR, illustrata nei mesi scorsi su MC.*

### TSR e device driver

Il menu file comprende ora tre opzioni «incredibili». Ricordiamo prima che, come ben noto, non è normalmente possibile esaminare con un debugger un programma residente; ancor meno un device driver. Nel primo caso, l'esecuzione del programma provoca solo l'installazione della sua parte residente «sopra» il debugger (cioè ad un indirizzo di memoria più alto), con l'unico effetto di costringerci poi a resettare la macchina. Nel secondo, semplicemente non c'è un programma da eseguire. Con il Turbo Debugger 2.0 invece si può.

Se si vuole mettere a punto un programma residente, si lancia il debugger normalmente, indicando il nome del TSR nella riga comando; poi si preme F9 per eseguirlo tutto d'un fiato. A questo punto si imposta un *breakpoint* nella parte residente del TSR, quindi si sceglie l'opzione *Resident* del menu *File*. Così ci si ritrova immediatamente alla prompt del DOS: lo stesso Debugger è diventato residente. Quando poi si attiva il TSR (premendo Ctrl-Shift, Alt-F11, o simili) istantaneamente si ritorna al Debugger posizionati sulla istruzione che si era selezionata; di qui si può poi proseguire come se si stesse lavorando su un programma normale. Alla fine, quando si esce dal Debugger, il TSR viene automaticamente disinstallato in modo da non costringere ad un reset.

Incredibilmente semplice ed efficace. È soprattutto incredibile che... funzioni perfettamente, come abbiamo potuto verificare con quella unit TSR.PAS che è stata illustrata su MC nei primi mesi di quest'anno.

Si può anche seguire un'altra strada, se si vuole lavorare su un programma già reso residente. Ovviamente il TSR deve essere stato compilato in modo da contenere le informazioni necessarie per il debugging simbolico (cioè con le opzioni /V o /ZI, secondo il linguaggio usato), le quali vanno però eliminate e poste in un file con estensione TDS mediante il programma di utilità TDSTRIP. Si usa poi un altro programma, TDMEM, per avere la mappa della memoria, in particolare il segmento del PSP di ogni programma presente in memoria (compresi KEYB, MOUSE, ecc.). Si fa partire il Debugger senza indicare alcun nome nella riga comando, poi si carica quel file TDS mediante l'opzione *Symbol load* del menu *File;* si dispone così dei simboli del programma e dei loro indirizzi *relativi;* per trasformare tali indirizzi in assoluti è infine necessario «rilocare» la *symbol table* del TSR mediante l'opzione *Table relocate* dello stesso menu *File*: questa chiede un indirizzo di segmento, e si deve rispondere dando l'indirizzo fornito da TDMEM. Questo dice il manuale. In realtà TDMEM fornisce il segmento del PSP del programma, che non coincide con quello in cui il TSR è stato caricato dal DOS; si deve quindi aggiungere un valore di 10h (ad esempio se TDMEM dà 1559, si deve digitare: 1569h). E comunque anche questo funziona.

Il secondo metodo è ovviamente meno comodo del primo, e viene proposto probabilmente solo perché rappresenta una variante di quello da seguire per lavorare su un device driver. Le differenze sono solo due: va usato il programma TDDEV invece di TDMEM, e soprattutto si deve disporre di due macchine. Già il Turbo Debugger 1.0 consentiva di operare con due macchine collegate mediante un cavo «null-modem» tra le rispettive porte seriali, in modo da tenere il Debugger su una e il programma da esaminare sull'altra; ora è possibile caricare il device driver su una macchina (mediante una riga nel file CONFIG.SYS) e rendere questa «remota» mediante il programma TDREMOTE; sull'altra macchina si può quindi procedere esattamente come nel caso dei TSR: reso residente il Debugger, la

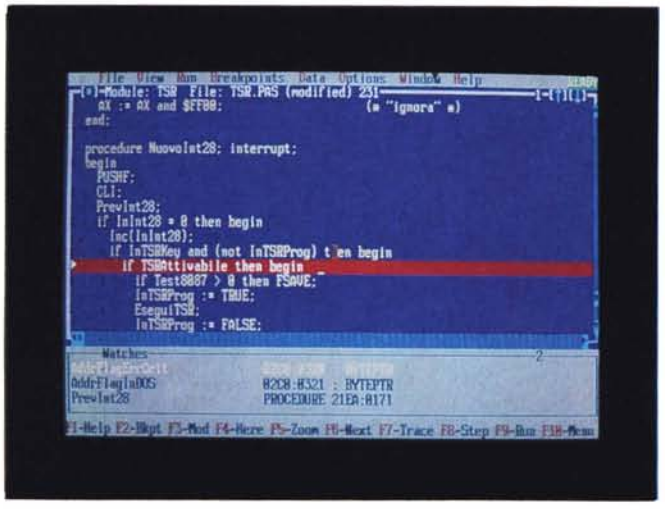


*Figura 5 — È ora possibile anche il debugging di un device driver. Nella figura, la routine di scrittura di singoli caratteri di un device driver che converte dal set di caratteri di una stampante IBM Graphics Printer a quello di una EPSON FX 85.*

macchina remota ritorna alla prompt del DOS, da cui si può quindi fare quanto necessario per attivare il device driver (abbiamo provato con un DD che avevamo preparato per poter stampare su una EPSON FX85 il set di caratteri IBM, sostituendo però il corsivo al sottolineato: si è quindi trattato di mandare alle stampe un file di prova). Fa una certa impressione ritrovarsi poi a poter seguire in ogni dettaglio l'esecuzione di codice normalmente inaccessibile.

### *Un Assembler più comodo e più efficiente*

Il Turbo Assembler 1.0 si presentava come un prodotto caratterizzato dalla notevole velocità di esecuzione e da una piena compatibilità con il Macro Assembler della Microsoft nelle sue diverse versioni, ma anche da estensioni destinate da una parte ad agevolare l'interfacciamento con C e Pascal, dall'altra a rendere più elegante e al tempo stesso più potente ed efficiente la sintassi del linguaggio (il cosiddetto modo «Ideal»). Tali caratteristiche rimangono tutte. Sottoponendo la versione 2.0 agli stessi test che vi avevamo proposto a gennaio dello scorso anno, si riscontrano tempi di assemblaggio e dimensioni dei file OBJ praticamente identici; la compatibilità con i prodotti Microsoft è stata portata fino a coprire il recente Quick Assembler (QASM); le estensioni già note rimangono, e ad esse se ne aggiungono altre. Troviamo ad esempio una direttiva PUBLICDLL che, oltre a definire PUBLIC etichette e procedure, le rende accessibili da programmi che girino sotto OS/2, consentendo così la realizzazione di *dynamic link libraries*; è ora possibile usare istruzioni PUSH e POP con più di un argomento (PUSH AX, BX invece di PUSH AX e poi PUSH BX); la direttiva COMM consente non più solo di definire variabili con «n» elementi, ma anche di precisare la dimensione di ognuno di questi (quasi come array bidimensionali).

Sono soprattutto due gli aspetti per i quali il nuovo Assembler fa registrare dei miglioramenti rispetto al predecessore. Avevamo rilevato a suo tempo che il Turbo Assembler 1.0 era in grado di convertire automaticamente tutti i salti condizionati (quali JNZ, JA, ecc.) per i quali non risultasse rispettata la distanza massima di 128 byte dalla istruzione di salto alla sua destinazione (ricordiamo che la conversione consiste nel sostituire un «JZ Zero» con un «JNZ NonZero» seguito da «JMP Zero»), ma anche che ciò avveniva efficacemente solo per i salti «all'indietro»: essendo infatti un Assembler a una sola passata, non era in grado di gestire adeguatamente i salti «in avanti», che venivano tutti convertiti in prima istanza, per essere poi riconvertiti in salti condizionati semplici se risultava rispettato il limite dei 128 byte; il codice di «aggiramento» già prodotto veniva rimpiazzato da una sequenza di tre NOP, con uno spreco di spazio e di tempo. Unica soluzione era l'uso accorto delle direttive JUMPS e NOJUMPS. La versione 2.0 dispone ora però di una opzione «/M[NumeroPassi]» che, se indicata nella riga comando, porta a «NumeroPassi» il numero massimo di passi che si devono eseguire: trasformato così il Turbo Assembler in un Assembler a più passate, l'inconveniente viene eliminato. Il prezzo da pagare per la produzione di codice più efficiente è rappresentato unicamente da un trascurabile allungamento dei tempi di assemblaggio, bilanciato peraltro dalla possibilità di abbandonarsi con maggiore libertà alla scrittura di codice con *forward reference,* anche fino a limiti che, come mostra IWHEREIS.ASM (uno dei numerosi file esemplificativi forniti con il prodotto), non sarebbero gestibili con una sola passata.

Altri miglioramenti sono di natura sintattica. Avevamo già apprezzato la disponibilità di direttive come MODEL, ARG, LOCAL e USES, nonché di estensioni della tradizionale PROC, che, secondo il linguaggio usato e il modello di memoria prescelto, agevolano sensibilmente la scrittura di moduli da linkare poi a programmi scritti in C o in Pascal. Rimanevano in pratica solo due problemi: i simboli PUBLIC contenuti in moduli da usare con programmi C dovevano essere preceduti da un trattino di sottolineatura, e soprattutto nel chiamare funzioni scritte in C o in Pascal si doveva tener conto dei diversi meccanismi richiesti dai due linguaggi per il passaggio dei parametri nello stack.

Tutto ciò limitava la possibilità di scrivere moduli in Assembler in modo indipendente dalle convenzioni dei linguaggi a cui tali moduli dovevano essere poi linkati. A ciò si è provveduto in primo luogo consentendo di specificare il «linguaggio destinazione» di direttive come EXTRN o PUBLIC: si può fare riferimento ad una funzione C *pippo* con «EXTRN C pippo», senza bisogno di quel trattino di sottolineatura che non sarebbe richiesto da una omonima funzione o procedura Pascal; quando si volesse in un secondo momento linkare il modulo ad un programma Pascal, basterebbe cambiare «EXTRN C» in «EXTRN PASCAL», senza bisogno di dare la caccia a quei trattini per tutto il sorgente. È stata poi estesa la sintassi della istruzione CALL, nel senso che ora è possibile indicare, oltre al nome della routine da chiamare, anche il linguaggio e quindi l'elenco dei parametri (ad esempio: «CALL FAR PTR pippo PASCAL, AX, BX, WORD PTR WORDVAL»): l'Assembler genererà automaticamente sia i PUSH nell'ordine corretto sia, nel caso di chiamata di una funzione C, l'incremento del registro SP subito dopo la CALL.

### *Conclusioni*

Sempre meglio. Dotato di una interfaccia utente analoga a quella del Turbo C++, che nella prova di luglio ci eravamo sentiti di definire «praticamente perfetta», arricchito della possibilità di ripercorrere a ritroso l'esecuzione di un programma, aggiornato nella sua capacità di padroneggiare anche le più complesse gerarchie di oggetti o classi, comodo da usare su qualsiasi hardware (tranne un giocattolo a due floppy da 360K) in quanto capace di sfruttarne a fondo le potenzialità, il nuovo Turbo Debugger mantiene il suo stato di strumento eccellente pur nel più avanzato contesto caratterizzato dai nuovi linguaggi orientati all'oggetto. Come se non bastasse, mette a nostra disposizione il debugging pure di programmi residenti e di device driver. Il Turbo Assembler 2.0, pur mantenendo l'originario equilibrio tra agilità d'uso, velocità ed efficienza, ha saputo superare i limiti discendenti dal compromesso iniziale acquistando la capacità di trasformarsi, con una semplice opzione nella riga comando, in un sofisticato Assembler a più passate. Allo stesso tempo, ha ulteriormente potenziato quelle caratteristiche che già in misura notevole avevano semplificato la scrittura di moduli da inserire in programmi C o Turbo Pascal. A ciò dovreste aggiungere che della confezione fa ora parte anche uno straordinario Turbo Profiler, che verrà esaminato in un prossimo numero.

La qualità complessiva del prodotto è indubbiamente elevata, nonostante il trasferimento di alcune informazioni dai manuali a tre o quattro file DOC su disco, e tale da giustificare la contropartita in denaro. Va comunque soprattutto sottolineato che la terna Debugger/Profiler/Assembler, in quanto concepita come strumento per la scrittura di programmi corretti ed efficienti in C, C++ e Turbo Pascal, dovrebbe indurre a valutare con attenzione l'acquisto di un linguaggio: ciò che in più offrono le versioni «Professional» rispetto a quelle normali viene a stento ripagato dal maggior prezzo. MC

L'informatica su misura

# Compaq & SPARTA Informatica

Sempre in prima linea nella ricerca di soluzioni informatiche ottimali, COMPAQ ha messo a punto due PC specificamente progettati per reti locali: COMPAQ DESKPRO 386N e COMPAQ DESKPRO 286N.

Entrambi i PC sono semplici da configurare e quindi sempre pronti ad essere adattati alle esigenze del momento, con schede di espansione a standard industriale, unità a dischetto, a disco fisso oppure senza disco.

La potenza elaborativa altamente sofisticata e la grande flessibilità sono unite ad una capacità di memoria espandibile su di una base di 1 megabyte RAM che arriva a 16 megabyte per il 386N ed a 13 megabyte per il 286N.

La qualità COMPAQ garantisce prestazioni eccellenti, che vengono esaltate dai possibili collegamenti con COMPAQ SYSTEMPRO e con i sofisticati server da tavolo COMPAQ.

Un'innovativa serie di funzioni di controllo di sicurezza multilivello assicura la protezione di dati e componenti interni. Il design compatto, la silenziosità ed il programma di utilità SETUP completano le caratteristiche dei due PC. Che, in più, presentano il vantaggio di un rapporto prezzo-prestazioni estremamente competitivo.

COMPAQ 386N e COMPAQ 286N: pensati per mete sempre più ambiziose e destinati ai professionisti migliori.

SPARTA INFORMATICA Via delle Sette Chiese, 142 - 00145 Roma
Tel.06/5141652-5141653-5137104-5137901• Fax:06/5126489 • Teleassistenza:06/5126752

Concessionario Autorizzato

COMPAQ