ASSEMBLER 8086 8088

Seconda parte

di Pierluigi Panunzi

Il set di istruzioni

Istruzioni di stringa

In questa puntata terminiamo l'argomento iniziato la volta scorsa e riguardante la gestione delle stringhe di dati, analizzando le ultime istruzioni che l'Assembler ci offre per tale scopo.

Le istruzioni di stringa - SCAS

È questa un'istruzione molto utile: dopo aver visto nella scorsa puntata le istruzioni di caricamento e spostamento di una stringa, questa volta abbiamo a che fare con un'istruzione che realizza un'altra operazione fondamentale, «primitiva» e cioè la scansione della stringa stessa alla ricerca di un certo elemento.

Come abbiamo già visto per le altre istruzioni sarà proprio la SCAS (che sta per «SCAn String»), nelle sue due versioni SCASB e SCASW, ad imporre che la ricerca avvenga rispettivamente su base «byte» o «word»: in entrambi i casi, da un lato c'è l'elemento i-esimo della stringa (byte o word), mentre dall'altro c'è il contenuto dell'onnipresente accumulatore (rispettivamente sotto l'aspetto di AL e di AX, come è consuetudine).

Ancora una volta per quanto riguarda l'indirizzamento dell'elemento della stringa, si deve tirare in ballo il registro di destinazione DI nonché «Extra Segment» (ES): l'elemento in esame sarà individuato (a livello di indirizzo fisico di memoria) dalla coppia «segment:offset» formata da ES:DI.

In particolare l'istruzione SCAS calcola la differenza tra il contenuto dell'accumulatore (AL o AX) ed il contenuto dell'elemento così individuato (byte o word), senza calcolare il risultato, ma bensì alterando in maniera opportuna tutti i flag, analogamente a quanto abbiamo già visto fare dall'istruzione CMP («CoMPare»).

Ulteriore fatto che ritroviamo nell'esecuzione di questa istruzione è l'aggiornamento automatico del puntatore all'elemento della stringa, grazie alla presenza del flag DF («Direction Flag»): nel caso in cui (prima dell'esecuzione della SCAS) il flag in esame sia nullo allora si avrà l'incremento automatico di DI al termine dell'esecuzione, mentre se il flag risulta settato, allora si avrà il decremento automatico.

Come la scorsa puntata crediamo di fare cosa gradita proponendo uno schemetto nel quale viene riportato il funzionamento dell'istruzione SCAS nei due casi possibili (figura A).

Per comprendere meglio il funzionamento di questa istruzione, molto semplice da usarsi, ma anche molto utile, come dicevamo, analizziamo il problema di ricercare se all'interno di una stringa di comando è presente un «blank», separatore ad esempio del comando da un parametro.

Useremo anche nell'esempio B l'istruzione-prefisso REPZ, della quale parleremo in dettaglio nel corso della puntata per formalizzarne le caratteristiche.

Analizziamo dunque il piccolo programma: innanzitutto nel segmento di dati abbiamo definito direttamente «COMMAND» come una stringa di 40 byte grazie alla direttiva DB (Define Byte), stringa che poi in realtà risulterà riempita da un nostro comando effettivamente impostato.

Nel programma vero e proprio, dopo l'inizializzazione del registro ES al segmento di dati e DI all'offset della stringa, abbiamo resettato il flag DF per mezzo dell'istruzione CLD in modo da effettuare la scansione della stringa per indirizzi successivi.

Inoltre abbiamo posto in CX il numero di byte costituenti la stringa COMMAND in esame che per generalità è stato definito con una direttiva EQU.

Tale valore in CX indicherà poi che la SCASB (grazie al prefisso REPZ) verrà eseguita al massimo un numero di volte pari alla lunghezza della stringa stessa: la lettera «Z» nel prefisso iterativo (come vedremo in dettaglio nel seguito della puntata) farà si che il

```
Istruzione SCASB
----------------

   AL - ES:DI

if DF = 0
 then DI  <--  DI + 1
 else DI  <--  DI - 1


Istruzione SCASW
----------------

   AX - ES:DI

if DF = 0
 then DI  <--  DI + 2
 else DI  <--  DI - 2
```

A

```
LUNGHEZZA EQU 40

DATA      SEGMENT
COMMAND   DB LUNGHEZZA DUP(?)
DATA      ENDS

CODE      SEGMENT
          ASSUME CS:CODE,ES:DATA
          MOV AX,DATA
          MOV ES,AX
          MOV DI,OFFSET COMMAND
          CLD
          MOV CX,LUNGHEZZA    ;numero max di iterate
          MOV AL,20H          ;il "blank"
          REPZ SCASB
          JZ TROVATO
          ....                ;non trovato
          ...
TROVATO   DEC DI              ;correzione del puntatore
          ...
CODE      ENDS
```

B

loop di confronto termini allorché si abbia la coincidenza tra l'elemento generico della stringa e l'accumulatore.

Sarà la condizione «di Zero», rappresentata dal flag ZF settato, a far appunto abbandonare l'esecuzione del loop per far poi saltare all'etichetta «TROVATO»: sappiamo infatti che l'istruzione SCASB sottrae il valore attuale dell'elemento della stringa (quello puntato da DI) dal contenuto costante dell'accumulatore ed in caso di identità tra i due termini viene appunto settato il flag di Zero.

Dal momento che l'istruzione SCASB effettua comunque al termine l'aggiornamento del puntatore DI, ecco che il valore in esso contenuto sarà, a meno di un'unità, l'offset della cella in cui si ha il valore contenuto nell'accumulatore: in particolare se il flag DF era resettato (tramite l'istruzione CLD, è il nostro caso), allora bisognerà incrementare DI con una «DEC DI», mentre viceversa il flag DF era stato settato da un'istruzione STD (indicante il decremento automatico), allora alla fine bisognerà incrementare DI (con una «INC DI»), per avere in entrambi i casi il valore corretto.

Nel caso invece di istruzione SCASW, il valore finale di DI (in caso di scansione terminata con successo) sarà valido a meno di due unità, dal momento che, lo ricordiamo, l'istruzione in esame incrementa o decrementa di due unità il contenuto di DI dopo aver effettuato la sottrazione: lasciamo perciò ai lettori il facile compito di arguire quali operazioni compiere su DI per riportarlo al valore esatto «di coincidenza».

Invece nel caso in cui non si abbia nessuna coincidenza tra gli elementi della stringa ed il contenuto dell'accumulatore, allora il loop verrà eseguito un numero di volte pari al contenuto iniziale del registro CX ed al termine il registro DI punterà al byte o alla word successiva all'ultimo byte o word della stringa in esame (nel caso che il flag DF fosse resettato all'inizio), mentre viceversa il registro DI punterà al byte o alla word precedenti il primo byte o la prima word della stringa.

Le istruzioni di stringa - CMPS

L'ultima istruzione di stringa, la CMPS, permette di effettuare la comparazione («CoMPare Strings») tra due stringhe, termine a termine (byte a byte o word a word) ogni volta effettuando la sottrazione tra i due elementi, senza effettuare il calcolo, ma settando i flag opportuni ed andando poi ad incrementare o decrementare i due registri puntatori.

Ma andiamo con ordine...

Innanzitutto i due elementi sono puntati rispettivamente da «(DS:)SI» e da «ES:DI», come è consuetudine nel caso delle stringhe, laddove abbiamo indicato con «(DS:)» il fatto che per default SI punta nel Data Segment, mentre viceversa DI va a puntare all'interno dell'Extra Segment.

Per quanto riguarda la sottrazione tra i due elementi, vale quanto detto per l'istruzione precedente ed a quelle considerazioni aggiungiamo che il programmatore non deve per forza servirsi del flag di Zero, ma a scelta di qualunque altro flag gli possa servire: supponendo che si voglia vedere che una stringa è «minore» di un'altra, ecco che il flag di Z non servirà granché, al contrario del Carry.

Infine, a seconda se il flag DF è settato o meno, l'istruzione provvederà a, rispettivamente, decrementare o incrementare entrambi i registri indice di un'unità o di due unità a seconda se rispettivamente si tratti di un'istruzione CMPSB o di una CMPSW in quanto anche in questo caso l'operazione di comparazione può avvenire sia tra stringhe di byte che tra stringhe di word.

Come al solito riportiamo nelle seguenti tabelline una rappresentazione schematica di quanto eseguito dall'istruzione in esame:

```
Istruzione CMPSB
----------------

  DS:SI - ES:DI

if DF = 0
 then SI  <--  SI + 1 ; DI  <--  DI + 1
 else SI  <--  SI - 1 ; DI  <--  DI - 1

Istruzione CMPSW
----------------

  DS:SI - ES:DI

if DF = 0
 then SI  <--  SI + 2 ; DI  <--  DI + 2
 else SI  <--  SI - 2 ; DI  <--  DI - 2
```

Va notato infine l'ordine con cui viene effettuata la sottrazione tra i due termini: l'elemento puntato da ES:DI viene sottratto dall'elemento puntato da DS:SI, fatto che risulta di fondamentale importanza allorché, come già accennato, si debba considerare non tanto l'uguaglianza tra due stringhe, ma la loro comparazione in termini di «maggiore» o «minore».

Per vedere ora un esempio di applicazione dell'istruzione di comparazione tra stringhe, facciamo qualcosa di nuovo rispetto a quanto fatto precedentemente: cioè partiamo dal problema che risolveremo con una subroutine.

Supponiamo dunque di voler scrivere una subroutine che effettui la comparazione tra due stringhe, entrambe poste nel Data Segment, i cui indirizzi (offset) ci vengono forniti rispettivamente in SI e DI e le cui lunghezze sono poste a loro volta nei registri BX e DX: già da questo vediamo che non abbiamo alcun «sentore» di quello che esiste nel Data Segment, né tantomeno sappiamo «il nome» delle due stringhe...

Come output dovremo fornire il flag di Zero settato nel caso di identità delle stringhe oppure il flag di Carry settato se la prima stringa (quella puntata da SI) risulta minore della seconda: supponiamo per semplicità di sapere che si tratta di stringhe di caratteri ASCII, ma ciò è solo un dettaglio ininfluente.

La nostra subroutine, che abbiamo battezzato «EQSB» (da «EQual String Byte») sarà dunque la seguente:

```
EQSB    PROC

        PUSH DS
        POP ES
        CLD
        CMP BX,DX
        MOV CX,BX
        JBE OK
        MOV CX,DX
OK:     CMPSB
        JNZ EXIT
        LOOP OK
        CMP BX,DX
EXIT:   RET

EQSB    ENDP
```

Dodici istruzioni... forse si può fare di meglio?! Comunque a parte considerazioni riguardo all'uso di una certa istruzione invece di un'altra a seconda dei propri gusti, a noi il programma, da una ventina di istruzioni nella versione iniziale, è riuscito così ed ora passeremo ad analizzarlo.

Innanzitutto abbiamo detto che le due stringhe sono puntate rispettivamente dai registri SI e DI, all'interno del Data Segment: ecco perché dunque bisogna inizializzare il registro ES (nel nostro caso deve puntare proprio all'interno del Data Segment) dal momento che così è richiesto dalle istruzioni di stringa per la coppia ES:DI.

A proposito: avete notato che in tutta la subroutine non si trova traccia dei due registri SI e DI?! Potenza delle istruzioni di stringa...

Dopo aver dunque resettato il flag di «Direction» per poter scandire le due stringhe verso indirizzi crescenti, dobbiamo innanzitutto controllare i valori posti nei registri BX e DX e cioè le lunghezze delle due stringhe: dato che per forza di cose la comparazione avviene tra elementi «esistenti» delle due stringhe, allora il numero massimo di confronti che dovremo effettuare sarà pari alla lunghezza della stringa più corta.

Perciò confrontiamo i valori contenuti in BX e DX e se il confronto sancisce che BX è minore o uguale di DX, allora nel registro «contatore» CX andrà proprio BX, mentre nel ca-

so in cui DX sia minore sarà lui a riempire il registro CX.

Arrivati dunque all'etichetta «OK» avremo:

— in CX il numero massimo di confronti da effettuare

— in DS: SI l'indirizzo della prima stringa

— in ES: DI l'indirizzo della seconda stringa

— in DF il valore 0

per cui siamo pronti per iniziare il confronto, che avviene dunque con la CMPSB.

Grazie al fatto che questa istruzione setta i flag a seconda dell'esito del confronto ci porterà subito a dire che, se il flag di Zero non è settato, allora le stringhe sono differenti e si uscirà dalla subroutine dall'etichetta «EXIT», badate bene, già con il corretto valore posto nel Carry!

In definitiva uscendo a questo punto, il flag di Zero non è settato ed il Carry viceversa è settato se la prima stringa è minore della seconda.

Viceversa se l'elemento fin qui controllato della prima stringa è identico al corrispondente della seconda stringa (stiamo all'interno di un loop e perciò si parlerà dell'elemento «i-esimo» della stringa e non già del «primo») allora il flag di Zero sarà settato e potremo eseguire l'istruzione successiva, la «LOOP OK», sulla quale torneremo forse già nella prossima puntata.

In parole povere l'istruzione in esame decrementa il contenuto del registro CX e finché non ottiene «0» fa saltare all'etichetta indicata, nel nostro caso «OK», il tutto senza alterare lo stato dei flag.

Ora si può vedere che il ciclo verrà eseguito per tutto il tempo che la prima stringa ha elementi identici a quelli della seconda: si potrà uscire dal loop solo nei due casi in cui da un certo punto in poi le stringhe sono differenti (ed è il caso già incontrato in precedenza) oppure se abbiamo analizzato per tutta la lunghezza della stringa più corta.

Ma a questo caso si aggiunge anche quello in cui le due lunghezze coincidevano e perciò ora basterà fare il confronto tra i valori originari di BX e DX per decidere quale delle stringhe è maggiore dell'altra ottenendo tra l'altro automaticamente il corretto settaggio dei flag di Zero e di Carry!

A questo punto dunque le due stringhe hanno tutti i caratteri identici e perciò sarà «minore» la stringa che ha la lunghezza minore: se BX è minore di DX allora verrà settato il Carry, proprio come volevamo.

Ma se le due stringhe avevano uguale lunghezza allora, dopo essere già usciti dal loop con il flag di Zero settato, il confronto tra BX e DX in un certo senso «rafforzerà» lo stato del flag di Zero...

Non contenti diciamo che il programma non si accorcia neanche nel caso in cui si decida di porre la lunghezza della prima stringa in CX, sfruttando le seguenti istruzioni per determinare il valore minore tra CX e DX:

```
        CMP CX,DX
        JBE OK
        XCHG CX,DX
OK:     ...
```

in quanto il valore contenuto in CX non rimane costante nel corso del programma ed allora il confronto finale dovrebbe avvenire comunque tra DX (rimasto inalterato) ed un altro registro (ancora BX per esempio), che deve però essere inizializzato subito con il valore di CX: l'istruzione che si risparmia con le tre istruzioni precedenti si ripaga appunto poi con l'inizializzazione di BX.

A questo punto tralasciamo, in quanto molto banale, di dare un esempio di un programma «chiamante», che dovrà provvedere ad inizializzare correttamente i registri puntatori DI e SI, nonché i due registri contenenti le lunghezze delle due stringhe e cioè BX e DX.

La ripetizione di istruzioni di stringa - REP

Come già abbiamo avuto modo di vedere, l'istruzione REP (che sta per «REPeat») altro non è che un «prefisso» anteposto ad un'istruzione di stringa che ne permette l'iterazione condizionata al contenuto del registro CX: per tutto il tempo che CX è diverso da «0», l'istruzione di stringa viene ripetuta.

Dalla tabellina seguente vediamo il comportamento del prefisso REP con le istruzioni di stringa:

```
Prefisso REP
------------

while CX <> 0
 execute string operation
 CX <- CX - 1
wend
```

Dalla tabella si vede che «prima» viene eseguita l'istruzione di stringa e «poi» viene decrementato il valore di CX, il tutto a patto che CX non sia nullo: perciò si potranno al massimo effettuare 65535 iterazioni (con un valore di CX pari a OFFFFH) e non 65536 in quanto un valore iniziale nullo di CX non fa eseguire per niente l'istruzione di stringa.

Per quanto riguarda le istruzioni che si possono ripetere, notiamo che mentre è «logico» ripetere istruzioni di spostamento (MOVS) e di memorizzazione o inizializzazione (STOSB), che devono essere eseguite in blocco per CX volte, non è molto «logico» effettuare comparazioni (CMPS) e scansioni (SCAS) in quanto entrambe le operazioni verrebbero eseguite comunque «ciecamente» fino alla fine senza che si possa sfruttare il fatto che si è avuta ad esempio un'uguaglianza tra i due termini a confronto.

Per finire è completamente «illogica», ancorché possibile da eseguire (!), è la ripetizione del caricamento dell'accumulatore (LODS), il quale evidentemente verrebbe continuamente ed inutilmente caricato con valori differenti. A meno che non si voglia inserire in loop di ritardo...

Abbiamo inoltre visto che esistono anche i «prefissi di iterazione condizionati», ottenibili per mezzo delle istruzioni REPZ o REPE (che stanno per «REPeat if Zero» o il che è lo stesso «REPeat if Equal»), contrapposte alle REPNZ o REPNE (che stanno per «REPeat if Non Zero» e «REPeat if Not Equal»), le quali hanno un'effettiva utilità se associate alle due istruzioni di stringa che alterano il flag di Zero e cioè la CMPS e la SCAS.

I primi due prefissi (perfettamente identici dal punto di vista dell'opcode) consentono di continuare l'esecuzione dell'iterazione fintantoché il flag di Zero risulta settato, mentre una condizione di «Non Zero» comporta l'arresto del loop.

Viceversa gli altri due prefissi permetteranno l'iterazione fino a che il flag di Zero non venga settato per effetto di una coincidenza tra i due termini in confronto.

A tale proposito si vede che è «illogico» usare le «REP condizionate» con le altre istruzioni di stringa che non settano i flag.

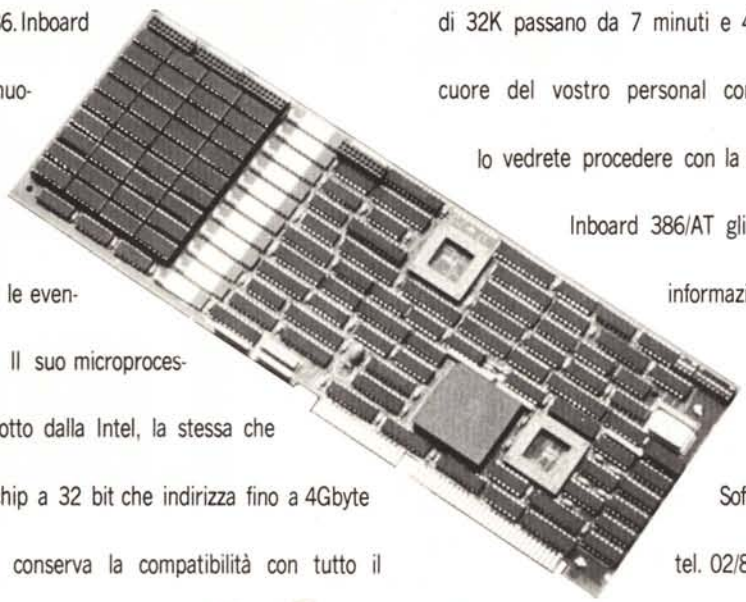
Diamo ora le due tabelline relative alle due coppie di prefissi or ora analizzati.

```
Prefisso REPZ o REPE (con CMPS e SCAS)
--------------------

while CX <> 0 and ZF = 0
 execute CMPS or SCAS
 CX <- CX - 1
wend

Prefisso REPNZ o REPNE (con CMPS e SCAS)
----------------------

while CX <> 0 and ZF <> 0
 execute CMPS or SCAS
 CX <- CX - 1
wend
```

Con questo abbiamo terminato l'analisi delle istruzioni di stringa: dalla prossima puntata analizzeremo le importantissime istruzioni di «controllo» del flusso di esecuzione di un programma, in generale i salti condizionali e non, le chiamate a e i ritorni da subroutine, i loop, ecc. MC

Inboard 386/AT. Bit bit hurrà!

La tristezza era calata sul mondo dei personal con un cuore 80286, da quando erano apparsi i microprocessori a 32 bit. Ma oggi è un giorno di festa: c'è la scheda Intel Inboard 386, che dà agli AT e compatibili una potenza di elaborazione confrontabile solo con quella dei computer basati sul chip 80386. Inboard costa molto meno di un nuovo personal, e funziona con tutte le applicazioni del PC DOS e con le eventuali schede opzionali. Il suo microprocessore è un 80386 prodotto dalla Intel, la stessa che lo ha inventato. È un chip a 32 bit che indirizza fino a 4Gbyte di memoria, conserva la compatibilità con tutto il software scritto per i suoi predecessori, ed è pronto a sfruttare i programmi che verranno creati per la nuova generazione dei PC. E, lavorando a 16 MHz, dimezza la durata della maggior parte delle operazioni. Un esempio: l'eliminazione delle trasparenze e la stampa di un file AutoCAD di 32K passano da 7 minuti e 41 secondi a 2'14. Sostituite il cuore del vostro personal computer: finalmente sollevato, lo vedrete procedere con la potenza e la velocità che solo Inboard 386/AT gli può dare. Se volete maggiori informazioni rivolgetevi ai migliori rivenditori oppure telefonate o scrivete a EIS - Editrice Italiana Software - Via Fieno,8 - 20123 Milano tel. 02/805.32.67 oppure 805.70.09.



intel®

Editrice Italiana Software DIVISIONE PRODOTTI SPECIALI

TRA IL PIÙ DIFFUSO ED IL NUOVO ESISTONO UNMILIO TOTTOMILASET DIFFE LIRA PIÙ

Sí, Amstrad PC-1512, nella sua versione base ti costa veramente meno, solo L. 1.390.000. Questo è un prezzo incredibilmente basso se consideri che comprende, oltre alla tastiera, al monitor grafico, all'unità di sistema e al drive per dischi, anche il mouse e ben quattro dischi di software.
Le prestazioni poi, sono un'altra meraviglia.
Amstrad PC-1512, confrontato a personal molto più costosi, esprime prestazioni superiori.
Infatti è decisamente più veloce e la sua memoria, di ben 512K RAM, è espandibile a 640K.
Puoi scegliere il tuo PC-1512 tra sei differenti versioni, con drive singolo, doppio, oppure con hard disc da 20 Mbyte.
Il monitor può essere scelto tra quello monocromatico a 16 toni di grigio o quello a 16 colori.
Sei configurazioni, sei modi intelligenti per garantirsi il massimo, senza per questo dover investire il capitale che altri personal ti chiedono. Amstrad PC-1512, ti dà tanto di più, ma ti chiede molto di meno.

Se il PC-1512 vi interessa e desiderate ricevere ulteriori informazioni, spedite il tagliando compilato a:
G.B.C. Italiana S.p.A. - Viale Matteotti, 66
20092 Cinisello Balsamo (MI) - Tel. (02) 61.81.801

MC.M.

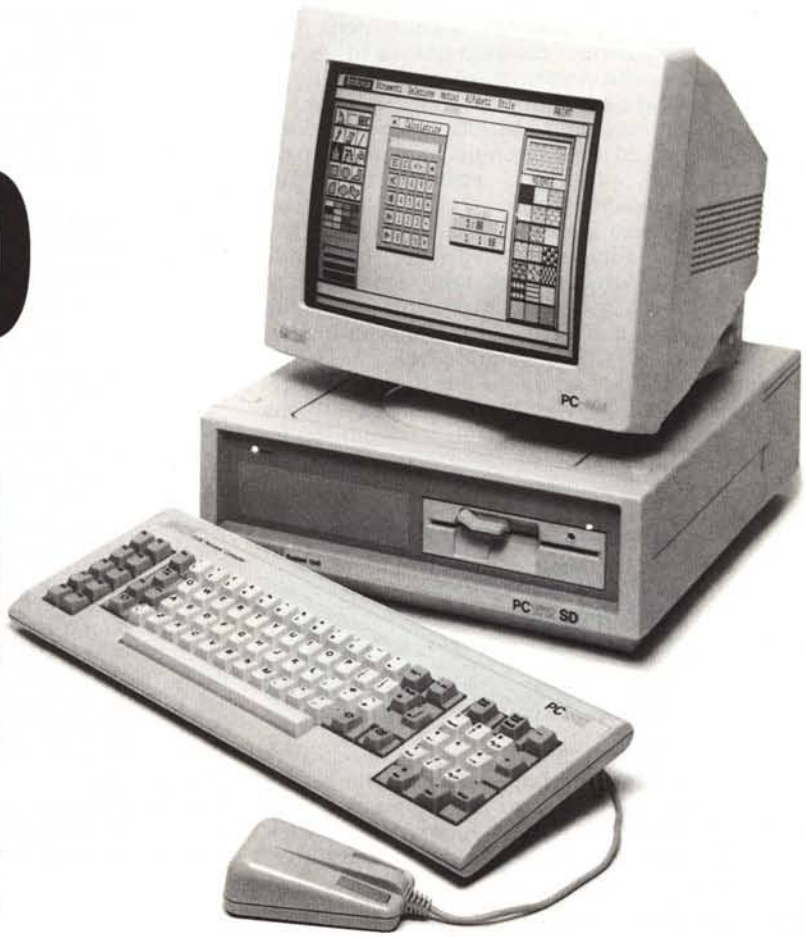
Cognome

Nome

Via N°

Città CAP.

PERSONAL COMPUTER AMSTRAD PC 1512 NETRECENTOQUARAN TECENTONOVANTA RENZE LIRA MENO



AMSTRAD

PC-1512 Versione Italiana

L. 1.390.000 + IVA

Distribuito in esclusiva da G.B.C. Italiana S.p.A.